PUBLICLY
AVAILABLE
SPECIFICATION

Pre-Standard

IEC PAS 62453-1

First edition
2006-05

Field Device Tool (FDT) interface specification –

Part 1:
Concepts and detailed description

Reference number
IEC/PAS 62453-1:2006(E)

Publication numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series. For example, IEC 34-1 is now referred to as IEC 60034-1.

Consolidated editions

The IEC is now publishing consolidated versions of its publications. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Further information on IEC publications

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology. Information relating to this publication, including its validity, is available in the IEC Catalogue of publications (see below) in addition to new editions, amendments and corrigenda. Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is also available from the following:

- **IEC Web Site (www.iec.ch)**

- **Catalogue of IEC publications**

  The on-line catalogue on the IEC web site (www.iec.ch/searchpub) enables you to search by a variety of criteria including text searches, technical committees and date of publication. On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.

- **IEC Just Published**

  This summary of recently issued publications (www.iec.ch/online_news/ justpub) is also available by email. Please contact the Customer Service Centre (see below) for further information.

- **Customer Service Centre**

  If you have any questions regarding this publication or need further assistance, please contact the Customer Service Centre:

  Email: custserv@iec.ch
  Tel: +41 22 919 02 11
  Fax: +41 22 919 03 00

PUBLICLY
AVAILABLE
SPECIFICATION

IEC PAS 62453-1

Pre-Standard

First edition
2006-05

Field Device Tool (FDT) interface specification –

Part 1:
Concepts and detailed description

International Electrotechnical Commission, 3, rue de Varembé, PO Box 131, CH-1211 Geneva 20, Switzerland
Telephone: +41 22 919 02 11 Telefax: +41 22 919 03 00 E-mail: inmail@iec.ch Web: www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

PRICE CODE XH

For price, see current catalogue

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

_____

Field Device Tool (FDT) interface specification –

Part 1: Concepts and detailed description

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
6) All users should ensure that they have the latest edition of this publication.
7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

A PAS is a technical specification not fulfilling the requirements for a standard but made available to the public.

IEC-PAS 62453-1 has been processed by subcommittee 65C: Digital communications, of IEC technical committee 65: Industrial-process measurement and control.

The text of this PAS is based on the following document:	This PAS was approved for publication by the P-members of the committee concerned as indicated in the following document

Draft PAS	Report on voting
65C/398A/NP	65C/411/RVN

Following publication of this PAS, which is a pre-standard, the technical committee or subcommittee concerned will transform it into an International Standard.

This PAS shall remain valid for an initial maximum period of three years starting from 2006-05. The validity may be extended for a single three-year period, following which it shall be revised to become another type of normative document or shall be withdrawn.

IEC 62453 consists of the following parts under the general title *Field Device Tool (FDT) interface specification:*

Part 1: Concepts and detailed description

Part 2: INTERBUS communication

Part 3: PROFIBUS communication

Part 4: HART communication

Part 5: FOUNDATION FIELDBUS communication

INTRODUCTION

This PAS 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 inter-operate seamlessly.

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.

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.

The attention of the reader is drawn to the fact that, throughout this PAS, references to versions other than Version 1.2.1 (this PAS) relate to earlier versions of this document published by the FDT Group (http://www.fdt-jig.org). For a list of changes introduced between Version 1.2 and this PAS, see Annex F.

Field Device Tool (FDT) interface specification –

Part 1: Concepts and detailed description

1 Scope

1.1 Performance

This part of IEC 62453 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 such a way as to support all significant fieldbus protocols. The main focus of this version of FDT does not lie on any specific fieldbus protocol. Due to the scalable structure of FDT it will be easy to add new schemas in order to extend the scope to new protocols.

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

This PAS is intended for developers who want to implement FDT components.

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.

1.2 Audience

This PAS 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[1] technology and the needs of the Process Control industry or of the field devices.

This PAS 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]

---

[1] Microsoft ActiveX is the trade name of the a product supplied by Microsoft. This information is given for convenience of users of this document and does not constitute an endorsment by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

[2] Microsoft COM is the trade name of the a product supplied by Microsoft. This information is given for convenience of users of this document and does not constitute an endorsment by IEC of the product named. Equivalent products may be used if they can be shown to lead to the same results.

2 Normative references

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

ISO/IEC 19501:2005 *Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
ActiveX
component technology based on the Microsoft Component Object Model (COM/DCOM). Former standard was OLE controls (OCX)

3.2
Addressing
communication protocol specific identifier. Due to that the namespace, the format, etc., is defined by the given protocol

3.3
Channel
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

3.4
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

3.5
Communication Channel
device can be connected to. In general such a channel is the master of an underlying fieldbus system

3.6
Configuration
System configuration created by configuring the plant components and the topology

3.7
Configure
setting parameters at the instance data set as well as the logical association of plant components to build up the plant topology (offline)

3.8
Connection
established data path for communication with an selected device

3.9
Data
xxxx See transient- and persistent data

3.10
DCS Manufacturer / System Manufacturer
in this context the manufacturer of the engineering system

3.11
Device
piece of hardware which can be connected to a FIELDBUS system. It can be either a master, a slave, a bus coupler, or...

3.12
Device manufacturer
manufacturer of fieldbus devices, usually slave devices

3.13
Document
to document means to generate the documentation

3.14
Documentation
documentation is the human readable information about a device instance. Within the context of FDT, the printed documentation and the documentation provided via XML as well is meant. The documentation can consist of several documents

3.15
Device Type Manager (DTM)
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

3.16
Fielddevice
sensor or actor for measuring or positioning as a plant component

3.17
Frame Application
represents the components which build the DTM environment. This can be an engineering tool, a stand-alone tool or a web page

3.18
FDT Model
model to describe the interaction between DTM and Frame Application

3.19
GSD
text file containing a description of e.g. the PROFIBUS device with a predetermined syntax. It is delivered with e.g. the PROFIBUS device or can be obtained from the manufacturer

3.20
Multi user environment
environment which allows that more than one DTM instance can get access to an identical instance data set

3.21
Net
single bus system or a group of connected bus systems

3.22
Nested Communication
communication path built up by a cascaded sequence of communication channels

3.23
Parameterization
setting parameters at a device according to the task of the device (online)

3.24
Persistent Data
permanent stored data

3.25
Process
industrial process accessed through field devices. Data acquisition and control

3.26
Project Design
configuration of devices and process units to build up a net

3.27
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 within the responsibility of the Frame Application to guarantee the data consistency within a session

3.28
Transient Data
temporary data during configuration. Turn to persistent data after storage

4 Abbreviations

DCS	Distributed control system
DTM	Device Type Manager.
MIDL	Microsoft Interface Definition Language
OPC	OLE for Process Control
PLC	Programmable Logic Controller
SCADA	Supervisory/System control and data acquisition
UML	Unified Modelling Language
XML	Extensible Markup Language

5 Concept

5.1 Status

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.

5.2 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 (see Figure 1). 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.

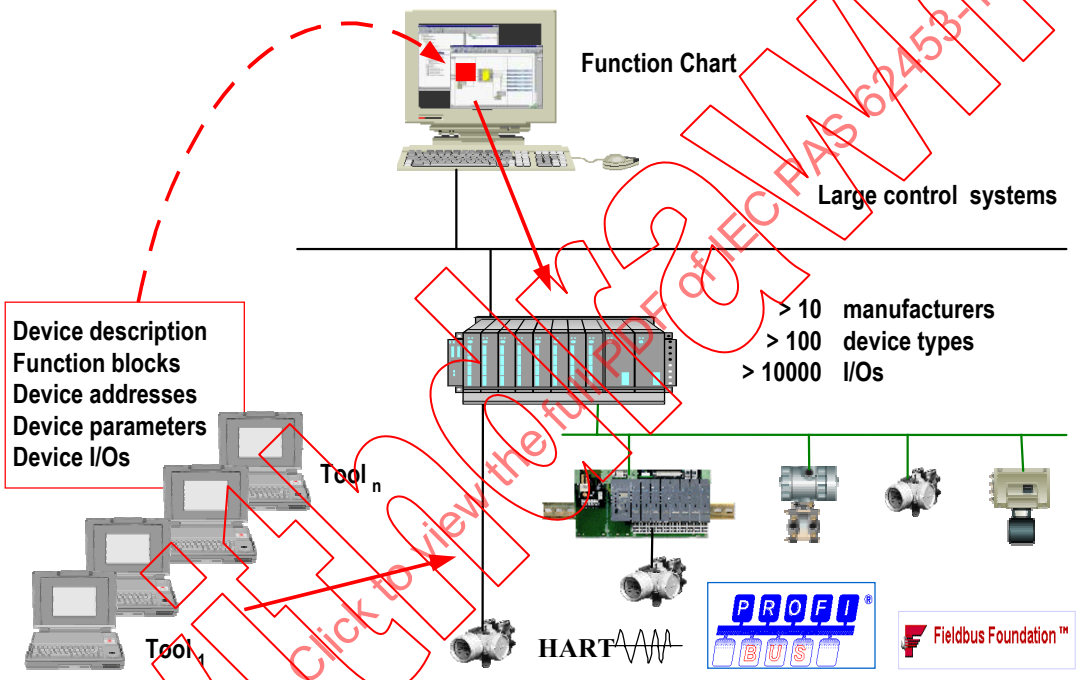

Figure 1 – Different tools and multiple data input have determined field device integration to date

5.3 Aims

In order to maintain the continuity and operational reliability of process control technology, it is necessary to fully integrate fieldbus devices as a subcomponent 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 are the following.

- 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 (see Figure 2). 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.

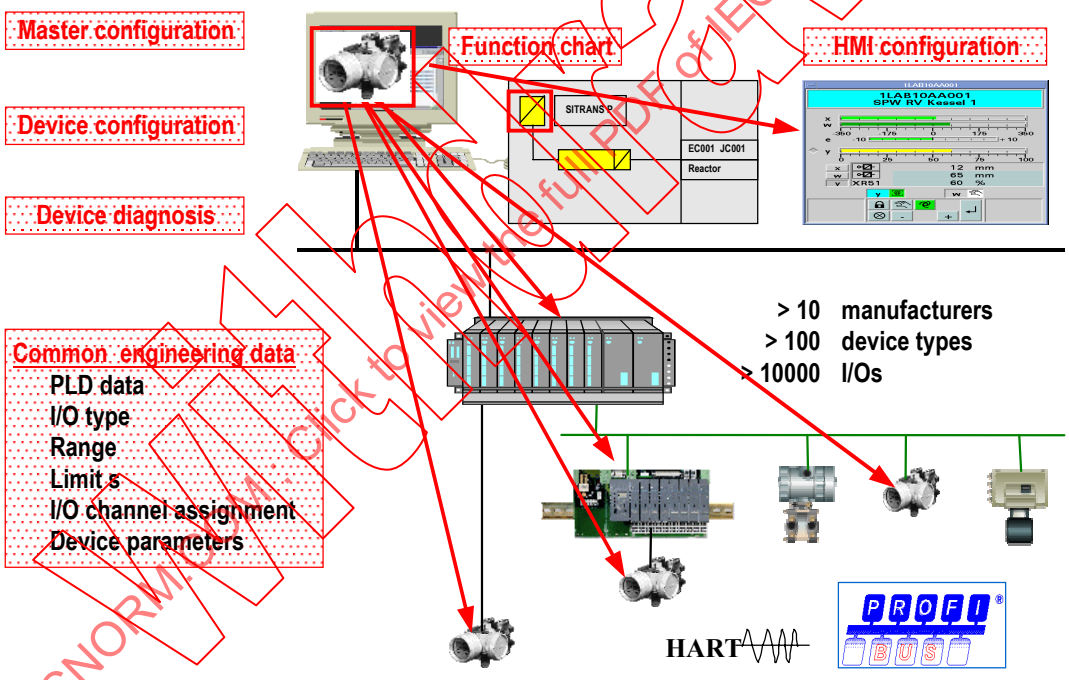

Figure 2 – The potential of the field bus technology cannot be used until the field bus has been homogeneously integrated into the engineering systems.

5.4 Technological orientation

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

Moreover, as Microsoft Windows has established itself as the *de facto* industry standard, there are few alternatives when choosing the operating system. The ActiveX technology introduced by Microsoft makes it possible to define interfaces which contain not only data but also functions. These possibilities have already been successfully used in conjunction with OPC (OLE for Process Control) definition.

Within FDT only interfaces in ActiveX technology are specified for the engineering components of field devices. If the engineering system implements the corresponding interfaces, the ActiveX technology provides the automatic integration of the components and takes care of the interaction between the engineering system and the software components of the devices. Furthermore, the FDT interface specification allows the integration of device components with integrated user interfaces as well as the embedding of ActiveX controls provided by the device component for special engineering tasks.

The underlying object model is based on a client-server architecture that is easily extensible for future functionality. The data exchange between the devices' software components and the engineering system is performed by means of XML (eXtensible Markup Language) which also allows for easily extending the content of the information that is exchanged in later versions of FDT. The implementation of FDT´s client server architecture is based on Microsoft COM.

5.5 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 stand-alone 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 stand-alone operation may be implemented in special cases for example when migrating from a stand-alone tool to a DTM. For reasons of data consistency, parallel operation of stand-alone solutions and DTMs running in the system's engineering tool accessing the same devices are not intended. Stand-alone 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 to 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

5.6 Migration to Device Type Manager (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 does not 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 stand-alone tools. FDT provides the openness to equip these tools with the FDT-interfaces and to build DTMs out of existing stand-alone tools. That way, the device manufacturer's investments can be protected. DTMs that are equipped with external tools are seen as a way to migrate towards FDT. The final goal of such a migration should be a DTM, which provides a well integrated ActiveX-based GUI.

In the long run, each device manufacturer has the freedom to choose from the following options for existing or new devices (see Figure 3).

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

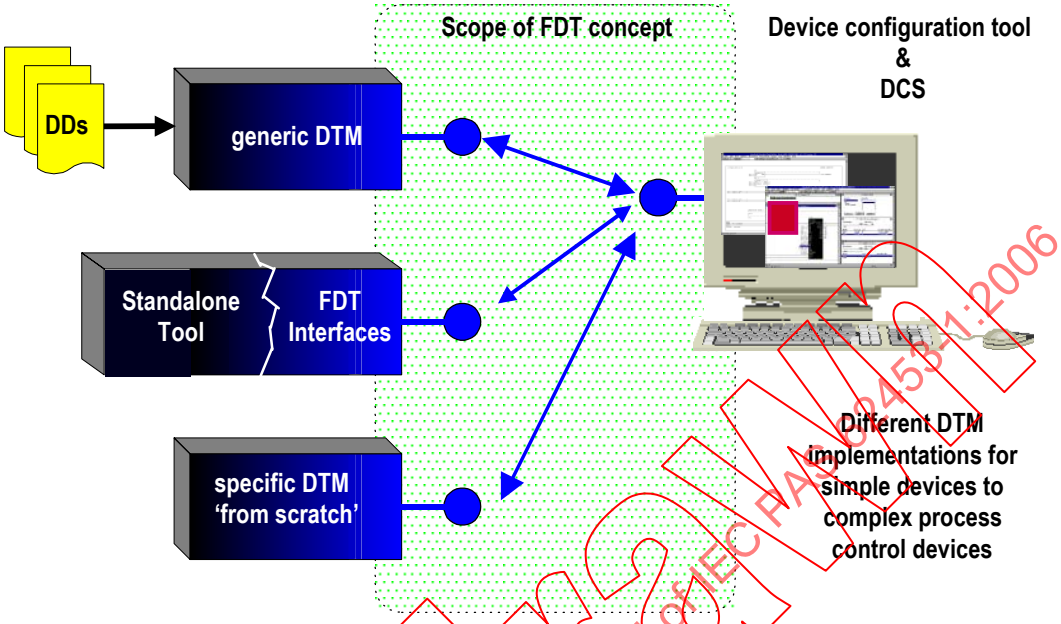

Figure 3 – DTM - implementations

6 FDT fundamentals

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

6.1 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 (see Figure 4).

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.

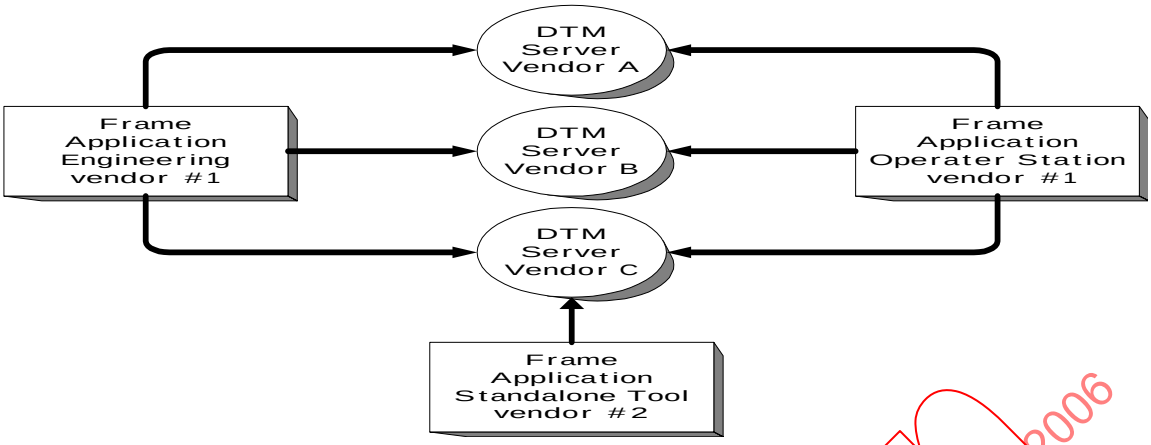

Figure 4 − General FDT Client/Server relationship

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

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

Within each channel the DTM can define one or more fieldbus specific parameters that give information about the channel like data types, ranges, alarms, etc. (see Figure 5).

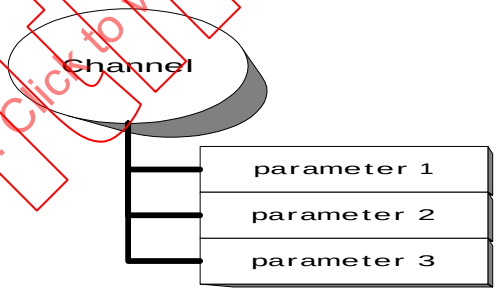

Figure 5 − Channel/Parameter relationship

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 subclause 10.2 Nested communication.

6.2 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.

6.3 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 in Figure 6.

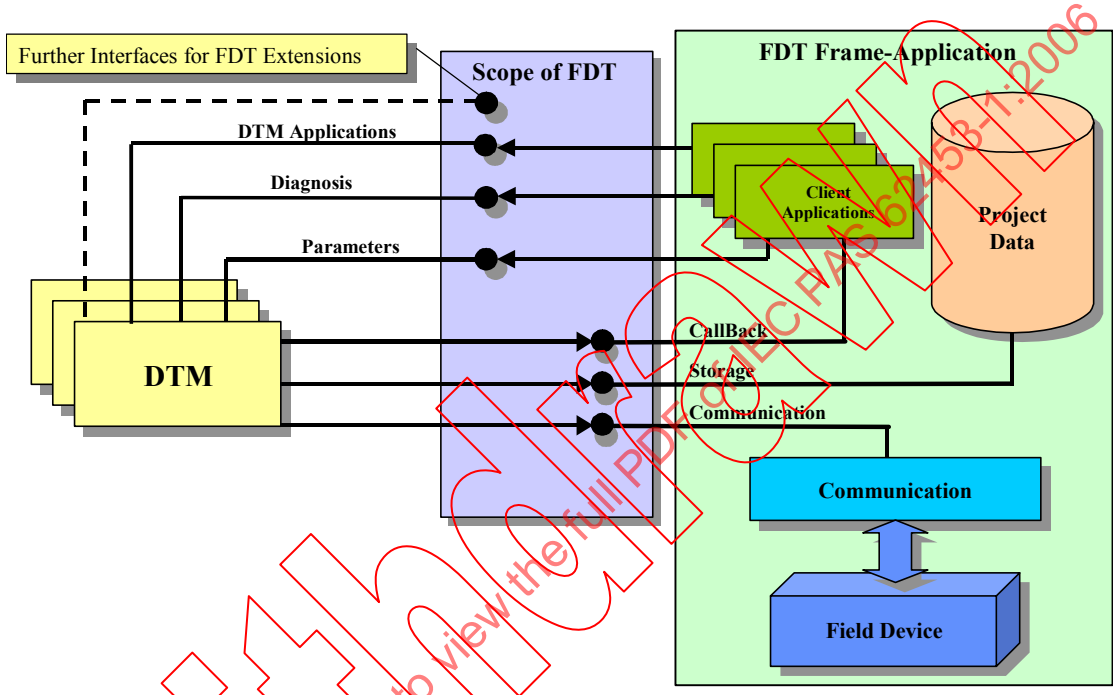

Figure 6 – FDT interfaces

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 stand-alone tools for device configuration. These applications are called 'Frame Applications'. Throughout this PAS 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 which is the server.

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

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

6.4 Overview of objects and interfaces

6.4.1 The Device Type Manager (DTM)

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

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

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

The interfaces IDtm and IDtmInformation have to be implemented by each DTM (see Figure 7). 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 in its own clause.

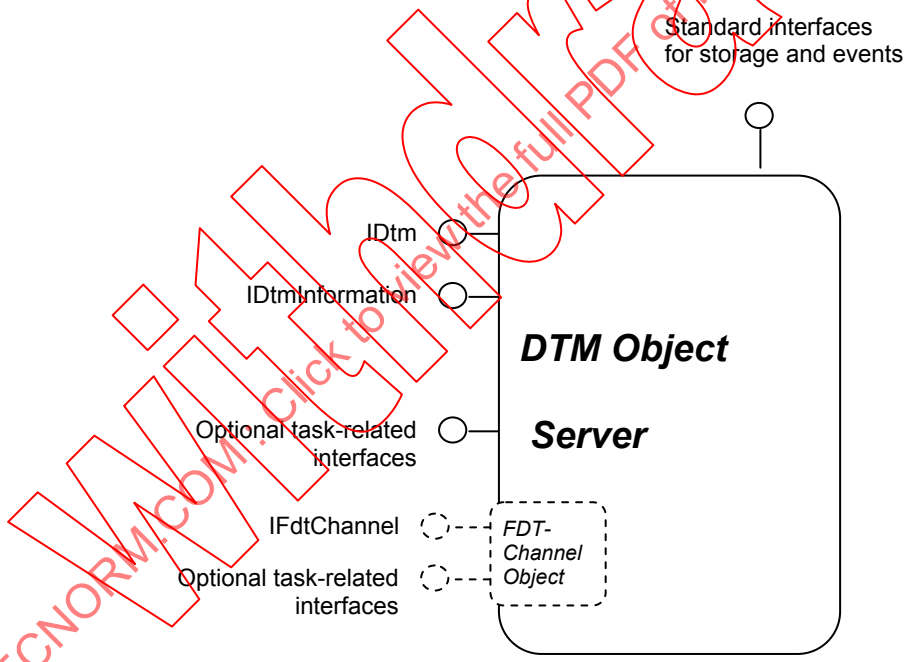

Figure 7 – DTM interfaces

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

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

6.4.2 The Block Type Manager (BTM)

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

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

The DTM is the root element for complete device representation . It is the starting point to collect the information from the BTMs belonging to the same device. The internal device structure is represented by this topology shown in Figure 8

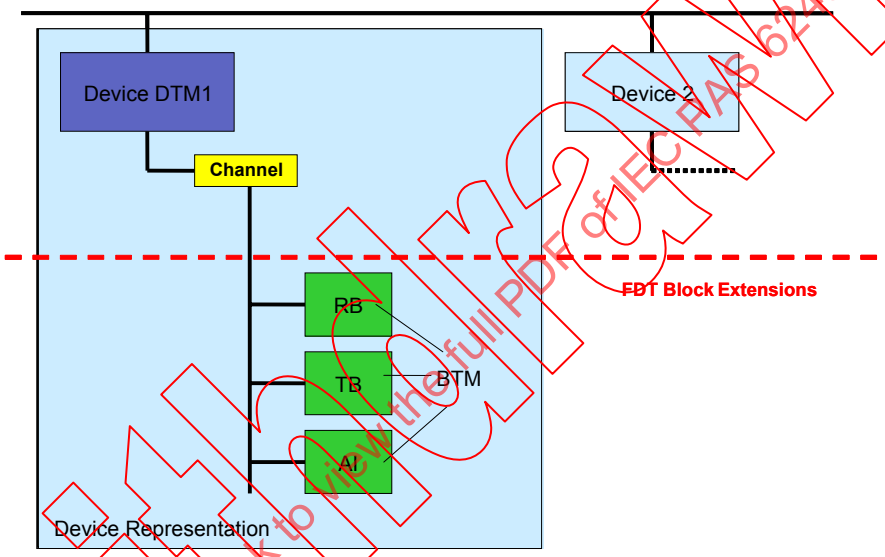

Figure 8 – Example of device architecture and components

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

6.4.3 The FDT Frame Application (FA)

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 stand-alone 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.

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 (see Figure 9). Which task-related interfaces are provided depends on the capability of the Frame Application. Each interface will be described in detail at its own clause.

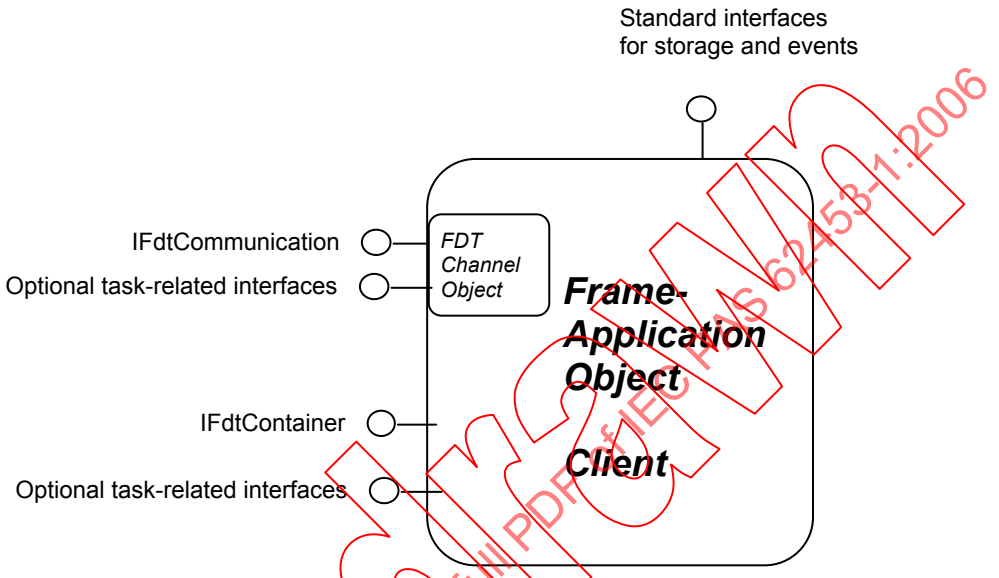

Figure 9 – Frame Application interfaces

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

To represent its communication capabilities a Frame Application can implement FDT-Channel objects. The Frame Application's channels represent the gateway from the FDT-specific to the Frame Application-specific communication. At least the interface IFdtCommunication must be implemented for a channel of a Frame Application. On the Frame Application's side the communication channel can be a PC I/O board or engineering topology with processing units and proprietary bus systems.

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

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

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

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

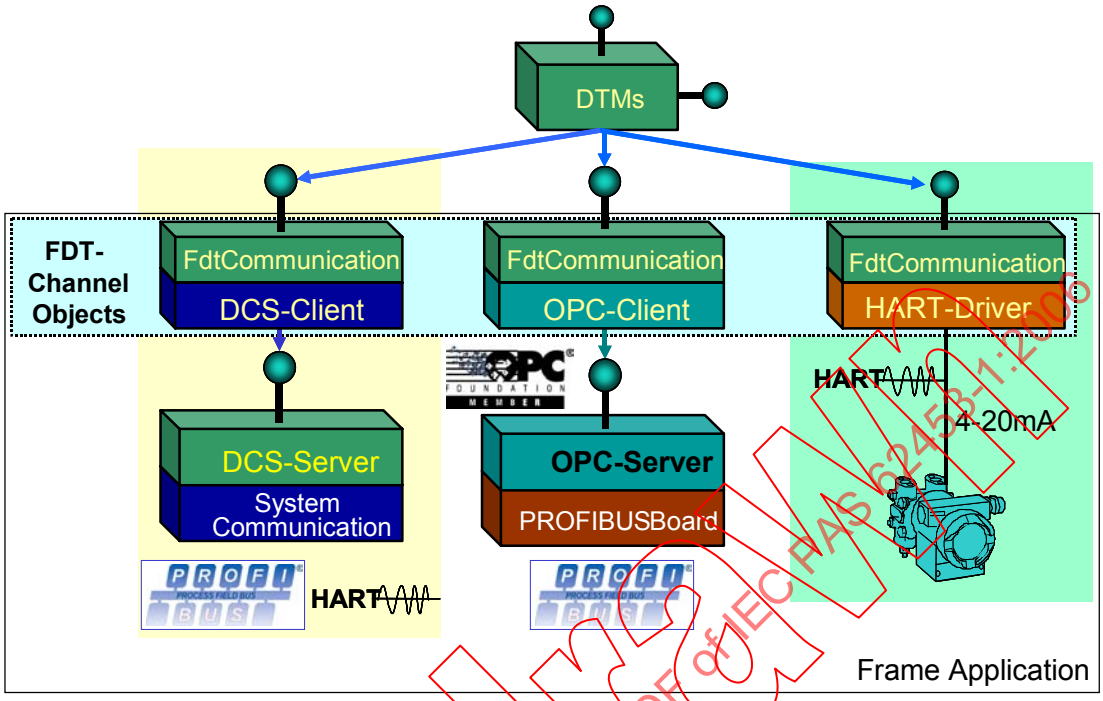

Figure 10 – The FDT communication layers

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 (see Figure 10). An engineering tool for example may configure its fieldbus master by its own manufacturer specific means.

6.5 Synchronization and serialization issues

On the 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.

6.6 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 (see Figure 11). Typically, in process control systems, multiple client applications like observing, channel assignment, or master configuration, need information about the configuration of a device.

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

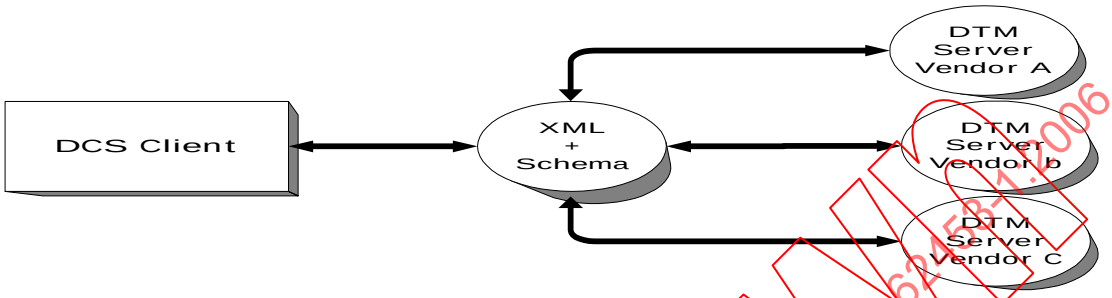

Figure 11 – FDT Client/Server relationship via XML

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

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

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

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

The FDT developer should always work with the DOM because

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

NOTE

In the case where 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; for example, more than one DTM for the same device on different working stations.

6.6.1 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 an XML document to simplify the DTM development and to have 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 (see Figure 12).

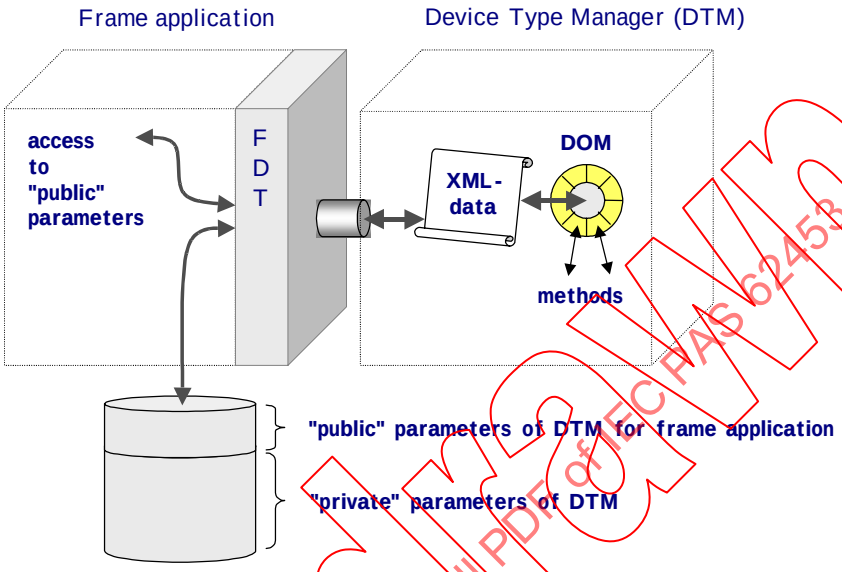

Figure 12 – Data access and storage

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 (see Figure 13).

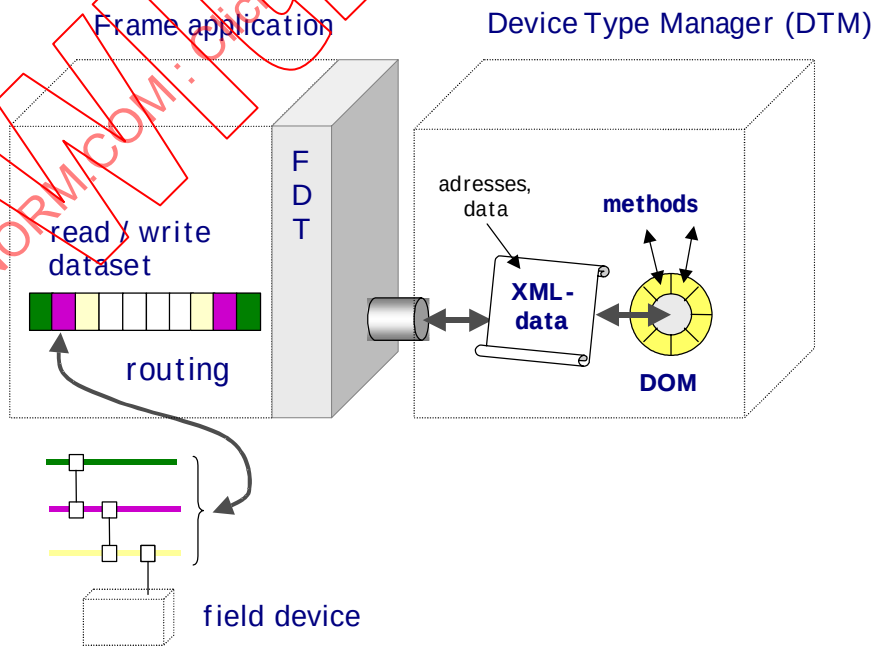

Figure 13 – Communication

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 (see Figure 14). A default style sheet is supported by the Frame Application.

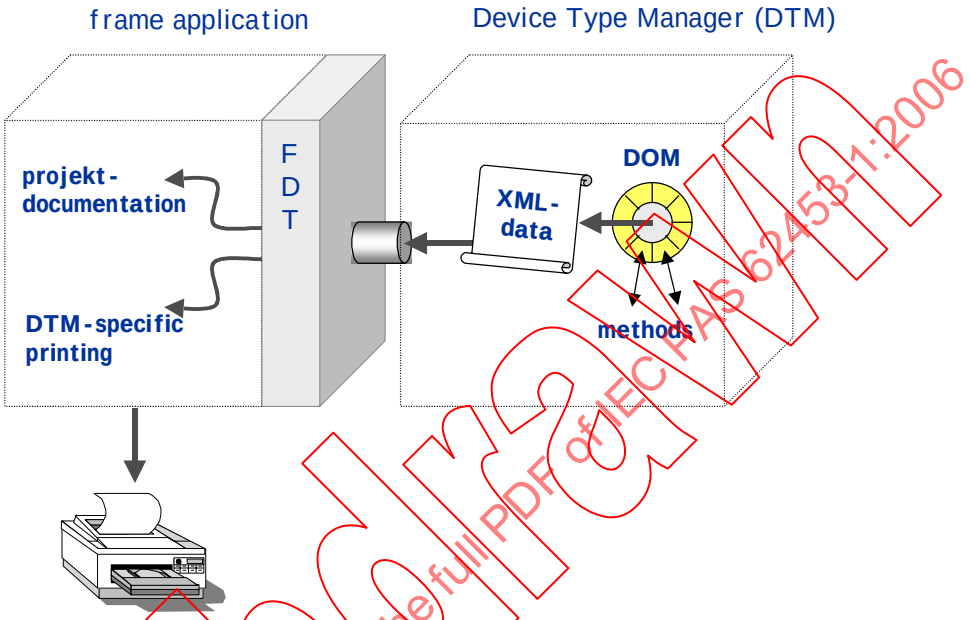

Figure 14 – Documentation

In the 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 (see Figure 15). The interchange format of this data is also an XML document.

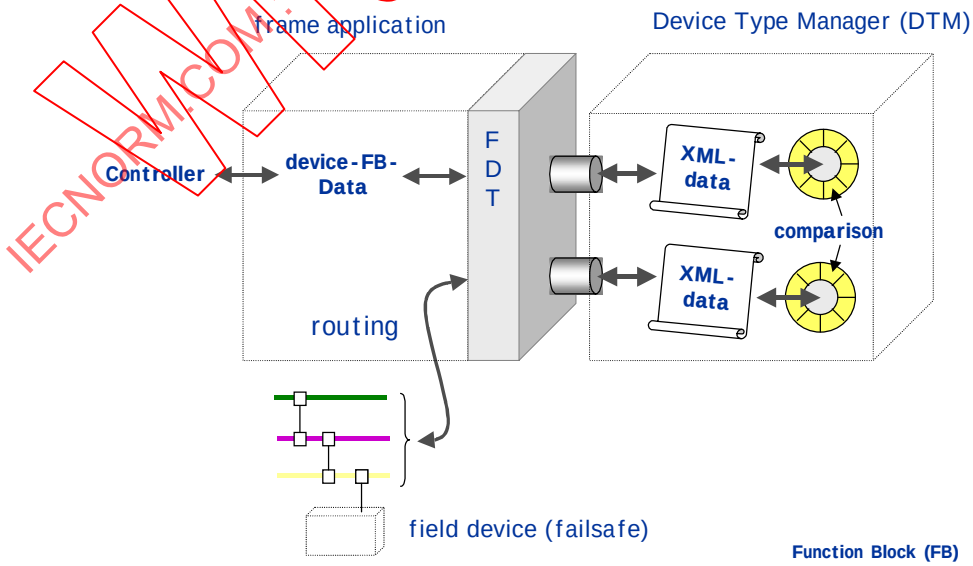

Figure 15 – Parameter verification in case of failsafe devices

6.7 Persistent storage story

6.7.1 Persistence overview

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

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

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

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

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

The Frame Application requests the storage of private data of a DTM. A DTM must be able to re-establish its complete state when this is requested by the Frame Application. This is done by calling the function IPersistXXX::Load of the DTM. Reload of a DTM object by calling IPersistXXX::Load several times may not be supported by all DTM suppliers. With an IPersistXXX:Save request a DTM must store its private data within the Frame Application. A DTM object for a new instance must be initialized if the IPersistXXX:InitNew method is called by the Frame Application.

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

NOTE

In order to simplify the DTM development, it is up to a DTM to implement one of the defined persistent interfaces (IPersistStreamInit or IPersistPropertyBag) according to the Microsoft standard. The Frame Application must be able to handle both.

References to DTMs do not belong to the instance data of a DTM. A DTM must not store any references to other DTMs. A DTM can get information concerning its parents or children via IFdtTopology::GetParentNodes() and IFdtTopology::GetChildNotes().

DTM should report data load errors via standard COM error mechanism (HRESULT not equal to S_OK). Optionally, DTM can write further human readable error information to standard COM global error info (Win32 SetLastError method).

6.7.2 Persistence interfaces

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

6.8 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.

6.9 Basic operation phases

6.9.1 Roles and access rights

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

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

Examples:

- An observer will not get access to calibration functions.
- An observer can not modify parameters.

See 9.1 Actors

6.9.2 Operation phases

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

There are five operation phases:

- Engineering
  Planning and configuring of a plant,
  no online access to the plant
- Commissioning, workshop
  Installing the plant and the devices,
  Scanning the network to verify a planned network,
  Downloading project data into the plant,
  Programming, diagnosis of the devices,
  Adjusting parameters

- Runtime

  The plant is completely commissioned and running.

  Very restricted access to configuration and parameterization data.

  Scan to verify the network

  Replacing defect devices.

  Reading process values and diagnosis information.

- Service

  This operation phase is used for service tool Frame Applications.

  Scan to create a topology. Scan to verify a network. All service related operations. Unrestricted access to the device.

- Not supported

  The application does not support the operation phases defined above.

  A DTM must offer all functions if the Frame Application passes “not supported”.

The following Table 1 describes the usage of operation phases in different Frame Application types.

Table 1 − Operation phases

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

For example, the DTM allows the maintenance actor during commissioning phase the complete Online parameterization, but during the runtime phase it does not allow it, or allows it only for some of its parameters.

6.10 Abstract FDT object model

This is the view on the device itself to integrate the I/O structure for Frame Application tasks like channel assignment and controller configuration. An object model of DTMs, BTMs and FDTChannels serves the access to this information (see Figure 16). Only DTM, BTM, Channel and presentation objects are implemented as COM objects. Information about the Frame Application, project or DCS channels is provided using XML documents. Also information about functionality and properties of the DTM, BTM and channel objects is available via XML documents and exchanged via COM interfaces of these objects.

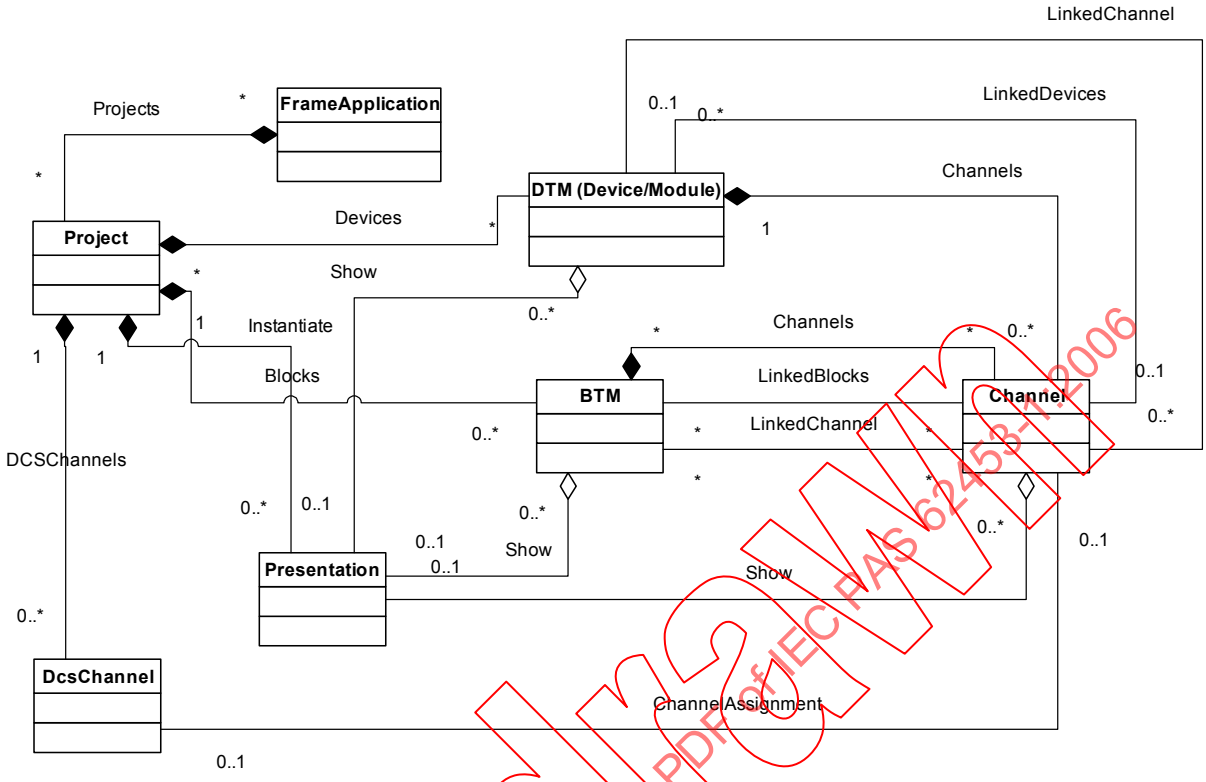

Figure 16 – FDT objects - device related

The following Table 2 provides a description of all objects of the FDT object model.

Table 2 – Description of FDT objects

Object	Description
Frame Application	The Frame Application is a logical object to represent an environment like an engineering system or a stand-alone tool. It controls the lifetime of DTM-instances within the project. A Frame Application can handle several projects
Project	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
DTM	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. A device can be subdivided into modules and submodules. A module is either a hard-ware module or a software module. Hardware modules are plugged into physical slots of the device. For example, an I/O module can be plugged into a Remote I/O device. A module can also be a software module. Software modules represent static or configurable substructures of a device like a closed loop module. These SW modules in general are device-specific and cannot be moved between devices. - A DTM may represent different types of devices, e.g. field devices, communication devices and gateway devices. - A Device-DTM represents a 'normal' field device that uses a Communication-Channel to communicate with the related field device - A Communication-DTM represents a communication device that provides communication capabilities via Communication-Channels (in sense of FDT) but does not need any communication capabilities of a parent DTM - A Gateway-DTM is a Communication-DTM that provides communication capabilities via Communication-Channels (in sense of FDT) and requires communication capabilities of a parent DTM

Object	Description
BTM	A BTM is used to represent flexible software objects, like function blocks. These software blocks are more flexible than Software modules; for instance, they may be moved between devices. Also a protocol may require modelling of device structures as blocks.
FdtChannel	FdtChannel-Object could behave in 2 ways: As a 'Communication-Channel' and a 'Process-Channel'. - Communication-Channel is an object that provides access to communication infrastructure. It is used by Device-DTM or Gateway-DTM as a service provider for communication. - Process-Channel represents a single process value, its optional state information, and its assigned ranges and alarms. A channel either belongs to a device or a module DTM. - An FdtChannel-Object could implement both behaviors.
Presentation	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
DcsChannel	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. A DcsChannel belongs to the Frame Application.

All the associations shown in the object model above are listed in the Table 3.

Table 3 – Relations between FDT objects

Name	Description
Devices	Within a project it is important to know all field devices. This association enumerates all field devices in the project. It also forces DTMs with private instance database to collaborate with the project for creation and removal of field devices. The removal of the project forces all DTM instances to be removed. A DTM instance (field device) cannot be part of more than one project.
Channels	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
Show	The instances of presentation objects are associated to the instance of their DTM business object by this relationship. This association belongs to the type of application.
Instantiate	This association shows for ActiveX controls that the presentation objects of a DTM are at least instantiated and embedded by the Frame Application.
Channel-Assignment	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 The Frame Application (project) is responsible for handling this association.
LinkedDevice	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() The Frame Application (project) is responsible for handling this association.
LinkedChannel	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() The Frame Application (project) is responsible for handling this association.

6.11 Fieldbus independent integration

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

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

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

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

6.12 Scanning and DTM assignment

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

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

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

Refer to 8.10.8 for more information about the general concept.

7 FDT version interoperability guidance

7.1 Overview

FDT is a component based standard, which is constantly enhanced to new improved versions. Control system environments typically run for 10-15 years in contrast. If hardware and software components in a control system have to be exchanged, a mixture of components designed for different FDT versions may emerge.

This raises the question, how components based on different FDT versions can cooperate. This clause FDT Version Interoperability Guide goes further into this question and provides solutions.

7.2 General

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

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

2. Component Interoperability : Components of different FDT versions must interoperate properly. DTMs of newer FDT versions must run in older Frame Applications and vice versa. Communication must work even if FDT versions of Device-DTMs and Communication-DTMs differ.

A limiting condition for future FDT-enhancements is to ensure maximum compatibility of different FDT versions. This results in a maximum interoperability of FDT components originally designed for different FDT versions.

FDT takes care about version interoperability on two different levels:

a) Specification Level: The FDT specification targets full compatibility of different specification releases (e.g. schemas of the newer releases are compatible with older versions). Properly designed FDT components will achieve compatibility without extra implementations.

b) Implementation Level: FDT provides test tools to examine the FDT compliance of FDT components. Although mentioned here, this is not in the scope of this PAS.

The FDT version is composed by a major, a minor, a release and a build number (e.g. 1.2.0.3 where 1 is the major , 2 is the minor number, 0 the release and 3 the build number). Compatibility is defined slightly different with respect to the major number:

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

7.3 Component interoperability

Interoperability of Frame Application and DTM components is shown in the following Table 4.

Table 4 – Interoperability between components of different versions

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

Note O – optional ✓– assured

The FDT specification ensures that DTMs and Frame Applications of the same major version cooperate regardless of minor version, e.g. FDT version 1.2 is compatible to FDT version 1.2.1. In this case the additional functionality defined in the higher FDT version may not be provided.

The situation is different if the major version number changes.

A DTM with a higher major version (than the Frame Application) may not function within a Frame Application with lower major version. Interoperability in this case is optional. In order to function in the 'old' Frame Application the DTM needs to behave according to the FDT version provided by the Frame Application.

A Frame Application with a higher major version (than the DTM) may refuse to work with a DTM with lower major version. Interoperability in this case is optional. In order to work with the 'old' DTM the Frame Application needs to provide both interface sets (see section 'Design Rules'). In this situation the Frame Application behaves like an 'old' FDT version toward the DTM (old version number, old XML schemas).

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

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

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

7.4 FDT type library

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

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

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

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

7.5 DTM and device versions

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

If ClassID and ProgID of a DTM have changed, FDT will provide a mechanism to upgrade to the newer DTM (see also sub clause 7.6 and 10.25).

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

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

7.6 Persistence

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

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

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

7.7 Nested communication

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

7.7.1 Data exchange

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

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

7.7.2 Communication channel upgrade

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

7.7.3 Scenarios

7.7.3.1 Parent component version 1.2

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

7.7.3.2 Parent component newer than version 1.2

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

7.7.3.3 DTM version 1.2

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

7.7.3.4 Nested communication

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

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

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

Examples:

a) A DTM of a Profibus – HART Remote I/O can support HART FDT communication compatible to FDT version 1.2.1 although its parent component supports only FDT version 1.2. In this case the Remote I/O DTM must be able to map all FDT1.2.1 HART transaction requests (e.g. HART burst mode) to appropriate FDT 1.2 profibus compatible transaction requests

b) A Multiplexer DTM of version 1.2.1 with a parent component supporting only FDT version 1.2 may not be able to provide functionality of the higher FDT version (e.g. because it depends on device-initiated data transfer).

7.7.4 OnAddChild

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

7.8 Implementation hints

7.8.1 Interfaces

Mandatory interfaces defined in a new FDT minor version may not be defined in older versions. Therefore components designed for the older FDT version will not implement these interfaces. Although the system runs under the newer FDT version defined by the Frame Application the components need to be aware that an interface may not be supported if the server component is based on a lower FDT version.

Example: A DTM for FDT version 1.2.1 must not rely on the interface IDtm2. Although this interface may be defined mandatory for a DTM with FDT version 1.2.1, it must be possible to connect the DTM to an other DTM with FDT version 1.2 which supports only IDtm.

7.8.2 Persistence

A DTM should add version information to it's dataset, e.g. as a property of the property bag.

8 FDT interface specification

8.1 Overview of the FDT interfaces

The FDT interface specification includes the following objects:

- DTM
- BTM
- Presentation Objects
- DTMActiveXControl
- BTMActiveXControl
- FdtChannelActiveXControl
- FdtChannel
- Frame Application

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

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

8.2 FDT objects

8.2.1 FDT object model

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

The interfaces provided by DTM, ActiveX and Frame Application are shown in following Figure 17. After the ActiveX is created by the Frame Application, the DTM cooperates with the ActiveX object.

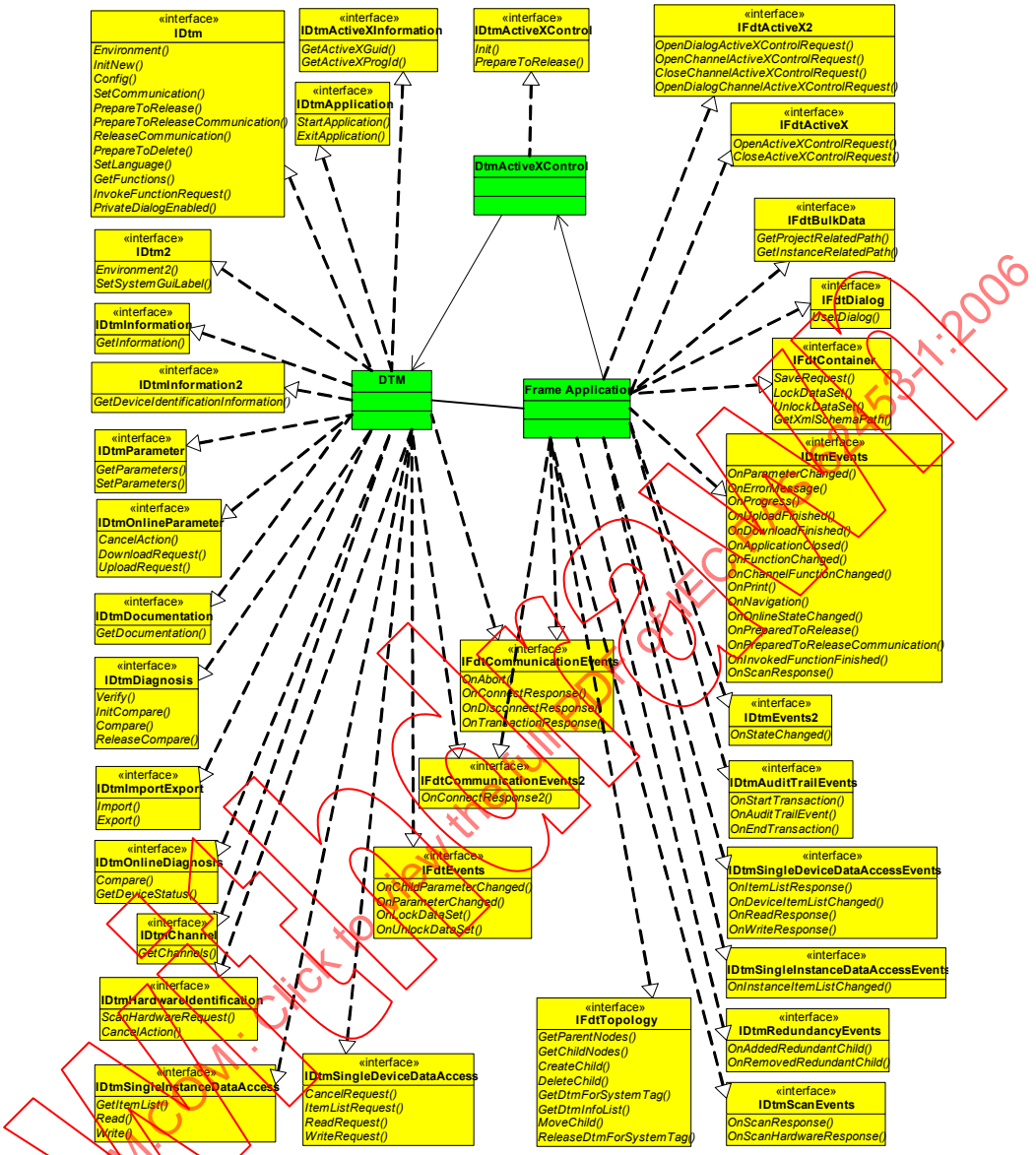

Figure 17 – FDT objects- DTM, DtmActiveXControl and Frame Application

The interfaces provided by a channel are shown in following Figure 18. The DTM accesses communication by sending requests to the interface IFdtCommunication and receives responses by the interface IFdtCommunicationEvents. An hierarchy of Channels and DTMs may be used to provide nested communication.

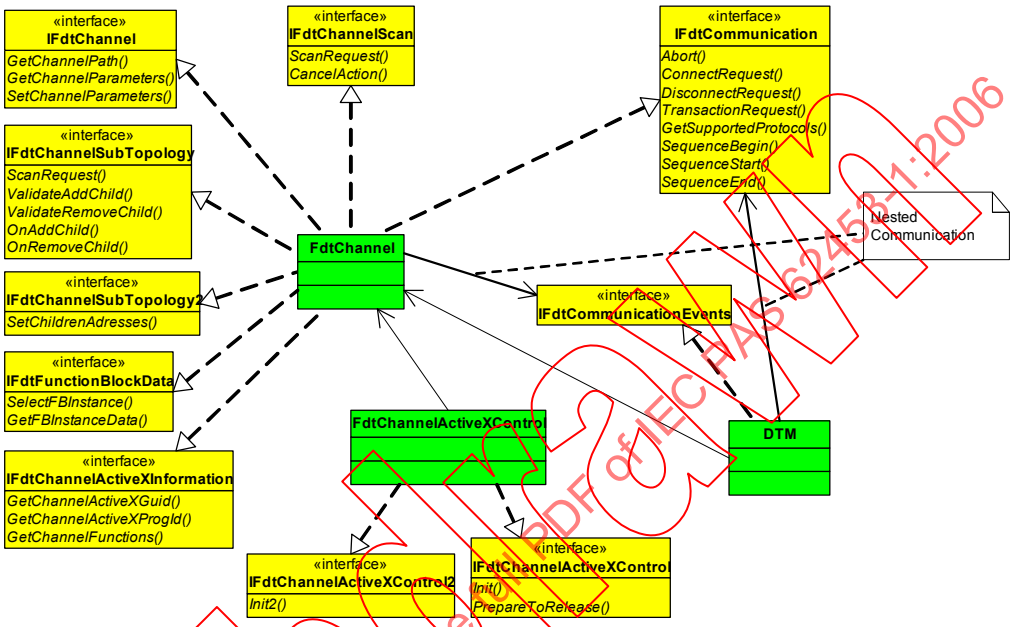

Figure 18 – FDT objects- FdtChannel

The interfaces related to a BTM are shown in the following Figure 19.

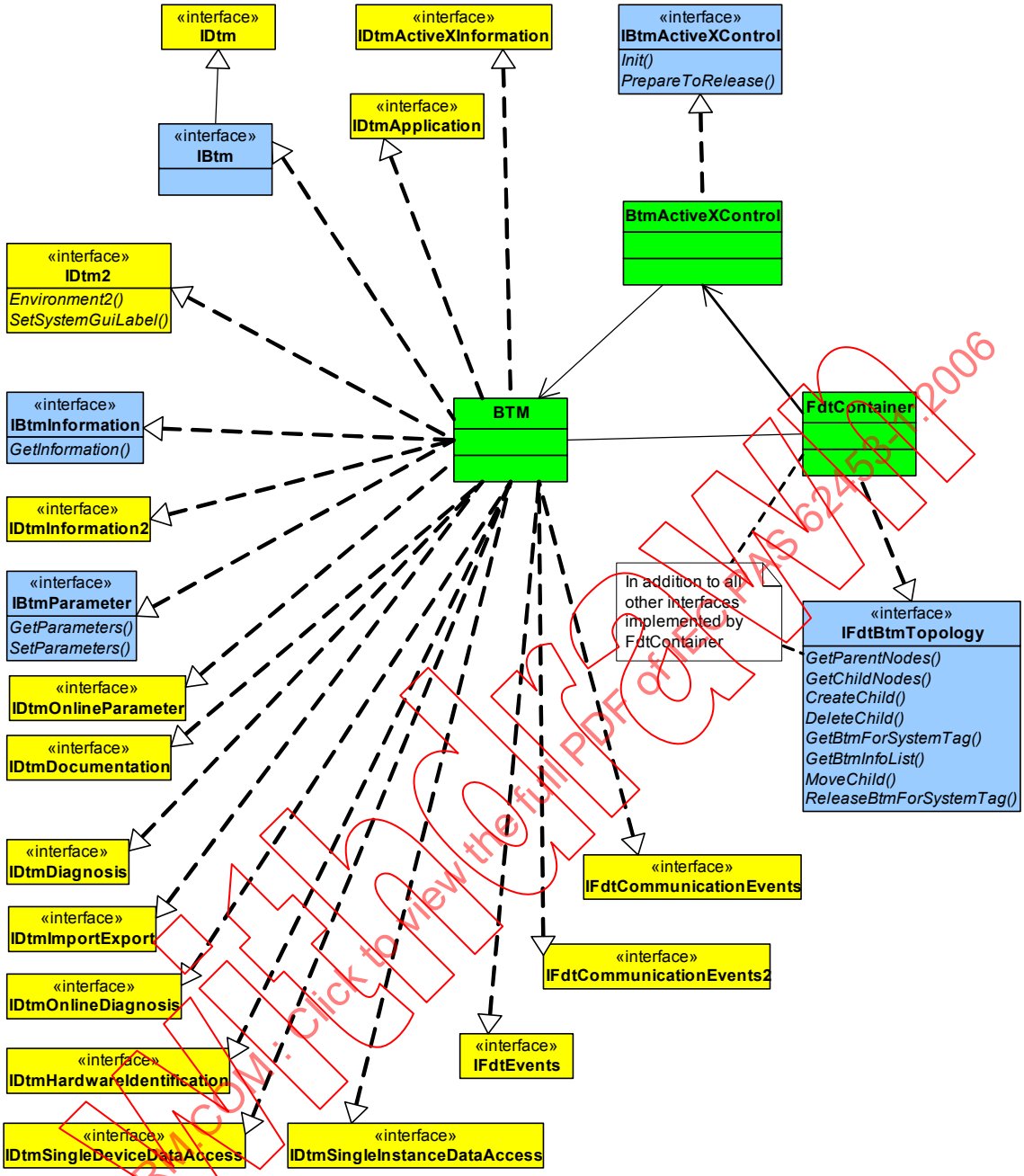

Figure 19 – FDT objects-- BTM and BtmActiveXControl

The FDT specific datatypes “FdtUUIDString”, “FdtXmlDocument” and “FdtXPath” are derived are derived from datatype string by inheritance, see Figure 20.

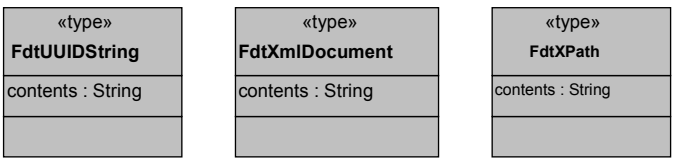

Figure 20 – FDT data types

8.2.2 DTM state machine

The following state machine shows the different states of a DTM, see Figure 21.

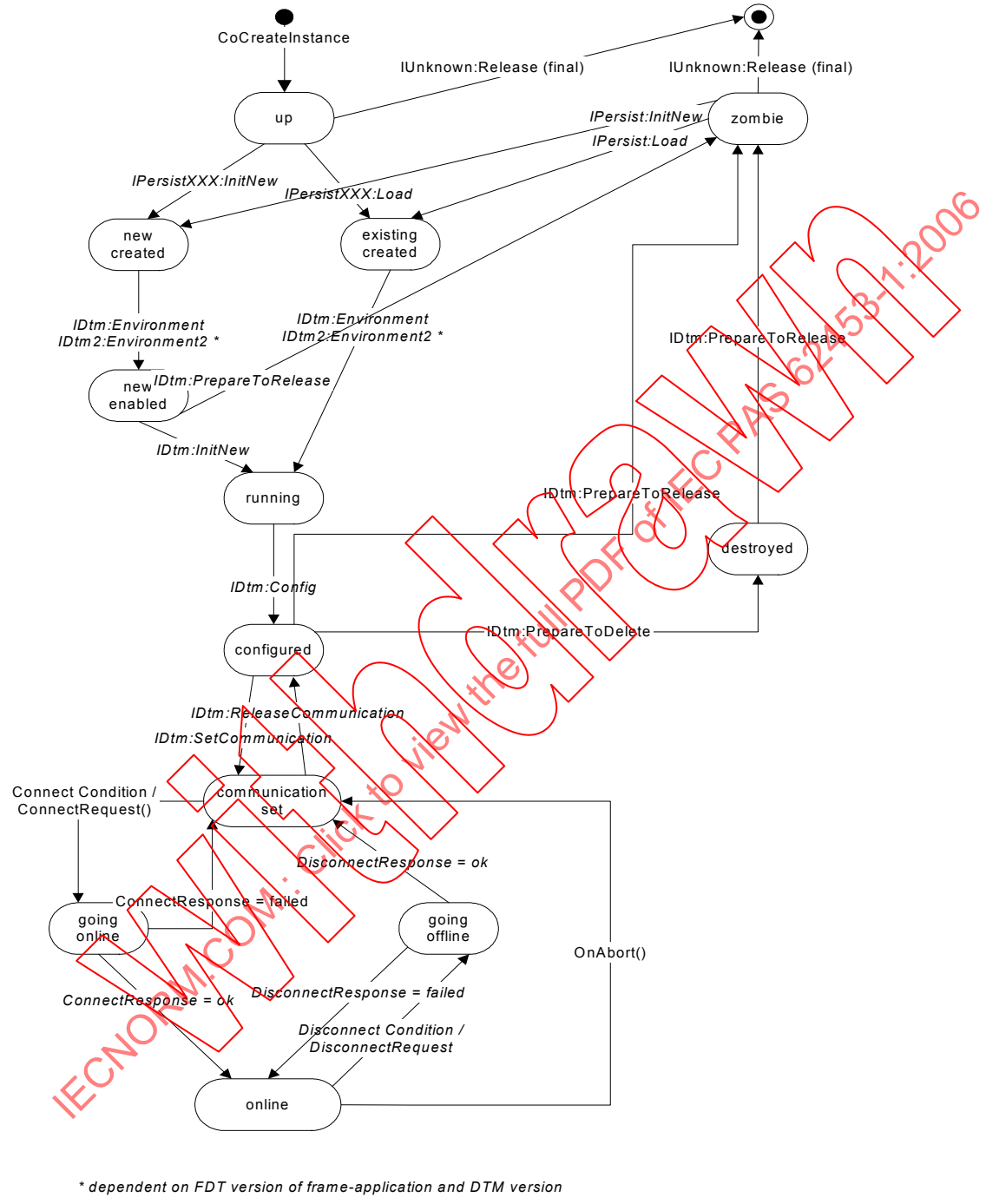

Figure 21 − State machine of a DTM

Table 5 below defines the interfaces of a DTM which can be used by a Frame Application at the shown states.

Table 5 − Availability of DTM methods in different states

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

Interface / Method	up	new created	new enabled	existing created	running	configured	going offline	online	going online	destroyed	communication set	zombie
IFdtChannelActiveXInformation					√	√	√	√	√		√	
IFdtChannelSubTopology												
OnAddChild()					√	√	√	√	√		√	
OnRemoveChild()					√	√	√	√	√		√	
ScanRequest()								√	√		√	
ValidateAddChild()					√	√	√	√	√		√	
ValidateRemoveChild()					√	√	√	√	√		√	
IIFdtChannelSubTopology2												
SetChildrenAdresses()					√	√	√	√	√		√	
IFdtChannelScan												
ScanRequest()								√	√		√	
CancelAction()								√	√		√	
IFdtCommunication												
Abort()							√	√	√		√	
ConnectRequest()							√	√	√		√	
DisconnectRequest()							√	√	√		√	
TransActionRequest()							√	√	√		√	
GetSupportedProtocols()						√	√	√	√		√	
SequenceBegin()							√	√	√		√	
SequenceStart()							√	√	√		√	
SequenceEnd()							√	√	√		√	
IFdtFunctionBlockData						√	√	√	√		√	
IDtmSingleInstanceDataAccess												
GetItemList()						√	√	√	√		√	
Read()						√	√	√	√		√	
Write()						√	√	√	√		√	
IDtmSingleDeviceDataAccess												
CancelRequest()							√	√	√		√	
ItemListRequest()						√	√	√	√		√	
ReadRequest()								√	√		√	
WriteRequest()								√	√		√	

NOTE 1 At the Zombie State not all DTMs must support reload instance via IPersistXXX interfaces, e.g. DTMs written in Visual Basic.

NOTE 2 Concerning the transition between states and methods with asynchronous behavior: The call of methods, which are defined with an asynchronous behavior (e.g. PrepareToRelease()) will start the transition. The related end state will be reached, when the according method was called (e.g. OnPreparedToRelease()).

NOTE 3 Concerning the transition between states in case of errors: If the method, which leads to the transition between states, fails (e.g. return value is FALSE or a COM error appears), the state is left unchanged.

Table 6 below defines the interfaces of a Frame Application which can be used by a DTM (starting from FDT version 1.2.1) at the shown states. A Frame Application must be aware that after IDtm:Environment an older DTM may call any container interface method.

Table 6 − Availability of Frame Application interfaces

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

8.3 Device Type Manager

8.3.1 Interface IDtm

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

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

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

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

8.3.1.1 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

In general it is expected that a DTM adapts the provided functionality according to the role of the current user (see 8.3.1.3).

8.3.1.2 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.

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

Comments

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

8.3.1.3 GetFunctions

HRESULT GetFunctions(

[in] FdtXmlDocument operationState,

[out, retval] FdtXmlDocument* result);

Description

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

Parameters	Description
operationState	XML document containing the current operation phase specified by the FDTOperationPhaseSchema

Return value

Return Value	Description
Result	XML document containing actual supported functions specified by the DTMFunctionsSchema

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. These data are available as soon as the DTM is instantiated but the information may change if it is instance-specific.

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

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

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

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

The asynchronous behavior is described in the according clauses.

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

It is expected that a DTM adapts the provided list of functions according to the role of the current user (see clause 8.3.1.1). As long as the DTM does not have valid user information (e.g. in state “running”), it should behave like the user with the least rights (“Observer”) is using the DTM.

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

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

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

Comments

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

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

8.3.1.4 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 subdevice 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

None

8.3.1.5 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 IDtmApplication::StartApplication(), IDtmActiveXInformation::GetActiveXGuid() or IDtmActiveXInformation::GetActiveXProgId() and can work asynchronous. If the called function is finished or terminated by the user the DTM sends 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.

8.3.1.6 PrepareToDelete

HRESULT PrepareToDelete (

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description

Return value

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

Behavior

The method is used to inform a DTM that it has to clean up, e.g. its log files or protocols. After this function call the data set will be deleted by the Frame Application. The Frame Application is responsible to ensure the pre-conditions for the delete. That means that the Frame Application must ensure that all DTM instance related to this data set are shut down (either Zombie-State or released). If the DTM returns FALSE the Frame Application can inform the user to close the user interfaces or can terminate them by ExitApplication() or IDtmActiveXControl::PrepareToRelease(). In general a DTM will finish its current communication process during the release of its user interfaces.

Comments

None

8.3.1.7 PrepareToRelease

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

Description

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

Parameters	Description

Return value

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

Behavior

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

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

Comments

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

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

8.3.1.8 PrepareToReleaseCommunication

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

Description

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

Parameters	Description

Return value

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

Behavior

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

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

Comments

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

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

See also 8.9.1.13 OnProgress and 8.3.1.11 SetCommunication

8.3.1.9 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 means that the DTM must not open any dialog windows.

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

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

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

Comments

None

8.3.1.10 ReleaseCommunication

HRESULT ReleaseCommunication(

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description

Return value

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

Behavior

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

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

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

Comments

See also clause 8.9.1.11, OnPreparedToReleaseCommunication().

8.3.1.11 SetCommunication

HRESULT SetCommunication(

[in] IFdtCommunication* communication,

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description
Communication	Interface pointer of a communication channel

Return value

Return Value	Description
TRUE	Pointer accepted
FALSE	Invalid communication pointer

Behavior

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

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

Comments

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

- For each DTM, which acts as a root concerning the chain of interfaces for nested communication, the method IDtm::SetCommunication() must be called with a NULL pointer as parameter 'communication'. This leads to the transition from 'configured' to 'communication set' concerning the DTM state machine (refer to 8.2.2, DTM State Machine).
- To set up the chain for nested communication, the Frame Application must start to hand over the interface pointer from the DTM, which acts as a root concerning the chain of interfaces for nested communication. To release the chain, according to 4.3.1.8, IDtm::PrepareToReleaseCommunication(), the Frame Application should act vice versa.

8.3.1.12 SetLanguage

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

Description

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

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

Return value

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

Behavior

The Frame Application sets the language during initialization of the DTM. So all presentation objects of the same instance have the same language. Also the messages on the event

interfaces like OnErrorMessage() and the human readable contents of the XML documents like at the interface IDtmDocumentation or IDtmInformation have to use the requested language. If a DTM does not support the requested language, it uses the current language in case it is already initialized or sets its default language on the first initialization.

Comments

The supported languages of a DTM are listed within the DTMInformationSchema. One DTM instance is always initialized with one language. It is up to a DTM whether it can change the current language of an user interface while the user interface is shown. The output format for numbers, currencies, times, and dates will be based on the regional options of the operation system.

8.3.2 Interface IDtm2

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

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

8.3.2.1 Environment2

HRESULT Environment2(

[in] BSTR systemTag,

[in] IFdtContainer* container,

[in] FdtXmlDocument frameInfo,

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description
systemTag	Identifier for the device instance; set by the Frame Application
Container	Back pointer to the Frame Application
frameInfo	XML document containing frame version information by the DTMEnvironmentSchema

Return value

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

Behavior

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

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

Comments

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

8.3.2.2 SetSystemGuiLabel

HRESULT SetSystemGuiLabel (

[in] FdtXmlDocument systemGuiLabel,

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description
systemGuiLabel	XML document containing a unique human readable identifier of the DTM instance in the context of the Frame Application. Document specified by DTMSystemGuiLabelSchema

Return value

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

Behavior

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

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

Comments

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

8.3.3 Interface IDtmActiveXInformation

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

8.3.3.1 GetActiveXGuid

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

Description

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

Parameters	Description
functionCall	XML document containing the function id for the requested user interface specified by the 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 NULL string.

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

Comments

None

8.3.3.2 GetActiveXProgId

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

Description

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

Parameters	Description
functionCall	XML document containing the function id for the requested user interface specified by the DTMFunctionCallSchema

Return value

Return Value	Description
result	ProgId for an ActiveX control

Behavior

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

None

8.3.4 Interface IDtmApplication

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

8.3.4.1 ExitApplication

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

Description

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

Parameters	Description
invokeId	Identifier for the started application. Same value as provided in the corresponding call of IDtmApplication::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 shutdown procedure for the user interface. During this shutdown 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.

8.3.4.2 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.

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
windowTitle	Window title required by the Frame Application

Return value

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

Behavior

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

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

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

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

Comments

None

8.3.5 Interface IDtmChannel

This interface is used for accessing channel objects. On the 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 the case of communication channels, these channel objects are used to build the communication chain for Nested Communication.

8.3.5.1 GetChannels

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

Description

Returns the channel objects of a DTM.

Parameters	Description

Return value

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

Behavior

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

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

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

Comments

None

8.3.6 Interface IDtmDocumentation

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

8.3.6.1 GetDocumentation

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

Description

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

Parameters	Description
functionCall	XML document containing the function id for the requested document specified by the 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 PAS is defined by the passed function call id, which is available via IDtm::GetFunctions() Only functions with the attribute 'printable' = TRUE will be supported. The Frame Application can use a XSL style sheet to transform the returned XML document to HTML. Nesting DTM specific style sheets can be used to transform the XML document into a DTM specifc HTML.Within these nested style sheets also hyperlinks to additional documents or into the World Wide Web can be placed.

Comments

For an example style sheet please have a look at DTMDocumentationStyle.xsl. Refer to Annex C.

8.3.7 Interface IDtmDiagnosis

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

8.3.7.1 Compare

HRESULT Compare(

[out, retval] VARIANT_BOOL* result);

Description

Returns TRUE if the complete data sets are equal.

Parameters	Description

Return value

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

Behavior

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

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

In case of errors the DTM should inform the Frame Application via the callback interface IDTMEvent::OnErrorMessage().

Comments

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

8.3.7.2 InitCompare

HRESULT InitCompare(

[in] BSTR systemTag,

[out, retval] VARIANT_BOOL* result);

Description

Initializes a DTM for comparison of two device instances

Parameters	Description
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().

8.3.7.3 ReleaseCompare

HRESULT ReleaseCompare(

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description

Return value

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

Behavior

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

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

Comments

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

8.3.7.4 Verify

HRESULT Verify(

[out, retval] VARIANT_BOOL* result);

Description

Returns TRUE if the complete data set is valid.

Parameters	Description

Return value

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

Behavior

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

Comments

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

8.3.8 Interface IDtmImportExport

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

8.3.8.1 Export

HRESULT Export(

[in] IStream* stream,

[out, retval] VARIANT_BOOL* result);

Description

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

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

Return value

Return Value	Description
TRUE	The operation succeeded
FALSE	The operation failed

Behavior

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

Comments

None

8.3.8.2 Import

HRESULT Import(

[in] IStream* stream,

[out, retval] VARIANT_BOOL* result);

Description

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

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

Return value

Return Value	Description
TRUE	The operation succeeded
FALSE	The operation failed

Behavior

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

IDtmImportExport interface. If this interface is not provided by a DTM the Frame Application uses one of the IPersistXXX interfaces for export/import.

Comments

None

8.3.9 Interface IDtmInformation

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

8.3.9.1 GetInformation

HRESULT GetInformation(

[out, retval] FdtXmlDocument* result);

Description

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

Parameters	Description

Return value

Return Value	Description
Result	XML document containing static DTM information specified by the 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

None

8.3.10 Interface IDtmInformation2

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

8.3.10.1 GetDeviceIdentificationInformation()

HRESULT GetDeviceIdentificiationInformation (

[in] FdtXmlDocument typeRequest,

[in] FdtUUIDString protocolId,

[out, retval] FdtXmlDocument* result);

Description

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

Parameters	Description
typeRequest	Defines the DTMDeviceType or BTMBlockType for which the identification is requested. (XML according to TypeRequestSchema)
protocolID	Defines UUID of protocol for which the device identification is requested

Return value

Return Value	Description
result	Protocol specific device identification information for a DTMDeviceType specified by a field bus specific schema. (FDTxxxDeviceTypeIdentSchema where xxx is the protocol name) If method was called at a Communiction-DTM, then XML document according to DTMDeviceTypeIdentSchema is returned and must not be transformed

Behavior

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

Comments

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

See usage of information returned by IDtmInformation2::GetDeviceIdentificiationInformation() in 8.10.8.

8.3.11 Interface IDtmOnlineDiagnosis

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

8.3.11.1 Compare

HRESULT Compare(

[out, retval] FdtXmlDocument* result);

Description

Returns an XML document containing the result of the compare.

Parameters	Description

Return value

Return Value	Description
Result	XML document containing the result of the compare specified by the 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.

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

Comments

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

8.3.11.2 GetDeviceStatus

HRESULT GetDeviceStatus(

[out, retval] FdtXmlDocument* result);

Description

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

Parameters	Description

Return value

Return Value	Description
Result	XML document containing the status of the device specified by the 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

A BTM must return the status of the related block.

8.3.12 Interface IDtmOnlineParameter

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

8.3.12.1 CancelAction

HRESULT CancelAction(

[in] FdtUUIDString invokeId,

[out, retval] VARIANT_BOOL* result);

Description

Cancels an active parameter-upload or download.

Parameters	Description
invokeId	Identifier of the action to be canceled

Return value

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

Behavior

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

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

Comments

None

8.3.12.2 DownloadRequest

HRESULT DownloadRequest(

[in] FdtUUIDString invokeId,

[in] FdtXPath parameterPath,

[out, retval] VARIANT_BOOL* result);

Description

Sends the request to write online data to the device.

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

Return value

Return Value	Description
TRUE	Request accepted
FALSE	Request cannot be performed

Behavior

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

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

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

Comments

None

8.3.12.3 UploadRequest

HRESULT UploadRequest(

[in] FdtUUIDString invokeId,

[in] FdtXPath parameterPath,

[out, retval] VARIANT_BOOL* result);

Description

Sends the request to read online data from a device.

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

Return value

Return Value	Description
TRUE	Request accepted
FALSE	Request cannot be performed

Behavior

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

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

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

Comments

None

8.3.13 Interface IDtmParameter

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

8.3.13.1 GetParameters

HRESULT GetParameters(

[in] FdtXPath parameterPath,

[out, retval] FdtXmlDocument* result);

Description

Returns an XML document with the device-specific parameters.

Parameters	Description
parameterPath	FdtXPath within the XML document

Return value

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

Behavior

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

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

The state of the received data is classified within the DTMParameterSchema.

The DTM must always return within the XML document the current DtmDeviceType. This means that, in the case of an update of the DTM, the changed DtmDeviceType must be returned instead of the DtmDeviceType given during IDtm::InitNew().

Comments

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

See also clause: State machine of instance data

8.3.13.2 SetParameters

HRESULT SetParameters(

[in] FdtXPath parameterPath,

[in] FdtXmlDocument xmlDocument,

[out, retval] VARIANT_BOOL* result);

Description

Sets changes of device-specific parameters.

Parameters	Description
parameterPath	FdtXPath within the XML document
xmlDocument	XML document specified by the DTMParameterSchema

Return value

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

Behavior

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

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

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

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

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

Comments

See also clause: State machine of instance data

8.3.14 Interface IFdtCommunicationEvents

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

8.3.14.1 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 the result of an invoked function or can occur “spontaneously” (e.g. if the absence of a device is noted automatically by the underlying communication infrastructure).

Comments

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

8.3.14.2 OnConnectResponse

HRESULT OnConnectResponse(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument response);

Description

Provides the response of ConnectRequest() identified by the invoke id.

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

Return value

Return Value	Description

Behavior

Via this method the sender of the 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

None

8.3.14.3 OnDisconnectResponse

HRESULT OnDisconnectResponse(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument response);

Description

Provides the response of DisconnectRequest() identified by the invoke id.

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

Return value

Return Value	Description

Behavior

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

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

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

Comments

None

8.3.14.4 OnTransactionResponse

HRESULT OnTransactionResponse(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument response);

Description

Provides the response of TransactionRequest() identified by the invoke id.

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

Return value

Return Value	Description

Behavior

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

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

Comments

None

8.3.15 Interface IFdtCommunicationEvents2

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

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

8.3.15.1 OnConnectResponse2

HRESULT OnConnectResponse2(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument parentinformation,

[in] FdtXmlDocument response);

Description

Provides the response of ConnectRequest() identified by the invoke id.

Parameters	Description
invokeId	Unique identifier for the request
parentinformation	Version information of the parent component according to FDTConnectResponseSchema
response	Fieldbus-protocol-specific information about the established connection specified by a fieldbus specific schema, e.g., like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

Return value

Return Value	Description

Behavior

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

The method provides

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

Comments

None

8.3.16 Interface IFdtEvents

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

8.3.16.1 OnChildParameterChanged

HRESULT OnChildParameterChanged(
[in] BSTR systemTag);

Description

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

Parameters	Description
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 an XML document containing the instance specific changes. The Frame Application will send this document via IFdtEvents::OnParameterChanged() to all DTMs which reference the same device instance. Concerning the according parent DTMs (primary parent, secondary parents, see IFdtTopology::GetParentNodes()), the Frame Application must implement the following behavio.:

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

Comments

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

8.3.16.2 OnLockDataSet

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

Description

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

Parameters	Description
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.

8.3.16.3 OnParameterChanged

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

Description

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

Parameters	Description
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 DTM specific information. The Frame Application will send this document via IFdtEvents::OnParameterChanged() to all DTMs that reference the same device instance.

Comments

None

8.3.16.4 OnUnlockDataSet

HRESULT OnUnlockDataSet(

[in] BSTR systemTag,

[in] BSTR userName,

[out, retval] VARIANT_BOOL* result);

Description

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

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

Comments

None

8.3.17 Interface IDtmHardwareIdentification

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

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

8.3.17.1 ScanHardwareRequest

HRESULT ScanHardwareRequest(

[in] FdtUUIDString invokeID,

[in] FdtXmlDocument request,

[out, retval] VARIANT_BOOL* result);

Description

Requests scan for availability of hardware and corresponding identification information.

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

Return value

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

Behavior

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

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

Comments

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

8.3.17.2 CancelAction

HRESULT CancelAction (

[in] FdtUUIDString invokeID,

[out, retval] VARIANT_BOOL* result);

Description

Cancels active hardware check request.

Parameters	Description
invokedID	Unique identification of scan hardware request

Return value

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

Behavior

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

If the DTM cannot cancel the operation then it returns FALSE and will fire the IDtmScanEvents::OnScanHardwareResponse() event when the operation has finished.

Comments

None

8.3.18 Interface IDtmSingleDeviceDataAccess

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

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

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

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

8.3.18.1 CancelRequest

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

Description

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

Parameters	Description
invokedID	Unique identification of a read or write request

Return value

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

Behavior

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

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

Comments

None

8.3.18.2 ItemListRequest

HRESULT ItemListRequest(

[in] FdtUUIDString invokeId)

Description

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

Parameters	Description

Return value

Return Value	Description
invokeId	Unique identifier for the request

Behavior

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

Comments

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

8.3.18.3 ReadRequest

HRESULT ReadRequest(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument itemSelectionList);

Description

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

Parameters	Description
InvokeId	Unique identifier for the request
itemSelectionList	List of required items described by a DtmItemSelectionList specified by the DTMItemListSchema

Return value

Return Value	Description

Behavior

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

Execution of the ReadRequest() method must not change the data of the instance data set.

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

Comments

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

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

8.3.18.4 WriteRequest

HRESULT WriteRequest(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument itemList);

Description

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

Parameters	Description
invokeId	Unique identifier for the request
itemList	List of required items described by a DtmItemList specified by the DTMItemListSchema

Return value

Return Value	Description

Behavior

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

Execution of the WriteRequest() method must not change the data of the instance data set.

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

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

Comments

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

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

8.3.19 Interface IDtmSingleInstanceDataAccess

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

8.3.19.1 GetItemList

HRESULT GetItemList(

[out, retval] FdtXmlDocument* result);

Description

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

Parameters	Description

Return value

Return Value	Description
Result	XML document containing a DtmItemInfoList with the actual available parameters specified by the DTMItemListSchema

Behavior

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

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

Comments

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

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

8.3.19.2 Read

HRESULT Read(

[in] FdtXmlDocument itemSelectionList,

[out, retval] FdtXmlDocument* result);

Description

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

Parameters	Description
itemSelectionList	List of required items described by a DtmItemSelectionList specified by the DTMItemListSchema

Return value

Return Value	Description
Result	Requested data as DtmItemList specified by the DTMItemListSchema

Behavior

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

Comments

None

8.3.19.3 Write

HRESULT Write(

[in] FdtXmlDocument itemList,

[out, retval] FdtXmlDocument* result);

Description

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

Parameters	Description
itemList	List of required items described by a DtmItemList specified by the DTMItemListSchema

Return value

Return Value	Description
Result	Data as DtmItemList that contains the device data of the successfully written data specified by the DTMItemListSchema (may differ to the written value due to e.g. rounding procedures within the device)

Behavior

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

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

Comments

None

8.4 DTM ActiveXControl

8.4.1 Interface IDtmActiveXControl

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

8.4.1.1 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 its supports this functionality.

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

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

Comments

None

8.4.1.2 PrepareToRelease

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

Description

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

Parameters	Description

Return value

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

Behavior

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

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

Comments

None

8.5 FDT Channel

8.5.1 Interface IFdtChannel

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

8.5.1.1 GetChannelParameters

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

Description

Returns an XML document with fieldbus dependent channel specific parameters.

Parameters	Description
parameterPath	FdtXPath within the XML document
protocolId	UUID of a fieldbusprotocol. This UUID may be • one of the UUIDs returned by GetSupportedProtocols() and specified by DTMProtocolsSchema • a UUID defined within the appropriate <ChannelReference> element of the DTMParameter document returned by the DTM

Return value

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

Behavior

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

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

It is recommended, to return instead of an empty document a document based on the FDTBasicChannelParameterSchema.

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

Comments

None

8.5.1.2 GetChannelPath

HRESULT GetChannelPath(

[out, retval] FdtXPath* result);

Description

Returns an identifier for a channel.

Parameters	Description

Return value

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

Behavior

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

<systemTag>/<id>

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

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

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

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

<systemTag>/<id>//<systemTag>/<id>

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

<systemTag>/<id>

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

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

Comments

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

8.5.1.3 SetChannelParameters

HRESULT SetChannelParameters(

[in] FdtXPath parameterPath,

[in] FdtUUIDString protocolId,

[in] FdtXmlDocument xmlDocument,

[out, retval] VARIANT_BOOL* result);

Description

Sets changes of channel specific parameters.

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

Return value

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

Behavior

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

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

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

Comments

See also clause relating to State machine of instance data.

8.5.2 Interface IFdtChannelActiveXInformation

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

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

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

Comments

None

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

Comments

None

8.5.2.3 GetChannelFunctions

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

Description

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

Parameters	Description
operationState	XML document containing the current operation phase specified by the 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.

This 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 in the according clauses.

Comments

None

8.5.3 Interface IFdtCommunication

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

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

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

8.5.3.1 Abort

HRESULT Abort(

[in] FdtXmlDocument fieldbusFrame);

Description

Aborts a communication link to a device without any response.

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

Return value

Return Value	Description

Behavior

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

Comments

None

8.5.3.2 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
invokeId	Unique identifier for the request
protocolId	UUID of a fieldbusprotocol to be used. Identifies type of fieldbus specific schema
fieldbusFrame	Fieldbus-protocol-specific information how to connect. The structure is specified by a fieldbus specific schema, e.g., like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

Return value

Return Value	Description
TRUE	Request sent
FALSE	Request refused

Behavior

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

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

The response will be provided by IFdtCommunicationEvents2::OnConnectResponse().

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

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

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

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

Comments

See also clause FDT ‘Use cases and Scenarios’.

8.5.3.3 DisconnectRequest

```
HRESULT DisconnectRequest(
        [in] FdtUUIDString invokeId,
        [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
invokeId	Unique identifier for the request
fieldbusFrame	Fieldbus-protocol-specific information how to release the connection specified by a fieldbus specific schema e.g. like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema

Return value

Return Value	Description
TRUE	Request sent
FALSE	Request refused

Behavior

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

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

The response will be provided by OnDisconnectResponse().

Comments

See also clause FDT 'Use cases and Scenarios'.

8.5.3.4 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 an XML document with fieldbus protocol UUIDs specified by DTMProtocolsSchema. Only protocols supported by the configured sub-device can be returned.

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

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

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

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.

8.5.3.5 SequenceBegin

HRESULT SequenceBegin(

[in] FdtXmlDocument fieldbusFrame,

[out, retval] VARIANT_BOOL* result);

Description

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

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

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

Return value

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

Behavior

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

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

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

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

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

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

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

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

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

Comments

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

8.5.3.6 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.

8.5.3.7 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.

8.5.3.8 TransactionRequest

HRESULT TransactionRequest(
[in] FdtUUIDString invokeId,
[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
invokeId	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 an XML document with communication parameters specified by a fieldbus specific.

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

The response will be provided by OnTransactionResponse().

If a TransactionRequest() is called as part of a sequence definiton and the given SequenceTime is expired, the return value has to be false.

Comments

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

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

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

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

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

8.5.4 Interface IFdtChannelSubTopology

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

8.5.4.1 OnAddChild

HRESULT OnAddChild(

[in] BSTR childSystemTag);

Description

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

Parameters	Description
childSystemTag	SystemTag of the newly added child instance

Return value

Return Value	Description

Behavior

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

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

Comments

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

8.5.4.2 OnRemoveChild

HRESULT OnRemoveChild(

[in] BSTR childSystemTag);

Description

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

Parameters	Description
childSystemTag	SystemTag of the child DTM which was removed

Return value

Return Value	Description

Behavior

The channel is informed that a device, specified by its systemTag, was removed from the sub topology.

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

Comments

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

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

8.5.4.3 ScanRequest

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

Description

Requests the asynchronously scan of the sub topology.

Parameters	Description
invokeId	Unique identifier for the request

Return value

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

Behavior

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

Comments

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

8.5.4.4 ValidateAddChild

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

Description

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

Parameters	Description
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

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

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

See also 8.5.4.1.

8.5.4.5 ValidateRemoveChild

HRESULT ValidateRemoveChild(

[in] BSTR childSystemTag,

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description
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

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

8.5.5 Interface IFdtChannelSubTopology2

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

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

8.5.5.1 SetChildrenAddresses

HRESULT SetChildrenAddresses (

[in] FdtXmlDocument dtmDeviceList,

[out, retval] FdtXmlDocument* result);

Description

Requests bus address setting for specified device list.

Parameters	Description
dtmDeviceList	XML according DTMDeviceListSchema defining device instances where to set the address

Return value

Return Value	Description
Result	XML document containing result of address setting. DTMDeviceListSchema

Behavior

Requests setting of bus address for one device or a list of devices via IDtmParameter interface of the corresponding child DTMs. The request may specify that the called communication channel should open a user interface to request device address settings from user. To get a qualified response, error information is included in the returned document.

Comments

As part of executing this method the method IFdtActiveX2:OpenDialogActiveXControlRequest()

method may be used to request address selection from the user.

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

8.5.6 Interface IFdtChannelScan

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

8.5.6.1 ScanRequest

HRESULT ScanRequest(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument request,

[out, retval] VARIANT_BOOL* result);

Description

Requests the asynchronously scan of the sub topology.

Parameters	Description
invokeId	Unique identifier for the request
Request	Information about the address range(s) to scan. FDTScanRequestSchema

Return value

Return Value	Description
TRUE	Request of sub-topology scan accepted
FALSE	The operation failed. IDtmEvents:OnErrorMessage() assumed

Behavior

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

Comments

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

8.5.6.2 CancelAction

HRESULT CancelAction(

[in] FdtUUIDString invokeID,

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description
invokedID	Unique identification of scan request

Return value

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

Behavior

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

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

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

Comments

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

8.5.7 Interface IFdtFunctionBlockData

This interface must be supported by DTMs for failsafe devices

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

In a system that is able to provide failsafe access, it is important that all communication components support failsafe access. Communication channels have to support the propagation of the failsafe function calls. That is why the interface is mandatory for all channels with communication functionality. If the represented gateway device does not provide failsafe functionality itself, it is sufficient to pass the function call to the underlying communication.

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

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

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

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

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

8.5.7.1 GetFBInstanceData

HRESULT GetFBInstanceData(

[in] BSTR* systemTag,

[out, retval] FdtXmlDocument* result);

Description

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

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

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

Parameters	Description
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 driver that contains the corresponding Proxy FB of the DTM, reads the individual device parameter set directly out of the bus master and passes them to the DTM.

If there is still no assignment of the DTM to a Proxy FB, at first it executes the browsing function as it is described for SelectFBInstance ().

If a Communication-DTM is not able to support failsafe functionality, it will return a valid xmlDocument that contains no "FDTFailSafeData" element.

Comments

None

8.5.7.2 SelectFBInstance

HRESULT SelectFBInstance(

[in] BSTR* systemTag,

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description
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 Proxy FBs that contain individual device parameters of devices. The user has to select the Proxy FB of the corresponding failsafe device for the DTM. The bus-master-DTM stores the assignment of the Proxy FB to the calling DTM-Instance.

The mechanism of assigning a DTM to a failsafe device is defined in more detail in section "4.5.3 F Parameter Assignment Paths" of the document "PROFIBUS Profile for Safety Technology, Version 1.20, 23-Oct-2002" .

If a Communication-DTM is not able to support failsafe functionality, it will return a "false" success.

Comments

None

8.6 Channel ActiveXControl

8.6.1 Interface IFdtChannelActiveXControl

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

8.6.1.1 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

None

8.6.1.2 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 sent 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

None

8.6.2 Interface IFdtChannelActiveXControl2

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

8.6.2.1 Init2

HRESULT Init2(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument functionCall,

[in] IFdtChannel* channel,

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description
invokeId	Identifier for the started application
functionCall	XML document containing the function id for the requested function or user interface specified by the DTMFunctionCallSchema
Channel	Pointer to the channel business object

Return value

Return Value	Description
TRUE	Channel initialized
FALSE	The operation failed

Behavior

Sets the callback pointer of an ActiveX control to the according FdtChannel. The functionCall document informs the instance of the ActiveXControl about the context, it is started.

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

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

Comments

This function replaces the former IFdtChannelActiveXControl::Init() function that did not provide the current functionCall document to the channel ActiveX control.

A Frame Application according to FDT version 1.2.1 must use this method instead of the former method.

8.7 Block Type Manager

The BTM follows the rules specified for the DTM. A set of new interfaces is defined to replace the corresponding DTM interfaces and to be used with the BTM. The new interfaces follow the specification for the corresponding interfaces for the DTM, but are applicable to the BTM and are using the schemas specified for the BTM.

These interfaces are:

- IBtm
- IBtmInformation

- IBtmParameter

8.7.1 Interface IBtm

This interface is the main interface of a BTM. IBtm methods have the same behavior as specified with the interface IDtm. The only difference is that the methods are applied on a block type object (not on a device). The same methods are used for a DTM and for a BTM and the corresponding XML schemas definitions reflect the differences between the Block and the Device Type Manager objects.

For the IBtm interface, the only difference is in the schemas used in IDtm::InitNew() method.

8.7.1.1 Config

For description of this method refer to the method IDtm::Config()..

8.7.1.2 Environment

For description of this method refer to the method IDtm::Environment().

8.7.1.3 GetFunctions

For description of this method refer to the method IDtm::GetFunctions().

8.7.1.4 InitNew

The deviceType parameter changes as follows:

Parameters	Description
deviceType	XML document containing the manufacturer specific data like unique identifier for a block type specified by BtmInitSchema

For description of the method refer to IDtm::InitNew().

8.7.1.5 InvokeFunctionRequest

For description of this method refer to the method IDtm::InvokeFunctionRequest().

8.7.1.6 PrepareToDelete

For description of this method refer to the method IDtm::PrepareToDelete().

8.7.1.7 PrepareToRelease

For description of this method refer to the method IDtm::PrepareToRelease().

8.7.1.8 PrepareToReleaseCommunication

For description of this method refer to the method IDtm::PrepareToReleaseCommunication().

8.7.1.9 PrivateDialogEnabled

For description of this method refer to the method IDtm::PrivateDialogEnabled().

8.7.1.10 ReleaseCommunication

For description of this method refer to method IDtm::ReleaseCommunication().

8.7.1.11 SetCommunication

For description of this method refer to method IDtm::SetCommunication().

8.7.1.12 SetLanguage

For description of this method, refer to method IDtm::SetLanguage().

8.7.2 Interface IBtmInformation

IBtmInformation methods have the same behavior as specified with the interface IDtmInformation. The only difference is that the GetInformation method is applied on a block type object (not on a device). The new XML schema definition reflects the differences between the Block and the Device Type Manager objects.

8.7.2.1 GetInformation

The result parameter changes as follows:

Parameters	Description
result	XML document containing static BTM information specified by the BtmInformationSchema

For description of the method refer to IDtmInformation::GetInformation().

8.7.3 Interface IBtmParameter

This interface allows a Frame Application the access to BTM parameters. The

IBtmParameter methods have the same behavior as specified with the interface IDtmParameter. The only difference is that the methods IDtmParameter::GetParameters() and IDtmParameter::SetParameters() are applied to a block type object (not to a device). The new XML schema definition reflects the differences between the Block and the Device Type Manager objects.

8.7.3.1 GetParameters

The return parameter changes as follows:

Parameters	Description
result	XML document with the block type device specific parameters specified by the BtmParameterSchema

For a description of the method refer to IDtmParameter::GetParameters().

8.7.3.2 SetParameters

The input parameter xmlDocument changes as follows:

Parameters	Description
xmlDocument	XML document specified by the BtmParamterSchema. This document details block type specific parameters

For a description of the method refer to IDtmParameter::SetParameters().

8.8 BTM ActiveXControl

8.8.1 Interface IBtmActiveXControl

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

8.8.1.1 Init

HRESULT Init(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument functionCall,

[in] IBtm* btm,

[out, retval] VARIANT_BOOL* result);

Description

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

Parameters	Description
invokeId	This is a unique identifier for the started application
functionCall	XML document containing the function id for the requested function or user interface specified by the DTMFunctionCallSchema
Btm	Pointer to the BTM business object

Return value

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

Behavior

Set the callback pointer of an ActiveX control to the relevant BTM.

For detailed description of the method refer to IDtmActiveXControl::Init() but note that the ActiveX application is associated to the Block Type Manager.

Comments

None

8.8.1.2 PrepareToRelease

For a description of this method, refer to the method IDtmActiveXControl::PrepareToRelease() but note that the ActiveX control is associated to the Block Type Manager.

8.9 Frame Application

8.9.1 Interface IDtmEvents

This interface is the callback-interface for the DTMs.

8.9.1.1 OnApplicationClosed

HRESULT OnApplicationClosed(
[in] FdtUUIDString invokeId);

Description

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

Parameters	Description
invokeId	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 invokeId was set at the startup of an application or during the initialization of the ActiveX control

8.9.1.2 OnDownloadFinished

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

Description

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

Parameters	Description
invokeId	Identifier of the download request
success	Notification by a DTM, whether the asynchronously download function call IDtmOnlineParameter::DownloadRequest() is successfully finished

Return value

Return Value	Description

Behavior

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

Comments

None

8.9.1.3 OnErrorMessage

HRESULT OnErrorMessage(
[in] BSTR systemTag,
[in] BSTR errorMessage).

Description

Notification by a DTM about errors during a function call.

Parameters	Description
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 is up to the Frame Application to handle the information. In case of errors or warnings, the human readable string can be displayed in a dialog box of the Frame Application.

Comments

In order to give user helpful hints concerning errors, it is recommended to use this method in cases, where a function call failed and IFdtDialog::UserDialog() not be used.

8.9.1.4 OnFunctionChanged

HRESULT OnFunctionChanged(
[in] BSTR systemTag);

Description

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

Parameters	Description
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

None

8.9.1.5 OnChannelFunctionChanged

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

Description

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

Parameters	Description
systemTag	Identifier of the device instance
channelPath	Identifier of the channel

Return value

Return Value	Description
TRUE	
FALSE	

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

None

8.9.1.6 OnInvokedFunctionFinished

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

Description

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

Parameters	Description
invokeId	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

None

8.9.1.7 OnNavigation

HRESULT OnNavigation(
[in] BSTR systemTag);

Description

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

Parameters	Description
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 Application specific application will be started to guarantee a unique navigation behavior for the user.

The Frame Application can identify the calling DTM by the systemTag.

Comments

None

8.9.1.8 OnOnlineStateChanged

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

Description

A DTM sends an notification about its online state.

Parameters	Description
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

The DTM is allowed to send this notification if state has not changed to inform Frame Application that it is still in the same state. This may for example happen if no connection can be established (onlineState=FALSE).

The Frame Application should ignore these additional notifications.

8.9.1.9 OnParameterChanged

HRESULT OnParameterChanged(

[in] BSTR systemTag,

[in] FdtXmlDocument parameter);

Description

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

Parameters	Description
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

This notification could also be used by a Frame Application to trigger an update, e.g. to visualize the topology information.

The parent DTM gets only a notification, because the XML document, exchanged via OnParameterChanged(), is DTM specific and cannot be interpreted by a parent DTM IDtmParameter::GetParameters(). A parent DTM, which receives such a notification, can update its child-specific data by calling GetParameters() at the child DTM.

8.9.1.10 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
TRUE	
FALSE	

Behavior

The DTM has released all references to other components. After the Frame Application has received this notification it can release the DTM.

Comments

None

8.9.1.11 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 IDtm::SetCommunication(). After the Frame Application has received this notification it can call IDtm::ReleaseCommunication() and release the communication pointer itself.

Comments

None

8.9.1.12 OnPrint

HRESULT OnPrint(

[in] BSTR systemTag,

[in] FdtXmlDocument functionCall);

Description

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

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

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

None

8.9.1.13 OnProgress

HRESULT OnProgress(

[in] BSTR systemTag,

[in] BSTR title,

[in] short percent,

[in] VARIANT_BOOL show).

Description

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

Parameters	Description
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

This method should be used by DTMs during functions, which may take a longer time, to inform the Frame Application and at least the user about ongoing activities. If a DTM cannot determine the real progress, it can be useful to change the title for example.

It is up to the Frame Application how to handle the information.

Comments

None

8.9.1.14 OnScanResponse

HRESULT OnScanResponse(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument response);

Description

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

Parameters	Description
invokeId	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 an XML document which contains a list of fieldbus-related information to identify the connected devices. If no devices could be found the list will be empty.

Comments

Within FDT version 1.2.1 this method is obsolete. Only DTMs based on FDT version 1.2 are allowed to call this method.

8.9.1.15 OnUploadFinished

HRESULT OnUploadFinished(

[in] FdtUUIDString invokeId,

[in] VARIANT_BOOL success);

Description

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

Parameters	Description
invokeId	Identifier of the upload request
success	Notification by a DTM, whether the asynchronously upload function call IDtmOnlineParameter::UploadRequest() is successfully finished

Return value

Return Value	Description

Behavior

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

Comments

None

8.9.2 Interface IDtmEvents2

This interface is the callback-interface for DTMs supporting FDT version 1.2.1 or higher version. This interface extends the interface IDtmEvents by a new method. This interface is mandatory.

A DTM supporting 1.2.1 or higher version must call IDtmEvents2, if the Frame Application supports this interface. Instead of calling IDtmEvents::OnOnlineStateChanged() such a DTM then must use the IDtmEvents2::OnStateChanged() method.

8.9.2.1 OnStateChanged

HRESULT OnStateChanged(
[in] BSTR systemTag,
[in] FdtXmlDocument xmldoc);

Description

A DTM sends a notification about change in its state.

Parameters	Description
systemTag	Identifier of the device instance
xmldoc	XML document containing the function id for the requested function or user interface specified by the DTMStateSchema

Return value

Return Value	Description

Behavior

A DTM has performed a state transitions according to the DTM state machine between one of the following states:

communication-set

going-online

going-offline

online

If the transition was triggered by an error condition, e.g., ConnectResponse failed or OnAbort, the fdt:CommunicationError information must be provided within the XML document.

Comments

None

8.9.3 Interface IDtmScanEvents

This interface defines callback methods, called by DTMs when returning scan information.

8.9.3.1 OnScanResponse

HRESULT OnScanResponse (

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument response);

Description

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

Parameters	Description
invokedID	Unique identification of scan request

Return value

Return Value	Description
response	XML document containing the result of the topology scan as specified by a field bus specific schema (FDTxxxScanIdentSchema) . The fieldbus specific document must be transformed using protocol specific xsl to get a protocol independent document, which must validate against DTMScanIdentSchema

Behavior

The communication channel calls this method to return provisional results, final scan request results or error information.

Comments

Information regarding the type of response (provisional, final or error) is part of the response document.

8.9.3.2 OnScanHardwareResponse

HRESULT OnScanHardwareResponse (

[in] FdtUUIDString invokeID,

[in] FdtXmlDocument response);

Description

Notification by a DTM, that asynchronous scan hardware request operation has finished.

Parameters	Description
invokeID	Identifier of the request
Response	XML document containing the result of the scan hardware request specified by a field bus specific schema (FDTxxxScanIdentSchema) if request was called at a Gateway- or Device-DTM. The fieldbus specific document containing additional manufacturer specific extensions and must be transformed using protocol specific xsl to get a protocol independent document according DTMScanIdentSchema. If request was called at a Communication-DTM, then XML document according DTMScanIdentSchema is returned, which must not be transformed (ID entries which cannot be filled are left empty)

Return value

Return Value	Description

Behavior

Notification by a DTM, that asynchronous scan hardware request operation has finished. DTM passes XML containing information about found hardware.

If the operation was called in context of a Communication-DTM, then the XML contains information of all responsive hardware entities (interface cards, modems…), which can be handled by this DTM device type.

If the operation was called in context of a Gateway-/Device-DTM, then the XML only contains information of single device where the DTM was started for.

The XML contains error information if the scan hardware request failed.

Comments

None

8.9.4 Interface IDtmAuditTrailEvents

This interface must be used by all DTMs to send their audit trail information to the Frame Application. It is up to the Frame Application to implement the audit-trail-application itself which records the device specific information and supplies the user interface.

8.9.4.1 OnAuditTrailEvent

HRESULT OnAuditTrailEvent(

[in] BSTR systemTag,

[in] FdtXmlDocument logEntry);

Description

Notification by a DTM about changed data to be recorded by the Frame Application's audit trail tool.

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

Return value

Return Value	Description

Behavior

A DTM must call this function at the Frame Application to add an entry to the audit trail record.

Comments

The content of the log-entry depends on the DTM. The implementation of the audit trail tool is Frame Application-specific.

8.9.4.2 OnEndTransaction

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

Description

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

Parameters	Description
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.

8.9.4.3 OnStartTransaction

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

Description

Notification by a DTM that the following changes should be recorded by the Frame Application's audit trail tool.

Parameters	Description
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

None

8.9.5 Interface IFdtActiveX

This interface must be provided by a Frame Application that supports a GUI.

8.9.5.1 CloseActiveXControlRequest

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

Description

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

Parameters	Description
invokeId	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

None

8.9.5.2 OpenActiveXControlRequest

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

Description

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

Parameters	Description
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.

In general it is expected that the ActiveX behaves like a modeless dialog.

8.9.6 Interface IFdtActiveX2

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

8.9.6.1 OpenDialogActiveXControlRequest

HRESULT OpenDialogActiveXControlRequest(

[in] BSTR systemTag,

[in] FdtXmlDocument functionCall,

[out, retval] VARIANT_BOOL* result);

Description

Request to open ActiveX control for a certain DTM in a modal dialog of Frame Application.

Parameters	Description
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 was instantiated by the Frame Application
FALSE	The operation failed (i.e. because Frame Application is running without user interface)

Behavior

This method is used to open an ActiveX user interface, which must be instantiated and embedded in a modal dialog by the Frame Application. The Frame Application will establish the link between the new user interface and the DTM via IDtmActiveXControl::Init().

This method behaves always modal. Frame Application at least has to ensure that all ActiveX controls of calling DTM are disabled; no further user input is possible.

The opened ActiveX user interface must call IFdtActiveX:CloseActiveXControlRequest() if it wants to be closed later. OpenDialogActiveXControlRequest() works synchronously so that the DTM is block until ActiveX user interface and corresponding dialog are closed.

Comments

A DTM always should use this method for more complex dialogs rather than calling IFdtDialog::UserDialog() with FDTUserMessageSchema XML containing variables to edit, because this is not supported by almost all Frame Applications. Call of IFdtDialog::UserDialog() should be preferred if FDTUserMessage XML only contains text lines.

Caller of OpenDialogActiveXControlRequest() have to be aware of some synchronization circumstances which arise if modal dialogs are opened. FDT 1.2 Addendum specification discusses this topic for IFdtDialog::UserDialog() and IDtmEvents:OnErrorMessage() calls (refer FDT Best Practices – A closer look at message loops).

8.9.6.2 OpenDialogChannelActiveXControlRequest

HRESULT OpenDialogChannelActiveXControlRequest(

[in] BSTR channelPath,

[in] FdtXmlDocument functionCall,

[out, retval] VARIANT_BOOL* result);

Description

Request to open ActiveX control for a certain channel object (identified by the channelPath) in a modal dialog of Frame Application.

Parameters	Description
channelPath	Identifier of the channel instance (as returned by IFdtChannel::GetChannelPath())

functionCall XML document containing the ActiveX function id for the requested user interface specified by the DTMFunctionCallSchema
Return value

Return Value	Description
TRUE	The requested ActiveX control was instantiated by the Frame Application
FALSE	The operation failed (i.e. because Frame Application is running without user interface)

Behavior

This method is used to open an ActiveX user interface, which must be instantiated and embedded in a modal dialog by the Frame Application. The Frame Application will establish the link between the new user interface and the channel via IFdtChannelActiveXControl2::Init2().

This method behaves always modal. Frame Application at least has to ensure that all ActiveX controls of related DTM are disabled; no further user input is possible.

The opened ActiveX user interface must call IFdtActiveX2:CloseChannelActiveXControlRequest() if it wants to be closed later. OpenDialogChannelActiveXControlRequest() works synchronously so that the call is block until ActiveX user interface and corresponding dialog are closed.

Comments

This function should be used for more complex dialogs rather than calling IFdtDialog::UserDialog() with FDTUserMessageSchema XML containing variables to edit, because this is not supported by almost all Frame Applications. Call of IFdtDialog::UserDialog() should be preferred if FDTUserMessage XML only contains text lines.

Caller of OpenDialogChannelActiveXControlRequest() have to be aware of some synchronization circumstances which arise if modal dialogs are opened. FDT 1.2 Addendum specification discusses this topic for IFdtDialog::UserDialog() and IDtmEvents:OnErrorMessage() calls (refer FDT V 1.2 – Mai 2003 - Addendum, chapter 2.30.4 – A closer look at message loops, paragraph DTM's point of view).

8.9.6.3 CloseChannelActiveXControlRequest

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

Description

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

Parameters	Description
invokeId	Identifier for the started ActiveX control

Return value

Return Value	Description
TRUE	The ActiveX control will be released by the Frame Application
FALSE	The operation failed

Behavior

This method is used by a DTM if it wants to close an ActiveX user interface which was instantiated and embedded by the Frame Application. The Frame Application will release the link between the user interface and the channel object. This is done via IFdtChannelActiveXControl::PrepareToRelease().

Comments

None

8.9.6.4 OpenChannelActiveXControlRequest

HRESULT OpenChannelActiveXControlRequest(
[in] BSTR channelPath,
[in] FdtXmlDocument functionCall,
[out, retval] VARIANT_BOOL* result);

Description

Request to start an ActiveX functionality defined by the function call id for a certain channel object (identified by the channelPath).

Parameters	Description
channelPath	Identifier of the channel instance (as returned by IFdtChannel::GetChannelPath())
functionCall	XML document containing the ActiveX 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 if the caller wants to open an ActiveX user interface which must be instantiated and embedded by the Frame Application. The Frame Application will establish the link between the new user interface and the related object via IFdtChannelActiveXControl::Init().

Comments

The caller of this method has also to take into account the additional information (application id and operation phase) passed via the XML document.

The Frame creates the Channel-ActiveX and assigns it to the channel object (referenced by channelPath) by calling IFdtChannelActiveXControl::Init().

8.9.7 Interface IFdtBulkData

The Bulk Data Interface offers DTMs the possibility to store a big amount of additional data like protocols of measured values or historical data of configuration changes. The DTMs are able to do that in a private way.

If access to these data is not possible, there must be no impact to the instance specific configuration data set of a DTM. The instance data set of a DTM must be always consistent to the configuration data stored via the standard IPersistXXX interface. It is in the responsibility of the Frame Application to create a backup strategy for this type of data.

The interface handles bulks of data at a device level.

8.9.7.1 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

Return Value	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.

8.9.7.2 GetProjectRelatedPath

HRESULT GetProjectRelatedPath(

[in] BSTR systemTag,

[out, retval] BSTR* result);

Description

Returns the project related path for bulk data. Returns a unique file system path (directory) for any combination of project and DTM type (e.g. it returns different paths for the same DTM type within two projects).

Parameters	Description
systemTag	Identifier of the device instance

Return value

Return Value	Description
result	Project related path (directory) for bulk data including a trailing backslash

Behavior

The Frame Application offers a DTM a way to request a project specific path to an area which could be used to store data in a DTM specific way. It is up to a DTM to decide in which way the data will be stored. It could be a binary format by using IStorage or PropertyBag or even an ASCII-File.

The Frame Application is responsible for providing 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.

8.9.8 Interface IFdtContainer

This is the main interface of the Frame Application. It supports the functions for the instance data management like locking within a multi user system.

8.9.8.1 GetXmlSchemaPath

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

Description

Returns a path where the default schemas are stored

Parameters	Description

Return value

Return Value	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

None

8.9.8.2 LockDataSet

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

Description

A DTM sends an notification to the Frame Application that it wants to have exclusive write access for the currently loaded data set.

Parameters	Description
systemTag	Identifier of the device instance

Return value

Return Value	Description
TRUE	Data set is locked for write access
FALSE	Data set could not be locked. DTM has read access only

Behavior

Via this method a DTM notifies the database that it wants to modify or delete the specified instance data set. It is up to the Frame Application to validate this request within a multi-user multi-session system. If the request fails, the DTM must not change any data and should set all input fields to ‘non edit’ in case of an open user interface.

It is in the responsibility of the Frame Application to reject write-requests if a DTM does not take care about its read only status.

A DTM must not lock it is instance data for the complete lifetime. Instead the DTM should try to lock data only if instance data is going to be modified and should unlock the data after the instance data is saved and no further modifications are expected.

Comments

Within a single user system the method always returns TRUE.

Examples for lock conditions are:

- a user interface is active, that allows changing the instance data;
- the DTM received the request to change data via its COM-interfaces (IDtmParameter, IDtmInstanceData).

8.9.8.3 SaveRequest

HRESULT SaveRequest(

[in] BSTR systemTag,

[out, retval] VARIANT_BOOL* result);

Description

Informs the Frame Application that it should store the changed data.

Parameters	Description
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.

Transient data remains in transient state until the Frame Application successfully completes IPersistXXX:Save().

Comments

This method is the only method to inform the Frame Application that it should store the changed data. Even if the IPersistXXX::IsDirty property is available, it will not be used by a Frame Application. The Frame Application could also initiate the persistence interface of a DTM by itself.

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

8.9.8.4 UnlockDataSet

HRESULT UnlockDataSet(

[in] BSTR systemTag,

[out, retval] VARIANT_BOOL* result);

Description

Notification to the Frame Application that the DTM wants to unlock a data set and needs only read access for the currently loaded data set.

Parameters	Description
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.

A DTM must not lock it is instance data for its complete lifetime. Instead the DTM should try to lock data only if instance data is going to be modified and should unlock the data after the instance data is saved and no further modifications are expected.
Comments

Within a single user system the method always returns TRUE.

8.9.9 Interface IFdtDialog

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

8.9.9.1 UserDialog

HRESULT UserDialog(

[in] BSTR systemTag,

[in] FdtXmlDocument userMessage,

[out, retval] FdtXmlDocument* result);

Description

Call the Container to display a message

Parameters	Description
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

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

None

8.9.10 Interface IFdtTopology

This interface provides the access to the complete system topology. A Frame Application has always to configure the sub-topology of a channel via the interface IFdtChannelSubTopology, so that the channel or at least the according DTM can validate the connections.

8.9.10.1 CreateChild

HRESULT CreateChild(

[in] FdtXmlDocument deviceType,

[in] FdtXPath channelPath,

[out, retval] BSTR* result);

Description

A DTM sends a request to the Frame Application to create a new instance data set of the specified device type.

Parameters	Description
deviceType	XML document containing the information specified by DTMInitSchema
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 clause 6.13.1 Instantiation of a New DTM. It is also in the responsibility of the Frame Application to insert the created DTM into the topology.

Comments

None

8.9.10.2 DeleteChild

HRESULT DeleteChild(

[in] BSTR systemTag,

[in] FdtXPath channelPath,

[out, retval] VARIANT_BOOL* result);

Description

Remove the DTM specified by systemTag from the topology identified by channelPath. If this was the last reference within the topology, remove the instance data set

Parameters	Description
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 IDtm::PrepareToDelete() with respect of the behavior. The operation will also fail if a sub-topology exists.

Comments

None

8.9.10.3 GetChildNodes

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

Description

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

Parameters	Description
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 an XML document containing the systemTags of all child DTMs of the DTM identified by its system tag and the channel path.

The topology information is globally accessible for all DTMs.

Comments

Only a Frame Application that supports Nested Communication has to implement this method.

8.9.10.4 GetDtmForSystemTag

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

Description

Return the associated DTM according the given system tag

Parameters	Description
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

None

8.9.10.5 GetDtmInfoList

HRESULT GetDtmInfoList(

[out, retval] FdtXmlDocument * result);

Description

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

Parameters	Description

Return value

Return Value	Description
Result	List of DtmInfo according the DTMInfoListSchema

Behavior

Returns an XML-document containing a list of DTM related information. This information could be used to create an XML document of type DTMInitSchema according the usage of CreateDtmInstance()

Comments

It is up to the Frame Application to decide which DTM information will be available via the list. E.g. the list could contain only information concerning HART devices even if PROFIBUS devices are installed.

8.9.10.6 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.

8.9.10.7 MoveChild

HRESULT MoveChild(

[in] BSTR systemTag,

[in] FdtXPath destinationChannelPath,

[out, retval] VARIANT_BOOL* result);

Description

Move a DTM defined by systemTag from the current position within the topology to the position defined by destinationChannelPath. The complete sub-topology will be moved.

Parameters	Description
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().

Comments

None

8.9.10.8 ReleaseDtmForSystemTag

HRESULT ReleaseDtmForSystemTag(

[in] BSTR systemTag,

[out, retval] VARIANT_BOOL* result);

Description

Release the associated DTM according the given system tag

Parameters	Description
systemTag	Identifier of the device to release

Return value

Return Value	Description
TRUE	The operation succeeded
FALSE	The operation failed

Behavior

Release the reference to the associated DTM according the given system tag. This method is used only in combination with GetDtmForSystemTag().

Comments

None

8.9.11 Interface IDtmRedundancyEvents

This Frame Application interface provides the access for parent components handling redundant slaves. A Frame Application not implementing this interface is not able to display redundancy information within it's topology information. (for redundancy refer clause 4.10.7)

8.9.11.1 OnAddedRedundantChild

HRESULT OnAddedRedundantChild(

[in] BSTR systemTag,

[in] FdtXPath channelPath,

[out, retval] VARIANT_BOOL* result);

Description

A parent component sends this event to the Frame Application if a Device-DTM handling a redundant device is added to the topology. The Frame Application is then able to display the instance at an additional redundant communication channel.

Parameters	Description
systemTag	Identifier of the Device-DTM instance representing a redundant slave
channelPath	Specifies the redundant channel path of the parent DTM to which the DTM is connected

Return value

Return Value	Description
TRUE	Operation succeeded
FALSE	Operation failed

Behavior

The parent component adds the Device-DTM specified by systemTag, handling a redundant slave, to the communication channel identified by channelPath. The Frame Application must not call ValidateAddChild() or OnAddChild() on this channel.

Comments

None

8.9.11.2 OnRemovedRedundantChild

HRESULT OnRemovedRedundantChild(

[in] BSTR systemTag,

[in] FdtXPath channelPath,

[out, retval] VARIANT_BOOL* result);

Description

A parent component sends this event to the Frame Application if a Device-DTM handling a redundant device is removed from the topology. The Frame Application is able to hide the instance at the additional redundant communication channel.

Parameters	Description
systemTag	Identifier of the device instance representing a redundant slave
channelPath	Specifies the redundant channel path of the parent DTM to which the DTM is connected

Return value

Return Value	Description
TRUE	Operation succeeded
FALSE	Operation failed

Behavior

The parent component removes the Device-DTM specified by systemTag, handling a redundant slave, from the communication channel identified by channelPath. The Frame Application must not call ValidateRemoveChild() or OnRemoveChild() on this channel.

Comments

None

8.9.12 Interface IDtmSingleDeviceDataAccessEvents

This interface is the callback interface for single device data access implemented by the Frame Application.

8.9.12.1 OnItemListResponse

HRESULT OnItemListResponse(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument response);

Description

Provides the response to ItemListRequest() identified by the invoke id. ItemListResponse provides an XML document containing a list of the available device specific parameters and process values. Within a DTM this list may contain items related to configuration parameters, process values as well as asset management related data like stroke counter. In DTM state

'configured' the returned item list is based on the current instance data set, which could be different from the configuration of the device. In this case Read- and WriteRequests may fail.

Parameters	Description
invokeId	Unique identifier for the request
response	XML document containing a DtmItemInfoList with the actual available parameters specified by the DTMItemListSchema Return Value

Return value

Return Value	Description

Behavior

The method provides a list of items that can be read or written from/to the DTM via ReadRequest() or written to the DTM via WriteRequest(). The source for this data is the device itself.

Items provided within these list may also be available as channel objects (provided by IDtmChannel::GetChannels()) or modeled as an exported variable (DtmVariable provided by IDtmParameter::GetParameters() or IBtmParameter::GetParameters()). The related items can be identified via the attribute 'semanticId' (refer to clause FDT Data Types).

Comments

The contents of the provided XML document may depend on the current configuration of the device. If the contents is changed, a DTM has to inform the Frame Application by sending IDtmSingleDeviceDataAccess::OnItemListChanged().

8.9.12.2 OnDeviceItemListChanged

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

Description

The DTM informs the Frame Application that the content of the item list has been changed.

Parameters	Description
systemTag	Identifier of the device instance

Return value

Return Value	Description

Behavior

Via this method a DTM informs the Frame Application that the content of the item list has changed (the available items in general, not the value). This may happen if the content of the list depends on the configuration of the device.

OnItemListChanged should not be fired in case of pending responses.

Comments

None

8.9.12.3 OnReadResponse

HRESULT OnReadResponse(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument response);

Description

Provides the response to ReadRequest() identified by the invoke id.

Parameters	Description
invokeId	Unique identifier for the request
Response	Received data as DtmItemList specified by the DTMItemListSchema. Return Value

Return value

Return Value	Description

Behavior

Via this method a Frame Application that sent the read-request, receives the requested data from the DTM.

Comments

None

8.9.12.4 OnWriteResponse

HRESULT OnWriteResponse(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument response);

Description

Provides the response to WriteRequest() identified by the invoke id.

Parameters	Description
invokeId	Unique identifier for the request
Response	Received data as DtmItemList that contains the device data of the successfully written data specified by the DTMItemListSchema (may differ to the written value due to e. g. rounding procedures within the device)

Return value

Return Value	Description

Behavior

Via this method a Frame Application may receive information from the DTM about the successfully written data.

Comments

None

8.9.13 Interface IDtmSingleInstanceDataAccessEvents

This interface is the callback interface for single instance data access implemented by the Frame Application.

8.9.13.1 OnInstanceItemListChanged

HRESULT OnInstanceItemListChanged(

[in] BSTR systemTag);

Description

The DTM informs the Frame Application that the content of the item list has been changed.

Parameters	Description
systemTag	Identifier of the device instance

Return value

Return Value	Description

Behavior

Via this method a DTM informs the Frame Application that the content of the item list has changed (the available items in general, not the value). This may happen if the content of the list depends on the configuration of the device.

OnItemListChanged should not be fired in case of pending responses.

Comments

None

8.9.14 Interface IFdtBtmTopology

This interface provides the access to the Block topology. IFdtBtmTopology methods have the same behavior as specified with the interface IFdtTopology. The only difference is that the methods are used to apply to a block type object (not to a device). The new XML schema definition reflects the differences between the Block and the Device Type Manager objects.

If a DTM has communication channels to support both - DTMs and BTMs, the IFdtBtmTopology interface will provide information only about BTMs. The information related to the DTM topology will be provided by the IFdtTopology interface.

8.9.14.1 CreateChild

The input parameter deviceType changes as follows:

Parameters	Description
deviceType	XML document containing the information specified by the BTMInitSchema For description of the method refer to IFdtTopology::CreateChild()

8.9.14.2 DeleteChild

For description of the method refer to IFdtTopology::DeleteChild().

8.9.14.3 GetChildNodes

For description of the method refer to IFdtTopology::GetChildNodes().

8.9.14.4 GetBtmForSystemTag

For description of the method refer to IFdtTopology::GetDtmForSystemTag().

8.9.14.5 GetBtmInfoList

For description of the method refer to IFdtTopology::GetDtmInfoList().

8.9.14.6 GetParentNodes

For description of the method refer to IFdtTopology::GetParentNodes().

8.9.14.7 MoveChild

For description of the method refer to IFdtTopology::MoveChild().

8.9.14.8 ReleaseBtmForSystemTag

For description of the method refer to IFdtTopology::ReleaseDtmForSystemTag().

8.10 General concepts

This subclause 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.

8.10.1 Task related FDT interfaces

All FDT interfaces are task related. Each object must implement a mandatory set of interfaces expected by all other objects. By implementing optional FDT interfaces an object is able to support additional functionality, e.g., a DTM may provide documentation in XML format or special diagnostics, a frame-application may provide audit trail functionality.

Each object is able to determine the availability of such optional interfaces of other objects during runtime.

All defined FDT interfaces are fixed and will never be changed. Additional future extensions will be based on additional optional interfaces. A DTM or frame-application is then able to add a higher FDT version support by implementing or using such additional FDT interfaces.

Depending on the functionality of a DTM, additionally to the default set of mandatory interfaces an extra set of interfaces may be mandatory to support (see Table 7).

Table 7 – Task related DTM interfaces

Device Type Manager	Availability	User Interface	ActiveX Control User Interface	Device with Online Data	Gateway DTM	Communi cation-DTM for PC/Fieldu s adapter
IPersistXXX	Mandatory					
IDtm	Mandatory					
IDtm2	Mandatory					
IDtmActiveXInformation	Optional		Mandatory			
IDtmApplication	Optional	Mandatory				
IDtmChannel	Optional				Mandatory	Mandatory
IDtmDocumentation	Mandatory					

Device Type Manager	Availability	User Interface	ActiveX Control User Interface	Device with Online Data	Gateway DTM	Communication-DTM for PC/Fieldus adapter
IDtmDiagnosis	Mandatory					
IDtmImportExport	Optional					
IDtmInformation	Mandatory					
IDtmInformation2	Mandatory					
IDtmOnlineDiagnosis	Mandatory					
IDtmOnlineParameter	Optional			Mandatory		
IDtmParameter	Mandatory					
IFdtCommunicationEvents	Optional			Mandatory	Mandatory	
IFdtCommunicationEvents2	Optional			Mandatory	Mandatory	
IDtmHardwareIdentification	Optional					
IDtmSingleDeviceDataAccess	Optional			Mandatory		
IDtmSingleInstanceDataAccess	Mandatory					
IFdtEvents	Mandatory					

The mandatory interfaces of a DTM-ActiveX are shown in Table 8.

Table 8 – Task related DTM-ActiveX interfaces

DTM ActiveX Control	Availability
IDtmActiveXControl	Mandatory

Depending on the functionality of a Channel, additionally to the mandatory default interface an extra set of interfaces may be mandatory to support (see Table 9).

Table 9 – Task related Channel interfaces

FDT Channel	Availability	Channel with User Interface	Channel of Gateway DTM	Communication-DTM for PC/Fieldus adapter
IFdtChannel	Mandatory			
IFdtChannelActiveXInformation	Optional	Mandatory		
IfdtChannelSubTopology	Optional		Mandatory	Mandatory
IfdtChannelSubTopology2	Optional		Mandatory	Mandatory
IfdtCommunication	Optional		Mandatory	Mandatory
IfdtFunctionBlockData			Mandatory	Mandatory
IfdtChannelScan			Mandatory	Mandatory

The mandatory interfaces of a Channel-ActiveX are shown in Table 10.

Table 10 − Task related Channel-ActiveX interfaces

FDT Channel ActiveXControl	Availability
IfdtChannelActiveXControl	Mandatory
IfdtChannelActiveXControl2	Mandatory

The mandatory interfaces of a BTM are shown in Table 11.

Table 11 − Task related BTM interfaces

BTM	Availability
IBtm	Mandatory
IDtmActiveXControlInformation	Mandatory
IDtmChannel	Optional
IDtmDocumentation	Mandatory
IDtmDiagnosis	Mandatory
IDtmImportExport	Optional
IBtmInformation	Mandatory
IDtmInformation2	Mandatory
IDtmOnlineDiagnosis	Mandatory
IDtmOnlineParameter	Mandatory
IBtmParameter	Mandatory
IFdtCommunicationEvents	Mandatory
IFdtCommunicationEvents2	Mandatory
IDtmHardwareIdentification	Mandatory
IDtmSingleDeviceDataAccess	Mandatory
IDtmSingleInstanceDataAccess	Mandatory
IFdtEvents	Mandatory

The mandatory interfaces of a Channel-ActiveX are shown in Table 12.

Table 12 − Task related BTM-ActiveX interfaces

BTM ActiveXControl	Availability
IBtmActiveXControl	Mandatory

Depending on the functionality of a Frame Application, not all interfaces defined for a Frame Application must be supported (see Table 13).

Table 13 − Task related Frame Application interfaces

Frame Application	Availability	With User Interface
IDtmEvents	Mandatory	
IDtmEvents2	Mandatory	
IDtmAuditTrailEvents	Mandatory	
IFdtActiveX	Optional	Mandatory
IFdtActiveX2	Optional	Mandatory
IFdtBulkData	Optional	
IFdtContainer	Mandatory	
IFdtDialog	Mandatory	
IFdtTopology	Mandatory	
IFdtBtmTopology	Mandatory	
IDtmScanEvents	Optional	
IDtmRedundancyEvents	Optional	
IDtmSingleDeviceDataAccessEvents	Mandatory	
IDtmSingleInstanceDataAccessEvents	Mandatory	

Furthermore, the prefixes FDT, DTM and BTM are reserved for identifiers and names defined in the FDT specification. This prevents conflicts of further releases with private extensions of interfaces or definitions.

In general, all FDT interfaces are designed with fieldbus- and manufacturer-neutral methods. Extensions for a new fieldbus are done via new XML schemas. Functional extensions for new tasks will be provided by new interfaces.

8.10.2 Return values of interface methods

Interface methods indicate success or failure of a method call by well-defined return values (marked as [out, retval]). COM errors (HRESULT) must not be used to return FDT function related errors, except if it is stated in the specification. If no return value is defined (e.g. for all Event methods), it is assumed that the method always succeeds.

8.10.3 Dual interfaces

All interfaces defined within the FDT Specification are implemented as dual interfaces. This decision was made to support C++, Visual Basic, Java and other COM compliant development languages. The functionality of an object is implemented in separate task-oriented interfaces, so that only the default interface is accessible via the dispatch interface. This prevents marshalling 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.

8.10.4 Unicode

All string parameters to the FDT interfaces are BSTRs and are therefore UNICODE strings.

Microsoft MIDL Version 3.0 or later is required to correctly compile the IDL code and generate proxy/stub software. Microsoft Windows NT 4.0 (or later), or Windows 95 with DCOM support is required to properly handle the marshalling 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.

8.10.5 Asynchronous vs. synchronous behavior

In general each function call is synchronous. Within FDT there are two special cases of asynchronous behavior.

- After starting the user interface of a DTM, the DTM works asynchronous to the Frame Application. Asynchronous in this case means that the user works with the DTM and the Frame Application is the server for communication and data access. This state ends by a notification to the Frame Application, when the DTM closes the user interface.
- While a DTM has opened its user interface, the DTM uses the asynchronous behavior at the communication interface. The time a communication function call needs to return depends on the system topology and the bus protocol and would block an application for seconds. Dividing a communication function call to a request and a response function causes a non-blocking behavior without the pain of multi-threading implementation. The DTM sends its request or several requests without being disturbed by incoming responses. When a response is available, the DTM gets a notification and can receive the response from the communication component. Due to this mechanism, on the 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.

8.10.6 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.

8.10.7 Slave redundancy

8.10.7.1 Redundancy scenarios

The scope of slave redundancy within FDT is one DTM for configuration of one device connected to either, see Figure 22.

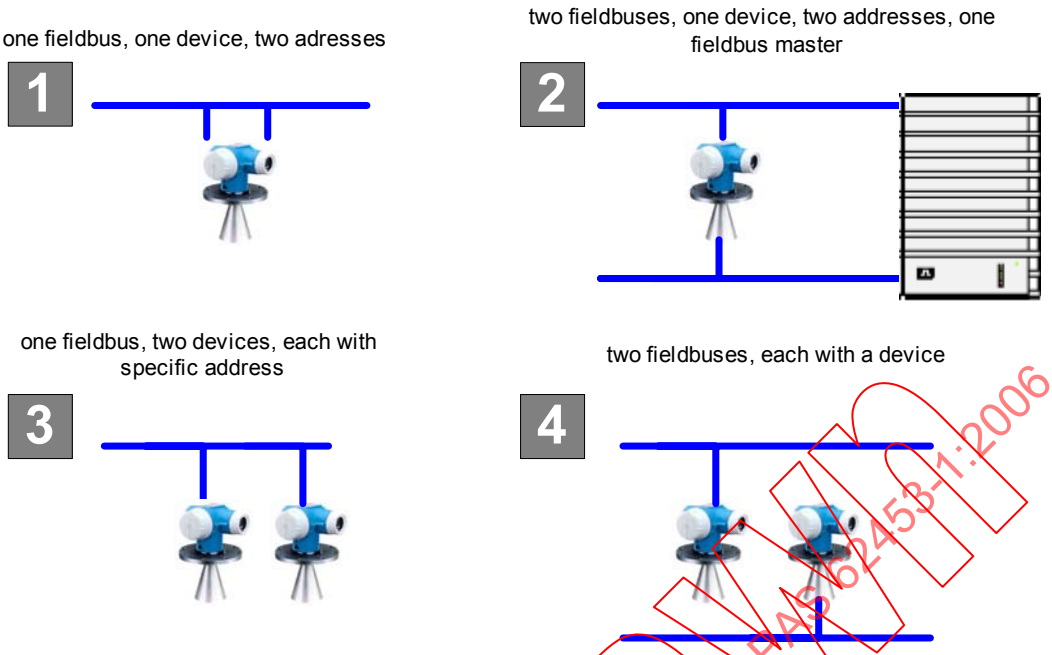

Figure 22 – Redundancy scenarios

- Scenario 1: one fieldbus using two different addresses
- Scenario 2: connected to two different fieldbuses controlled by one bus master.
- Separate redundant devices (scenario 3 and 4) can not be handled within FDT, these scenarios would require additional Frame Application functionality, e.g. with regard to data synchronization.

In Scenarios 1 and 2 redundant field buses must be managed by one redundancy aware parent component (e.g. a Communication-DTM) specific for the appropriate redundant field bus technology. This parent component should typically also be able to handle DTMs for redundant and not-redundant field devices concurrently.

8.10.7.2 Redundancy support in frame application

The DTMs able to handle slave redundancy can be used by any FDT Frame Application without need for specific redundancy support or knowledge. A FDT Frame Application may optionally implement the IDtmRedundancyEvents interface. Such a Frame Application is then able to display redundant slaves in it's topology view, e.g. a field device connected to two different lines or by using specific device icons within a topology treeview.

Within a Frame Application not implementing the IDtmRedundancyEvents interface a DTM representing a redundant slave can not be distinguished within the topology view. But the DTM and the parent component themselves typically display additional information, e.g. additional fieldbus addresses. Nevertheless within such a Frame Application these DTMs provide full functionality with regard to fieldbus redundancy.

As a redundant slave is represented by one DTM instance no additional functionality with regard to dataset synchronization or multiuser is required.

8.10.7.3 Parent component for redundant fieldbus

Fieldbus communication for redundant slaves is handled by a fieldbus and redundancy technology specific parent component, e.g. a specific Communication-DTM. All fieldbuses used to connect redundant slaves must be handled by one instance of such a parent component. In Scenario 2 for example, each fieldbus line is represented by an appropriate communication channel of the parent component. Therefore a Frame Application is able to

use such a parent component without knowledge about the physical redundant fieldbus structure.

During a ValidateAddChild() and a OnAddChild() the parent component is able to detect if a Device-DTM to be added is able to handle redundancy by examining the parameter document of the instantiated DTM. In this case a specific address selection dialog within the parent component can be used. This address selection is fieldbus and redundancy specific. It is up to the parent component if redundancy is handled automatically or only after user interaction.

If the Frame Application has implemented the IDtmRedundancyEvents Interface the parent component has to inform the Frame Application about the redundant DTM by calling the OnAddedRedundantChild() method.

The parent component must be able to handle all possible communication paths to the redundant device. Within a SetParameter() call complete redundant address information can be provided to the DTM instance handling this redundant device. This DTM is then able to use this information to display all available communication paths. During a ConnectRequest() the DTM provides this complete address information to the parent component. The parent component is then able to select the currently active communication path. The communication path can be changed within the parent component without need for interaction with the Device-DTM.

A parent component may also be able to handle a DTM representing a not redundant field device. In this case this DTM is handled without using the redundancy specific information within FDT xml documents.

8.10.7.4 Redundancy support in device-DTM

A DTM able to handle a redundant device provides additional information within it is parameter document.

After an OnAddChild() complete redundant address information can be given by the parent component to the Device-DTM. The Device-DTM is then able to detect if it is appropriate slave is used as a redundant slave. This complete address information must be used by the Device-DTM within a IFdtCommunication::ConnectRequest(), but can also be used for diagnosis and status information. Typically now additional redundancy handling is required within the Device-DTM itself.

A DTM handling a redundant device must check all addresses provided during a SetParameter() call. If the Device-DTM is not able to handle this addresses, e.g., because only a vendor-specific offset can be used, an error message should be created and a FALSE should be returned to the calling parent component. In this case the parent component is able to detect Device-DTMs which can not be used at the redundant fieldbus. A Device-DTM must save all redundancy information in it's instance dataset.

A DTM representing a redundant slave can be used as the child of a non-redundant aware parent component. In this case the Device-DTM must not use the additional FDT redundancy functionality, dialogs or redundant specific contents of FDT xml documents.

8.10.7.5 Scan and redundant slaves

Fieldbus scan provides for each address of a redundant slave one entry, a redundant slave can not be detected.

In scenario 1 mapping of redundant addresses to one instance is only possible for a DTM itself, in scenario 2 mapping can only be done based on project knowledge. Typically a scan is not executed on a redundant system.

8.10.7.6 Topology import-/export

A Frame Application not aware of DTMs handling redundant slaves is not able to provide redundancy information within a FDT topology export file.

A Frame Application aware of DTMs handling redundant slaves can only add a DTMNode element at ChannelNodes if the appropriate DTM instance has not been added to topology by a IDtmRedundancy::OnAddedRedundantChild() event call. Instead the appropriate DTM element of the topology document should contain a BusInformation element containing the redundant address information.

8.10.8 Field bus scanning and DTM assignment

8.10.8.1 Device identification

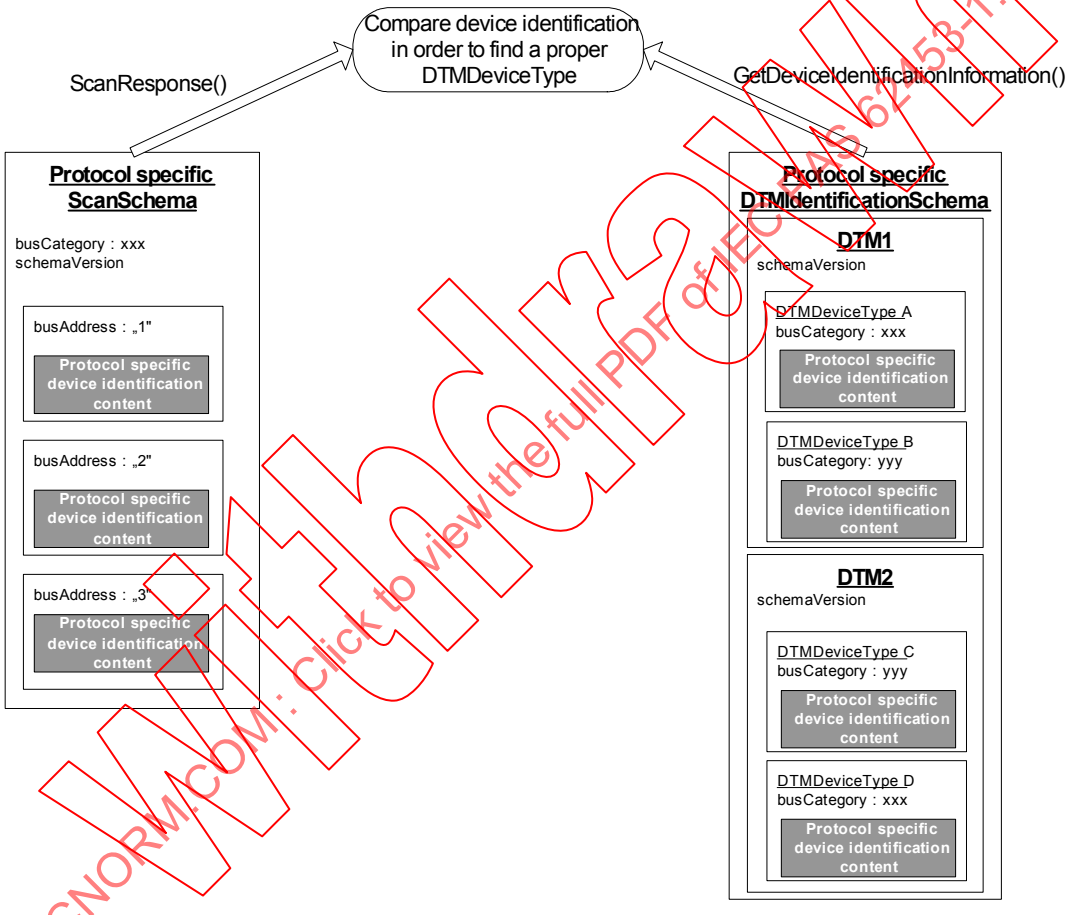

Figure 23 – Device identification

In order to identify a proper DTMDeviceType, a Frame Application has to compare identification information of scanned physical devices with identification information of a DTMDeviceType. These files must be converted to a fieldbus independent format using a fieldbus specific XSL transformation.

Figure 24 shows, how protocol specific schemas are integrated in the FDT specification and used by Frame Application and DTMs (example HART is shown).

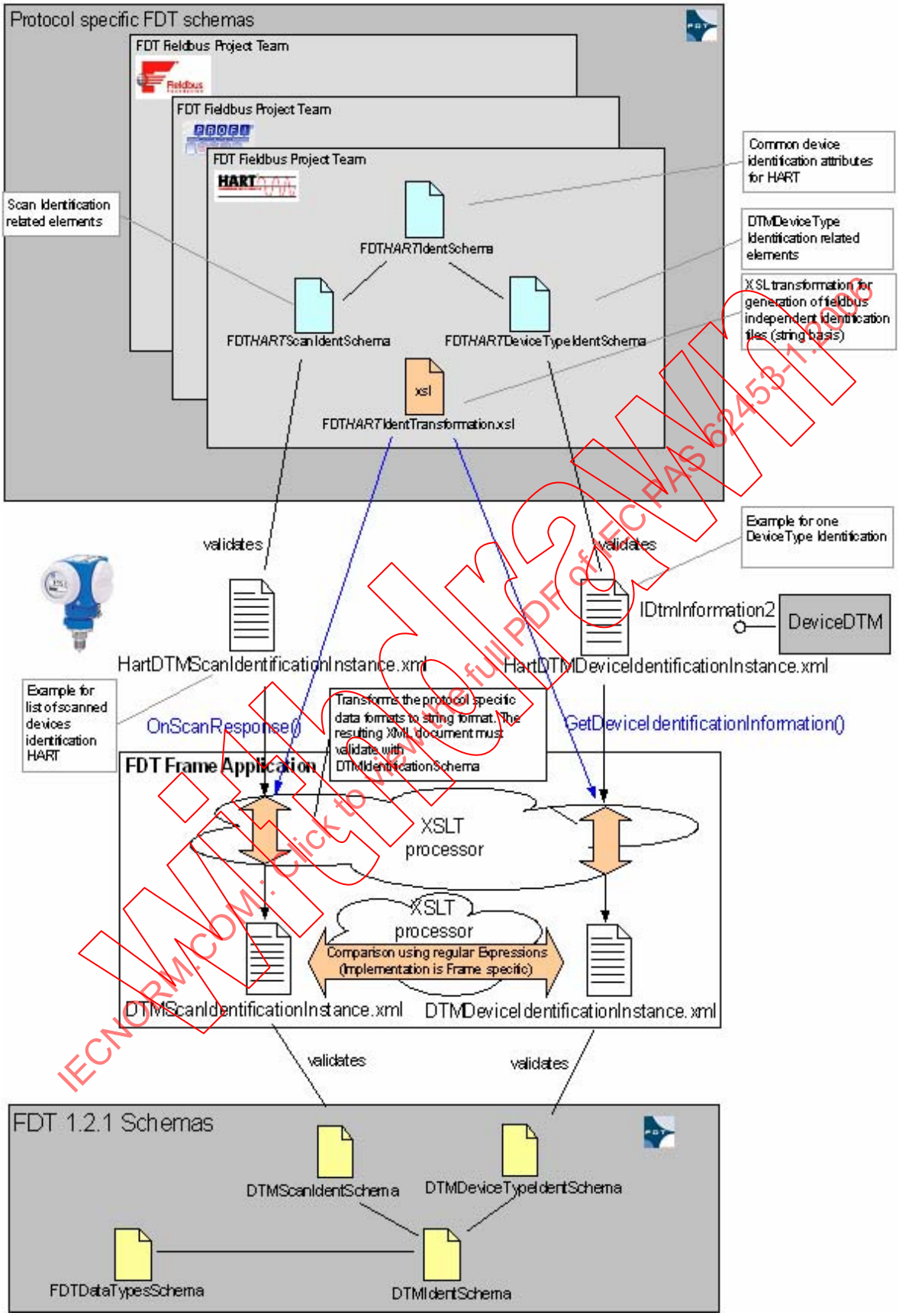

Figure 24 − Structural overview

8.10.8.2 Protocol specific transformation style sheet (xsl)

As shown in the structural overview Figure 24, the protocol specific FDT specification extension covers a transformation style sheet (.xsl) in addition to schemas. This xsl is to be used by a Frame Application in order to convert the protocol specific formats included in the identification (scan and DTM) XML documents into strings. The result must be validated against the protocol independent FDT Schemas (DTMScanIdentSchema and DTMDeviceTypeIdentSchema). The output can be used by a Frame Application to compare scan and DTM values based on string format. A DTM may define a range of supported physical device types by including regular expressions. In order to identify a matching DTMDeviceType, a Frame Application must implement a pattern matching according to the regular expression syntax defined below.

8.10.8.3 Semantic identification information

Table 14 lists identification elements, which have to be provided by Scan and DTM identification. After xsl transformation, the following values must be available in string format with the elements listed below in DTMScanIdentSchemaand DTMDeviceTypeIdentSchema.

Table 14 − Semantic identification information

Semantic element name	Description	Scan	DTM
IdAddress	Fieldbus address of the scanned physical device	x	-
IdBusProtocol	Protocol identification (enum)	x	x
IdBusProtocolVersion	Version of bus protocol		
IdManufacturer	Manufacturer identification	x	?
IdTypeID	Device type identification	x	?
IdSoftwareRevision	Tool relevant version of the physical device – Firmware Version	x	?
IdHardwareRevision	Hardware version of the physical device	x	?
IdTag	Tag name, which set in the physical device	x	-
IdSerialNumber	In order to get a common definition for all kind of protocols, a serial number is defined to be only unique for one manufacturer and device type. For world wide unique identification this attribute must always be combined with manufacturerID and deviceTypeID	x	-
IdDTMSupportLevel	Enum : genericSupport, profileSupport, blockspecificProfileSupport, specificSupport(=default)	-	x
NOTE X must be provided, - is not to be provided, ? optional			

If a semantic element cannot be defined for a fieldbus protocol, the value must be set to 'NOT_APPLICABLE'.

8.10.8.4 Device assignment

The comparison of scan result with DTM device identification information and the device assignment is done by the Frame Application based on it's internal rules.

Each element from the scan document can be compared with the DTM device identification information.

The element of the DTM device identification document must match to the value within the scan document.

8.10.8.5 Regular expression specification

If the element of the DTM device identification document contains a pattern, the Frame Application must use regular expressions to compare scan and DTM device identification information, see Table 15.

Table 15 − Regular expressions

MetaCharacter	Description
.	Matches any single character
[ ]	Indicates a character class. Matches any character inside the brackets (for example, [abc] matches "a", "b", and "c")
^	If this metacharacter occurs at the start of a character class, it negates the character class. A negated character class matches any character except those inside the brackets (for example, [^abc] matches all characters except "a", "b", and "c"). If ^ is at the beginning of the regular expression, it matches the beginning of the input (for example, ^[abc] will only match input that begins with "a", "b", or "c")
-	In a character class, indicates a range of characters (for example, [0-9] matches any of the digits "0" through "9")
?	Indicates that the preceding expression is optional: it matches once or not at all (for example, [0-9][0-9]? matches "2" and "12")
+	Indicates that the preceding expression matches one or more times (for example, [0-9]+ matches "1", "13", "666", and so on)
*	Indicates that the preceding expression matches zero or more times
??, +?, *?	Non-greedy versions of ?, +, and *. These match as little as possible, unlike the greedy versions which match as much as possible. Example: given the input "<abc><def>", <.*?> matches "<abc>" while <.*> matches "<abc><def>"
( )	Grouping operator. Example: ([0-9]+,)*[0-9]+ matches a list of numbers separated by commas (such as "1" or "1,23,456")
\	Escape character: interpret the next character literally (for example, [0-9]+ matches one or more digits, but [0-9]\+ matches a digit followed by a plus character). Also used for abbreviations (such as \a for any alphanumeric character; see table below). If \ is followed by a number n, it matches the nth match group (starting from 0). Example: <{.*?}>.*?</\0> matches "<head>Contents</head>". Note that in C++ string literals, two backslashes must be used: "\\+", "\\a", "<{.*?}>.*?</\\0>"
$	At the end of a regular expression, this character matches the end of the input. Example: [0-9]$ matches a digit at the end of the input
|	Alternation operator: separates two expressions, one of which matches exactly (for example, T|the matches "The" or "the")
!	Negation operator: the expression following ! does not match the input. Example: a!b matches "a" not followed by "b"

9 FDT session model and use cases

Figure 25 shows the UML syntax that is used throughout this clause.

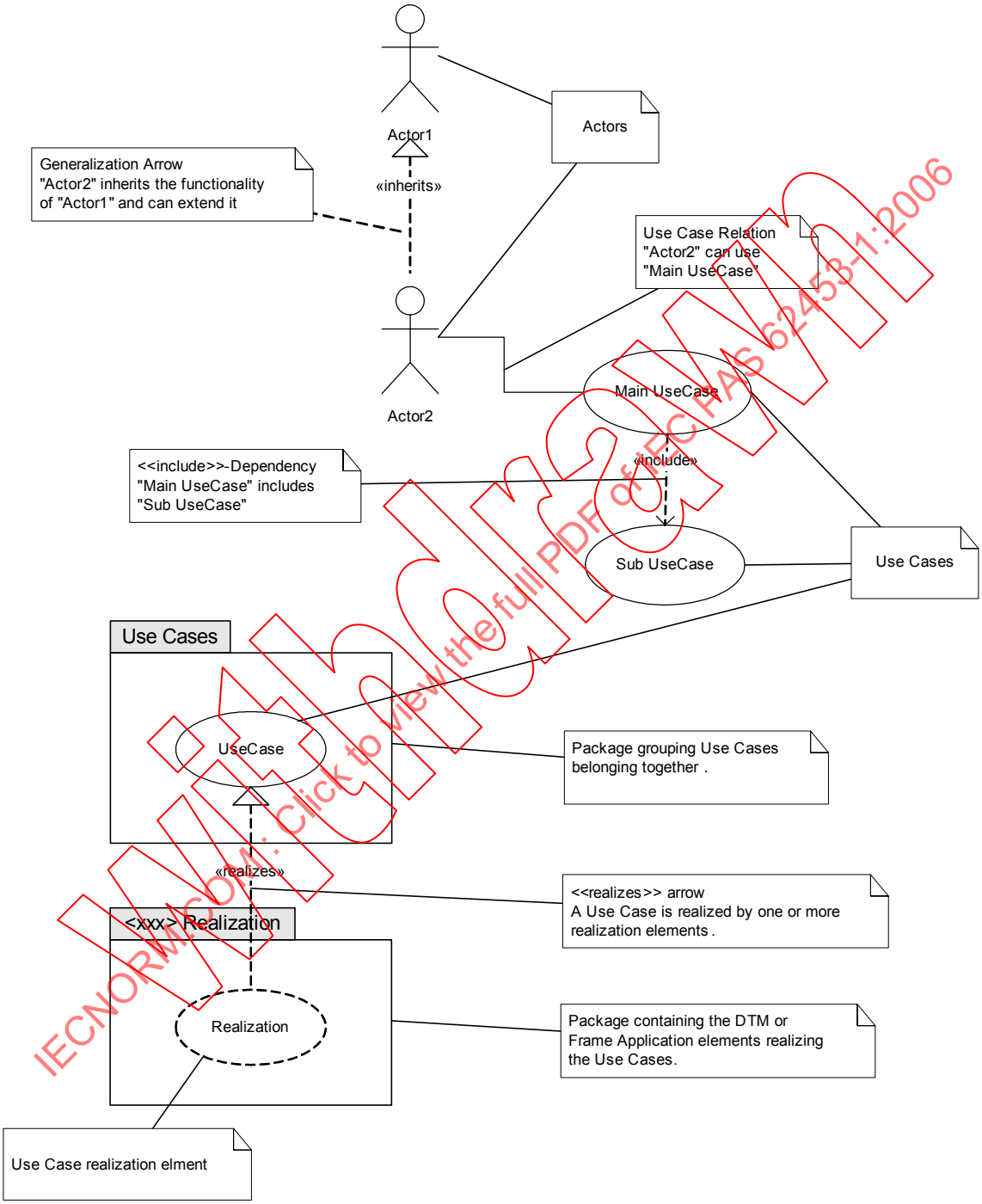

Figure 25 – UML syntax

Figure 26 shows the main use cases.

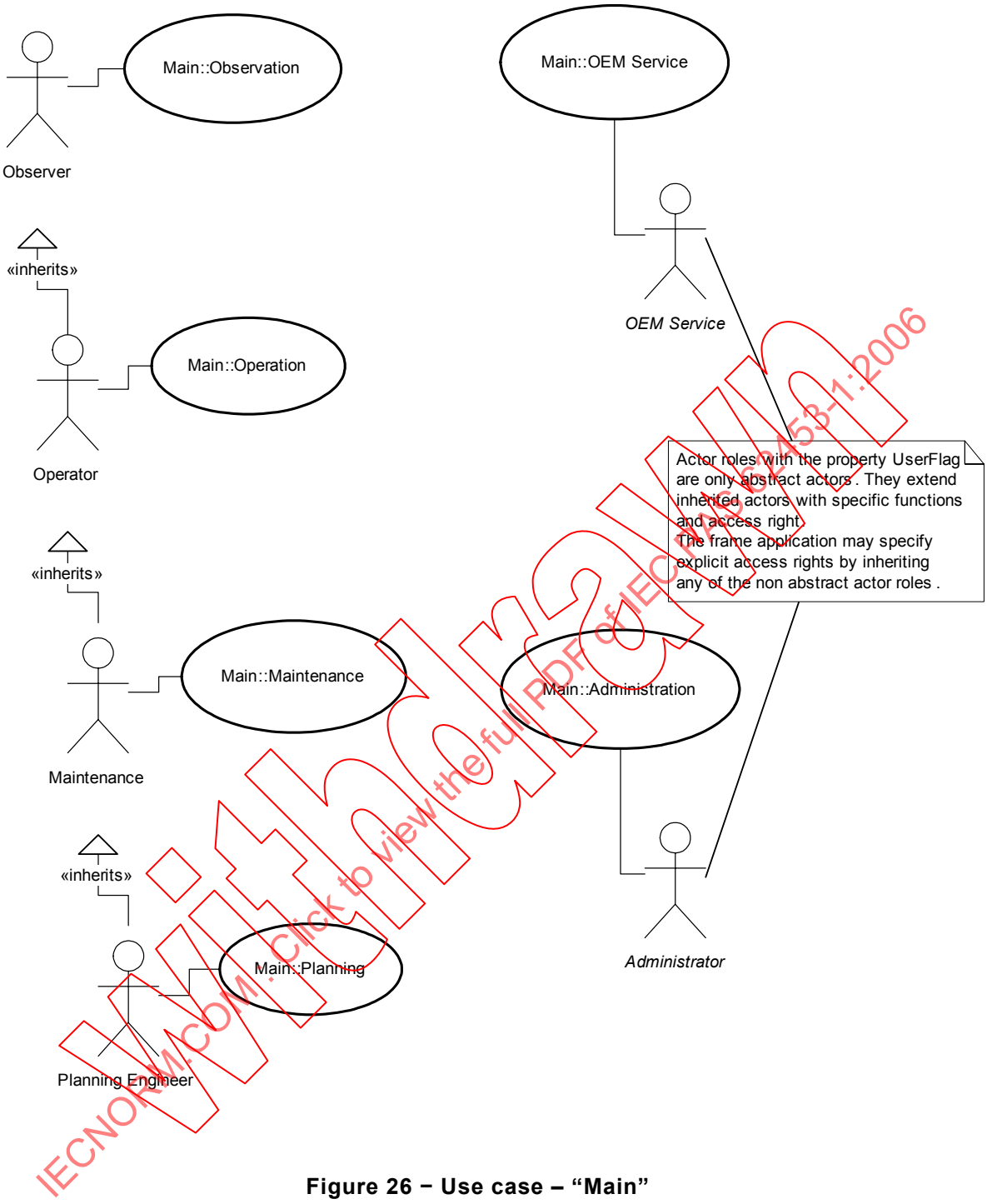

Figure 26 – Use case – "Main"

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

9.1 Actors

The actor types in the above displayed use case diagram are structured in a hierarchical way.

The "Maintenance" actor inherits the "Operator" actor. The "Planning Engineer" actors inherits the use cases of the "Maintenance" actor. Both the "Administrator" and the "OEM Service" can extend the inherited actors by additional use cases. Use cases, which are passed on to an actor of higher level, may act in an extended way to match their intention.

Table 16 will give a brief description of the actors' roles:

Table 16 – 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	His 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*
Note * 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*
NOTE * 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
NOTE * “User Flags” may be combined with any “User Level”, to specify the inherited actor role of an “Administrator” or “OEM Service” actor.	
Actor Name	**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
NOTE * “User Flags” may be combined with any “User Level”, to specify the inherited actor role of an “Administrator” or “OEM Service” actor.	

9.2 Use cases

The following subclause 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.

9.2.1 Observation

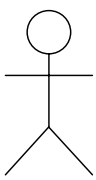

Figure 27 – Actor “Observer”

The “Observer” stands for a person that has the lowest access level. No use case is associated with this person.

9.2.2 Operation

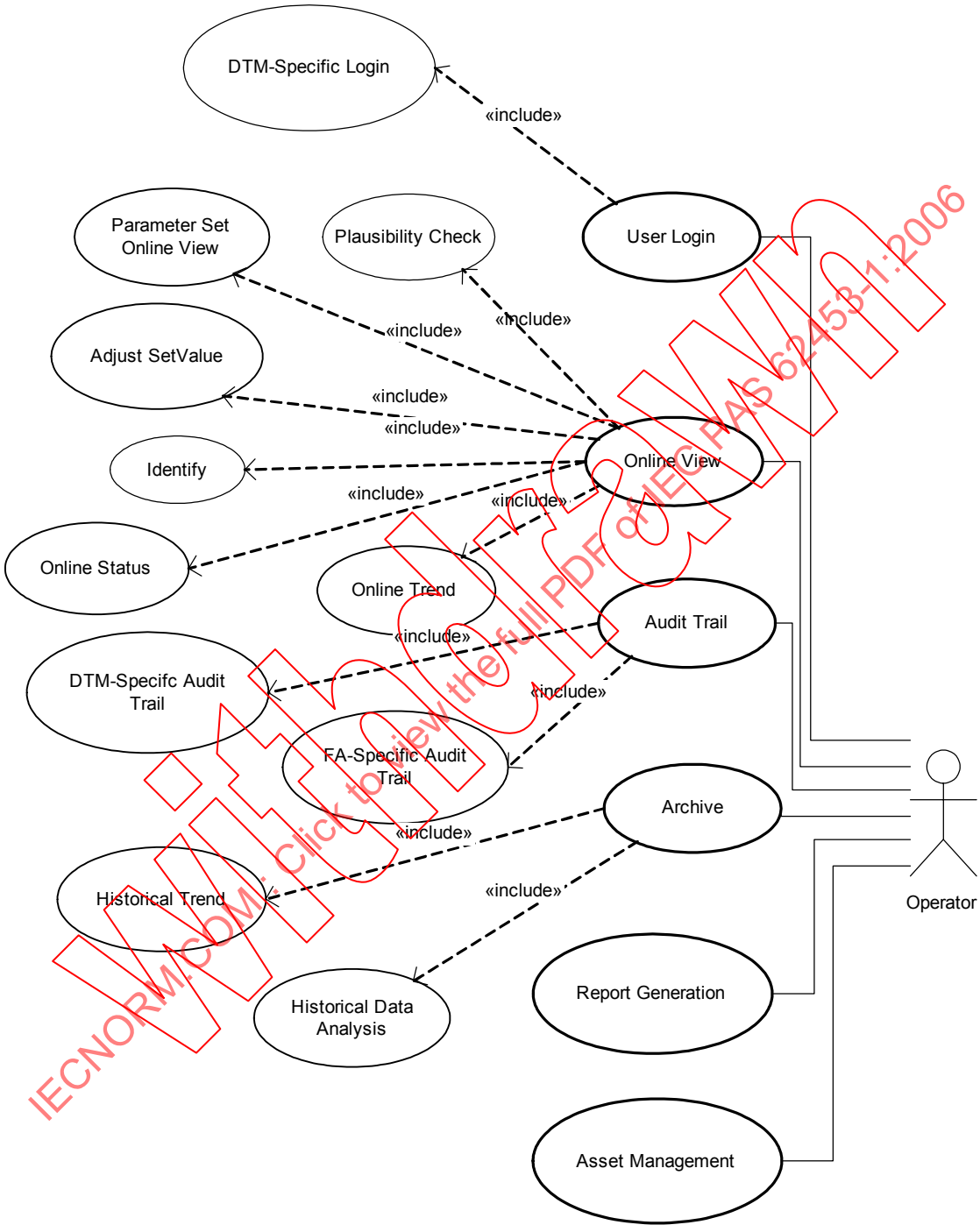

Figure 28 − Use case – “Operation”

Table 17 provides a description of the main use cases for the actor Operator.

Table 17 - Description of use case “User Login”

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
User level	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
UserLevel	OEM Service	Accessible only for OEM Service

Figure 29 shows how the use case is realized by DTM and Frame Application.

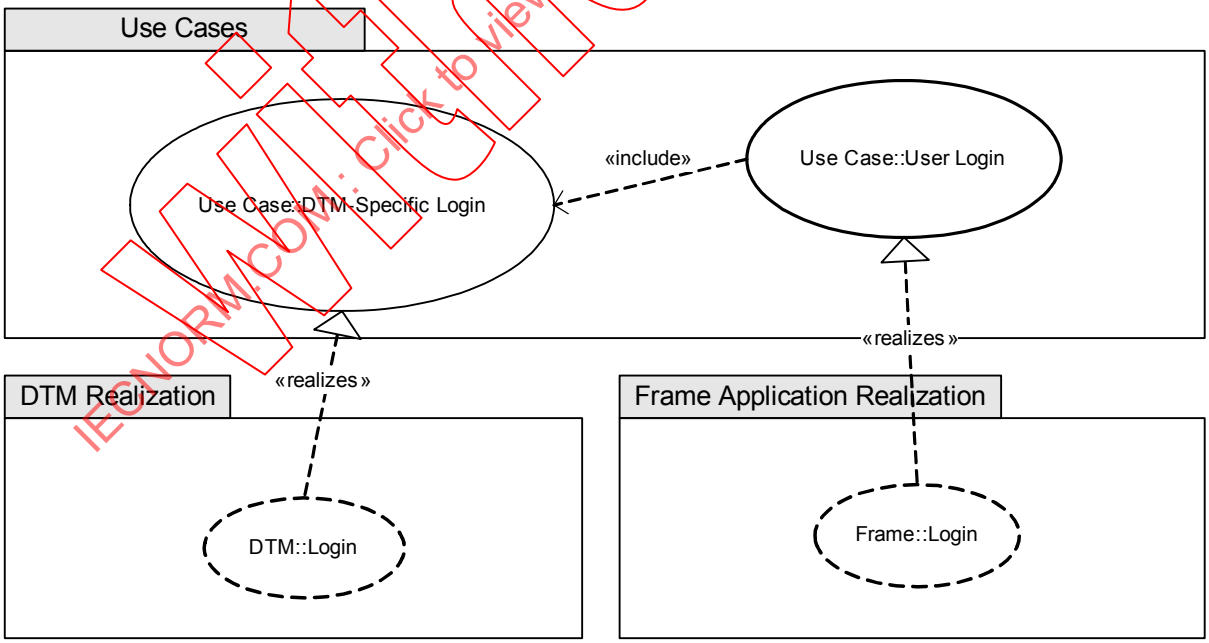

Figure 29 – Realization of use case “User Login”

In Table 18, the “Online View” use case is described.

Table 18 - Description of use case “Online View”

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.	
UserFlag		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 SetValues of a device (e.g. positioner, controllers)
Included use case		**Online Status**
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
Included use case		**Plausibility check**
Brief description		Every time the device parameters or the configuration data is changed a Plausibility check is automatically performed. As long as the Plausibility check fails, the data cannot be written to the device
Included use case		**Identify**
Brief description		Displays identification of device instance information e.g. address, TAG, Fieldbus-Rev., DD-Rev., Firmware. This information is protocol dependent. It also may contain device type specific information. Further information may be provided: references to documentation, area to add comments to the device instance. Refer to Device Identification clause in protocol specific FDT-JIG-Annex specifications

Figure 30 shows how the use case “Online View” is realized by the DTM.

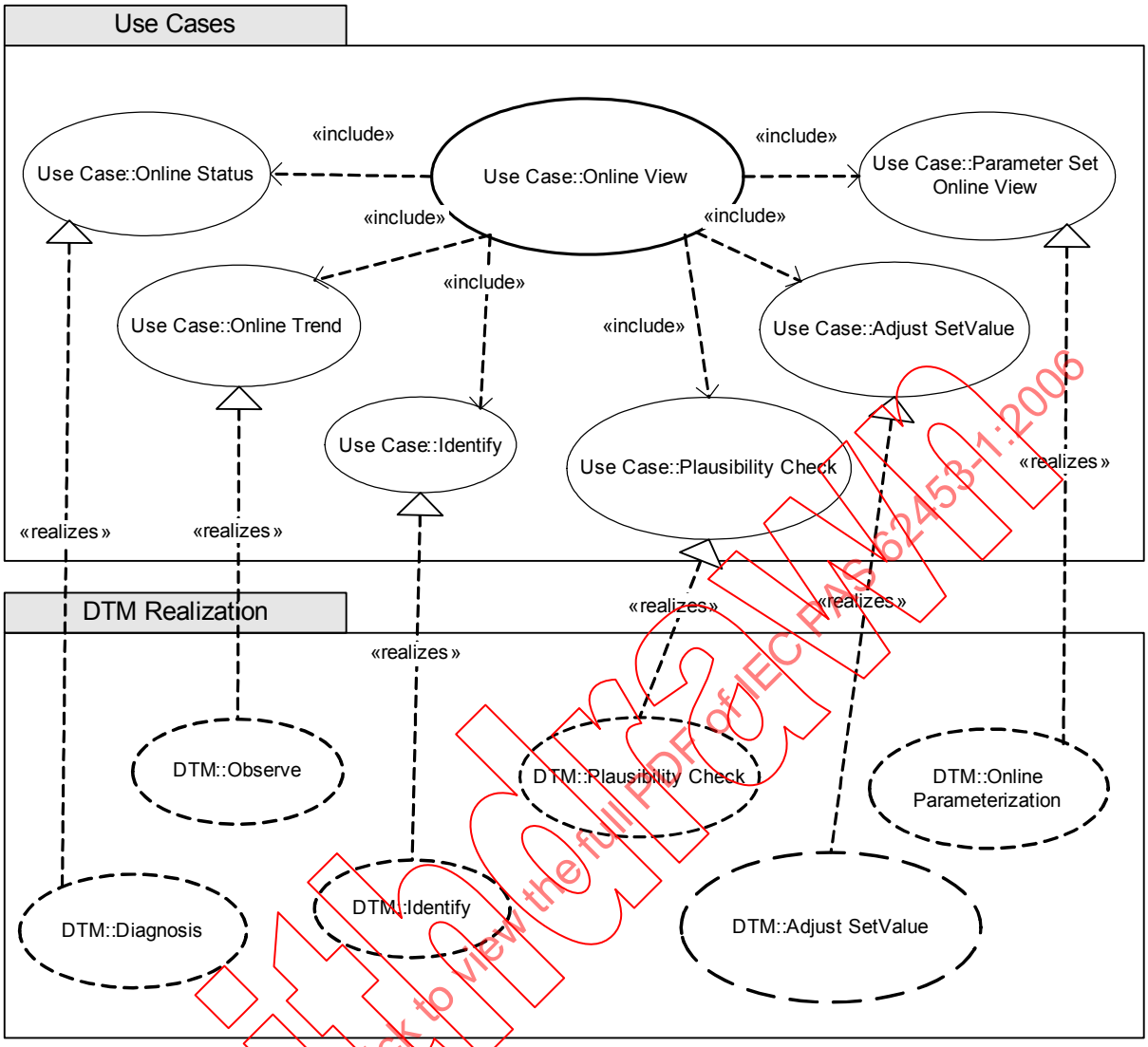

Figure 30 – Realization of use case "Online View"

Table 19 describes the use case "Audit Trail".

Table 19 - Description of use case "Audit Trail"

Use case		Audit Trail
Brief description		Use case to enable the actor viewing and deleting audit trail data
GUI		The component, which supports audit-trail, has to provide an adequate GUI
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 31 shows how the use case “Audit Trail” is realized by DTM and Frame Application.

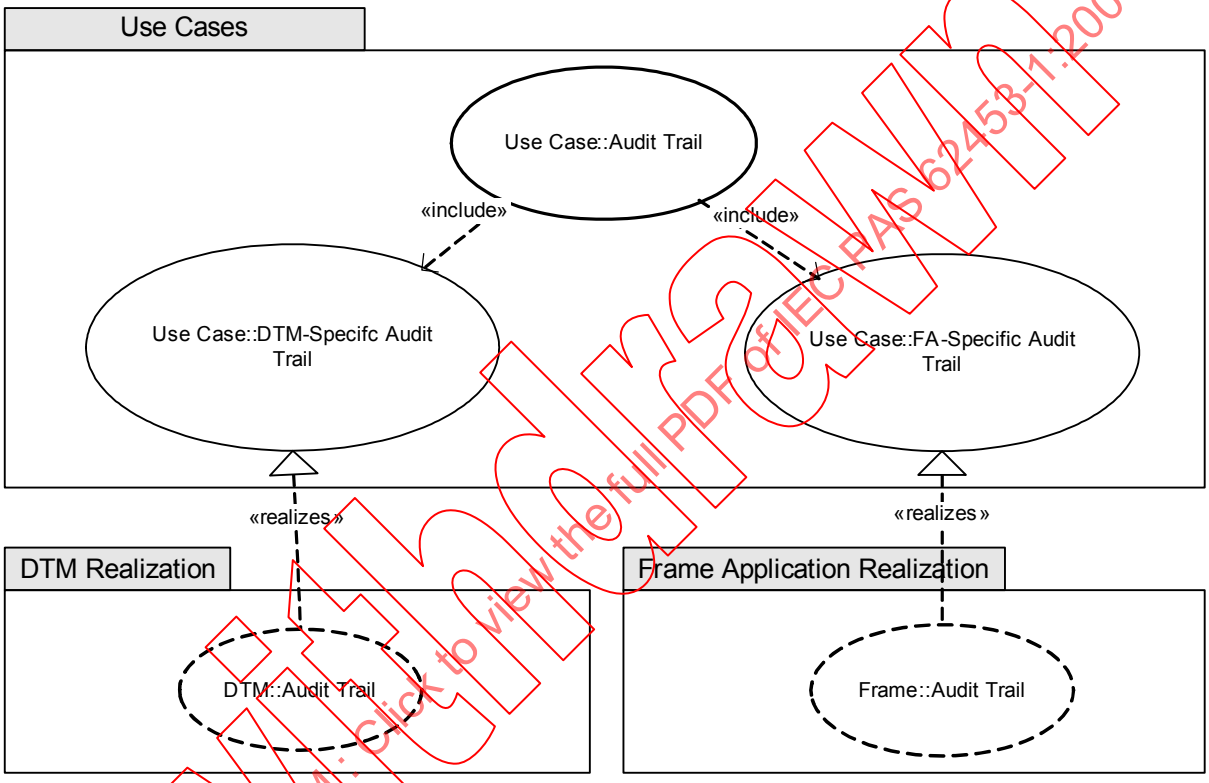

Figure 31 – Realization of use case “Audit Trail”

Table 20 provides a description of the use case “Archive”.

Table 20 - Description of use case “Archive”

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	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 32 shows how the use case “Archive” is realized by DTM and Frame Application.

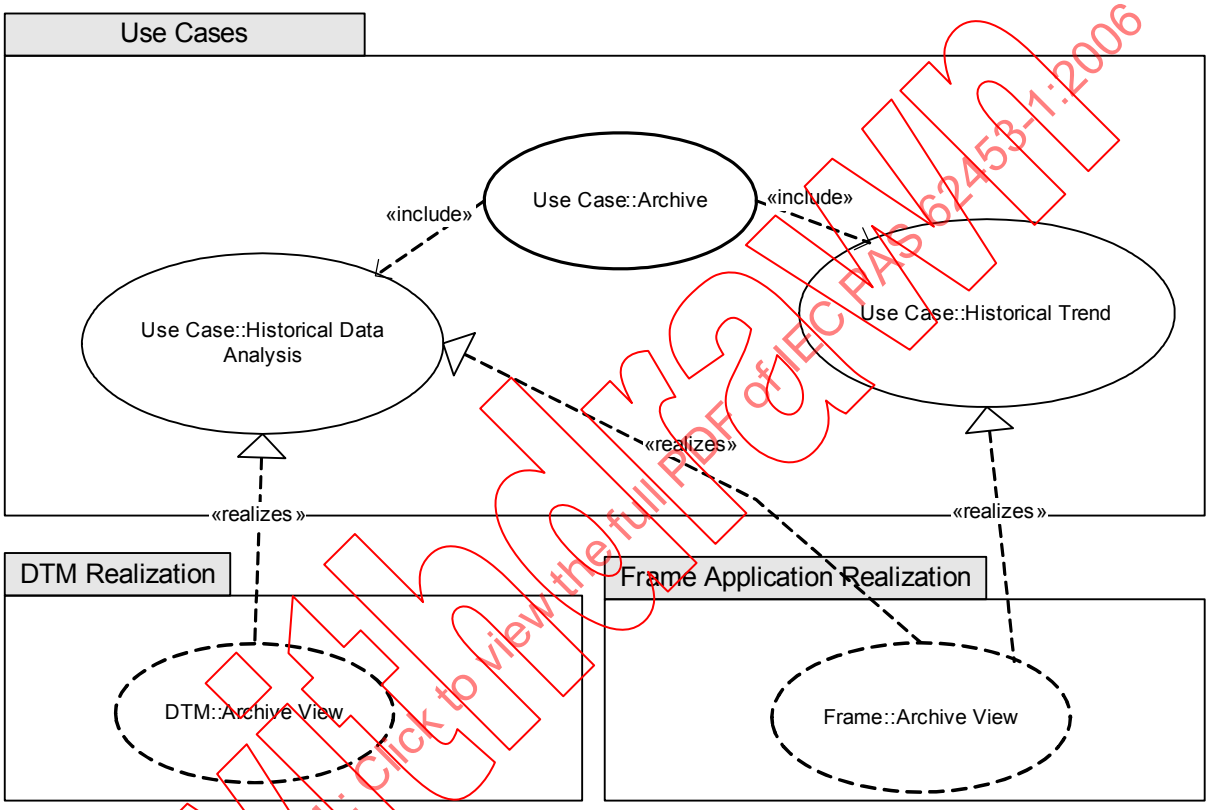

Figure 32 – Realization of use case “Archive”

Table 21 provides a description of the use case “Report Generation”.

Table 21 – Description of use case “Report Generation”

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, UserFlags	The printed information may depend on user level and user flags

Figure 33 show how the use case “Report Generation” is realized by DTM and Frame Application.

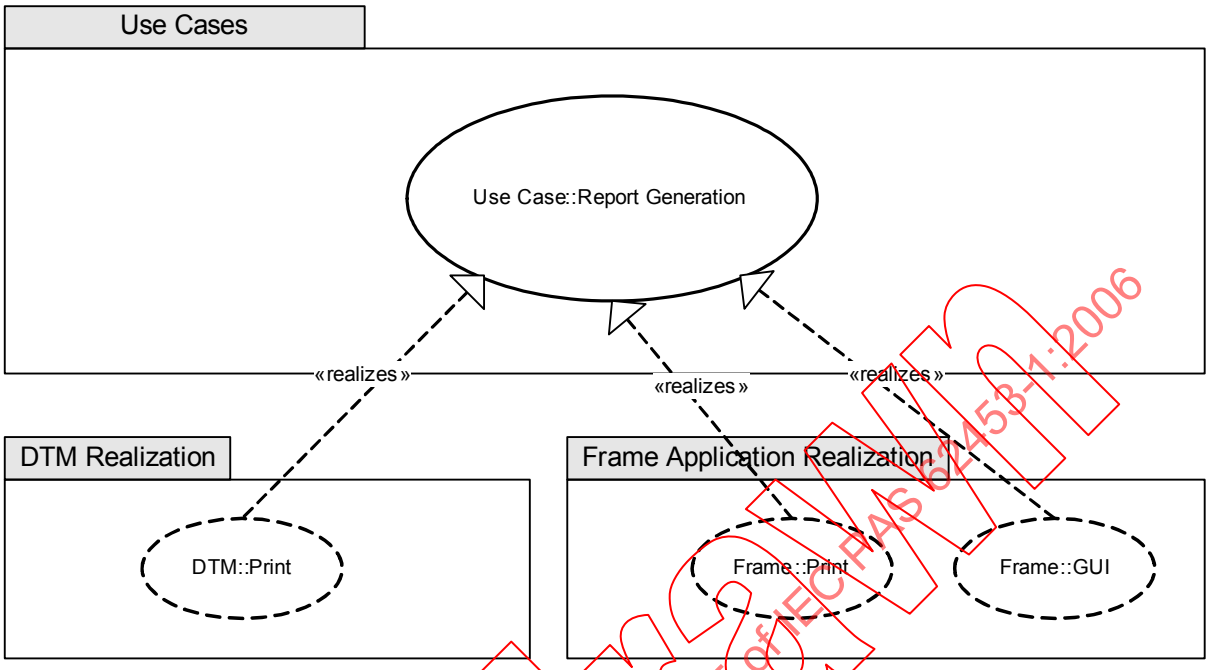

Figure 33 − Realization of use case “Report Generation ”

Table 22 provides a description of the use case “Asset Management”.

Table 22 – Description of use case “Asset Management”

Use case		Asset Management
Brief description		Frame Application provides mechanism for asset management
GUI		The component, which supports asset management functions, has to provide an adequate GUI.
Print		Frame Application 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 34 shows how the use case “Asset Management” is realized by DTM and Frame Application.

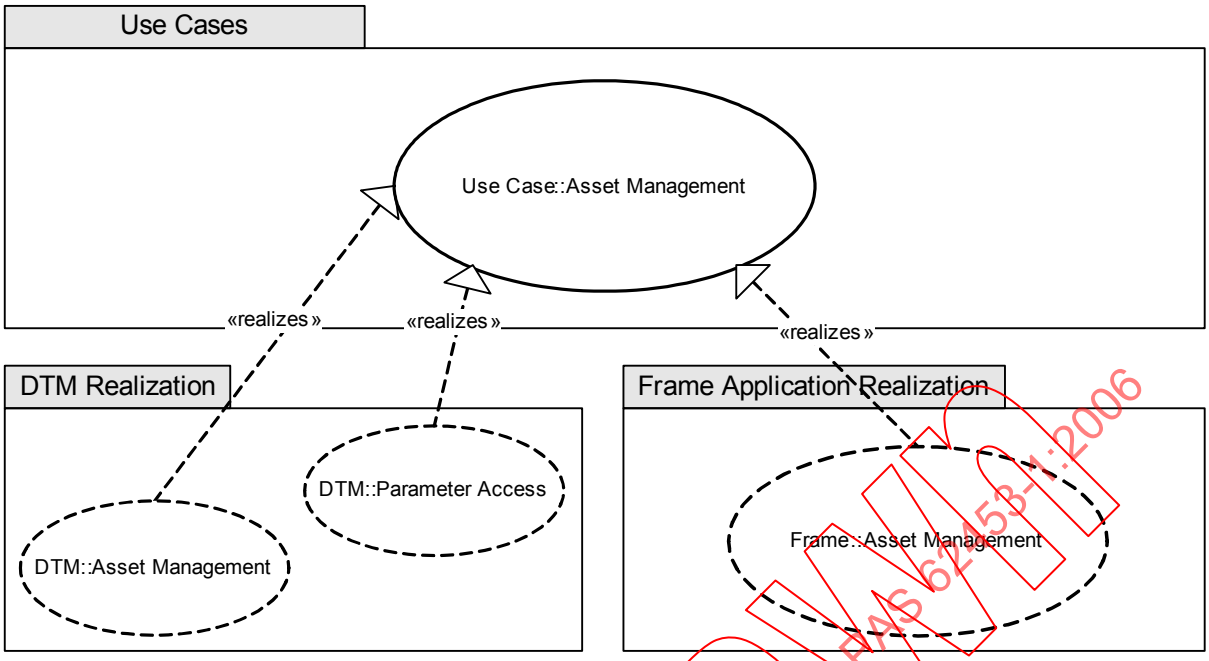

Figure 34 – Realization of use case "Asset Management"

9.2.3 Maintenance

Additionally to the use cases inherited from the actor “Operator”, the actor “Maintenance” may execute the use cases shown in Figure 35.

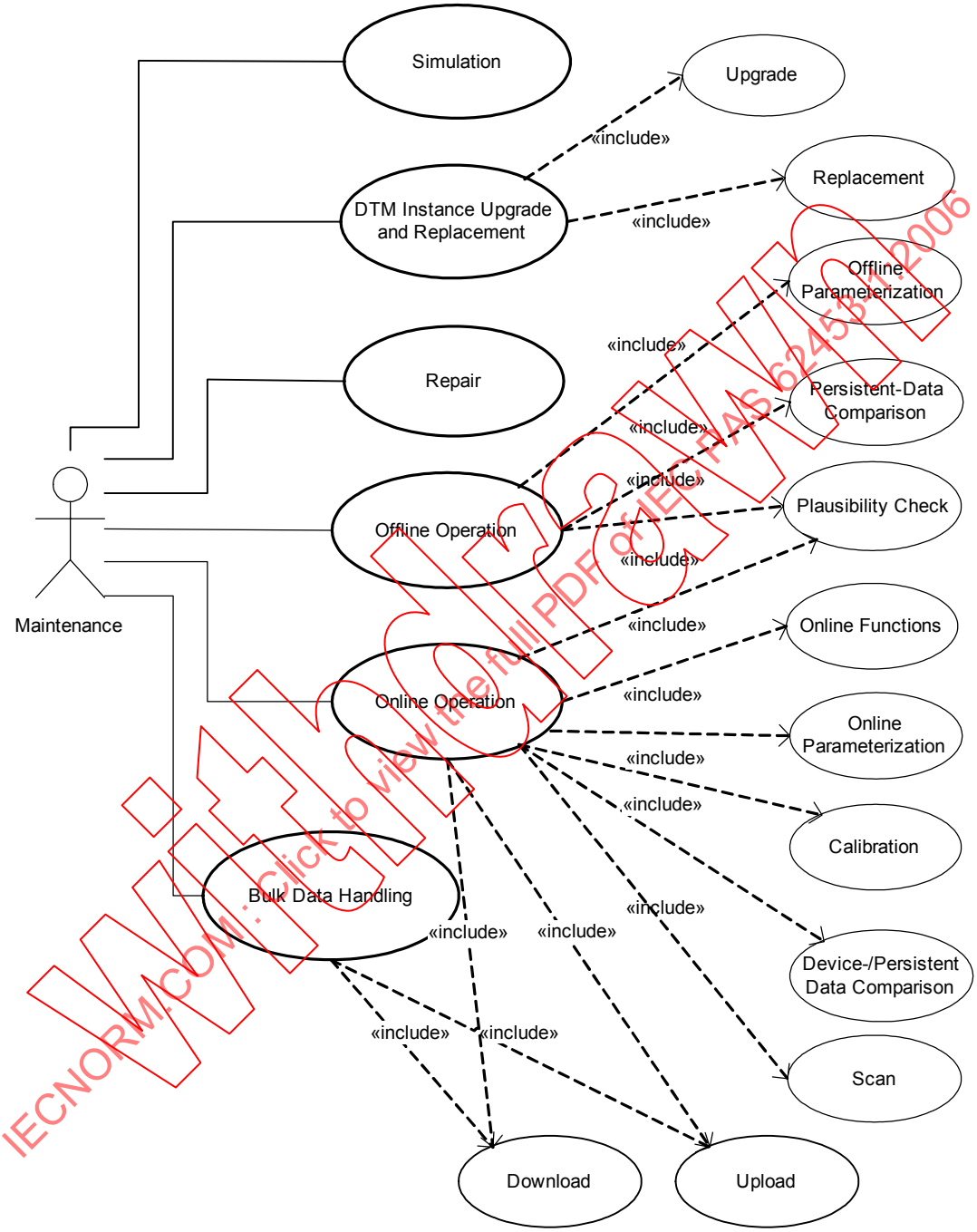

Figure 35 − Use case – “Maintenance”

All use cases of “Maintance” are described in the following, starting with use case “Simulation” in Table 23.

Table 23 – Description of use case “Simulation”

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

How the use case is realized by the DTM is shown in Figure 36.

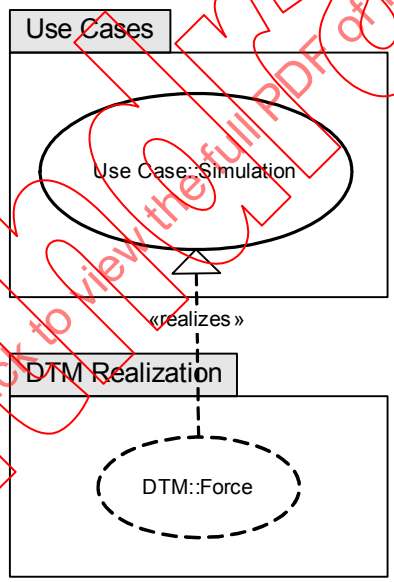

Figure 36 − Realization of use case “Simulation”

Table 24 provides a description of the use case “Offline Operation” and of included use cases.

Table 24 – Description of use case “Offline Operation”

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 • After display of a warning, inconsistent data may be stored • For safety reasons after display of a warning writing of data is prohibited
Users		The Plausibility check may be more tolerant for actors of higher level
Included use case		**Offline Parameterization**
Brief description		The user may change parameter values. The changes will only affect data in the Frame Application database after stored as persistent data. Data should be signed as transient data until they are stored
GUI		DTM
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		No connection necessary

Figure 37 shows how a DTM realizes the use case “Offline Operation”.

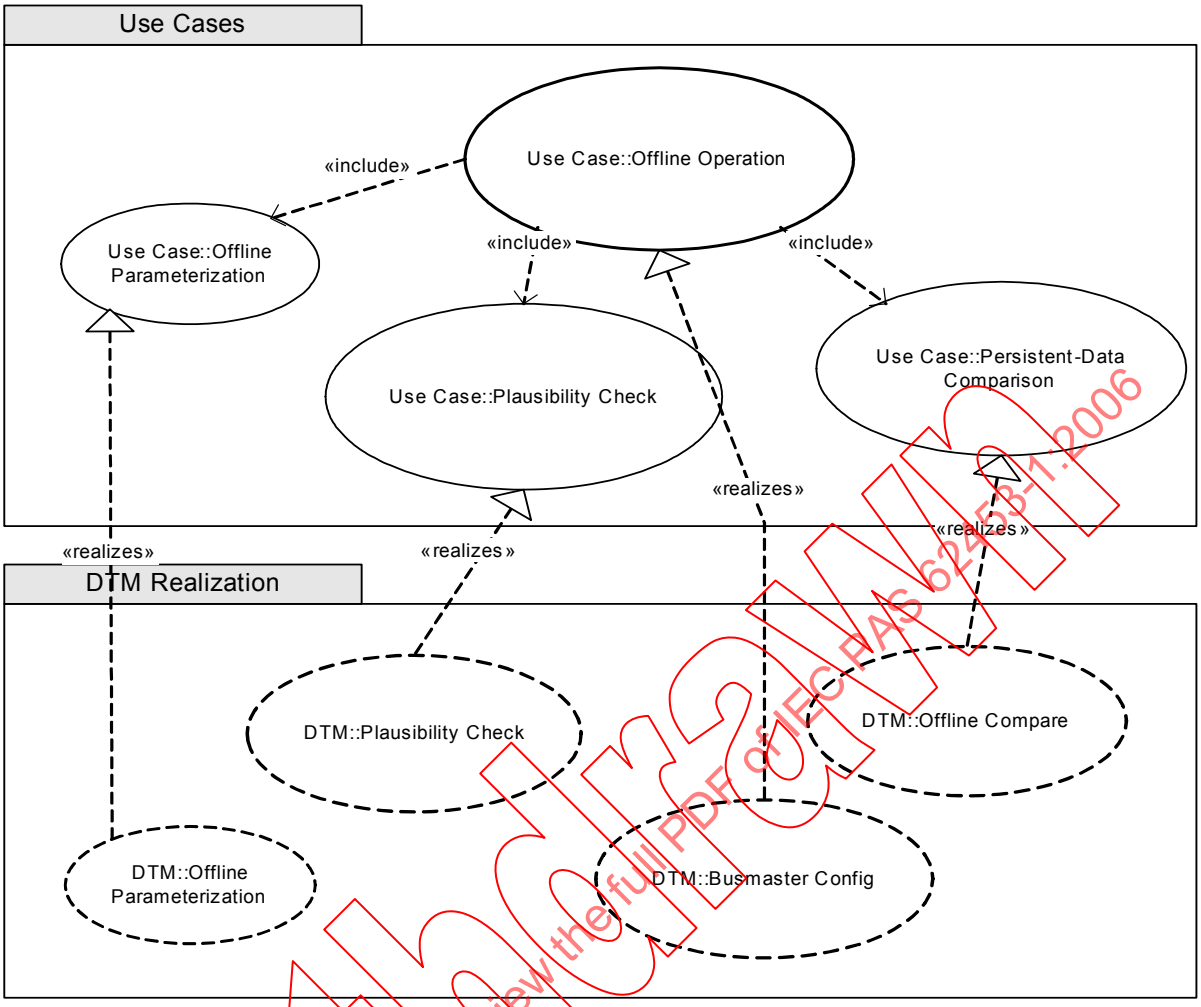

Figure 37 – Realization of use case "Offline Operation"

Table 25 provides a description of the use case "Repair".

Table 25 – Description of use case "Repair"

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
GUI		
Print		
Device connection		
UserLevel	Observer	No access
	Operator	
	Maintenance	Accessible
	Planning Eng.	
UserFlags	Administrator	No changes
	OEM Service	Accessible with extended functionality (see below)

Figure 38 show how a Frame Application realizes the use case “Repair”.

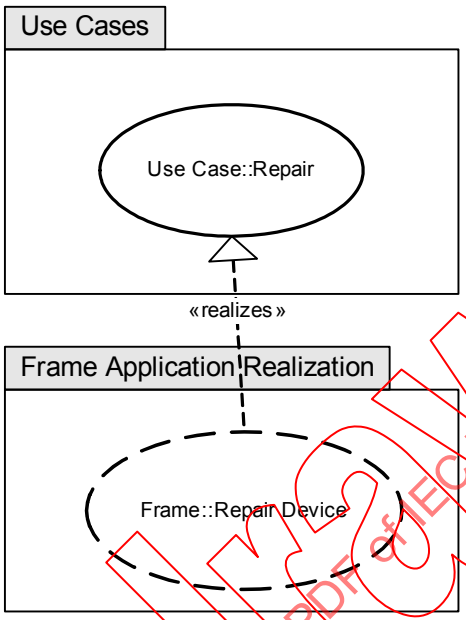

Figure 38 − Realization of use case “Repair”

Table 26 describes the use case “DTM Instance Upgrade and Replacement”.

Table 26 – Description of use case “DTM Instance Upgrade and Replacement”

Use case		**DTM Instance Upgrade and Replacement**
Brief description		This use case stands for operations which must be performed to upgrade or to replace a DTM. Upgrade or replacement are used when the ProgId of the new DTM is different to the ProgId of the previous DTM. The data format of the new DTM can be either compatible (Upgrade) or incompatible (Replacement) to the data format of the previous DTM.
UserLevel	Observer	No access
	Operator	
	Maintenance	Accessible
	Planning Eng.	
UserFlags	Administrator	No changes
	OEM Service	No changes
Included use case		**Replacement**
Brief description		This use case stands for operations which must be performed to replace a DTM in the case of incompatible data set format.
Included use case		**Upgrade**
Brief description		This use case stands for operations which must be performed to upgrade a DTM in the case of compatible data set format. Refer to 3.6 and 6.25

Figure 39 shows how the use case is realized by DTM and Frame Application.

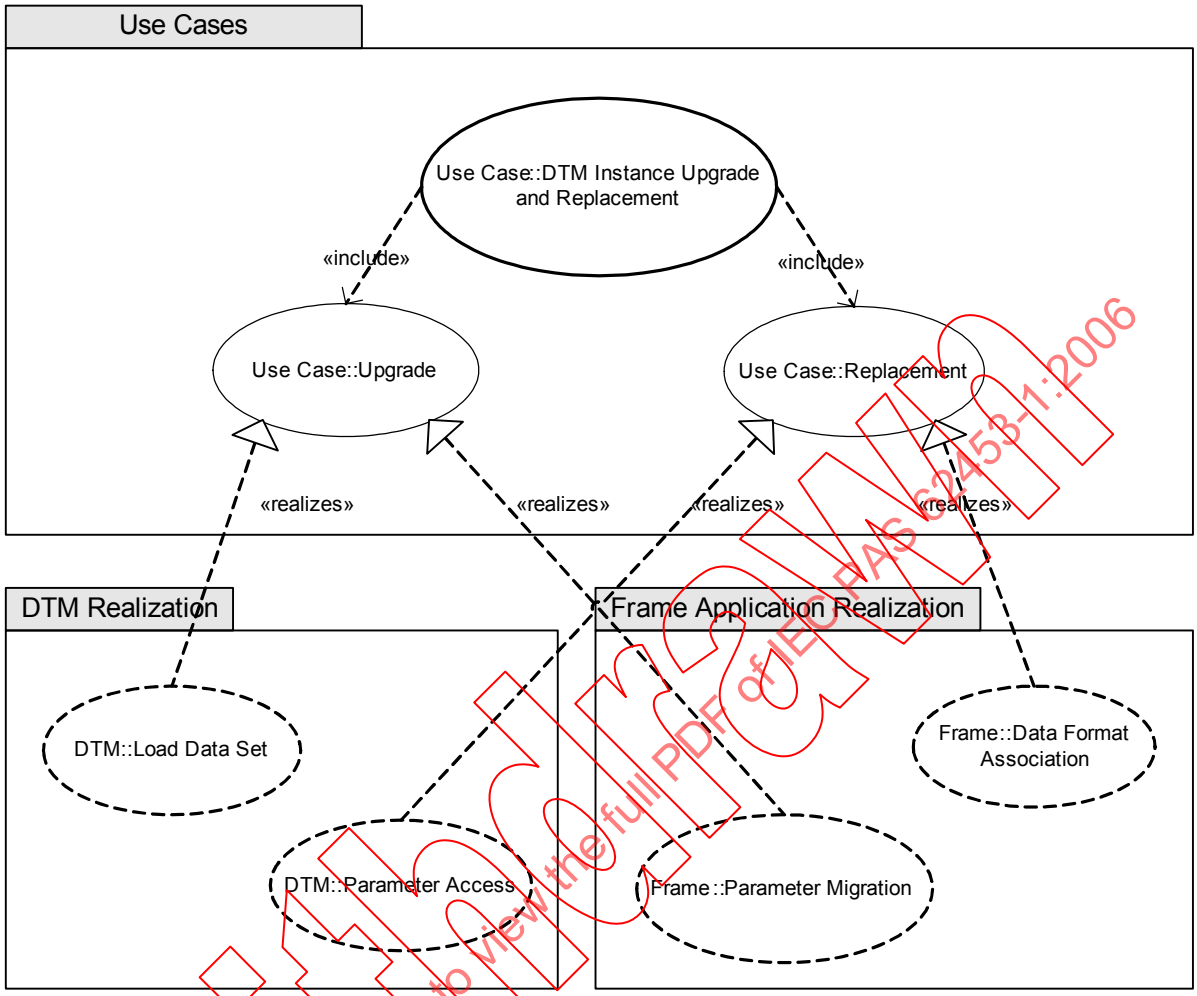

Figure 39 – Realization of use case "DTM Upgrade and Replacement"

Table 27 provides a description of the use case "Online Operation" and all included use cases.

Table 27 – Description of use case "Online Operation"

Use case		Online Operation
Brief description		This use case includes all operations which are performed directly with the device
GUI		DTM-specific
Print		DTM-specific
Device connection		Connected or disconnected
UserLevel	Observer	No access
	Operator	
	Maintenance	Accessible
	Planning Eng.	Fully accessible excluding OEM-specific parameter
UserFlags	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	**Online parameterization**
Brief description	When this use case starts all parameters are read from the device and exposed to the user within a GUI. Within the GUI the user may change parameters, depending on the user level. The device is updated immediately if the changed parameterization is in a verified state
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	**Calibration**
Brief description	This use case invokes calibration procedures to adjust the input for e.g. pressure transmitters or the current for the output
Included use case	**Plausibility check**
Brief description	Every time the device parameters or the configuration data is changed Plausibility check is automatically performed. As long as the Plausibility check fails, the data cannot be written to the device
UserLevel, UserFlag	The Plausibility check may be more tolerant for actors of higher level
Included use case	**Upload**
Brief description	Reads the whole parameter set from the device into the 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 upload process
Print	No support
Device connection	Needs a temporary connection
Included use case	**Scan**
Brief description	Retrieves the list of available devices via the IFdtSubtopology interface from a channel object. The channel object may be provided by a DTM or by Frame Application
GUI	The Frame Application may offer a GUI for representation of the list of devices
Print	No support
Device connection	Needs a temporary connection (The channel must have online access.)

Figure 40 shows how the use case is realized by DTM and Frame Application.

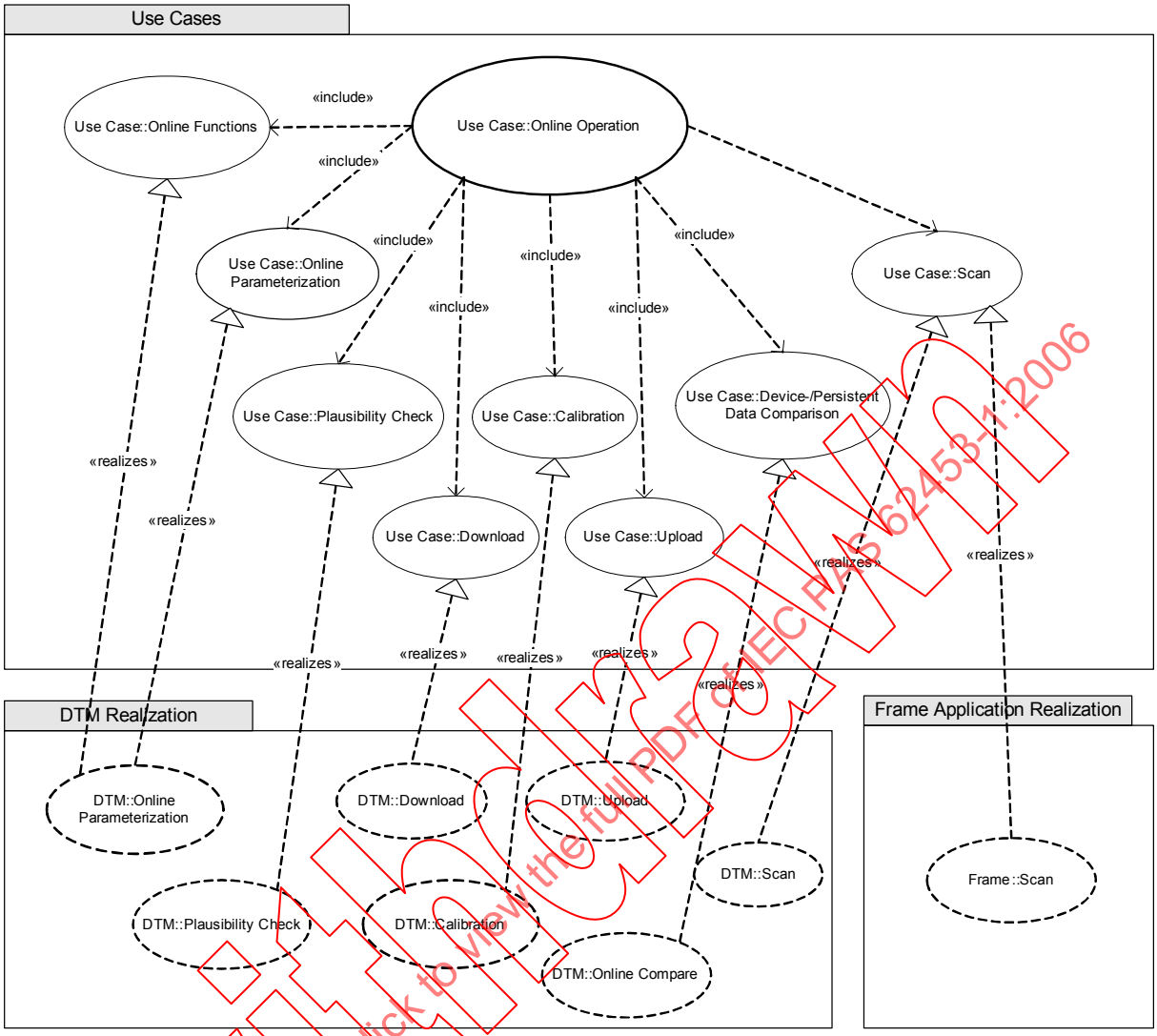

Figure 40 – Realization of use case “Online Operation”

Table 28 provides a description for the use case “Bulk Data Handling”.

Table 28 – Description of use case “Bulk Data Handling”

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. This use case handles bulk data on system level
GUI		Frame Application
Print		Not supported
Device connection		Needs a connection
UserLevel	Observer	No access
	Operator	
	Maintenance	
	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 41 shows how DTM and Frame Application provide realization for the use case “Bulk Data Handling”.

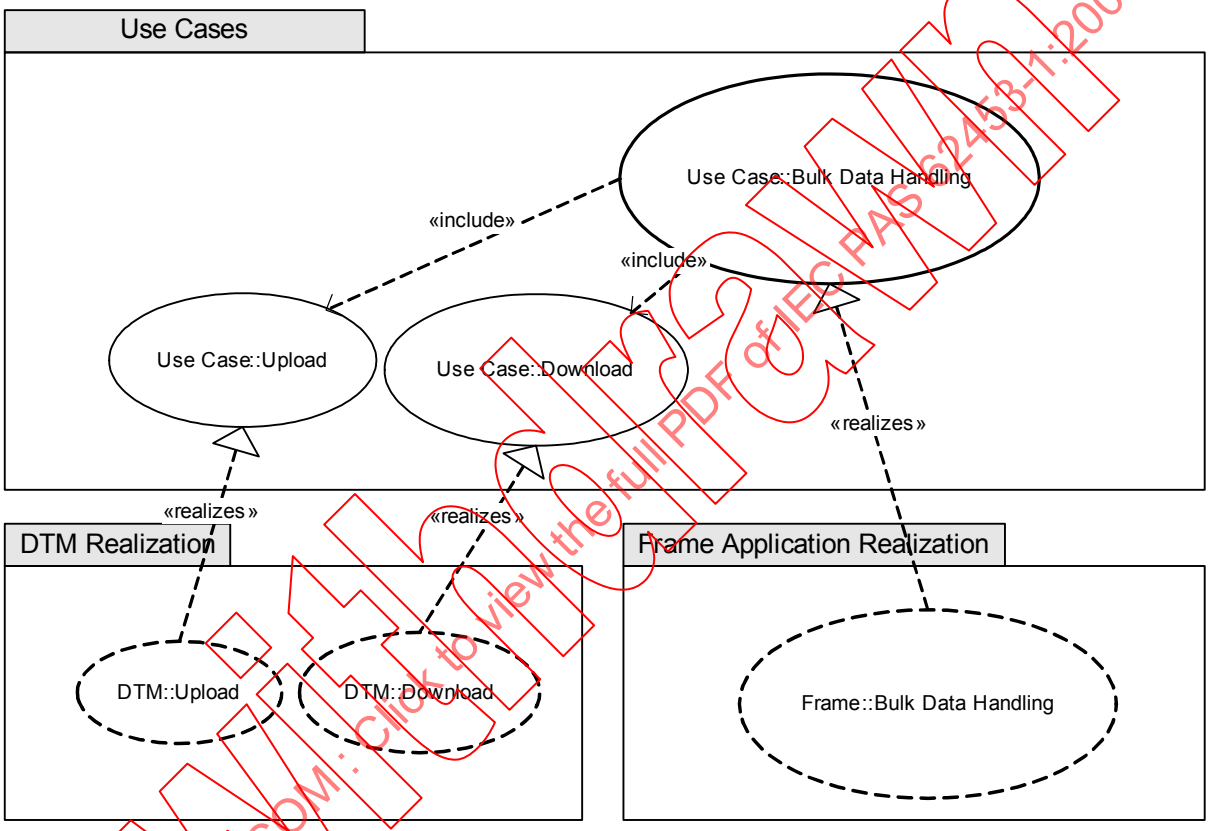

Figure 41 – Realization of use case “Bulk Data Handling”

9.2.4 Planning

Additionally to the use cases inherited from actor “Maintenance” the actor “Planning Engineer” has access to the “Planning” use cases as shown in Figure 42.

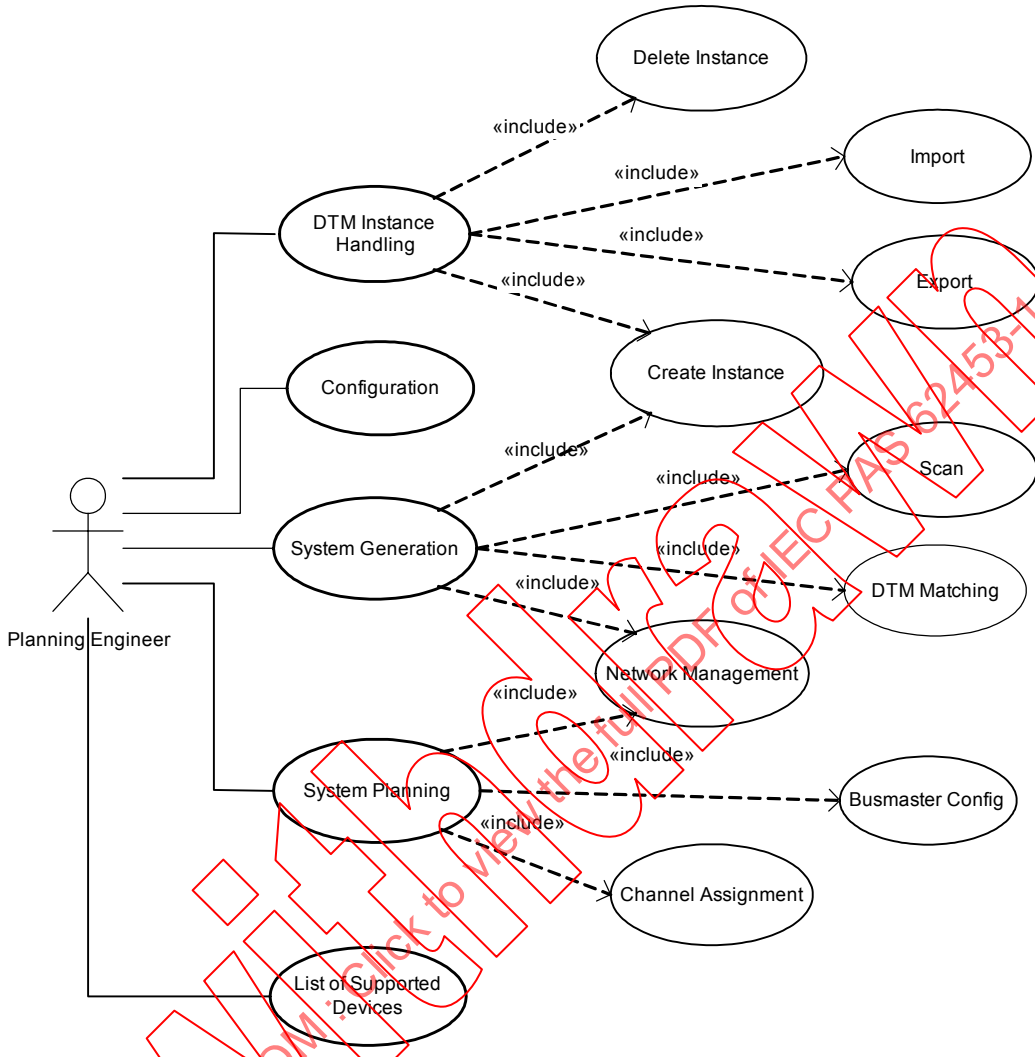

Figure 42 – Use case – “Planning”

Table 29 provides a description of the use case “DTM Instance Handling” and all related use cases.

Table 29 – Description of use case “DTM Instance Handling”

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	
	Operator	No access
	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 43 shows how the use case “DTM Instance Handling” is realized by DTM and Frame Application.

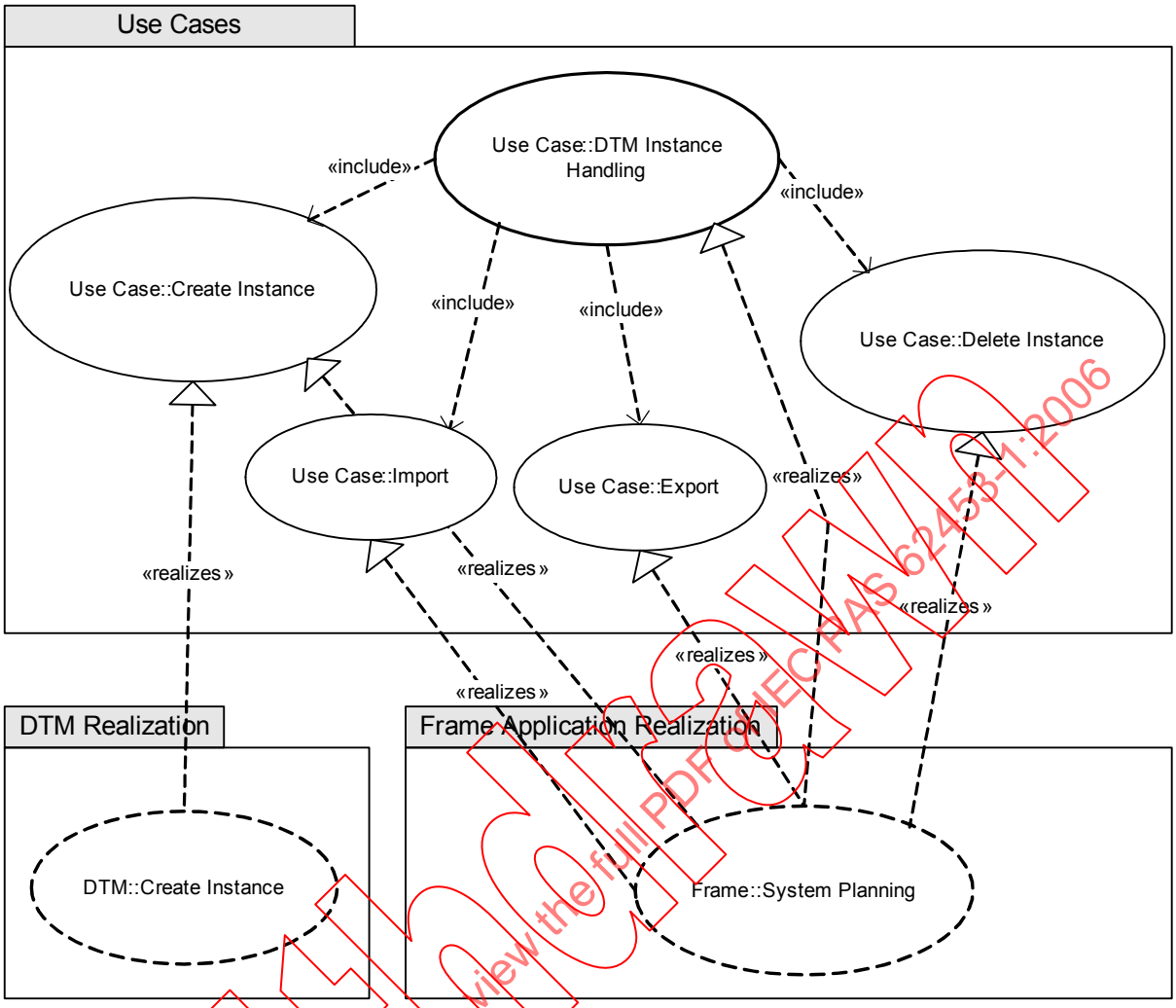

Figure 43 – Realization of use case "DTM Instance Handling"

Table 30 provides a description for the use case "Configuration".

Table 30 – Description of use case "Configuration"

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 44 shows how the use case is realized by DTM and Frame Application.

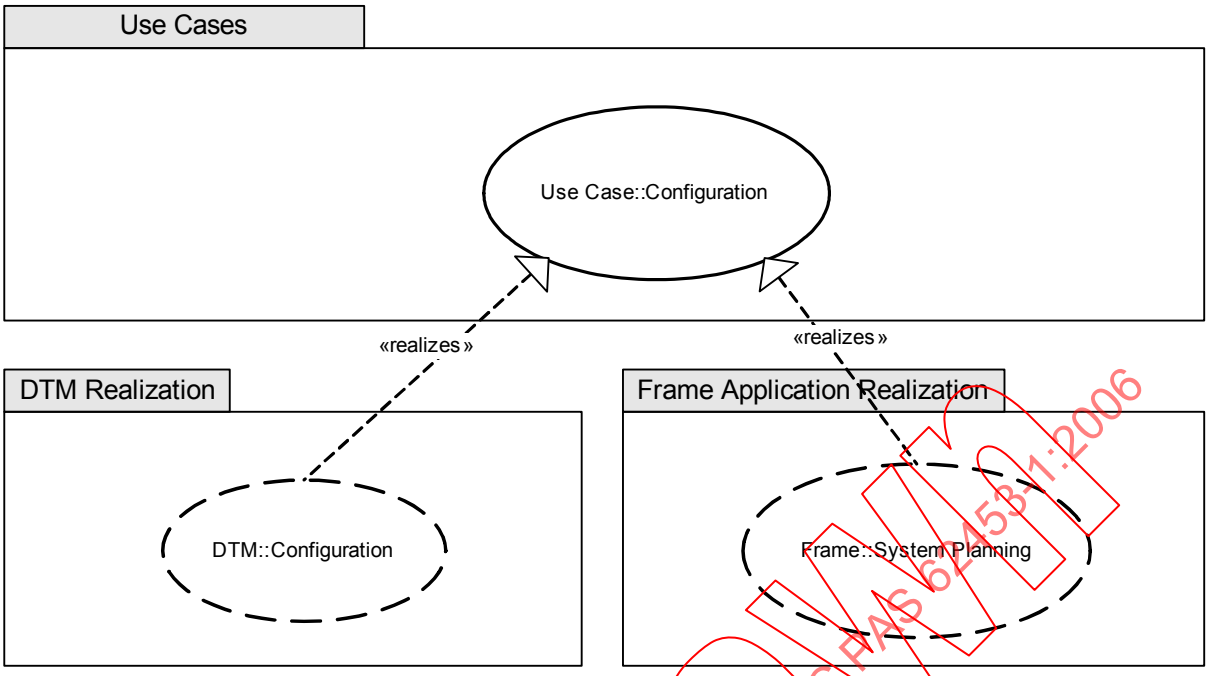

Figure 44 – Realization of use case "Configuration"

Table 31 provides a description for the use case "System Generation" and related use cases.

Table 31 – Description of use case "System Generation"

Use case		**System Generation**
Brief description		Use case to perform all necessary actions for the creatuib of topology based on the result of scan
GUI		Frame Application and DTM
Print		Frame Application and DTM
Device connection		Necessary
UserLevel	Observer	No access
	Operator	
	Maintenance	
	Planning Eng.	Accessible
UserFlags		No changes
Included use case		**Scan**
Brief description		Retrieves the list of available devices via the IFdtSubtopology interface from a channel object. The channel object may be provided by a DTM or by Frame Application
GUI		The Frame Application may offer a GUI for representation of the list of devices
Print		No support
Device connection		Needs a temporary connection (The channel must have online access.)
Included use case		**Network Management**
Brief description		Use case for network management purposes e.g. address setting as a function of a Communication-DTM
Included use case		**DTM matching**
Brief description		Use case for finding a DTM that matches best the information retrieved in the Scan use case

Included use case	Create Instance
Brief description	In this use case, a new device instance can be created and placed into the project After creation of a new device instance, the device parameter set must be instantiated. This action is performed by the DTM
GUI	Frame Application and DTM (if it supports device default selection)

Figure 45 shows how DTMs and Frame Application realize the use case “System Generation”.

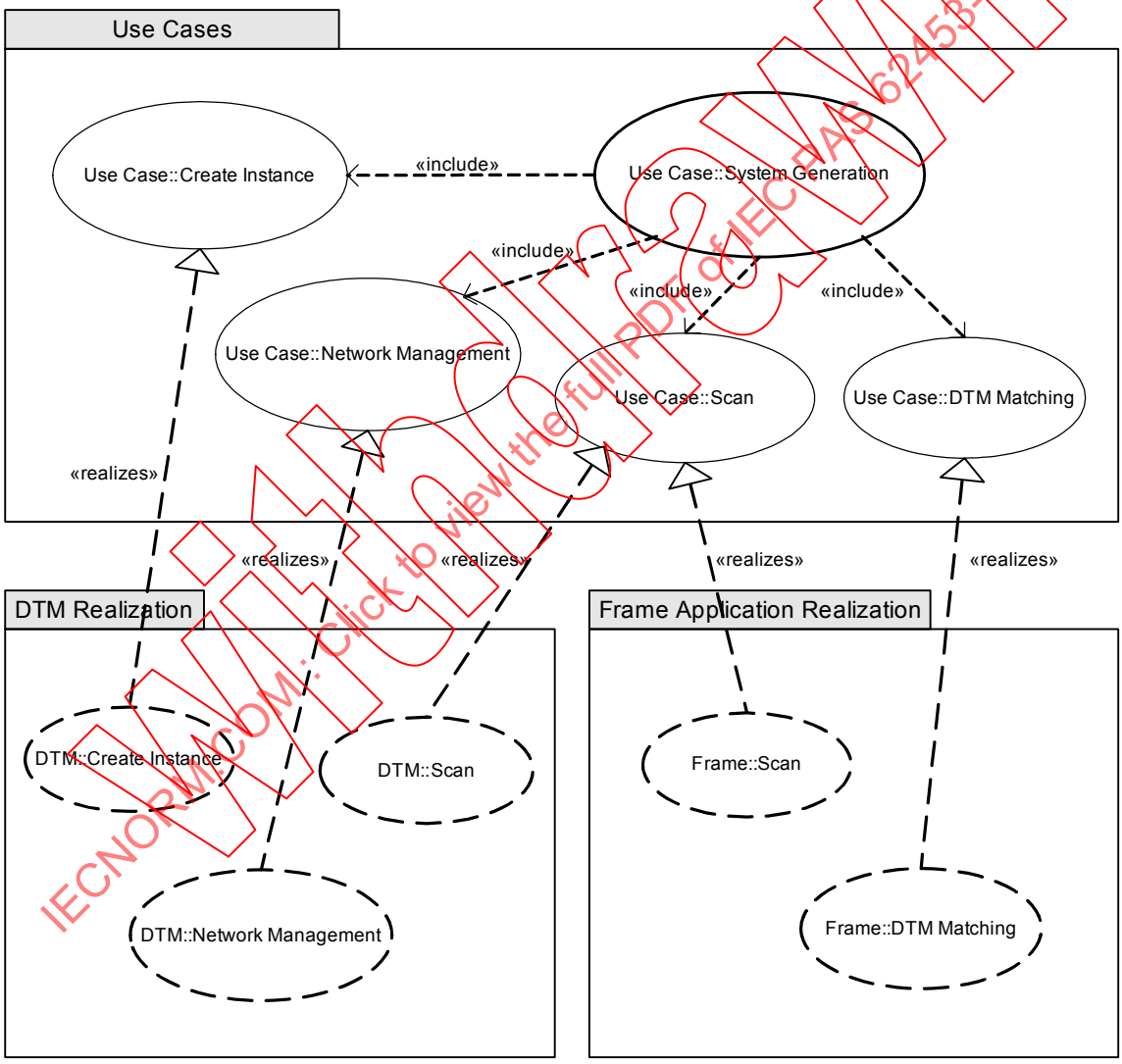

Figure 45 − Realization of use case “System Generation”

Table 32 provides description for the use case “System Planning”.

Table 32 – Description of use case “System Planning”

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		**Network Management**
Brief description		Use case for network management purposes e.g. address setting as a function of a Communication-DTM
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 46 shows how DTM and Frame Application realize the use case “System Planning”.

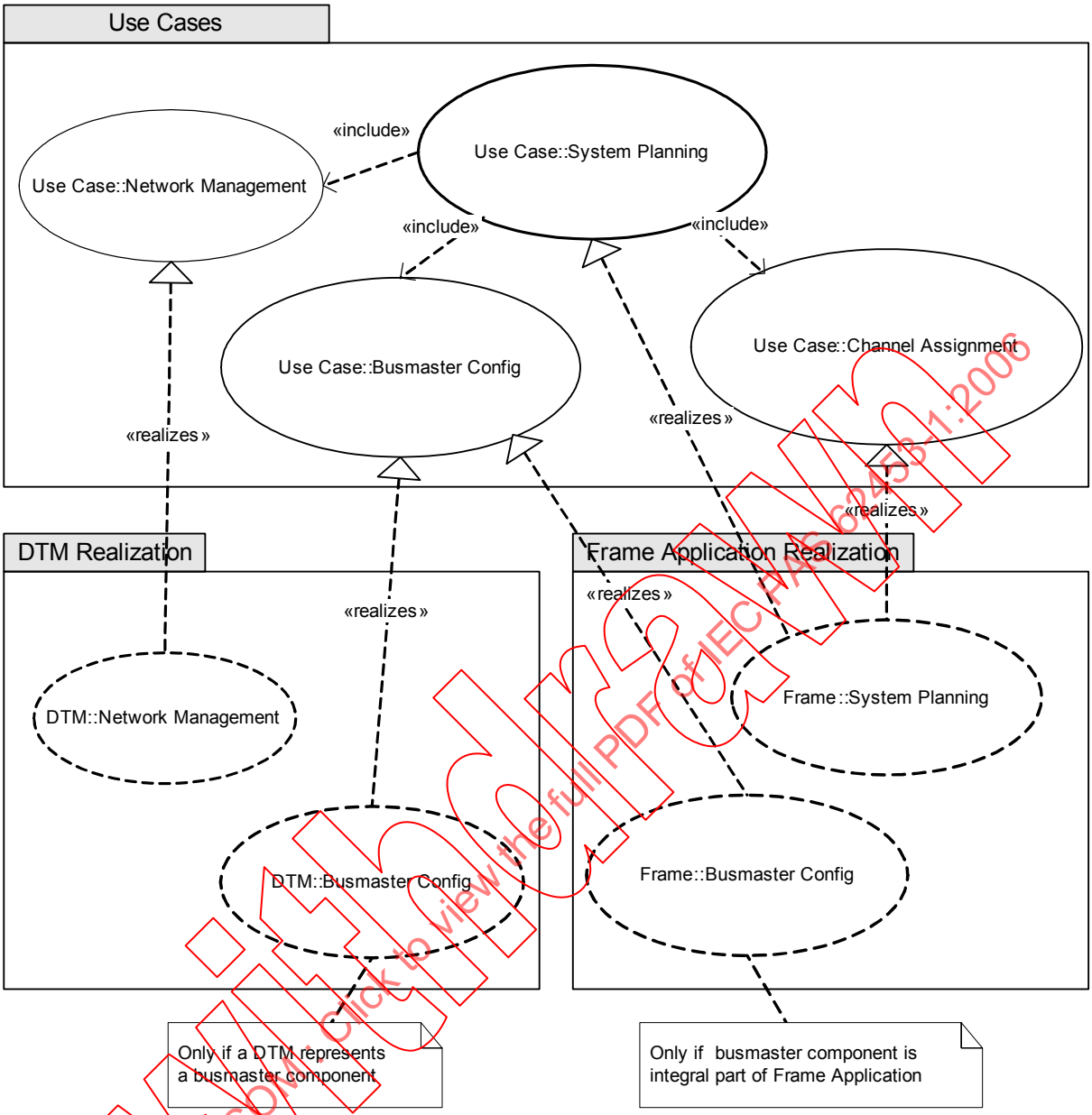

Figure 46 – Realization of use case "System Planning"

Table 33 provides a description for the use case "List of Supported Devices".

Table 33 – Description of use case “List of Supported Devices”

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	
	Operator	
	Maintenance	
	Planning Eng.	Accessible for viewing only
UserFlags	Administrator	Accessible for editing
	OEM Service	No changes
Included use case		
Brief description		
Included use case		
Brief description		

Figure 47 shows how the Frame Application realizes the use case “List of Supported Devices”.

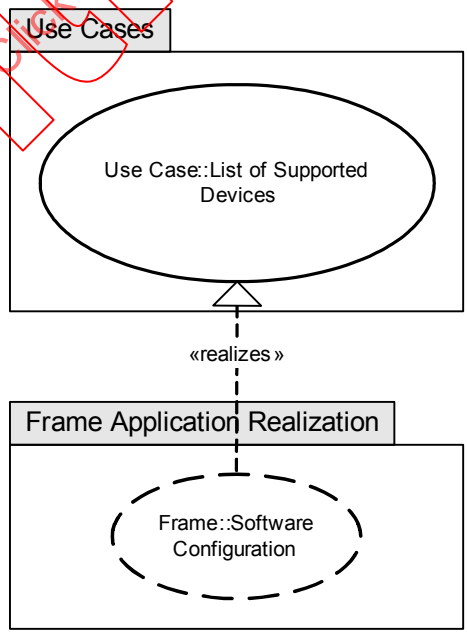

Figure 47 − Realization of use case “List of Supported Devices”

9.2.5 OEM service

The “OEM Service” actor may only have extended access to use cases of other actor roles, see Figure 48.

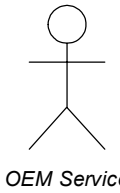

Figure 48 − No use case for actor “OEM Service”

9.2.6 Administration

The “Administrator” user flag allows execution of “Administration” use cases, see Figure 49.

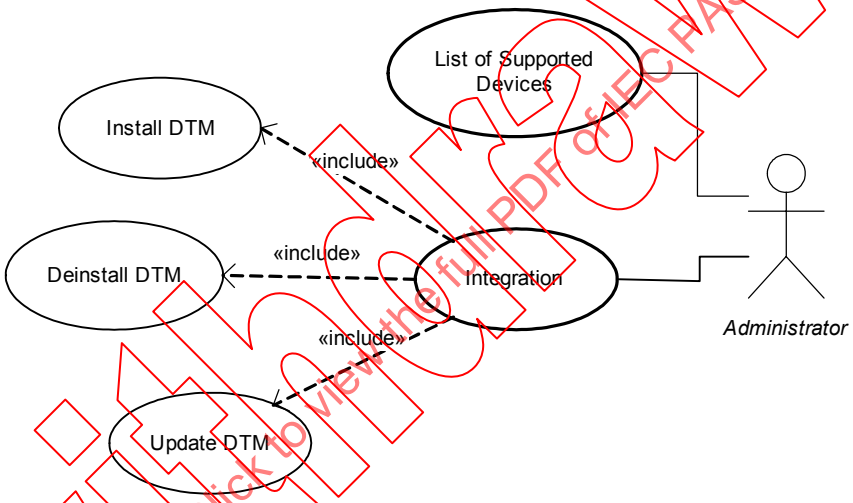

Figure 49 − Use case – “Administration”

Table 34 provides a description of the use case “Integration” and related use cases.

Table 34 – Description of use case “Integration”

Use case		Integration
Brief description		This use case enables the actor to install, deinstall or update 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
UserLevel	Observer	
	Operator	
	Maintenance	
	Planning Eng.	
User Flag	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 DTM**
Brief description	dito
GUI:	Responsibility of the DTM

Figure 50 shows how the “Integration” use case is implemented by DTMs and Frame Application.

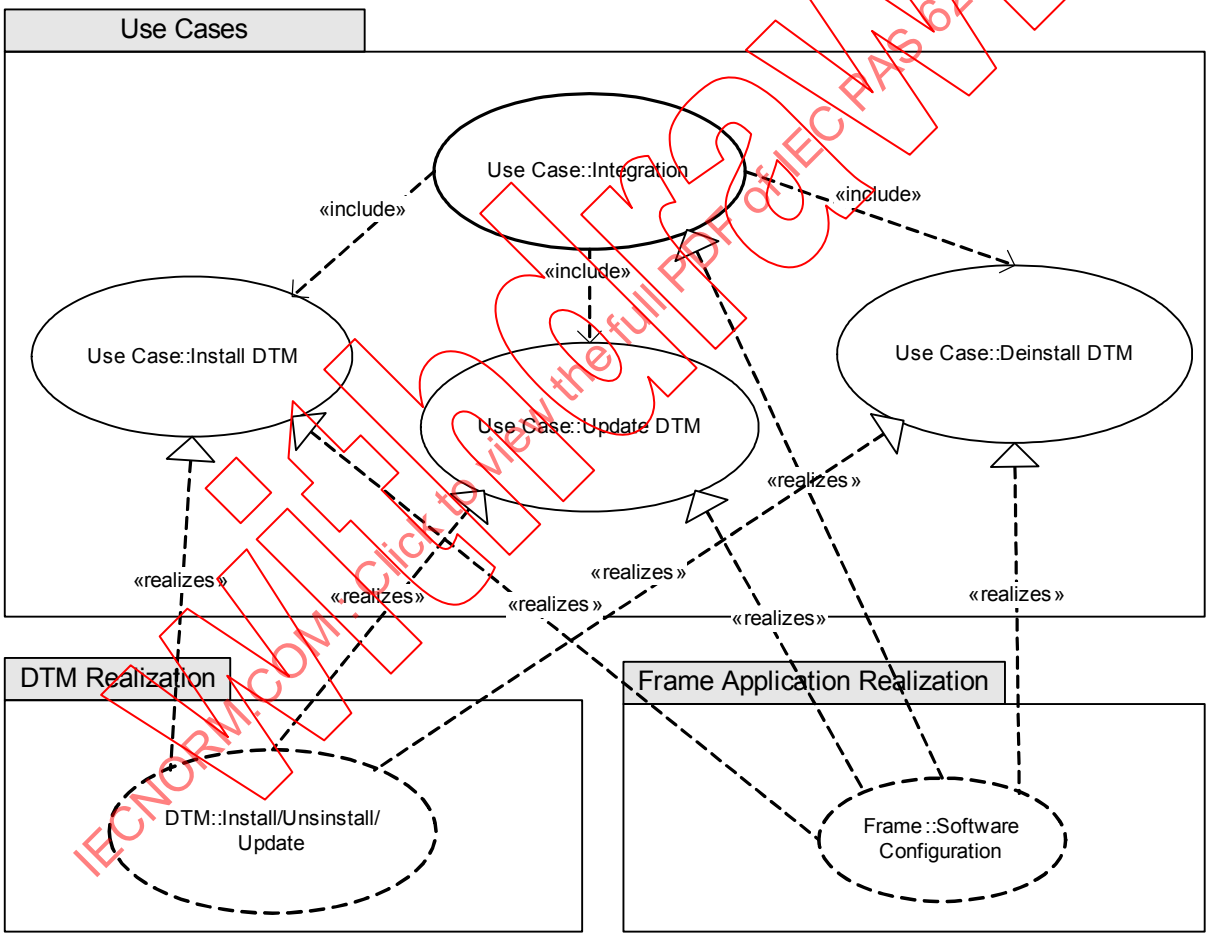

Figure 50 − Realization of use case “Integration”

9.3 DTM use case realization

As shown in the preceding sections, all Use cases are realized operations which are part of a DTM or Part of a Frame Application. In this section the operations contributing to the realization of the DTM related Use cases are explained.

The operations of the DTM may contain a graphical interface as well as a print function. Refer to 5.2.

The following description of the operations elements of the DTM is presented in tabular form, see Table 35. For each operation element the table contains the following entries.

- Brief description: Brief description of the operations.
- Extends Use case(s): Listing of all Use cases whose realization involves this operation.
- Invocation: Description showing how this operation is called.
- Application ID: Identification for application context.

Table 35 − Description of the operation elements of the DTM

Operation	**Adjust SetValue**
Brief description	Enables the actor to adjust the SetValues of a device (e.g. positioner, controllers)
Realizes use case(s)	Adjust SetValue
Invocation	The Adjust-SetValue-Operation is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtAdjustSetValue
Operation	**Archive View**
Brief description	A DTM may provide an archive for historical data. This realization element is for viewing the archived data
Realizes use case(s)	Archive
Invocation	The Archive View is called directly by the Frame Application
Application ID	-- not supported --
Operation	**Asset Management**
Brief description	The contents of the Asset Management have not been defined yet
Realizes use case(s)	Asset Management
Invocation	The Asset-Management-Operation is called directly by the Frame Application
Application 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.
Realizes use case(s)	Device-Specific Audit Trail
Invocation	Is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtAuditTrail
Operation	**Bus Master Configuration**
Brief description	GUI for configuration of the bus master
Realizes use case(s)	Bus Master Configuration
Invocation	Is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtManagement

Operation	Calibration
Brief description	Calibration procedures to adjust the input for e.g. pressure transmitters or the current for the output
Realizes use case(s)	Online Functions
Invocation	Is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtCalibration
Operation	**Configuration**
Brief description	The configuration operation lets a user define the structure of complex devices. Configuration is used online and offline.
Realizes use case(s)	Configuration
Invocation	is called directly by the Frame Application
Application 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
Realizes use case(s)	Create Instance
Invocation	is called directly by the Frame Application
Application ID	-- not supported --
Operation	**Delete Instance**
Brief description	Deletes the data
Realizes use case(s)	Delete Instance
Invocation	is called directly by the Frame Application
Application ID	-- not supported --
Operation	**Diagnosis**
Brief description	This operation realizes all diagnostic methods a DTM provides for a device check
Realizes use case(s)	Online Status
Invocation	is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtDiagnosis
Operation	**Download**
Brief description	The Download Operation writes the current parameterization to the field device
Realizes use case(s)	Download
Invocation	is called directly by the Frame Application
Application 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
Realizes use case(s)	Simulation
Invocation	Is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtForce
Operation	**Identify**
Brief description	Displays identification of a device instance information e.g. address, TAG, Fieldbus-Rev., DD-Rev., Firmware. These information is protocol dependant. It also may contain device type specific information. Further information may be provided: references to documentation, area to add comments to the device instance
Realizes use case(s)	Parameter Set Online View
Invocation	The Identify is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtIdentify
Operation	**Install/Uninstall/Update**
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
Realizes use case(s)	Install DTM, Deinstall DTM, Update DTM
Invocation	Can be started directly by the Frame Application or by the surface of the operating system
Application ID	-- not supported --
Operation	**Load Data Set**
Brief description	DTM loads data from a storage with a compatible data set format. Additional check for data compatibility must be done by the DTM
Realizes use case(s)	Upgrade
Invocation	is called directly by the Frame Application
Application 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
Realizes use case(s)	DTM-Specific Login
Invocation	is called during other operations, when an “OEM Service” actor accesses operations with OEM-specific function elements
Application ID	-- not supported --

Operation	Main operation
Brief description	Entry point for a combination of different functions of a DTM (e.g. combination of parametrization, configuration, diagnosis, measured values)
Realizes use case(s)	
Invocation	The Main operation is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtMainOperation
Operation	**Network Management**
Brief description	Function of a Communication-DTM to manage the network information
Realizes use case(s)	Network Management
Invocation	The Network Management is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtNetworkManagement
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
Realizes use case(s)	Online Trend
Invocation	is called directly by the Frame Application
Application 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 IDtmDiagnostics::Compare() function
Realizes use case(s)	Persistent Data Comparison
Invocation	is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtOfflineCompare
Operation	**Offline Parameterize**
Brief description	This operation enables editing of the instance persistent data set
Realizes use case(s)	Offline Parameterization
Invocation	is called directly by the Frame Application
Application 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
Realizes use case(s)	Device-/Persistent Data Comparison
Invocation	is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtOnlineCompare

Operation	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
Realizes use case(s)	Online parameterization, Online Functions, Parameter Set Online View
Invocation	is called directly by the Frame Application
Application ID	FDTApplicationIdSchema:fdtOnlineParameterize
Operation	**Parameter Access**
Brief description	DTM provides read and write access via SingleDataAccess interfaces
Realizes use case(s)	Replacement
Invocation	is only called indirectly via other operations
Application ID	-- not supported --
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
Realizes use case(s)	Plausibility check
Invocation	is only called indirectly via other operations
Application ID	-- not supported --
Operation	**Print**
Brief description	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
Realizes use case(s)	The print operation participates in the realization of several operations of the DTM. Every application context a DTM supports with a GUI, should also support printing
Invocation	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
Application ID	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
Operation	**Scan**
Brief description	A channel provides a list of available devices.
Operation	**Upload**
Brief description	Loads the current configuration from the device to the device instance data set of the DTM
Realizes use case(s)	Upload
Invocation	is called directly by the Frame Application
Application ID	-- not supported --

9.4 Frame Application use case realization

This section describes any operation the Frame Application must provide for the realization of the FDT Uses Cases. The operations are again listed in tabular form (see Table 36) and introduced in a brief description explaining their function in the context of the FDT-concept.

Table 36 − Description of the operation elements of the Frame Application

Operation	**GUI**
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**
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	Frame Application provides asset management functions
Operation	**Audit Trail**
Brief description	Frame Application provides mechanism for storage and display of audit trail information
Operation	**Bulk Data Handling**
Brief description	Frame Applications provide an option to apply individual functions like the Upload and Download of groups of devices. This operation realizes the parameterization and management of a wide range of devices
Operation	**Busmaster Config**
Brief description	If Frame application provides access to Fieldbus communication this operation allows configuration of the communication
Operation	**Data Format Association**
Brief description	Frame Application manages the format information of stored data sets and is able to associate DTM to the data sets. Refer to 6.25.2
Operation	**DTM matching**
Brief description	Frame Application compares the results from Scan to the information that is available in the Device Catalogue and finds the correct DTM for a physical device. Refer to 6.2.1.1

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	**Parameter Migration**
Brief description	Frame Application provides mechanism to copy data from one DTM instance to an other DTM instance. This mechanism is handled with SingleDataAccess interfaces and may need additional user interaction
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

10 FDT sequence charts

10.1 DTM peer-to-peer communication

For a DTM each connection is established as a peer-to-peer connection. This clause describes the communication function calls from a DTM developer's point of view.

10.1.1 Establish a peer-to-peer connection between DTM and device

The connection is established by methods calls to the IFdtCommunication pointer, which was provided to the DTM (see Figure 51).

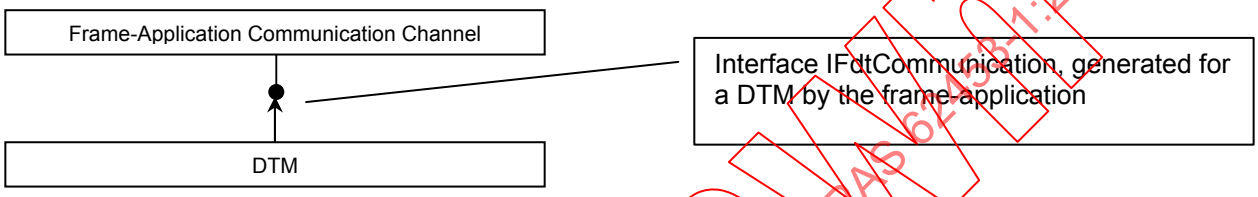

Figure 51 – Peer-to-peer connection between DTM and device

10.1.2 Asynchronous connect for a peer-to-peer connection

The connect request is handled asynchronously (see Figure 52).

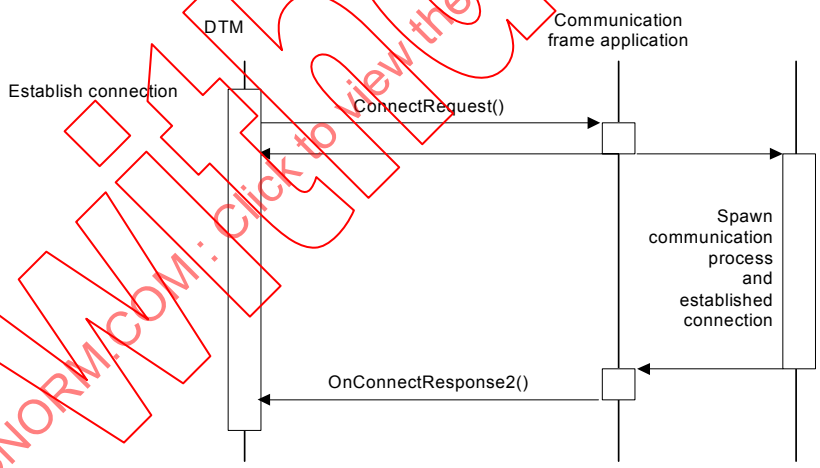

Figure 52 – Asynchronous connect (peer to peer)

Used methods:

IFdtCommunication::ConnectRequest()

IFdtCommunicationEvents2::OnConnectResponse2()

10.1.3 Asynchronous disconnect for a peer-to-peer connection

Also the disconnect is handled in an asynchronous way (see Figure 53).

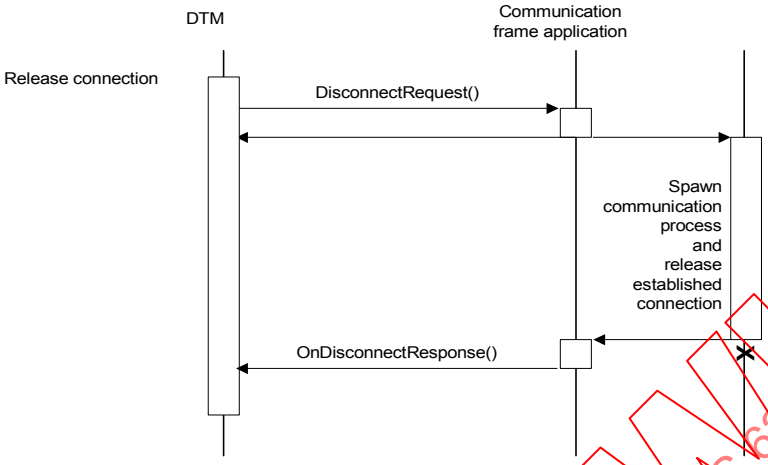

Figure 53 – Asynchronous disconnect (peer to peer)

Used methods:

IFdtCommunication::DisconnectRequest()
IFdtCommunicationEvents::OnDisconnectResponse()

10.1.4 Asynchronous transaction for a peer-to-peer connection

Transaction requests are handled asynchronously (see Figure 54).

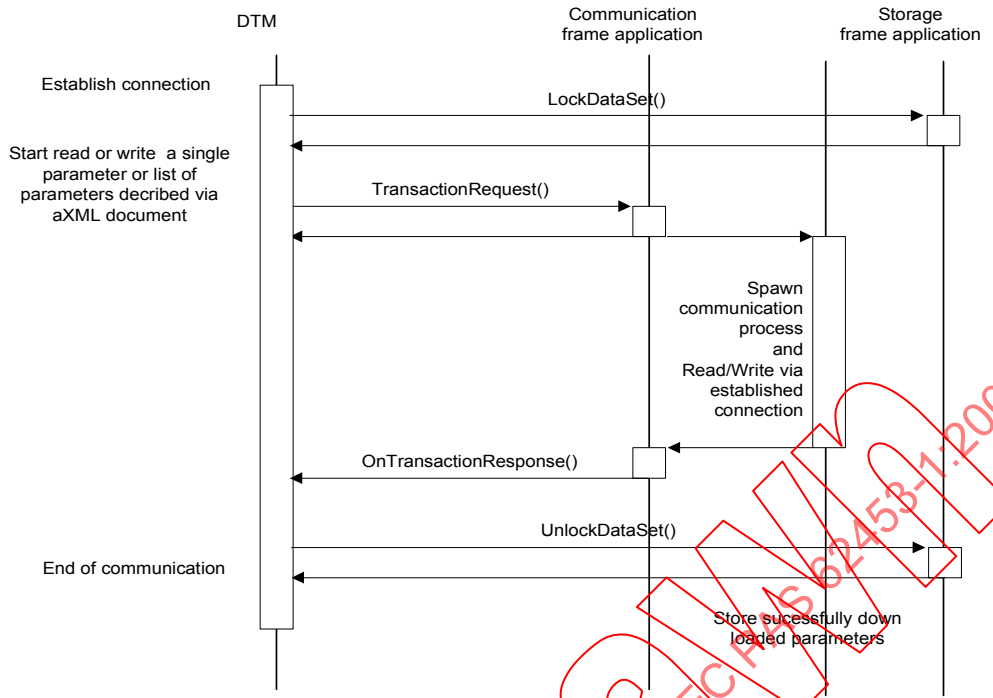

Figure 54 – Asynchronous transaction (peer to peer)

Used methods:

IFdtCommunication::TransactionRequest()

IFdtCommunicationEvents::()

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

10.2 Nested communication

This subclause is important for DTM developers who support a device with gateway functionality (e.g. Remote I/O). This subclause describes the communication function calls from the point of view of a developer of a communication component.

Nested communication is used to establish the connection to a device on a sub-system. For example, a DTM calls a field device which is connected to a channel of a Remote I/O.

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

The DTMs which have gateway functionality (Remote I/O) have to provide an FdtChannel with communication interfaces for each channel with gateway functionality.

Furthermore always the parent (DTM with gateway functionality or, at least, the Frame Application) is responsible for the communication addressing of its sub-devices. Therefore it has to set parameters like ‘tag’ and ‘BusInformation’ according to the communication protocol. (see also IDtmParameter::SetParameters())

Figure 55 shows the system topology for the following examples.

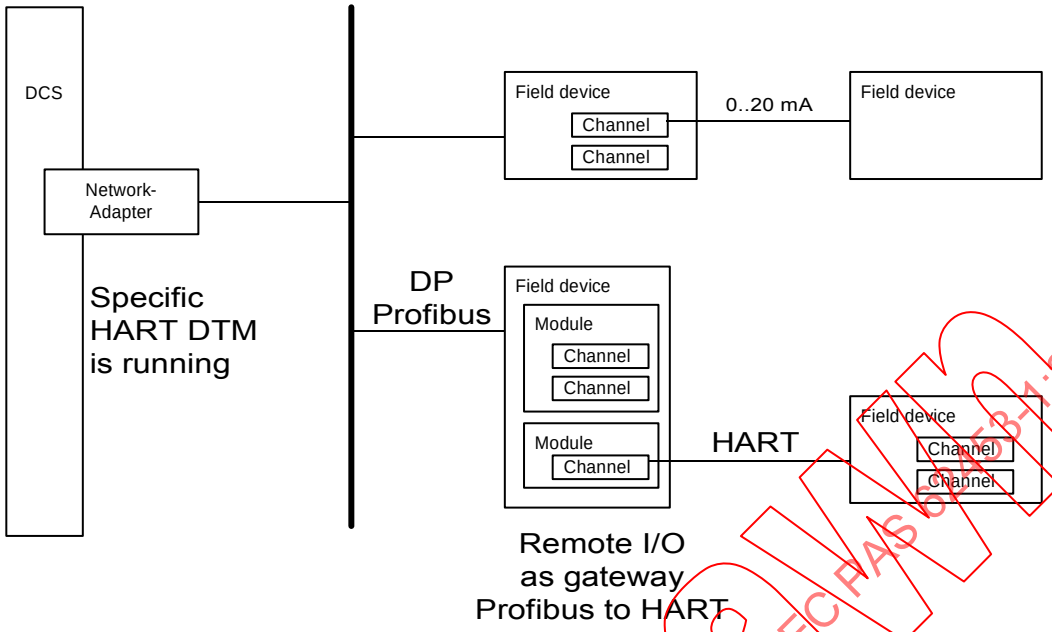

Figure 55 – System-topology

A connection from DCS to field device in such an hierarchical communication system is called “system connection” in the following clauses and sub-clauses.

10.2.1 Generate system topology

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.

10.2.1.1 Frame applications point of view

The Frame Application is responsible to generate and manage the topology. The sequence shows how the Frame Application manages the relation between DTMs and Communication-Channels (see Figure 56).

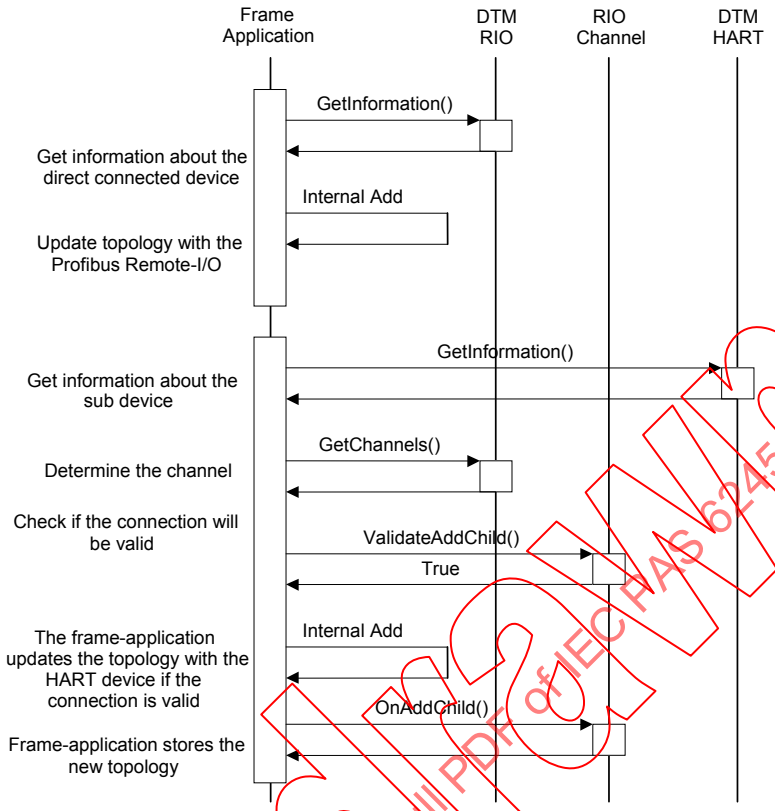

Figure 56 – Generation of system topology by Frame Application

Used methods:

IDtmInformation::GetInformation()

IDtmChannel::GetChannels()

IFdtChannelSubTopology::ValidateAddChild()

IFdtChannelSubTopology::OnAddChild()

10.2.1.2 DTMs point of view

This sequence shows the generation of the topology triggered by a DTM, see Figure 57.

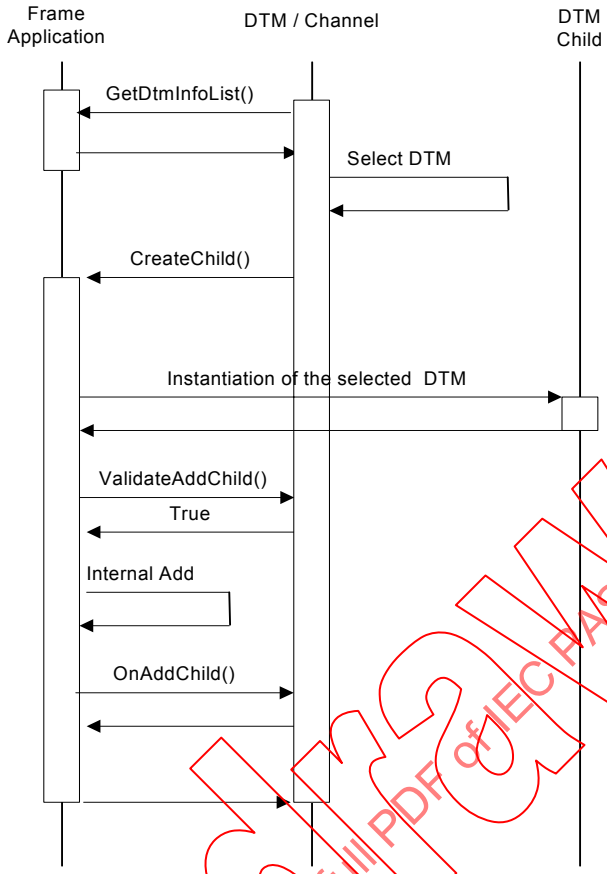

Figure 57 – Generation of system topology – participation of DTM

Used methods:

IFdtTopology::GetDtmInfoList()
IFdtTopology::CreateChild()
IFdtChannelSubTopology::ValidateAddChild()
IFdtChannelSubTopology::OnAddChild()

10.2.2 Establish a system connection between DTM and device

Figure 58 shows possible communication hierarchy in FDT.

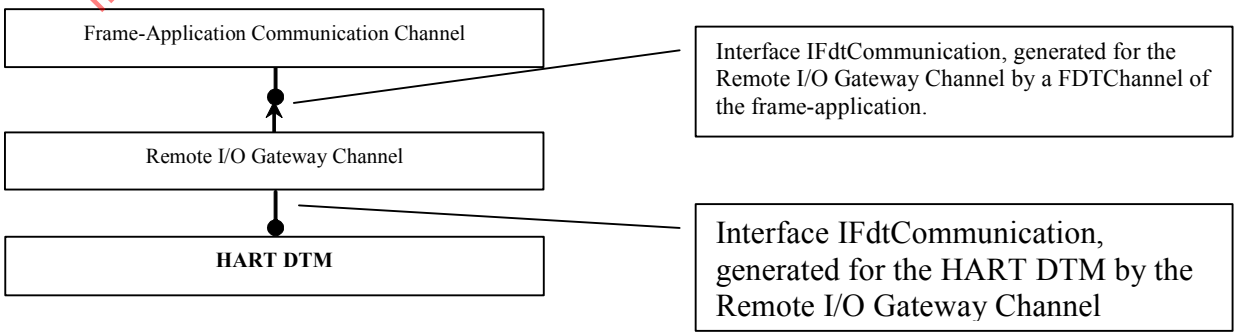

Figure 58 – System connection (across communication hierarchy)

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

Used methods:

IDtm::SetCommunication()

IFdtTopology::GetChildNodes()

IFdtCommunication::ConnectRequest()

IFdtCommunicationEvents2::OnConnectResponse2()

10.2.3 Asynchronous transaction for a system connection

Example HART/PROFIBUS.

The transaction-function-call of the HART DTM is realized as a read and write-function at the PROFIBUS communication component of the Remote I/O. With the PROFIBUS write function the communication component transfers the HART data to the HART Master at the Remote I/O. The answer of the HART device can be received from the HART Master by the according PROFIBUS read function call. The addressing for the read and write function call depends on the hardware and the configuration of the Remote I/O and is well known to the communication channel. (Figure 59)

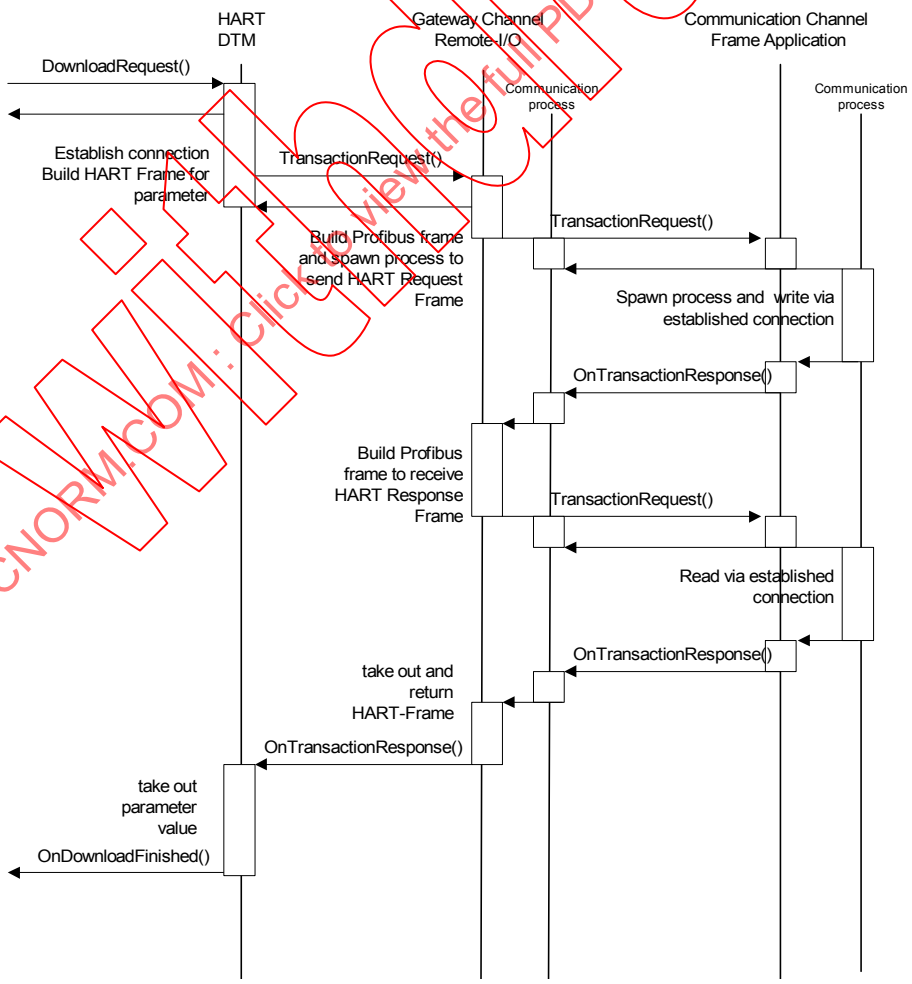

Figure 59 − Asynchronous transactions (system connection)

Used methods:

IFdtCommunication::TransactionRequest()

IFdtCommunicationEvents::OnTransactionResponse()

IDtmOnlineParameter::DownloadRequest()

IDtmEvents::OnDownloadFinished()

10.3 Topology scan

10.3.1 Scan network

Sequence diagram: scan network topology (see Figure 60).

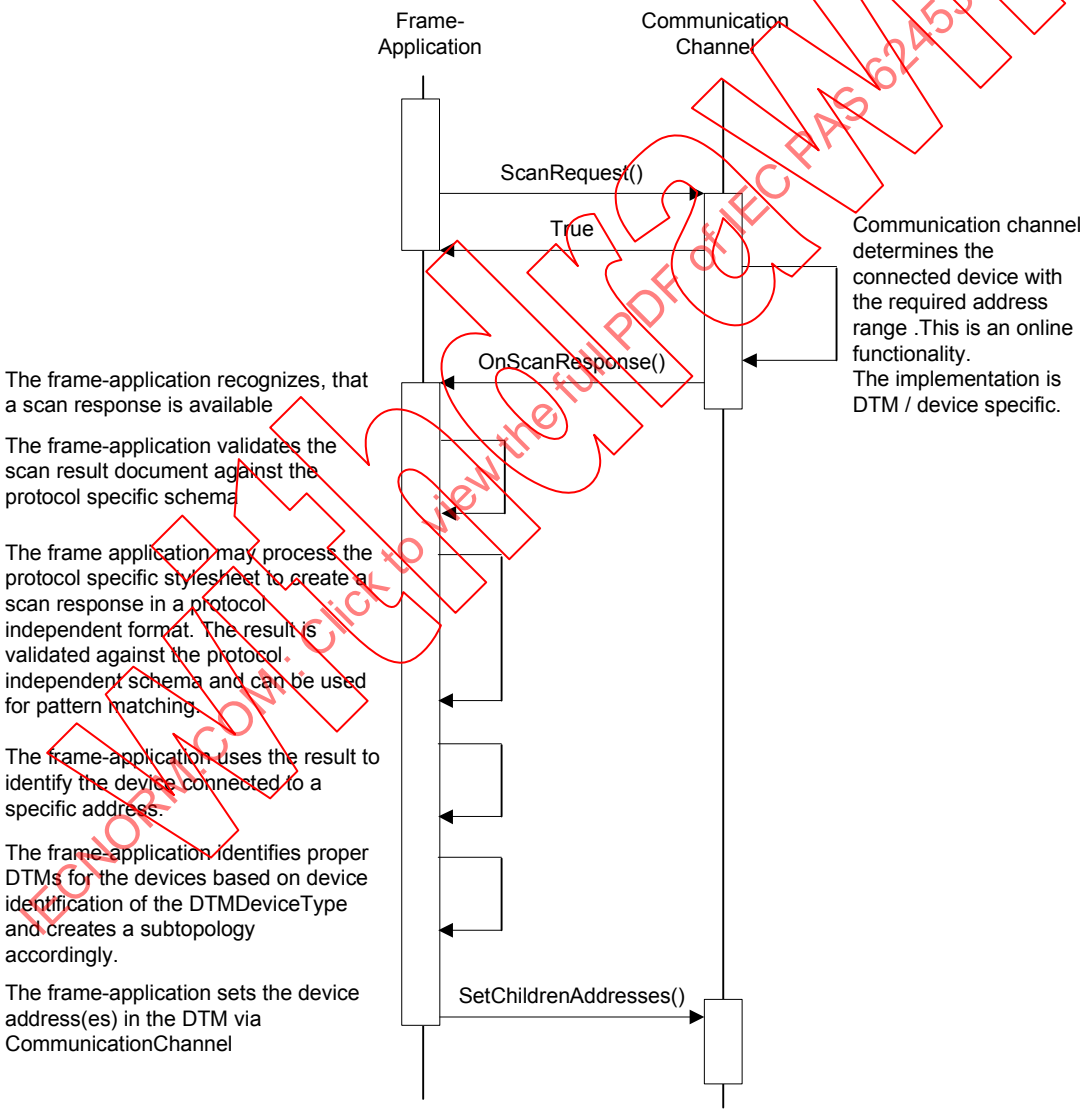

Figure 60 – Scan network topology

Used methods:

IFdtChannelScan::ScanRequest()

IDtmScanEvents::OnScanResponse()

IFdtChannelSubTopology2::SetChildrenAddresses()

10.3.2 Cancel topology scan

In this scenario Frame Application cancels an active scan request (see Figure 61).

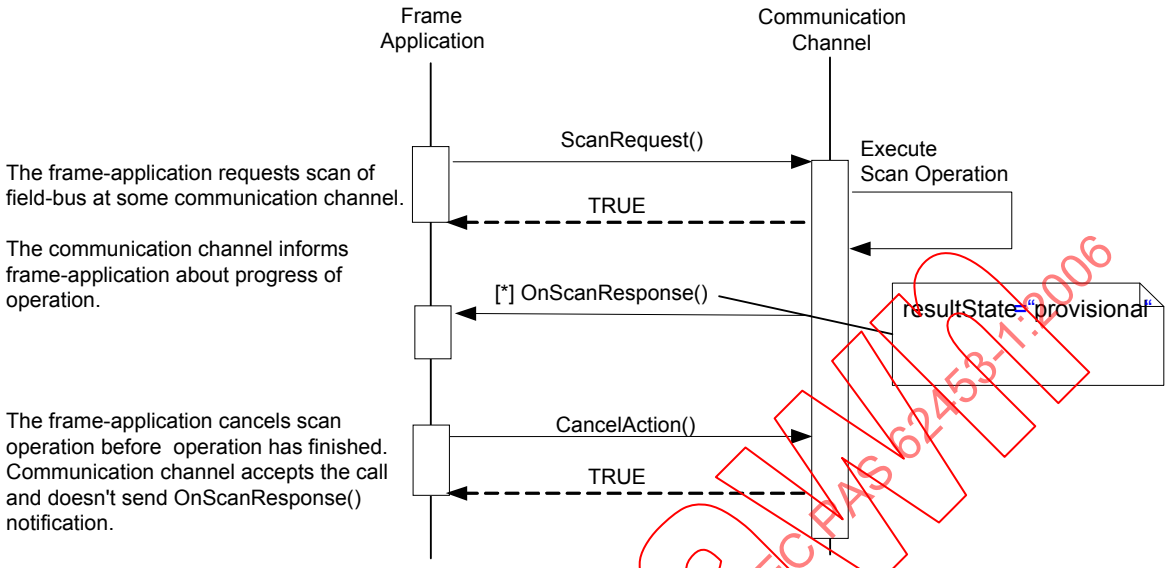

Figure 61 – Cancel topology scan

Used methods:

IFdtChannelScan::ScanRequest()

IFdtChannelScan:CancelAction()

IDtmScanEvents::OnScanResponse()

Behavior

In case of problems it is up to the communication channel to handle problems in a way that the communication channel returns final response with an error information.

If the final response is not returned in expected time, user or Frame Application can cancel this action. No further OnScanResponse() should be called.

In case of an error in a provisional scan result, it is up to the Frame Application to cancel the scanning or to handle the particular error in the result and continue the scan.

Progress events must be fired by communication channel while scanning is performed.

10.3.3 Provisional scan result notifications

In this scenario communication channel sends provisional Topology Scan result XMLs to the Frame Application (see Figure 62).

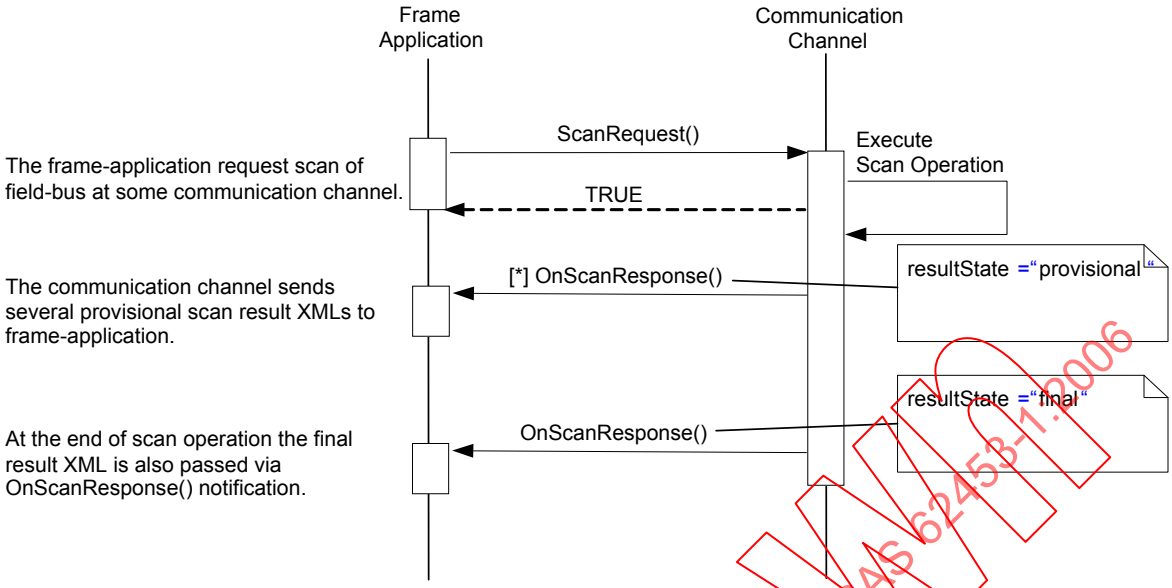

Figure 62 – Provisional topology scan

Used methods:

IFdtChannelScan::ScanRequest()

IDtmScanEvents::OnScanResponse()

10.3.4 Scan for communication hardware

Scan for communication hardware is needed for full automatic creation of a system topology out of an existing bus topology. Frame Application needs to check available communication hardware first and instantiate corresponding Bus Master DTM before scan for sub-topology is feasible (see Figure 63).

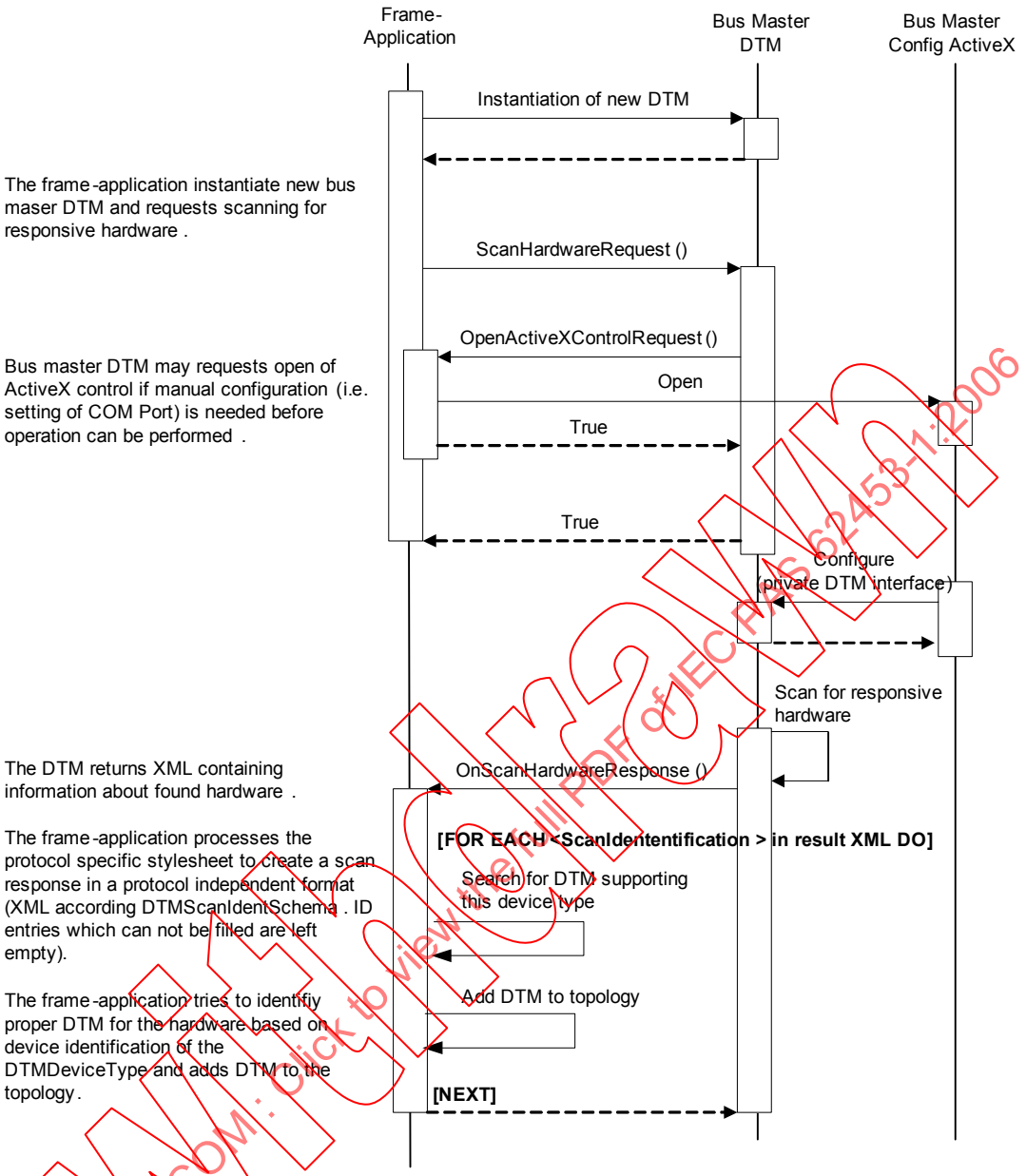

Figure 63 − Scan for communication hardware

Used methods:

IDtmHardwareIdentification::ScanHardwareRequest()

IDtmScanEvents::OnScanHardwareResponse()

IFdtActiveX::OpenActiveXControlRequest()

(or IFdtActiveX2:OpenDialogActiveXControlRequest())

10.3.5 Manufacturer-specific device identification

In this scenario a Frame Application scans an existing fieldbus network and uses DTM implementing IDtmHardwareIdentification interface to identify devices for which manufacturer-specific operation must be performed (see Figure 64).

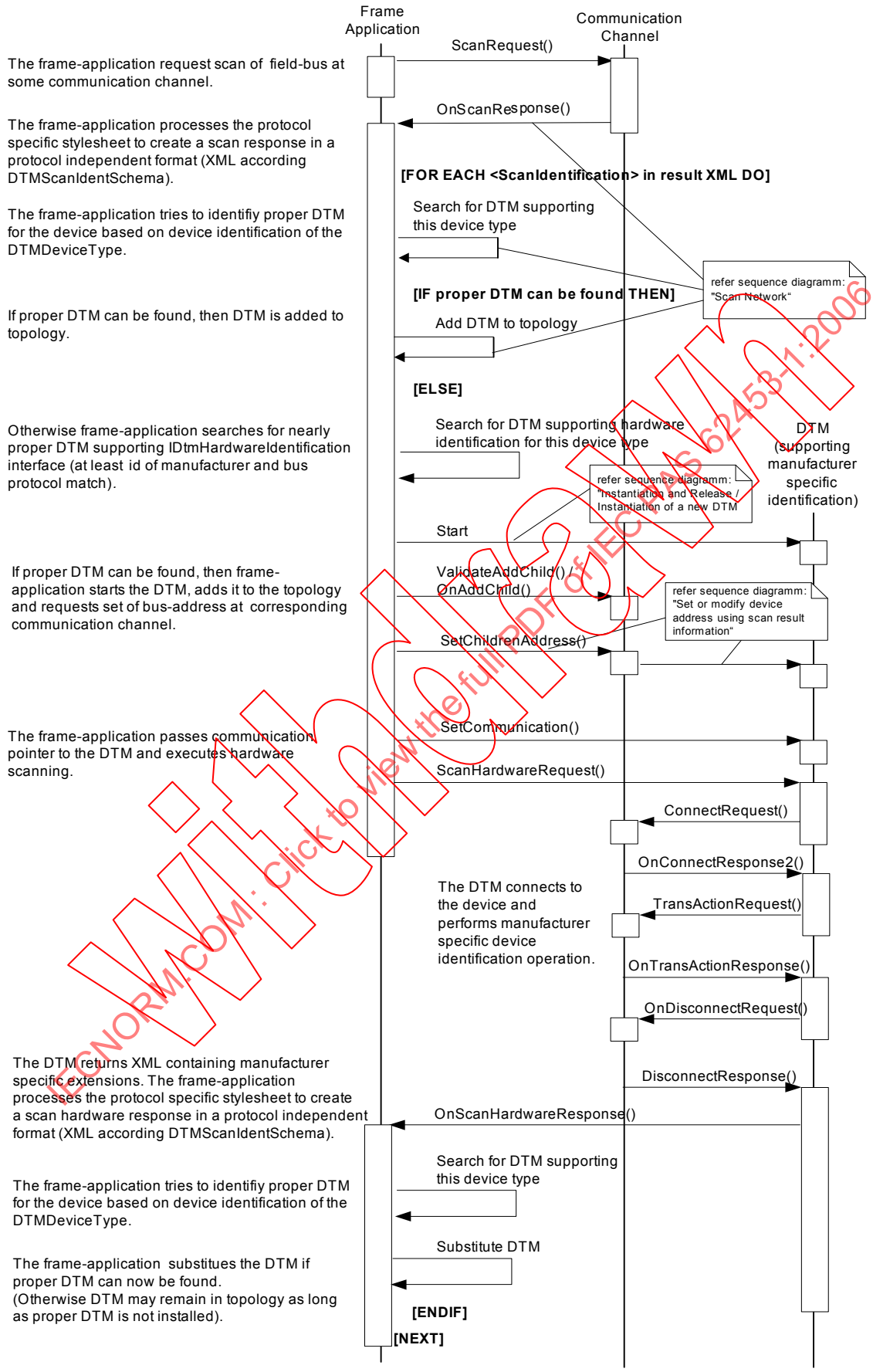

Figure 64 − Manufacturer specific device identification

Used methods:

IDtmHardwareIdentification:ScanHardwareRequest()

IDtmScanEvents::OnScanHardwareResponse()

IDtm::SetCommunication()

IFdtChannelSubTopology:ValidateAddChild()

IFdtChannelSubTopology::OnAddChild()

IFdtChannelScan::ScanRequest()

IFdtChannelSubTopology2::SetChildrenAddresses()

IDtmScanEvents::OnScanResponse()

IFdtCommunication::ConnectRequest()

IFdtCommunication::TransActionRequest()

IFdtCommunication::DisconnectRequest()

IFdtCommunicationEvents2::OnConnectResponse2()

IFdtCommunicationEvents::OnTransactionResponse()

IFdtCommunicationEvents::OnDisconnectResponse()

10.4 Registration of protocol specific FDT schemas

In order to add new protocol support to the FDT specification and to existing Frame Application installations, a CommDTM installation must provide protocol specific schemas. There are two cases.

c) Protocol specific schemas accepted and published by FDT-JIG. These schemas may be released independent on a FDT specification release.

d) Schemas of a proprietary protocol. In this case all CommDTMs and Device-DTMs are provided in one consistent package. Consistency of all related data must be ensured internally by DTM.

The following structure explains an overall workflow (see Figure 65).

- Installation of DTMs with a channel implementing IFdtCommunication.
- CommDTM is installed and contains in its setup the merge module of the protocol specific schemas.
- CommDTM copies the schemas to a CommDTM specific certain path.
- DTM library update

DTMDeviceType information is used for DTMCatalog information.

- IDtmInformation::GetInformation(), returns an XML document containing schema version and path of protocol specific schemas.
- Frame Application compares this information with the already known schemas.
- Frame Application checks, if schemas are already available in Frame Applications sub-schema path and copies the schemas; if not, or if higher version is offered.
- Frame Application is responsible to synchronize the schema files over all subdirectories.

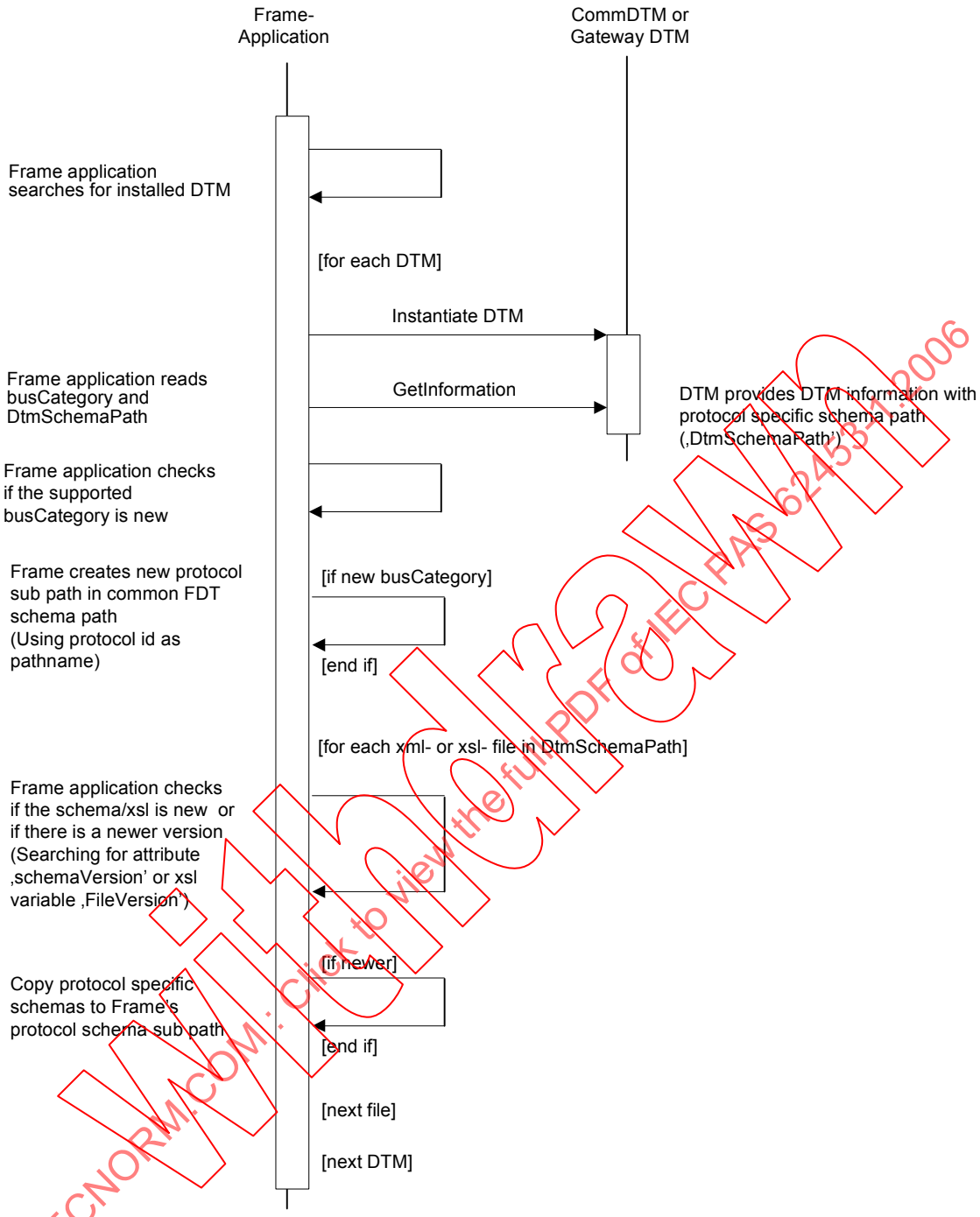

Figure 65 − Add protocol specific schemas to Frame Applications schema sub path

Used methods:

IDtmInformation::GetInformation()

After updating the schema cache, the Frame Application gets additional protocol specific device identification information from a DTM by calling IDtmInformation2::GetDeviceIdentificationInformation (see Figure 66).

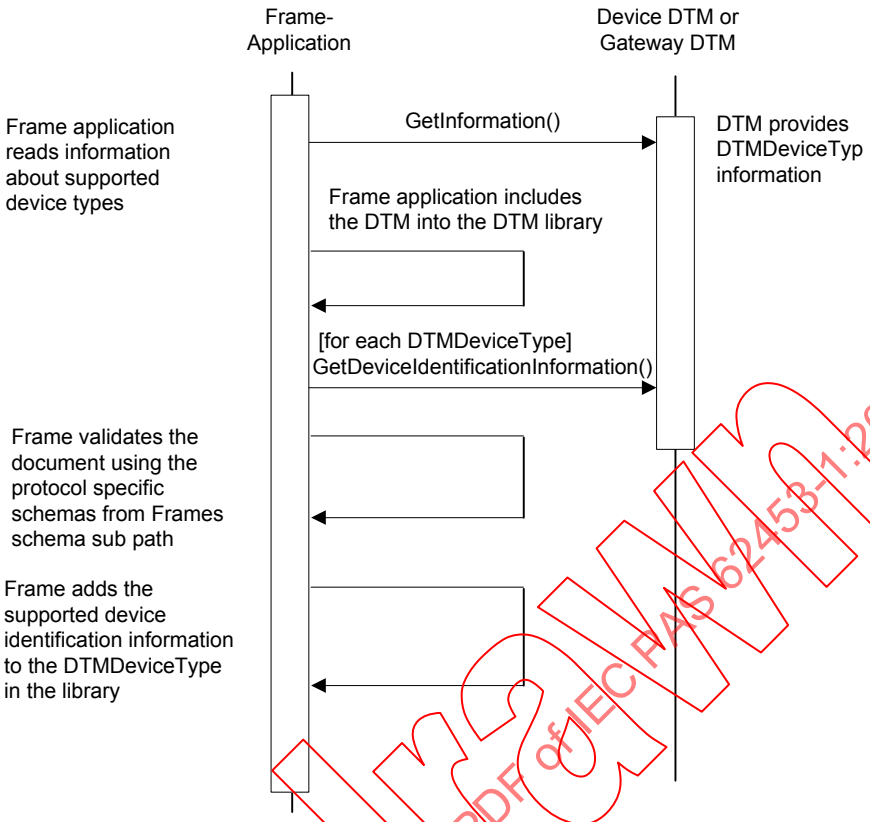

Figure 66 − Frame Application reads protocol specific device identification information of DTMDeviceTypes

10.5 Configuration of a fieldbus master

Device-specific bus parameters are needed to configure the Frame Application's fieldbus master or communication scheduler. To retrieve these parameters an interaction between DTMs and a master configuration tool is required. To provide a standard access to this bus-specific data, it is stored as a public data accessible by the predefined XML tag

<busMasterConfigurationPart>

<busMasterConfigurationPart> is a binary stream which contains the device specific bus information according to the Fieldbus-Protocol-Specification.

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

All other entries may be filled up with substitute values like zeros. The substituted values of the <busMasterConfigurationPart> structure will be set by the environment's master configuration tool according to the requirements of the complete bus.

Independent of the values filled in, it is very important that the structure generated by the DTM adheres to the definitions of the fieldbus-specification.

After all network participants have written their instance data, the master configuration tool can commission the fieldbus (see Figure 67). 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().

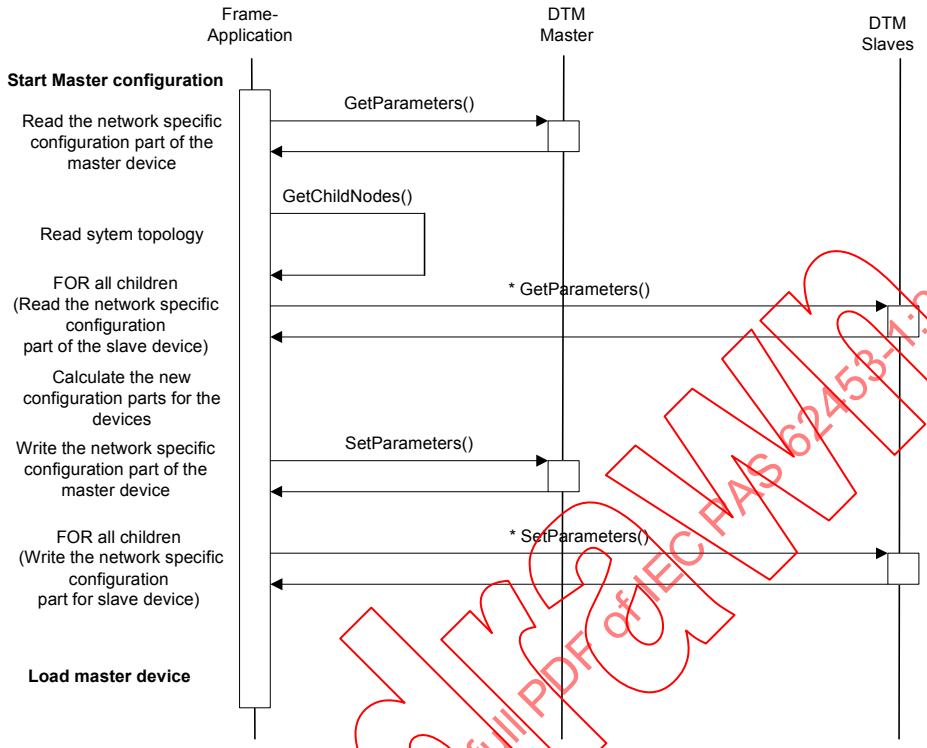

Figure 67 – Bus master configuration

Used methods:

IDtmParameter::GetParameters()

IDtmParameter::SetParameters()

IFdtTopology::GetChildNodes()

10.6 Starting and releasing applications

In general, the Frame Application uses the interface IDtmApplication to start an application or uses IDtmActiveXInformation to get the information about an ActiveX control for the required application. The following sequence chart (see Figure 68) 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().

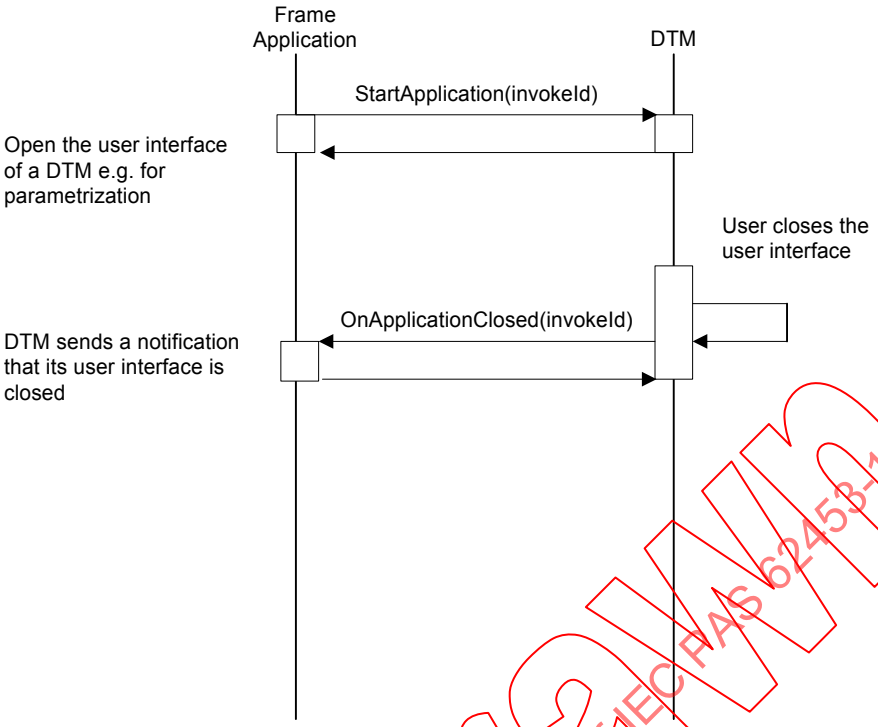

Figure 68 – Starting and releasing applications

Used methods:

IDtmApplication::StartApplication()

IDtmEvents::OnApplicationClosed()

10.7 Channel access

The information for the access of I/O data of a device within a Frame Application and the communication interfaces for nested communication are available via FdtChannel objects. These objects carry all address information for the configuration of a fieldbus master or a connected fieldbus controller. How to access such a channel object is fieldbus independent (see Figure 69). But the information which is accessible as an XML document depends on the specific fieldbus protocol.

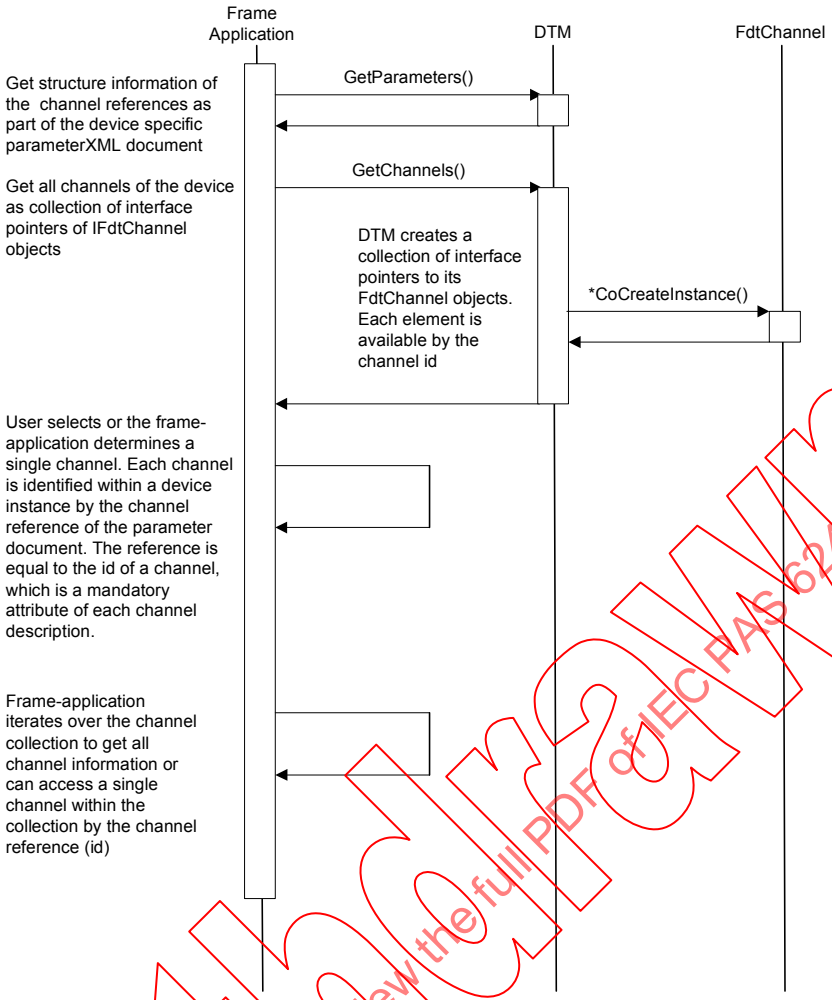

Figure 69 – Channel access

Used methods:

Standard Microsoft

IDtmParameter::GetParameters()

IDtmChannel::GetChannels()

10.8 DCS Channel assignment

During the channel assignment the relationship between a channel provided by the DTM and a channel within the Frame Application (DCS channel) is established by the Frame Application (see Figure 70).

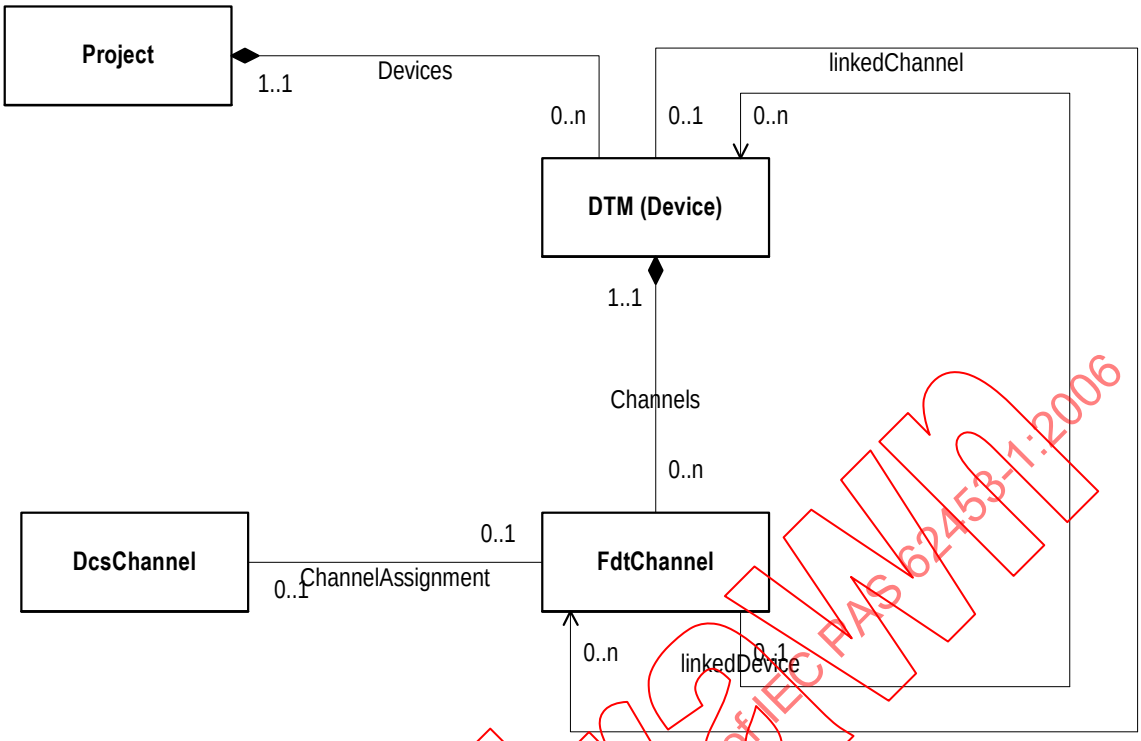

Figure 70 – DCS channel assignment single DTM

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 (see Figure 71).

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.

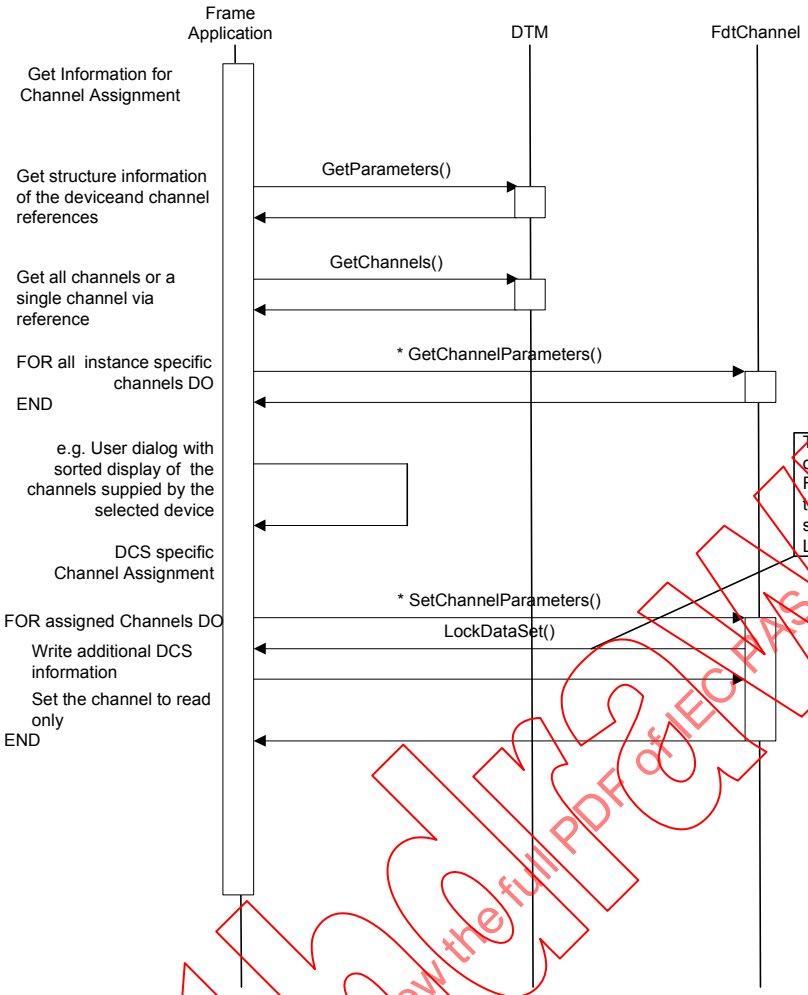

Figure 71 – Sequence of channel assignement for a single DTM

Used methods:

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

IDtmParameter::GetParameters()

IDtmChannel::GetChannels()

IFdtChannel::GetChannelParameters()

IFdtChannel::SetChannelParameters()

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.

The following figure shows a sub structure with DTMs for modular devices especially for remote I/Os with modules of different vendor (see Figure 72). The connection of the Device-DTM and its module DTMs is established via the standard topology methods. At this substructure 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.

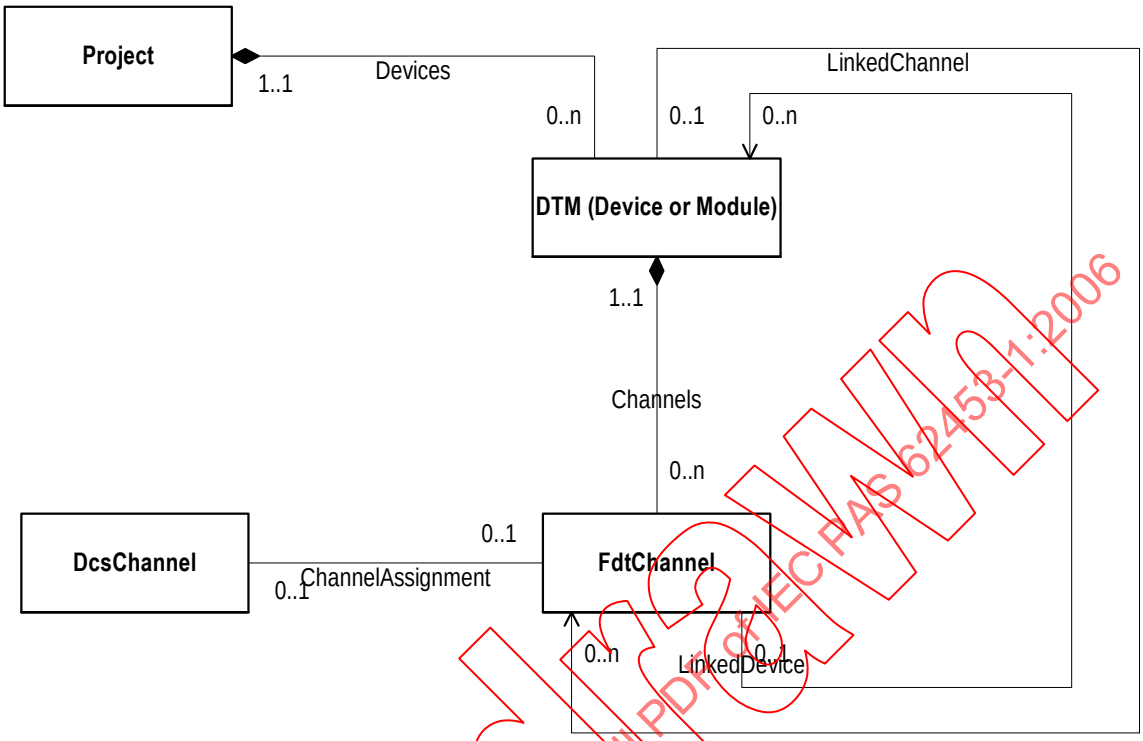

Figure 72 – Modular DTM structure

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

So if a DTM instance represents only a part of a device, the information for channel assignment has to be collected by the Frame Application.

In the case of a modular device like remote I/Os, the gateway is signed as main DTM and is the starting 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 (see Figure 73).

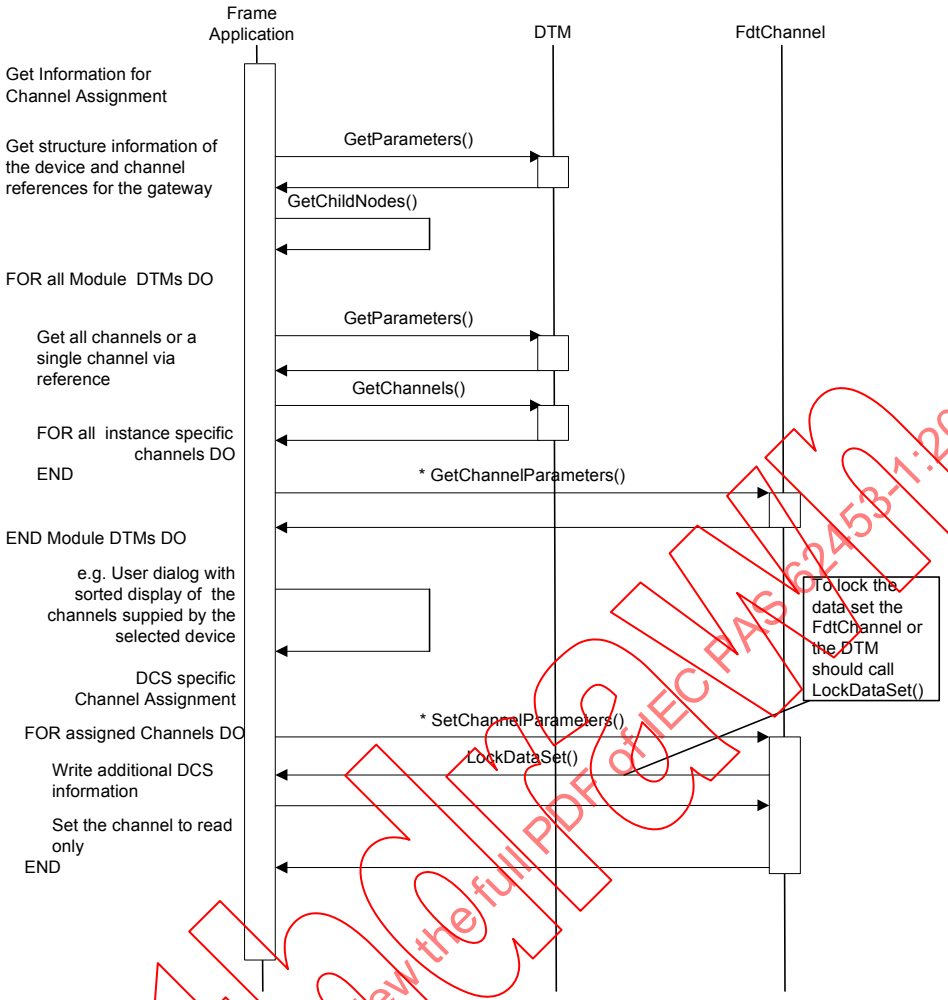

Figure 73 – Channel assignment for modular DTMs

Used methods:

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

IDtmParameter::GetParameters()

IDtmChannel::GetChannels()

IFdtChannel::GetChannelParameters()

IFdtChannel::SetChannelParameters()

IFdtTopology::GetChildNodes()

10.9 Printing of DTM specific documents

In general the Frame Application uses IDtmDocumentation for its documentation. This interface allows the Frame Application to ask a DTM for context-specific documents identified by the information passed via an XML document (see Figure 74). 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.

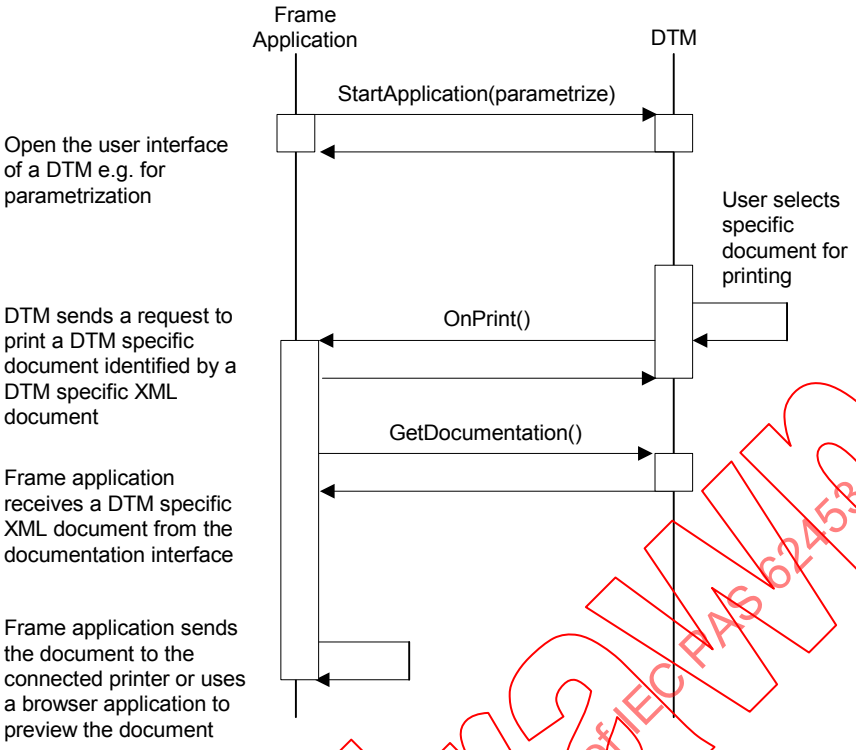

Figure 74 – Printing of DTM specific documents

Used methods:

IDtmApplication::StartApplication()

IDtmEvents::OnPrint()

IDtmDocumentation::GetDocumentation()

10.10 Printing of frame application specific documents

IDtmDocumentation allows the Frame Application to ask a DTM for context-specific documents identified by the information passed via an XML document (see Figure 75).

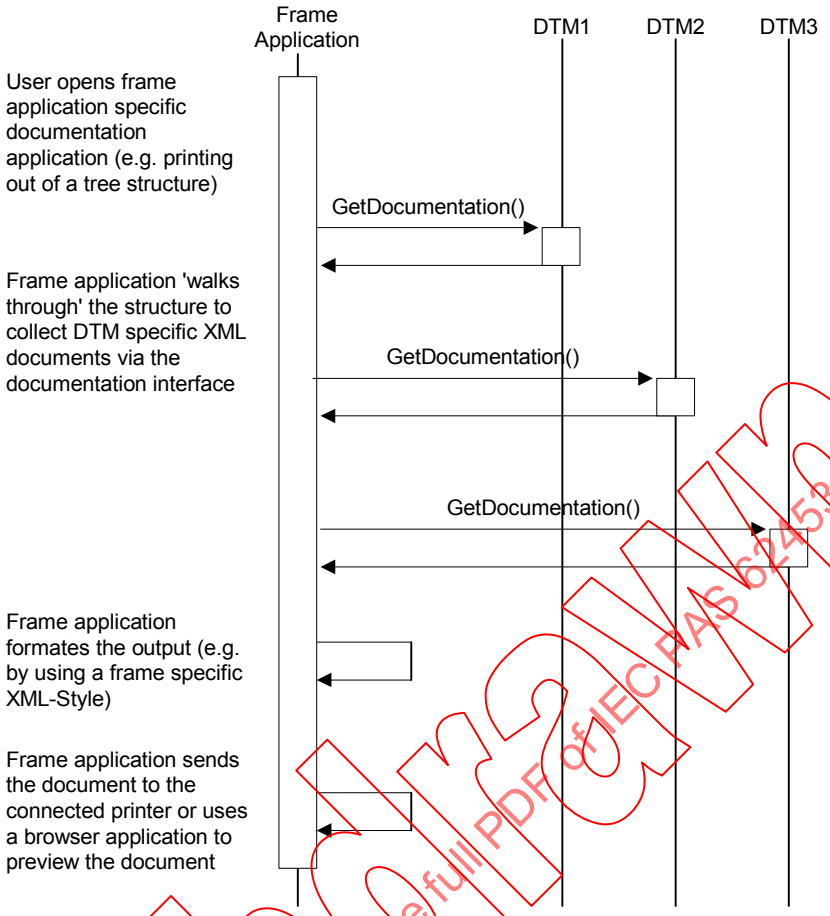

Figure 75 – Printing of frame application specific documents

Used methods:

IDtmDocumentation: GetDocumentation()

10.11 Propagation of changes

Within a multi-user environment, it is common that more than one DTM have access to the same data set. To synchronize DTMs which are started by several users on different workplaces FDT provides a notification mechanism via OnParameterChanged() (see Figure 76). 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.

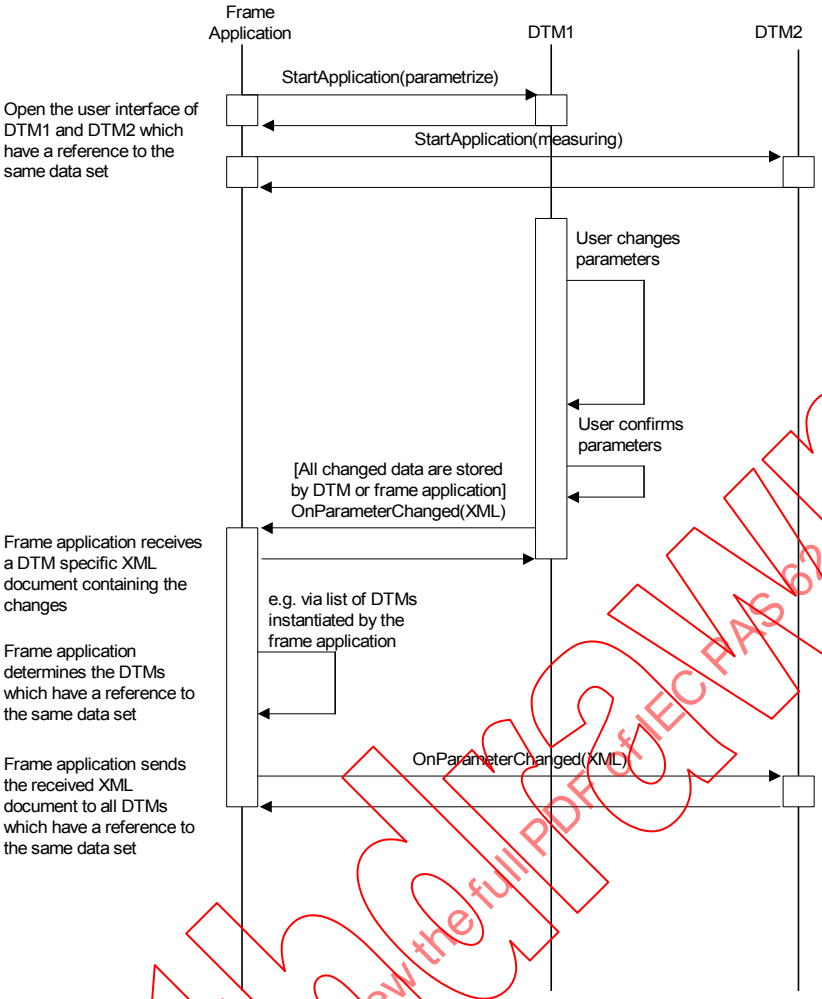

Figure 76 – Propagation of changes

Used methods:

IDtmApplication::StartApplication()

IDtmEvents::OnParameterChanged()

IFdtEvents::OnParameterChanged()

10.12 Locking

Within a multi-user environment, it is common that more than one DTM have access to the same data set. To synchronize DTMs which are started by several users on different workplaces FDT provides a locking mechanism. Target for this event mechanism is that only one DTM has read/write access to the data set. All other DTMs have read access only.

For this reason a DTM must lock it is data set only if required and only during modification of the data. After the instance data is saved by the Frame Application and is not further under modification the DTM must unlock it is data set immediately to enable concurrent access to the device data by other DTM instances within a multi-user environment.

- Before opening an ActiveX the DTM must try to lock the dataset. If successful all user input fields can be enabled, in the other case user input fields must be disabled. After closing all ActiveX controls in case of a locked datatset the DTM should request to save the dataset and unlock the data set after Frame Application has saved the dataset.
- Before an upload the DTM must also try to lock the dataset, after upload DTM should call IFdtContainer::SaveRequest() and unlock dataset after Frame Application has saved dataset.
- A DTM can only unlock the dataset if no user interface with write-access is opened.

- If a DTM locks it is dataset after instantiation and unlocks during e.g. a IDtm::PrepareToRelease(), this DTM is not suitable for use within a multi-user environment.

10.12.1 Locking for non-synchronized DTMs

This locking mechanism is suitable for all DTMs which do not implement IFdtEvents::OnParameterChanged() (IFdtEvents::OnParameterChanged() returns E_NOTIMPL). The mechanism must be implemented to support multi-user environments (see Figure 77).

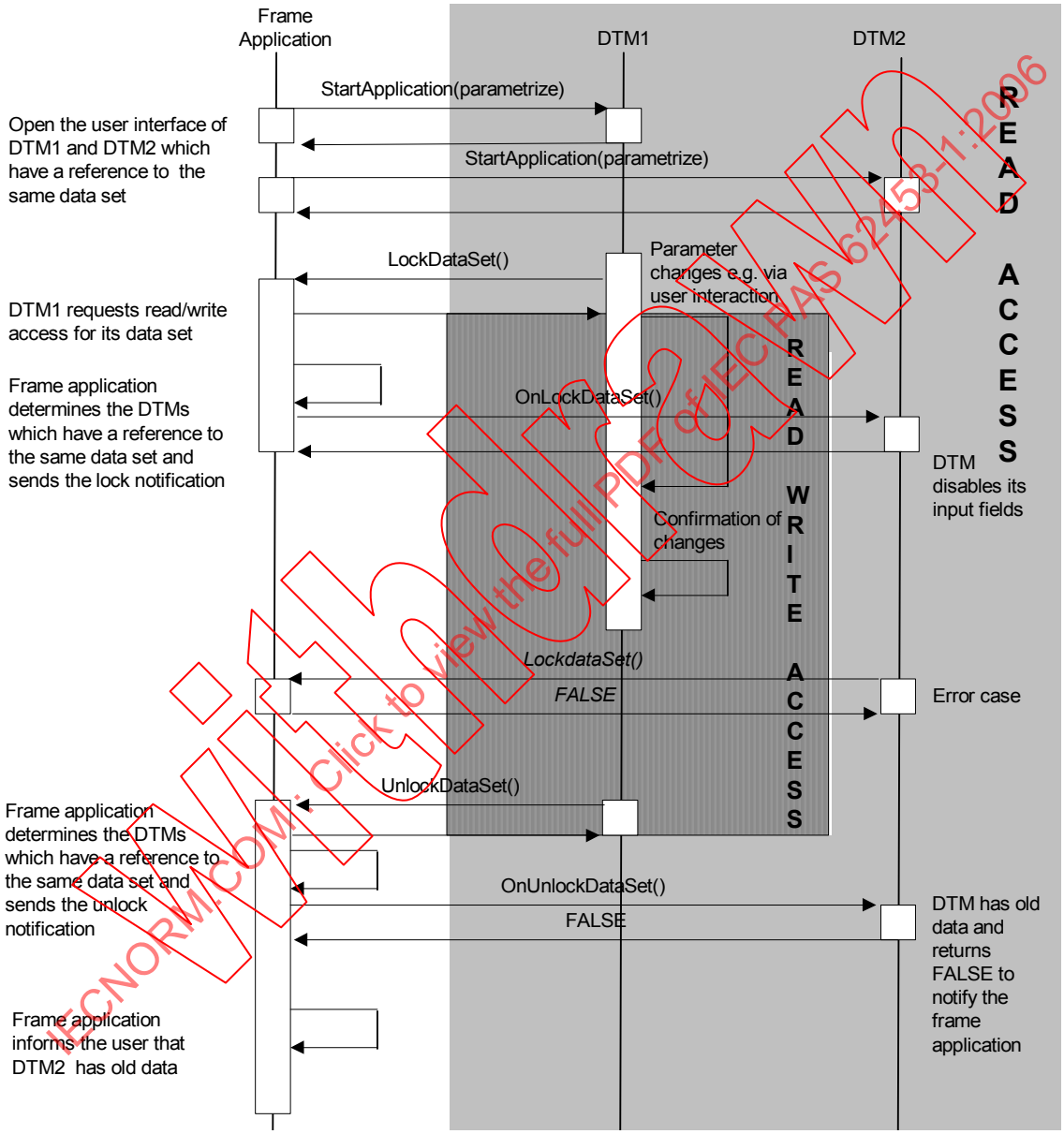

Figure 77 – Locking for non-synchronized DTMs

Used methods:

IDtmApplication::StartApplication()

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

IFdtEvents::OnLockDataSet()

IFdtEvents::OnUnlockDataSet()

10.12.2 Locking for synchronized DTMs

The synchronization of DTMs is an optional feature to provide a better handling for the user within a multi-user environment (see Figure 78).

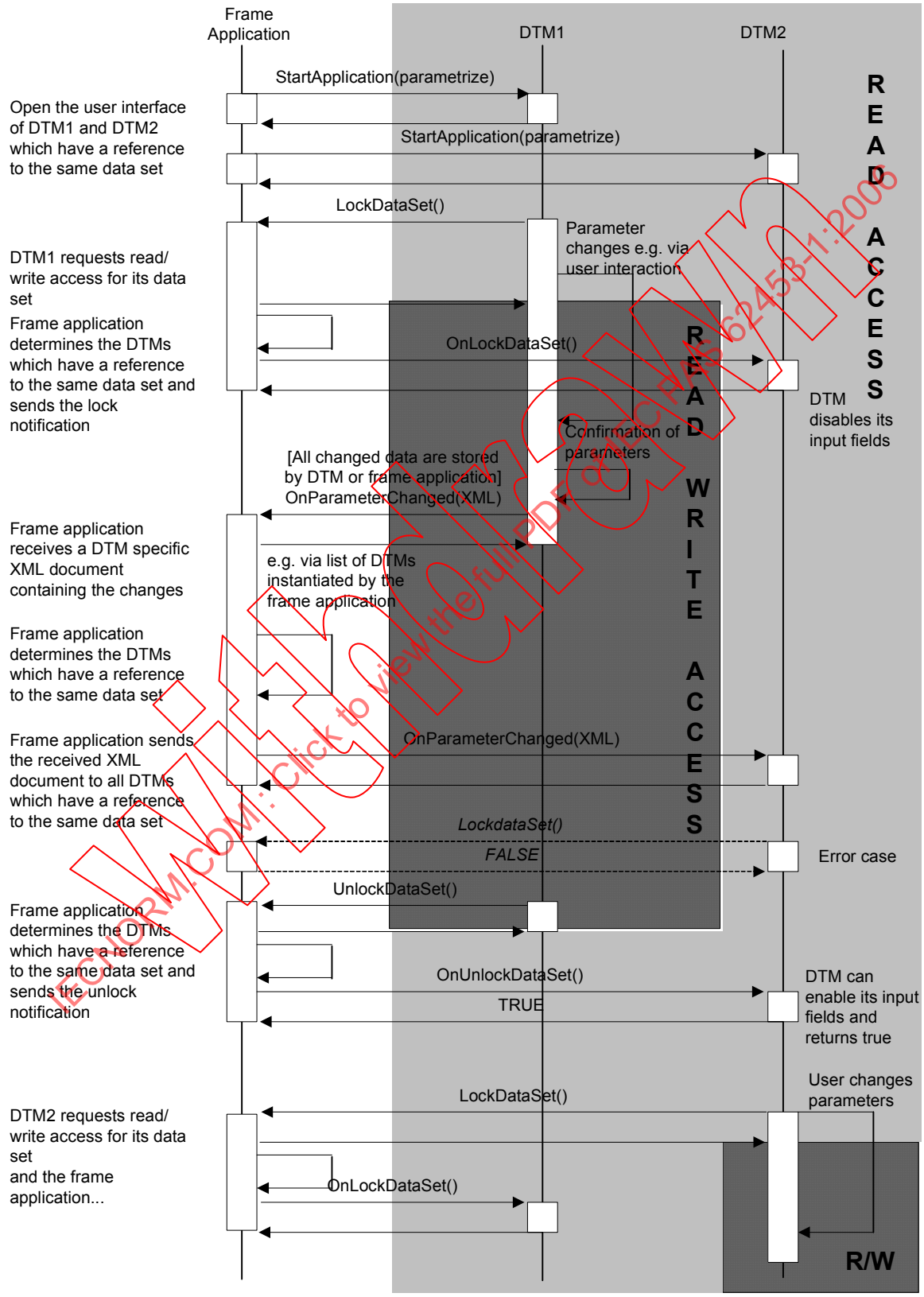

Figure 78 – Locking for synchronized DTMs

Used methods:

IDtmApplication::StartApplication()

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

IDtmEvents::OnParameterChanged()

IFdtEvents::OnParameterChanged()

IFdtEvents::OnLockDataSet()

IFdtEvents::OnUnlockDataSet()

10.13 Instantiation and release

10.13.1 Instantiation of a new DTM

After creation of a DTM business object the Frame Application must request the IPersistXXX interface pointer and call IPersistXXX::InitNew() (see Figure 79). Within this method, the DTM business object must initialize it is default device parameters.

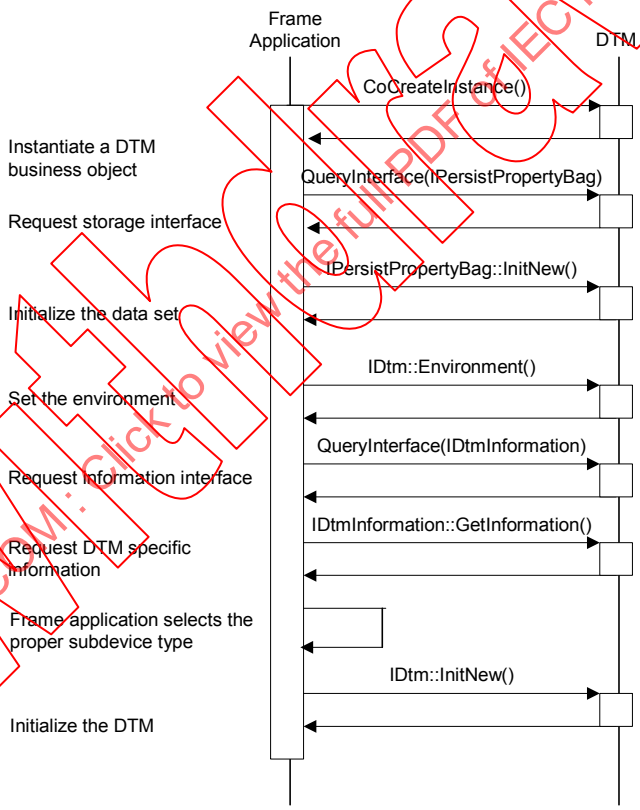

Figure 79 – Instantiation of a new DTM

Used methods:

Standard Microsoft

IDtm::Environment()

IDtm::InitNew()

IDtmInformation::GetInformation()

10.13.2 Instantiation of an existing DTM

After creation of a DTM business object for an existing instance the Frame Application must request the IPersistXXX interface pointer and call IPersistXXX::Load() with a reference to the stream object of the appropriate field device instance (see Figure 80). Within this method the DTM business object must initialize it is device parameters based on the information of the stream object.

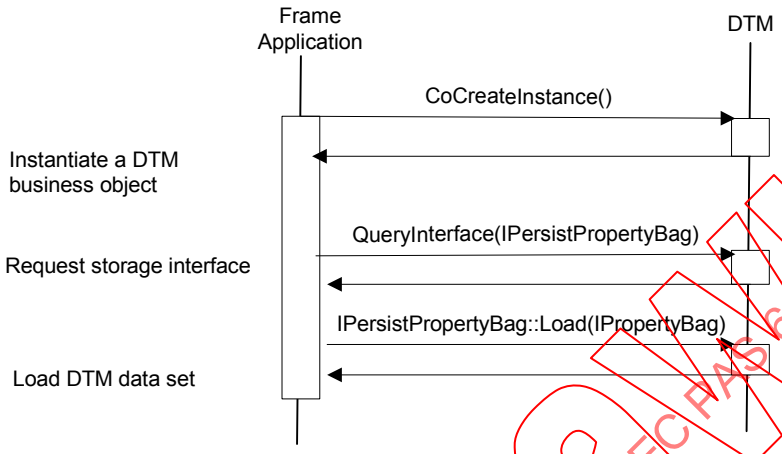

Figure 80 – Instantiation of an existing DTM

Used methods:

Standard Microsoft

10.13.3 Instantiation of a DTM ActiveX 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 (see Figure 81). Within the Init() method, the presentation object must initialize it is device parameters based on the information of the business object.

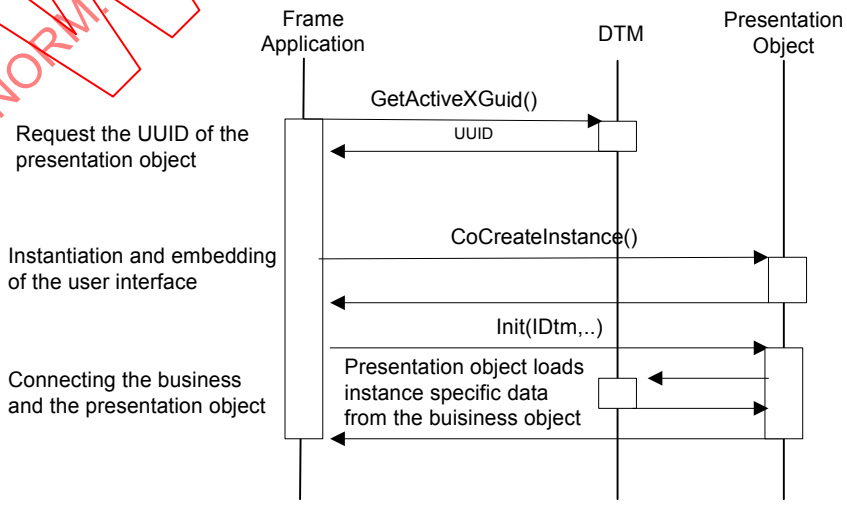

Figure 81 – Instantiation of a DTM user interface

Used methods:

Standard Microsoft

IDtmActiveXControl::Init()

IDtmActiveXInformation::GetActiveXGuid()

10.13.4 Release of a DTM user interface

If the Frame Application wants to release a presentation object of a DTM it first has to prepare the release by sending a notification to the presentation object (see Figure 82). After the IDtmActiveXControl::PrepareToRelease() method the presentation object must release all its connections to other components and can call DTM specific de-initialization methods.

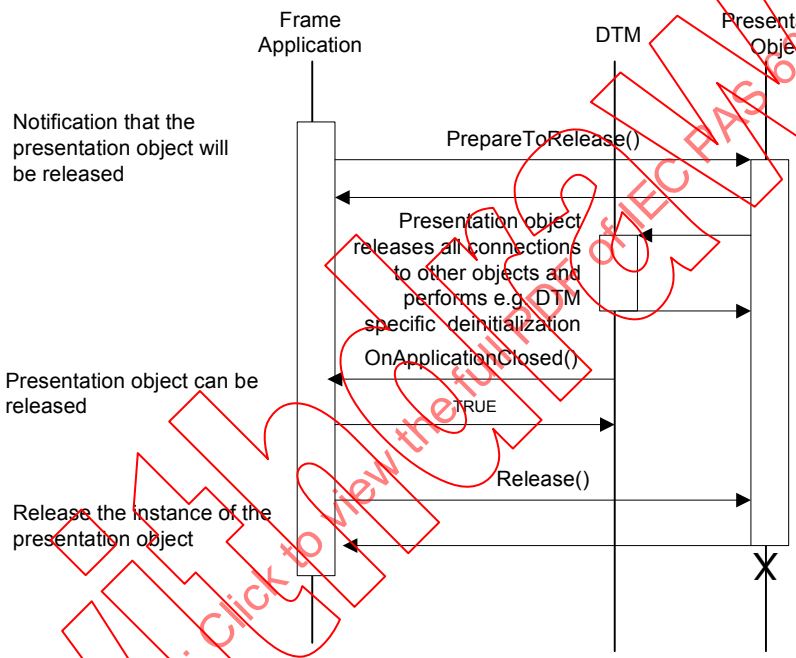

Figure 82 – Release of a DTM user interface

Used methods:

Standard Microsoft

IDtmActiveXControl::PrepareToRelease()

IDtmEvents::OnApplicationClosed()

10.14 Persistent storage of a DTM

10.14.1 State machine of instance data

10.14.1.1 Modifications

This state machine reflects the possible states of an instance data set concerning modifications (see Figure 83).

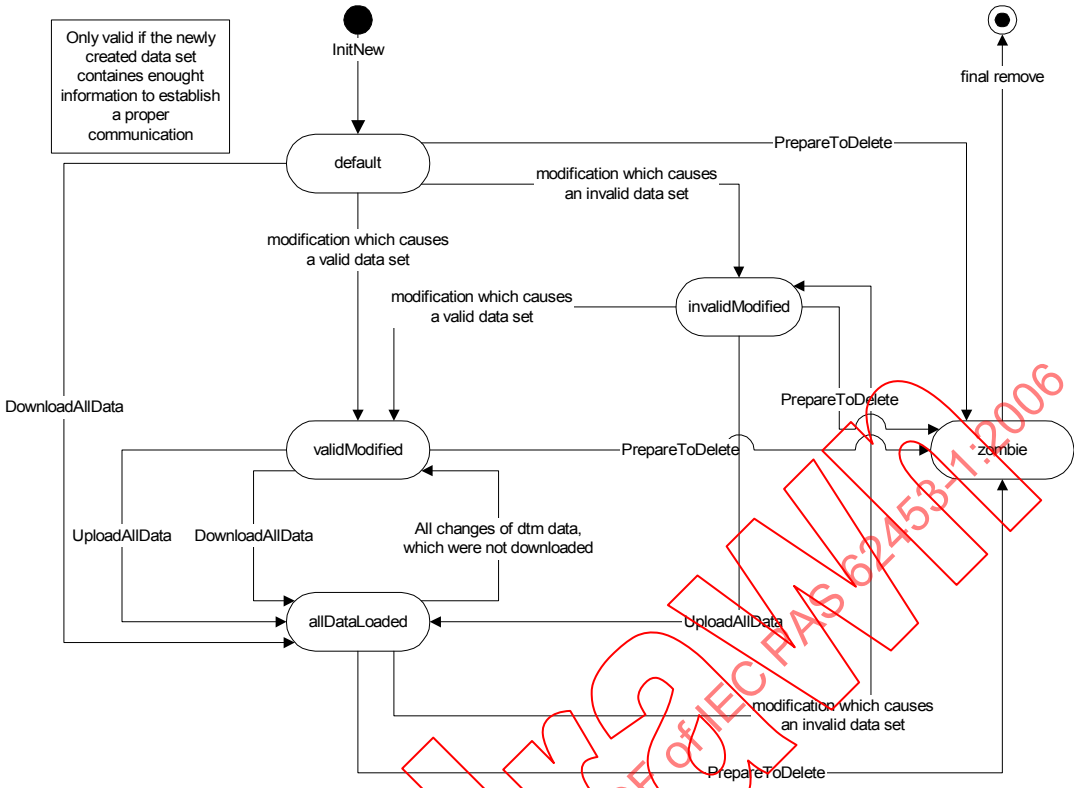

Figure 83 – State machine of instance data set

The meaning of the different states can be seen in Table 37.

Table 37 – Description of instance data set states

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 does not mean 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 cannot guarantee such a state
zombie	The instance data set is prepared to delete, no access to this data is allowed

10.14.1.2 Persistence states

This state machine reflects the possible states of an instance data set concerning the persistence of data (see Figure 84). Description of the states can be found in Table 38.

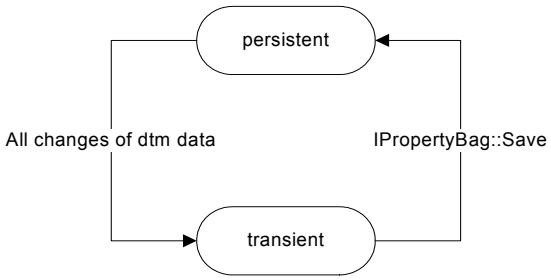

Figure 84 – Persistence states of a data set

Table 38 – Description of persistent states

State	Meaning
persistent	The content of the instance data set is persistent. This state will typically appear after the data set was saved by the Frame Application using the persistence interfaces of the DTM.
transient	The data set was modified. These changes are not saved in a persistent way. All modified instance data within the DTM is temporary and not synchronized with other DTM instances or with the Frame Application. A IFdtContainer::SaveRequest() call just informs the Frame Application that the data set should be stored.

10.14.2 Saving instance data of a DTM

To save the private data of a DTM object the Frame Application must request the IPersistXXX interface pointer and call IPersistXXX::Save() with a reference to the stream/property bag object of the appropriate field device instance (see Figure 85). Within this method the DTM business object must save all it is device parameters necessary to re-establish its complete state based on the information of the stream/property bag object within a IPersistXXX::Load() call.

After saving private data of a DTM the Frame Application is able to release the DTM business object.

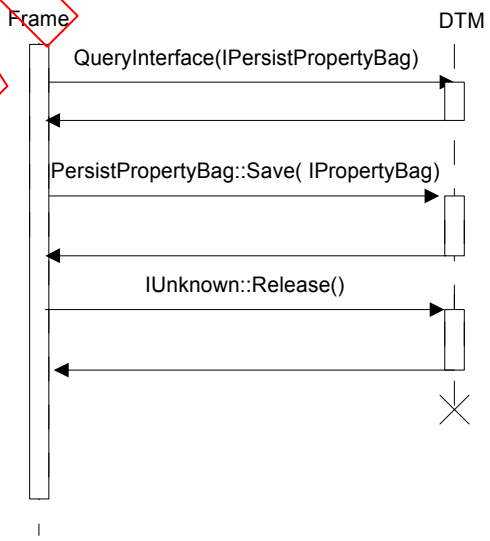

Figure 85 – Saving instance data of a DTM

Used methods:

Standard Microsoft

10.14.3 Reload of a DTM object for another instance

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

10.14.4 Copy and versioning of a DTM instance

After creation of a DTM object the Frame Application must request the IPersistXXX interface pointer and call IPersistXXX::Load() with a reference to the stream/property bag object of the appropriate field device instance (see Figure 86). Within this method the DTM business object must initialize it's device parameters based on the information of the stream object. To get a copy of the private data of a DTM business object the Frame Application must call IPersistXXX::Save() with a reference to a new stream/property bag object.

It is up to the Frame Application to handle the Frame Application specific versioning aspects and to manage the different instance data sets for a device.

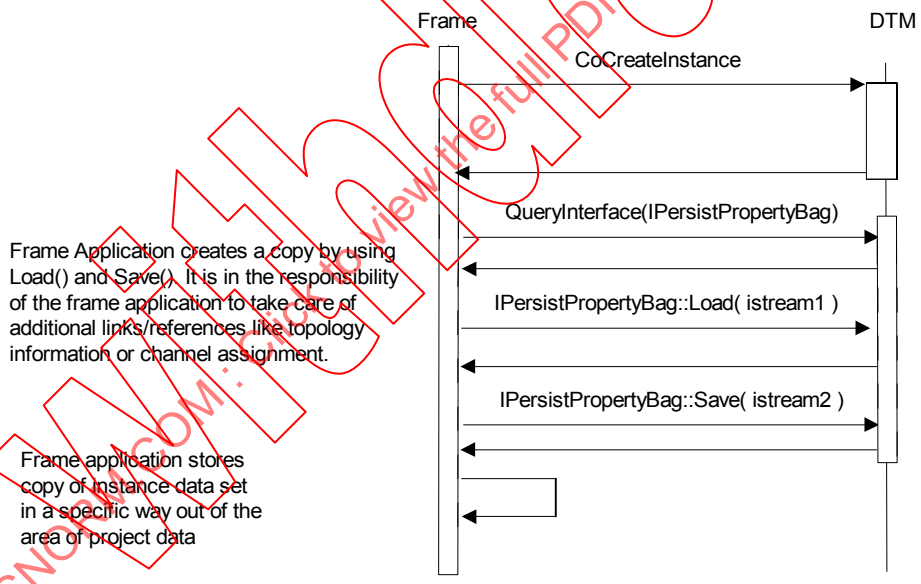

Figure 86 – Copy and versioning of a DTM instance

Used methods:

Standard Microsoft

10.15 Audit trail

Audit Trail services are implemented in the Frame Application. It stores all information about audit trail session like username, date and time, description and comment of the session and description of all audit-trail events within the session. It provides saving, analyzing a documentation of the audit trail sessions. For the DTM it offers the interface IDtmAuditTrailEvents.

When Audit Trail is started the Frame Application may ask the user for additional comments (see Figure 87).

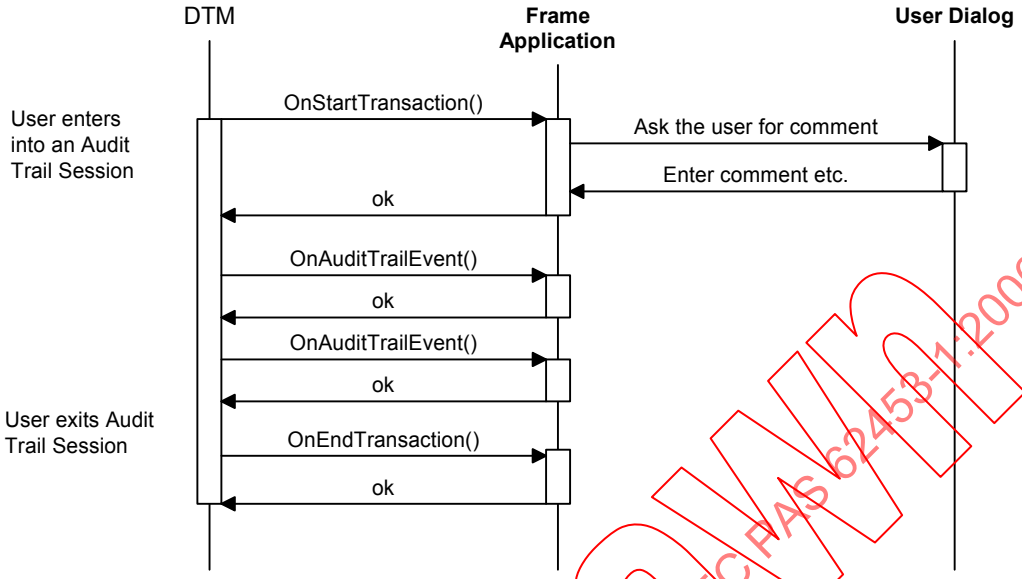

Figure 87 – Audit trail

Used methods:

IDtmAuditTrailEvents::OnStartTransaction()

IDtmAuditTrailEvents::OnAuditTrailEvent()

IDtmAuditTrailEvents::OnEndTransaction()

10.16 Comparison of two instance data sets

This subclause 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.

10.16.1 Comparison without user interface

The sequence chart for the comparison of two instance data sets is shown in Figure 88.

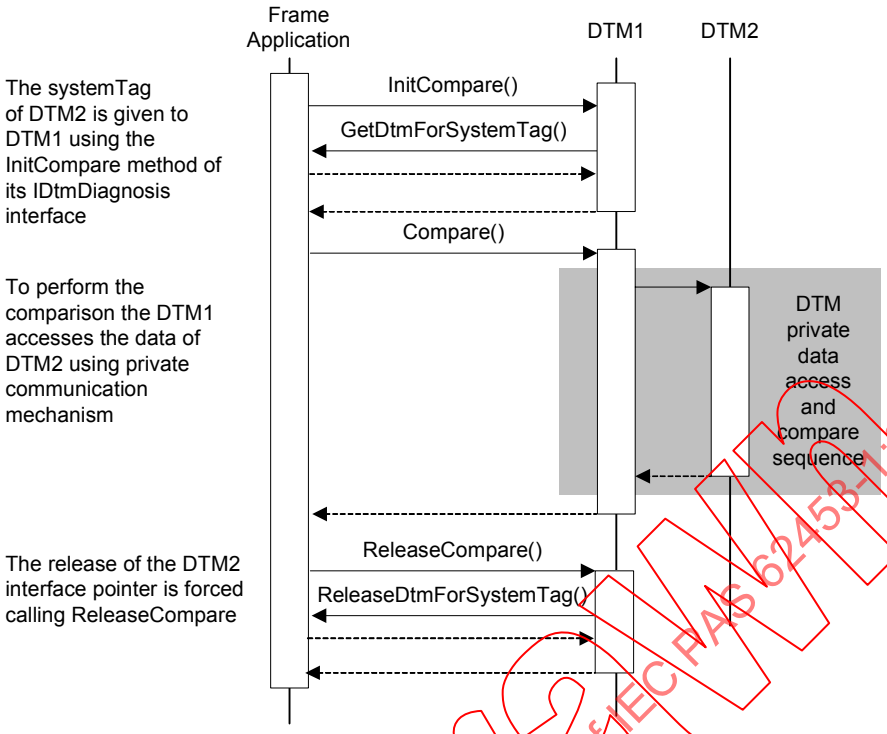

Figure 88 – Comparison without user interface

Used methods:

IDtmDiagnosis::InitCompare()

IDtmDiagnosis::Compare()

IFdtTopology::GetDtmForSystemTag()

IFdtTopology::ReleaseDtmForSystemTag()

10.16.2 Comparison with user interface

To invoke a user interface for comparison, the following sequence is to be performed (see Figure 89).

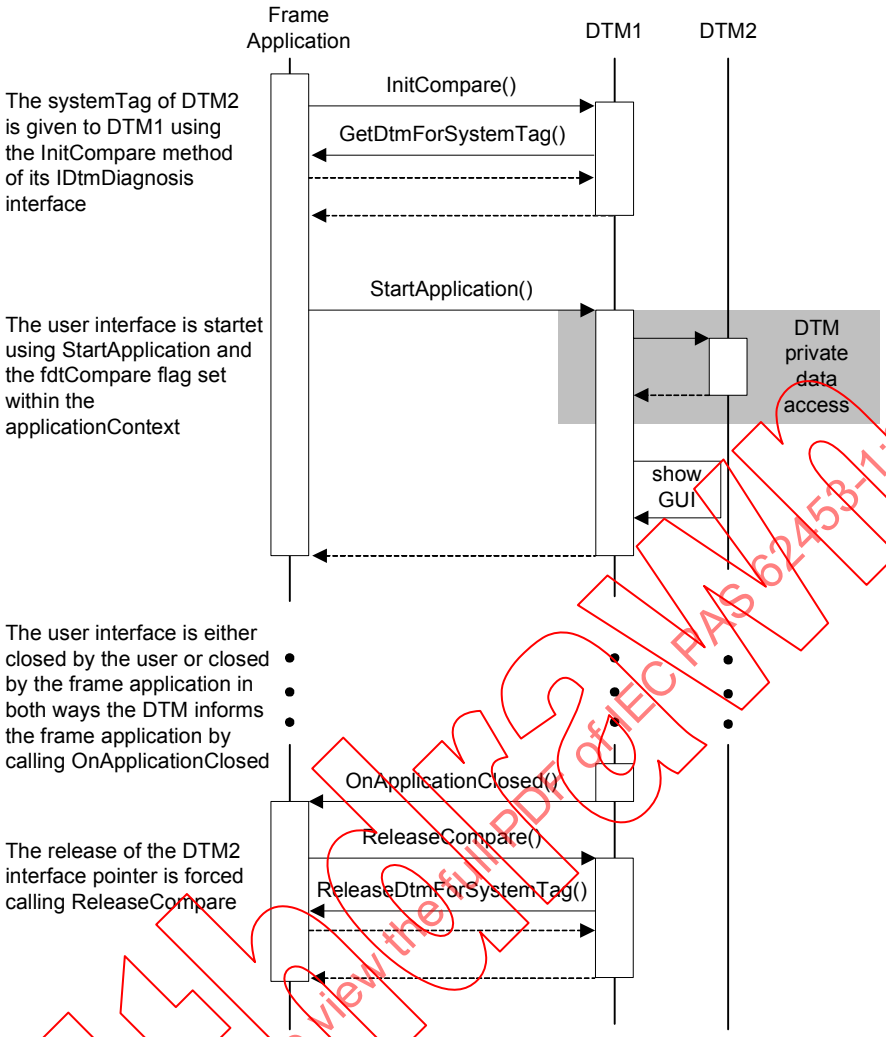

Figure 89 – Comparison with user interface

Equivalent to StartApplication() (as shown in the sequence chart) ActiveX controls can be used.

Used methods:

IDtmDiagnosis::InitCompare()

IDtmDiagnosis::ReleaseCompare()

IDtmApplication::StartApplication()

IDtmEvents::OnApplicationClosed()

10.17 Failsafe data access

The sequence chart (see Figure 90) shows how to access failsafe data from a device and its associated function block via two independent communication links.

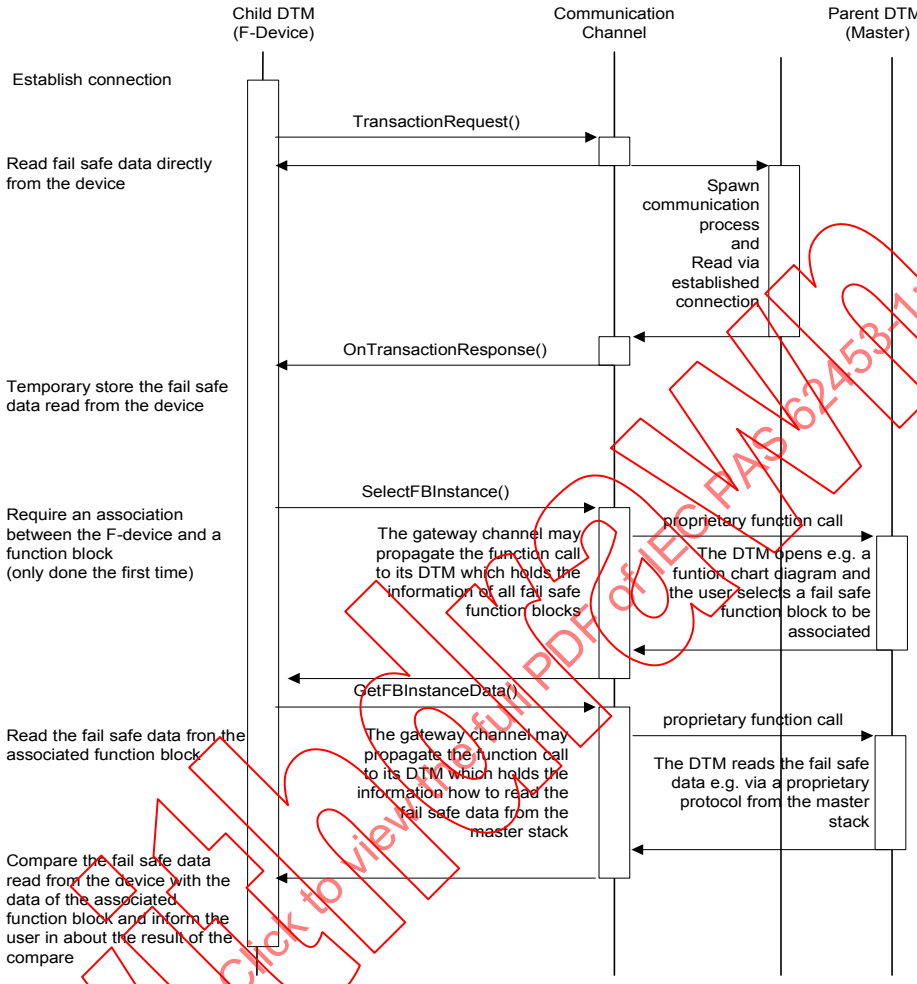

Figure 90 − Failsafe data access

Used methods:

IFdtCommunication::TransActionRequest()

IFdtCommunicationEvents::OnTransactionResponse()

IFdtFunctionBlockData::SelectFBInstance()

IFdtFunctionBlockData::GetFBInstanceData()

10.18 Set or modify device address with user interface

In this scenario the Frame Application requests a set of specific child device addresses at bus master DTMs. This sequence (see Figure 91) for example is started when a new DTM is added to the topology. Similar sequences can be used if a Frame Application offers manual change of address in context of a DTM.

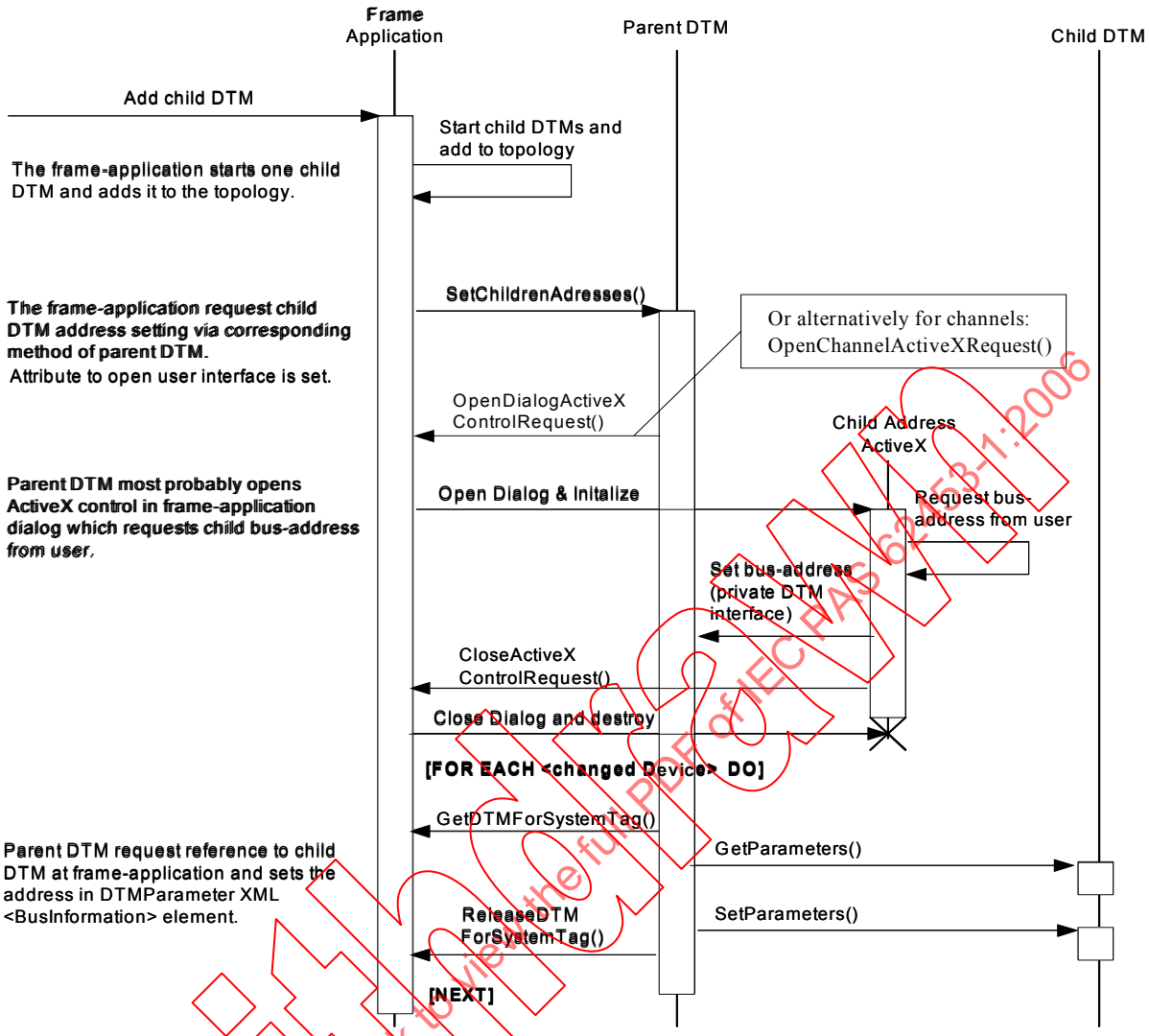

Figure 91 – Set or modify device address with user interface

Used methods:

IFdtChannelSubTopology2::SetChildrenAddresses()

IFdtActiveX2:OpenDialogActiveXControlRequest()

IFdtActiveX::CloseActiveXControlRequest()

IFdtTopology::GetDtmForSystemTag()

IFdtTopology::ReleaseDtmForSystemTag()

IDtmParameter::SetParameters()

IDtmParameter::GetParameters()

10.19 Set or modify known device addresses without user interface

In this scenario, the Frame Application requests set of device addresses at DTM, for example after a scanning or import operation (see Figure 92).

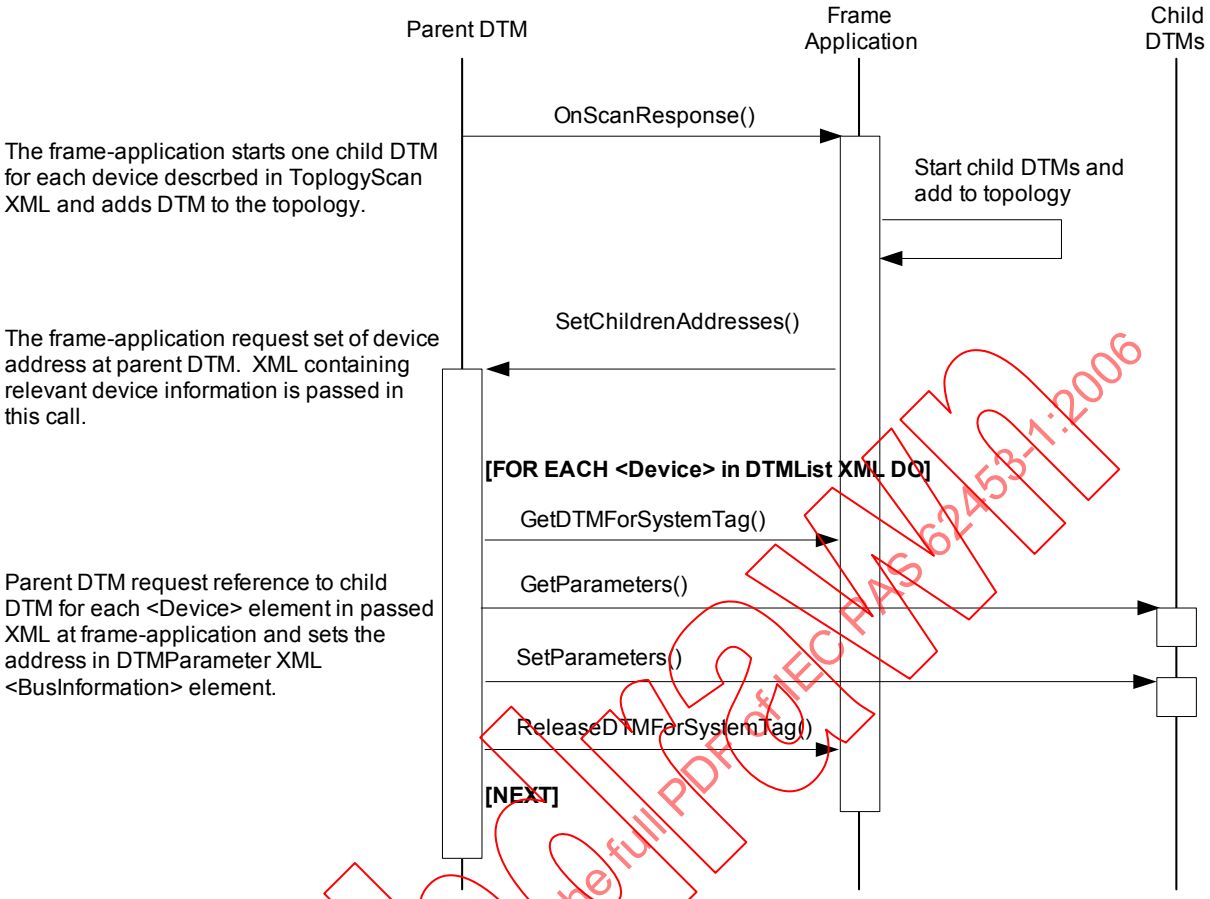

Figure 92 – Set or modify known device addresses without user interface

Used methods:

IFdtChannelSubTopology2::SetChildrenAddresses()

IDtmEvents::OnScanResponse()

IFdtTopology::GetDtmForSystemTag()

IFdtTopology::ReleaseDtmForSystemTag()

IDtmParameter::SetParameters()

IDtmParameter::GetParameters()

10.20 Display or modify all child device addresses with user interface

In this scenario Frame Application requests display or modification of all child device addresses at a Parent DTM. This sequence (see Figure 93) for example is started when a user selects the corresponding menu entry in context of a Communication DTM or a Gateway-DTM.

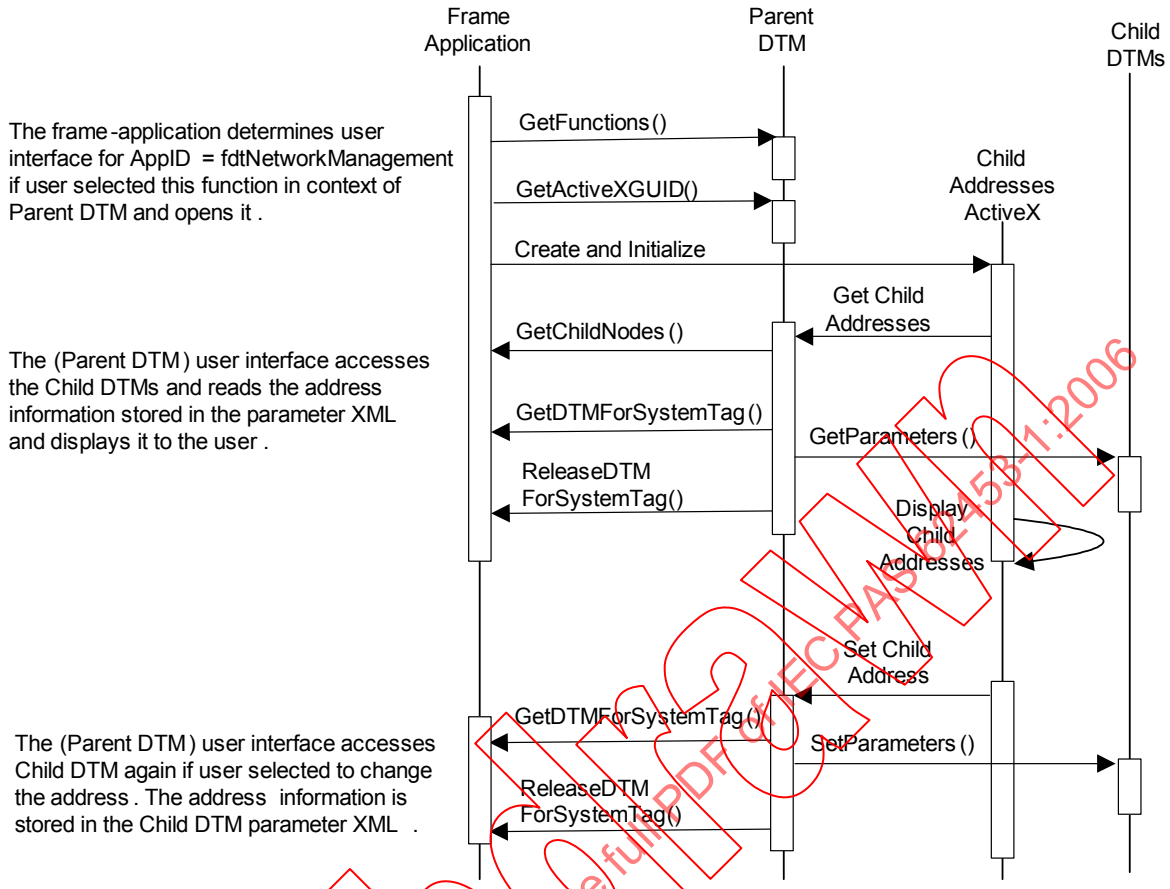

Figure 93 – Display or modify all child device addresses with user interface

Used methods:

IDtm::GetFunctions()

IDtmActiveXInformation::GetActiveXGuid()

IFdtTopology::GetChildNodes()

IFdtTopology::GetDtmForSystemTag()

IFdtTopology::ReleaseDtmForSystemTag()

IDtmParameter::SetParameters()

IDtmParameter::GetParameters()

10.21 Device initiated data transfer

Some protocols support data transfer services that are initiated by the device and not by the DTM. In order to support such services, the following approach is recommended (see Figure 94):

- the infrastructure (filter, service queue) for such services is initiated by a protocol-specific transaction request of the DTM; depending on the protocol this service may be acknowledged or not;
- the device initiated data transfer is transported by transaction responses to the initiating request (they carry the same invokeId);
- the infrastructure for these services is terminated by a protocol-specific transaction request of the DTM, which may carry the original invokeId as an attribute of the request XML document.

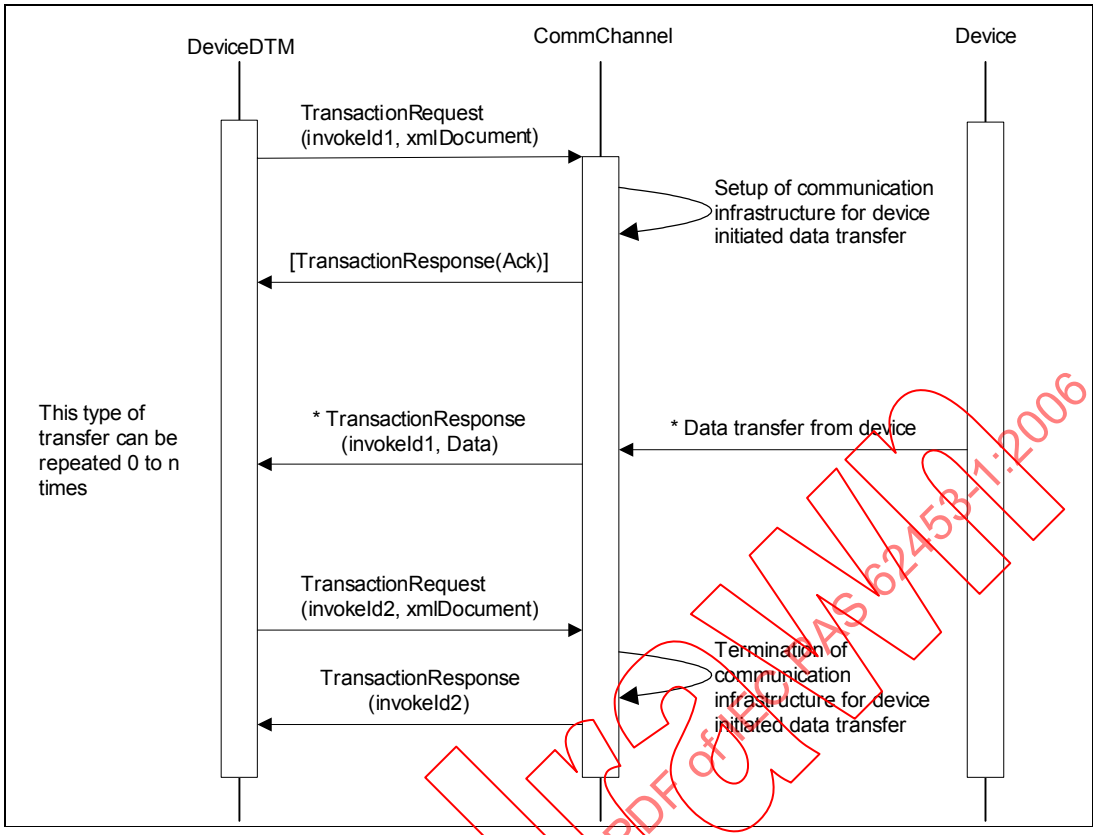

Figure 94 – Device initiated data transfer

Device initiated data transfer needs management by the communication infrastructure. That is why it is necessary to identify unambiguously the request to initialize and terminate this type of behavior.

The TransactionRequests to initiated or terminate device initiated data transfer will transport protocol specific XML-Documents.

The terminating TransactionRequest will contain the information necessary to terminate the device initiated data transfer. This may include the invokeId of the initiating TransactionRequest.

If a DTM initializes such services, it is required to terminate these services. The termination has to occur before the DTM can go offline (DisconnectRequest / Abort). If the connection is terminated by IFdtCommunicationEvents::OnAbort(), this service is terminated automatically.

The support of such services is protocol specific and device specific. If a communication channel is not able to support this type of service, it will return a protocol specific error document in the TransactionResponse, indicating that it is not able to support this feature.

10.22 Starting and releasing DTM user interface in modal dialog

DTM requests start of user interface over IFdtActiveX2 interface implemented by Frame Application (see Figure 95). This starts modal dialog and opens DTM user interface in it. DTM user interface exchanges data with DTM over private interface and request closing of dialog after work is finished. Frame Application closes the dialog and DTM user interface. Now IFdtActiveX2:OpenDialogActiveXControlRequest() call returns to the DTM.

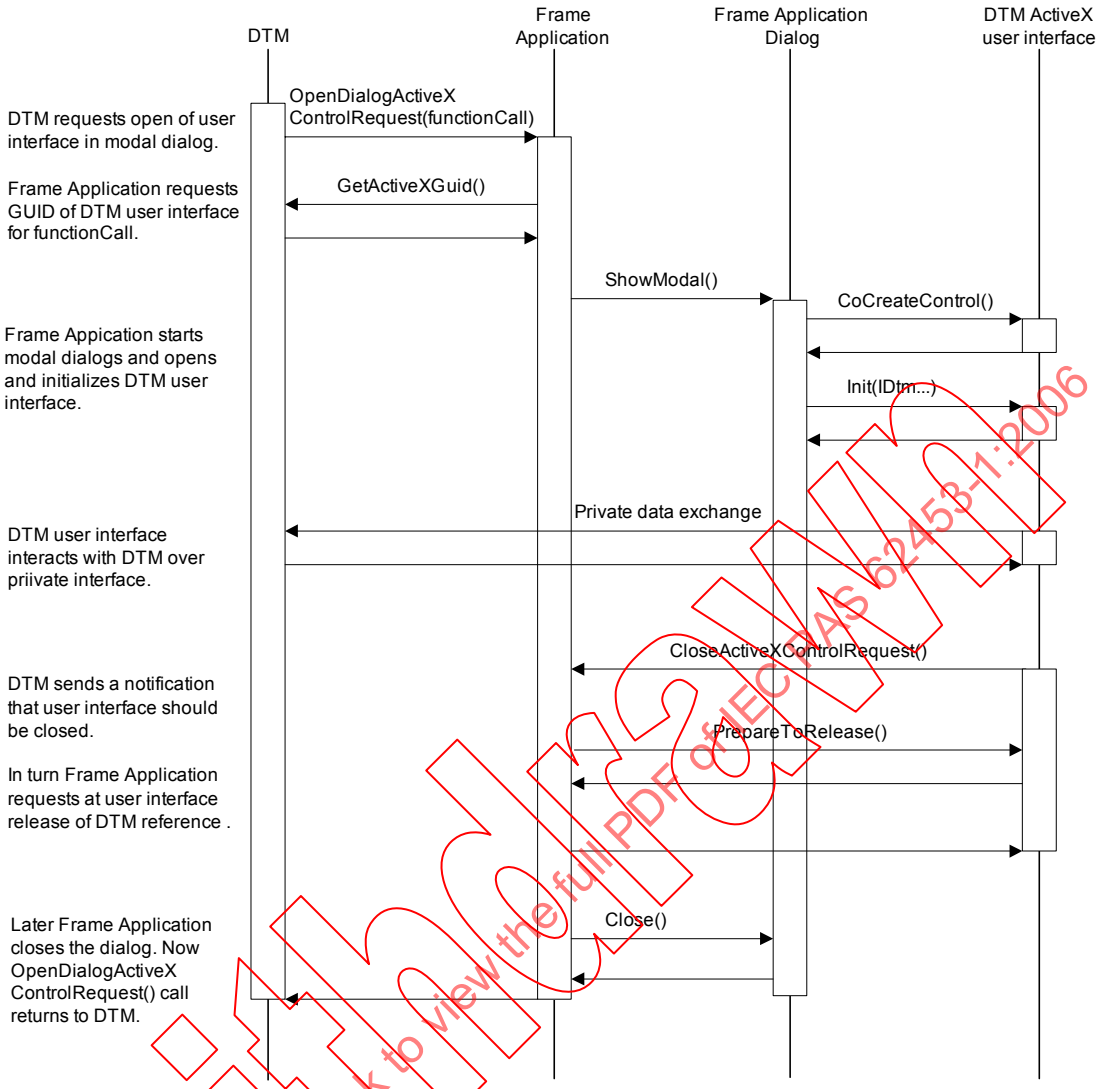

Figure 95 – Modal DTM user interface

Used methods:

Standard Microsoft

IDtmActiveXInformation::GetActiveXGuid()

IFdtActiveX2:OpenDialogActiveXControlRequest()

IFdtActiveX:CloseActiveXControlRequest()

IDtmActiveXControl:Init()

IDtmActiveXControl:PerpareToRelease()

10.23 Parent component handling redundant slave

A parent component, e.g. a Communication-DTM, handling a redundant fieldbus is able to detect a Device-DTM able to handle a redundant slave within execution of IFdtChannelSubTopology::OnAddChild() by examining the parameter document of this DTM (see Figure 96). The Communication-DTM selects the redundant communication paths (either automatically or using a dialog) and sets redundancy information to the Device-DTM by calling SetParameter().

Device-DTM provides this redundancy information within it is connect request to the Communication-DTM. For each such opened connection the Communication-DTM is able to use one of the available communication paths without need for further interaction with the Device-DTM.

A Frame Application implementing the IDtmRedundancyEvents interface is able to get topology information about redundant DTMs. In such a Frame Application the topology view may show this redundancy information. On the other hand Communication-DTM and Device-DTM of a redundant slave can be used in a Frame Application without knowledge about redundancy, because all redundancy functionality is handled by the Communication-DTM and it's child DTMs.

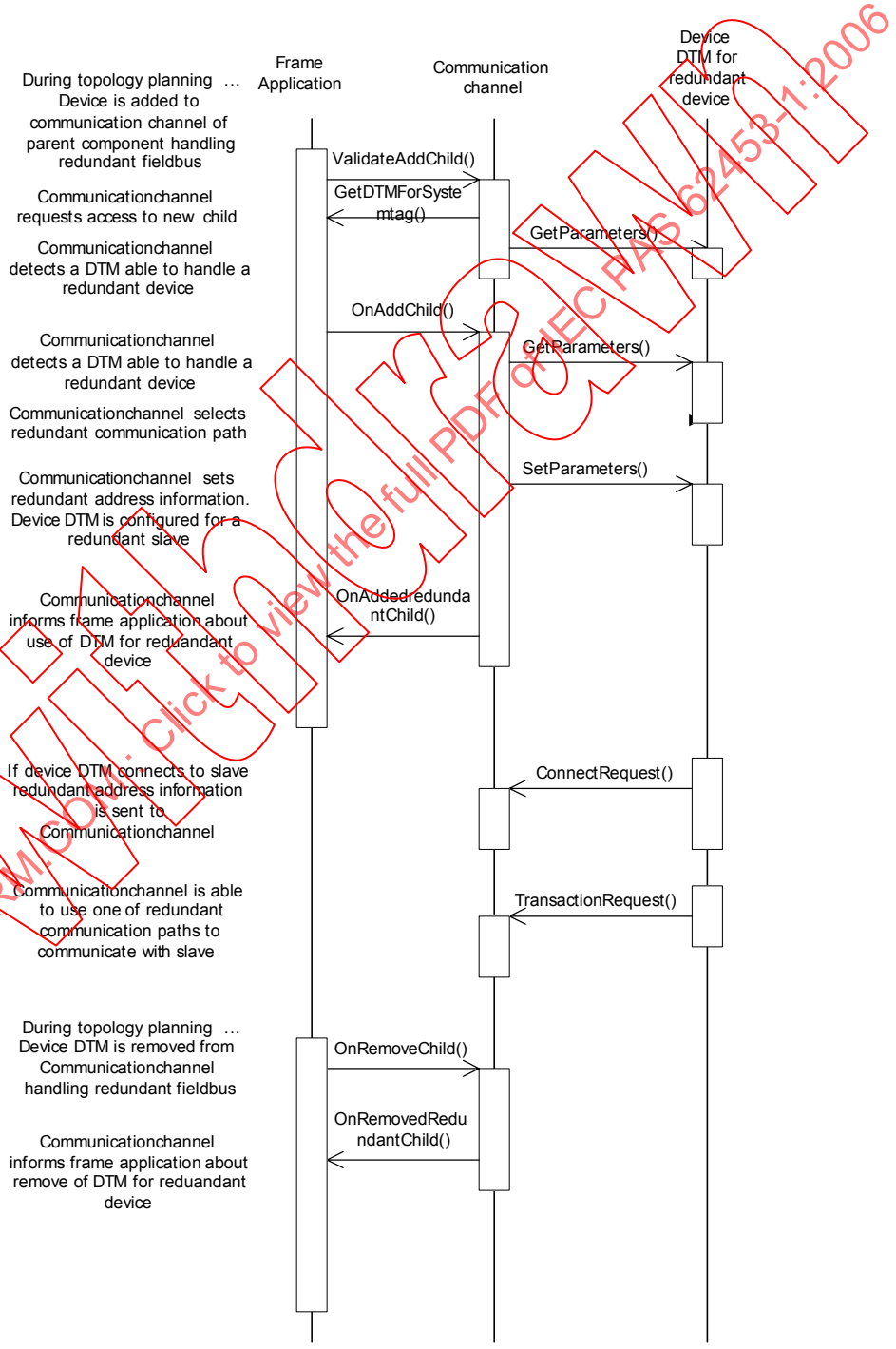

Figure 96 − Handling of a redundant slave

Used methods:

IDtmRedundancy::OnAddedRedundantChild()

IDtmRedundancyEvents::OnRemovedRedundantChild()

10.24 Initialization of a channel ActiveX control

To initialize a channel ActiveX control conformant to FDT this PAS, the Frame Application has first to check, if the interface IFdtChannelActiveXControl2 exists.

If it exists the Frame Application has to call the IFdtChannelActiveXControl2::Init2() function at this interface.

If it does not exist, the Frame Application calls the IFdtChannelActiveXControl::Init() function.

10.24.1 Supports IFdtChannelActiveXcontrol2

This diagram (see Figure 97) describes the sequence of the channel ActiveX control initialization for the case that the channel ActiveX control provides the IFdtChannelActiveXControl2 interface.

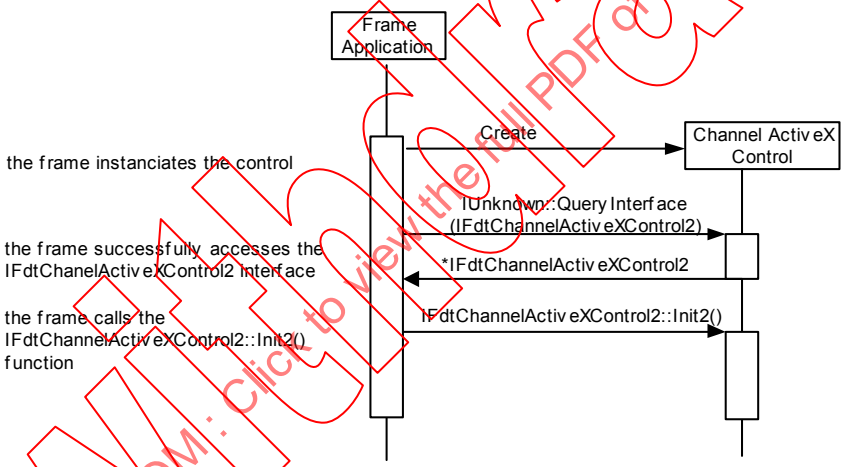

Figure 97 – Init of channel ActiveX with IFdtChannelActiveXControl2

Used methods:

IUnkown::QueryInterface

IFdtChannelActiveXControl2::Init2()

10.24.2 Does not support IFdtChannelActiveXControl2

This diagram (Figure 98) describes the sequence of the channel ActiveX control initialization for the case that the channel ActiveX control does not provide the IFdtChannelActiveXControl2 interface.

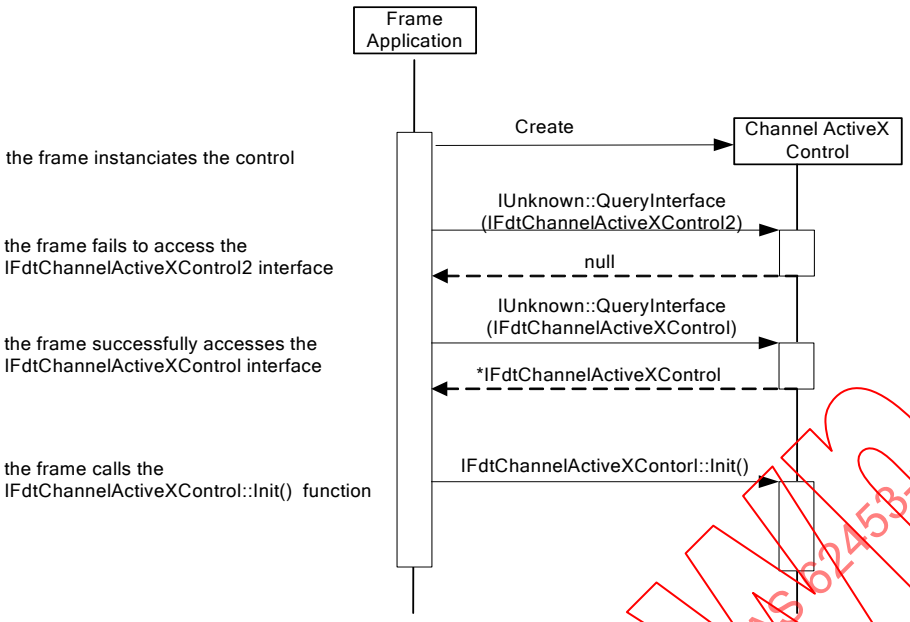

Figure 98 − Init of channel ActiveX without IFdtChannelActiveXControl2

Used methods:

IUnkown::QueryInterface

IFdtChannelActiveXControl::Init()

10.25 DTM upgrade

The first step to be resolved during upgrade is to verify if the saved data set can be associated with the new DTM.

Each DTM should expose an unique identifier (UUID) specifying its data set format. A DTM can provide a list of compatible data set formats it can load. These UUIDs are returned as part of IDtmInformation::GetInformation() method call.

If the CLSID and the ProgID of the DTM is not changed it is up to the DTM to handle version upgrade. The list of supported data set formats may not be considered by the Frame Application when data is loading in this case. Additional check for data compatibility must be done by the DTM when the data is loaded.

10.25.1 Saving data from a DTM to be upgraded

The first step is to store the information from the DTM that is for upgrade.

The Frame Application needs to store additional information about the DTM:

- CLSID of the DTM
- UUID of the stored data format
- Storage Type
- Additional information about the device.

This information is associated to the stored data and can be used later by the Frame Application to identify and manage data set.

The following diagram provides an illustration of that sequence (see Figure 99).

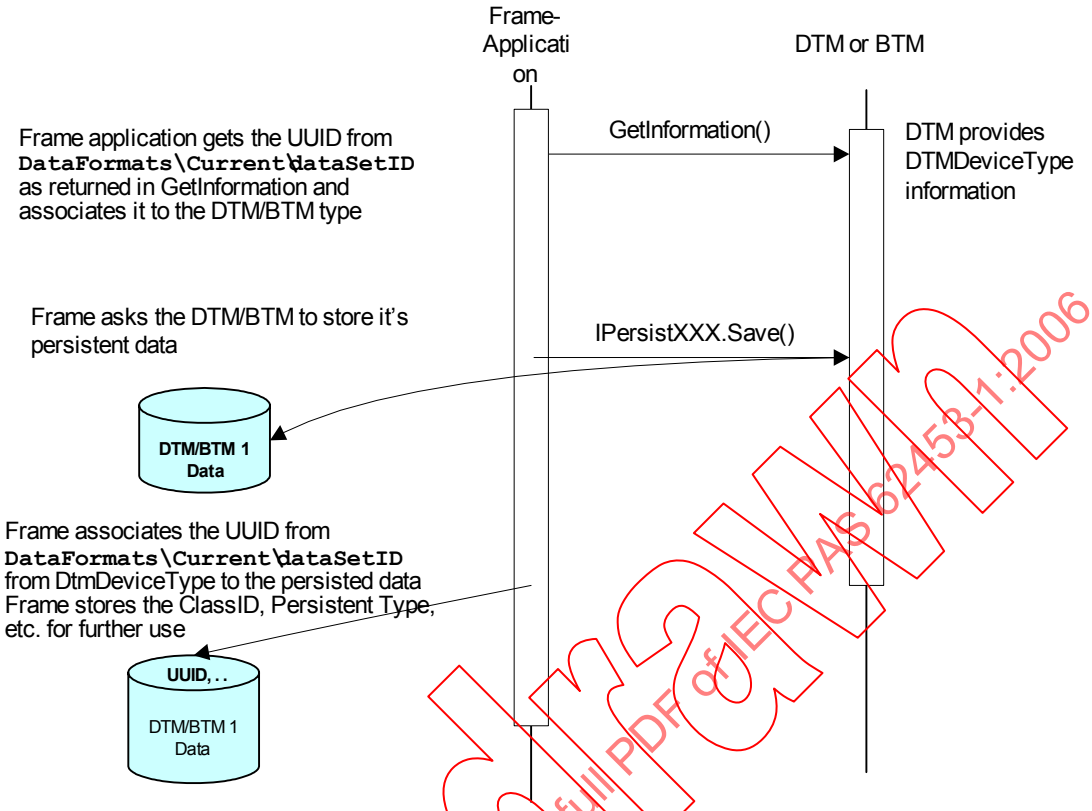

Figure 99 – Saving data from a DTM to be upgraded

10.25.2 Loading data in the replacement DTM

After the Frame Application created an new DTM, the newly created DTM loads the data set of the replaced DTM (see Figure 100).

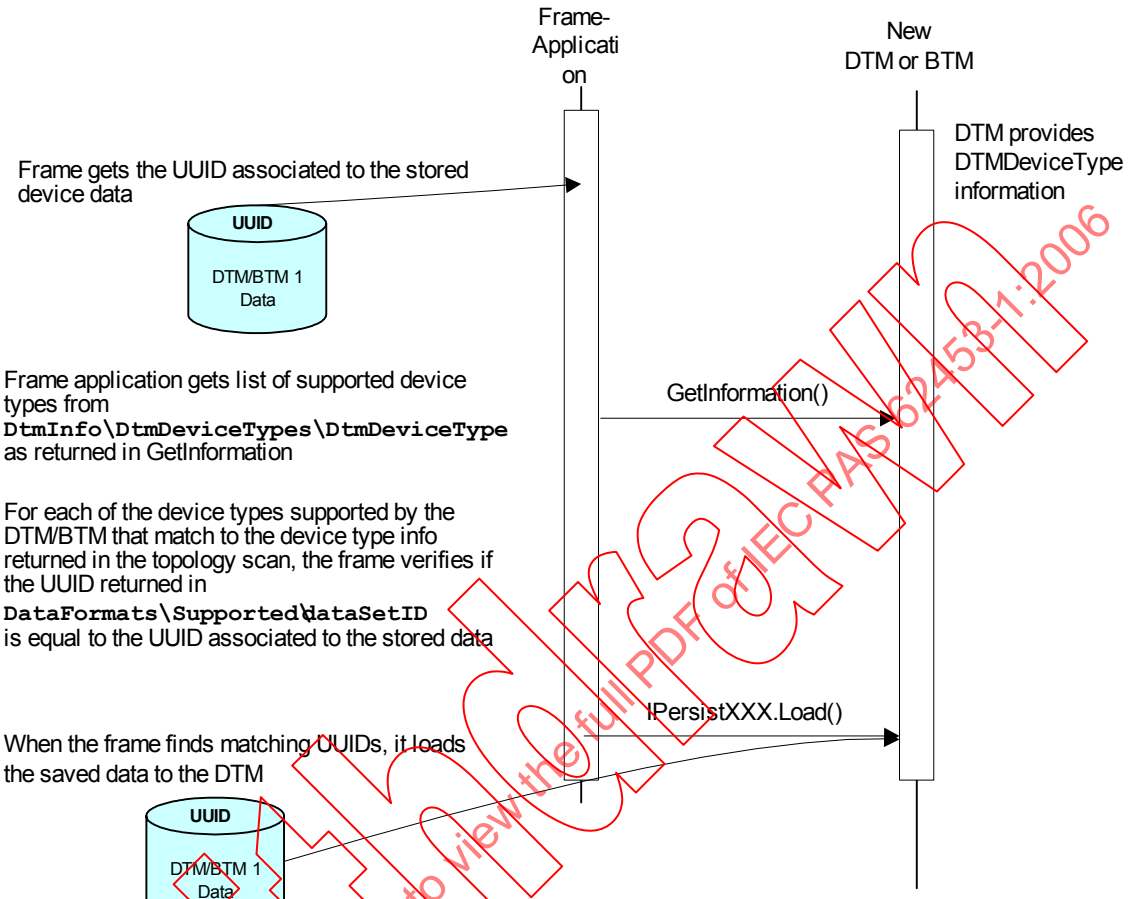

Figure 100 – Loading data in the replacement DTM

10.26 Usage of IDtmSingleDeviceDataAccess::ReadRequest/Write Request

This sequence chart (see Figure 101) gives an example regarding handling of ReadRequest() and WriteRequest().

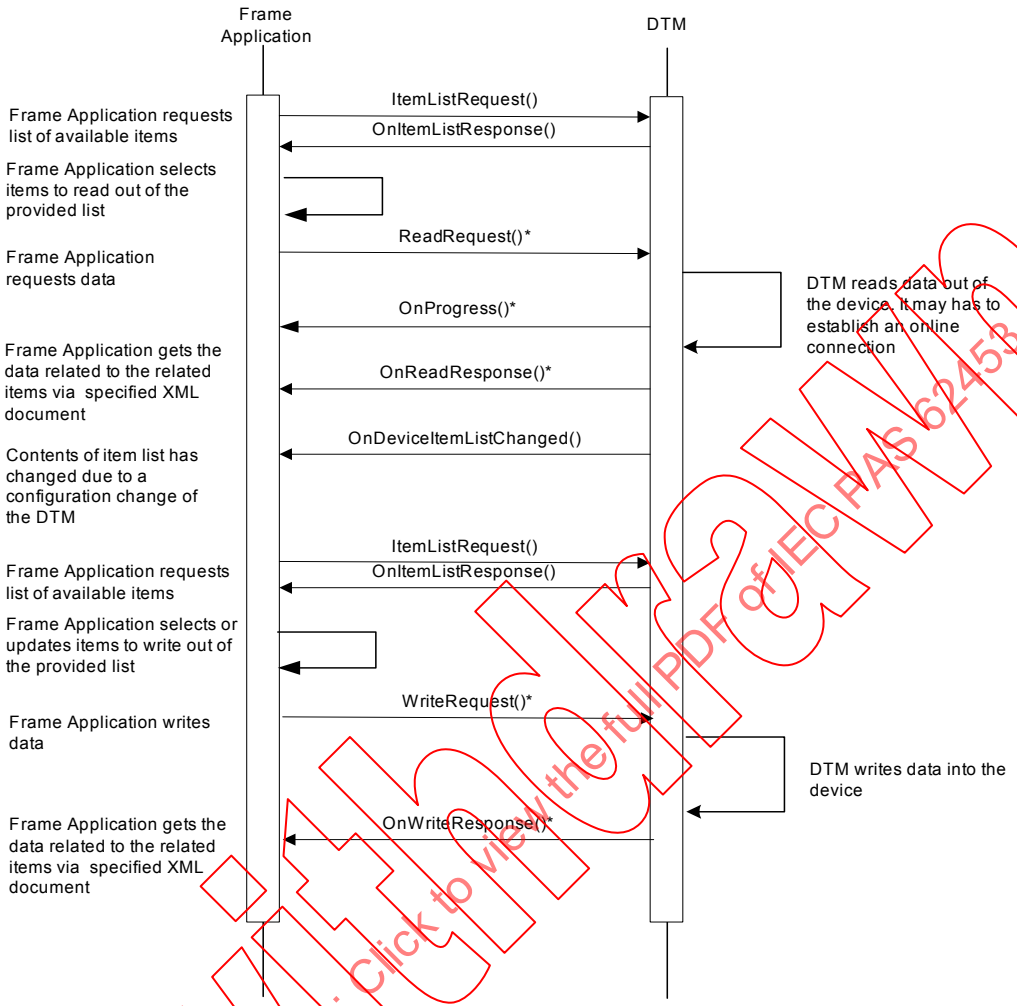

Figure 101 – Usage of IDtmSingleDeviceDataAccess

Used methods:

IDtmSingleDeviceDataAccess::ItemListRequest()

IDtmSingleDeviceDataAccessEvents::OnItemListResponse()

IDtmSingleDeviceDataAccess::ReadRequest()

IDtmSingleDeviceDataAccess::WriteRequest()

IDtmSingleDeviceDataAccessEvents::OnReadResponse()

IDtmSingleDeviceDataAccessEvents::OnWriteResponse()

IDtmEvents::OnProgress()

10.27 Instantiation of DTM and BTM

A BTM is created by the same mechanism as a DTM, which means the Frame Application always creates a BTM. If a DTM must create a BTM, it has to use the interface IFdtTopology of the Frame Application. BTMs are instantiated by:

- Frame Application according to the defined sequence defined;
- DTM triggers as described below.

The verification of assigned Child-BTMs is done by using the ValidateAddChild() method of the IFdtChannelSubTopology interface.

The general sequence is shown in the following chart (see Figure 102). The creation of BTMs is possible in any of the following states:

- Running
- Configured
- Communication set
- Going Online
- Going Offline
- Online.

The trigger for creating BTMs (shown in the chart by the event “Trigger to Initiate BTMs”) can originate from the following sources:

- The DTM GUI (e.g. the GUI of the running DTM provides a method “Add block”)
- A function exposed by the DTM (that is, a function without GUI)
- An event on the DTM (e.g., transition from state “Running” to state “Configured”)

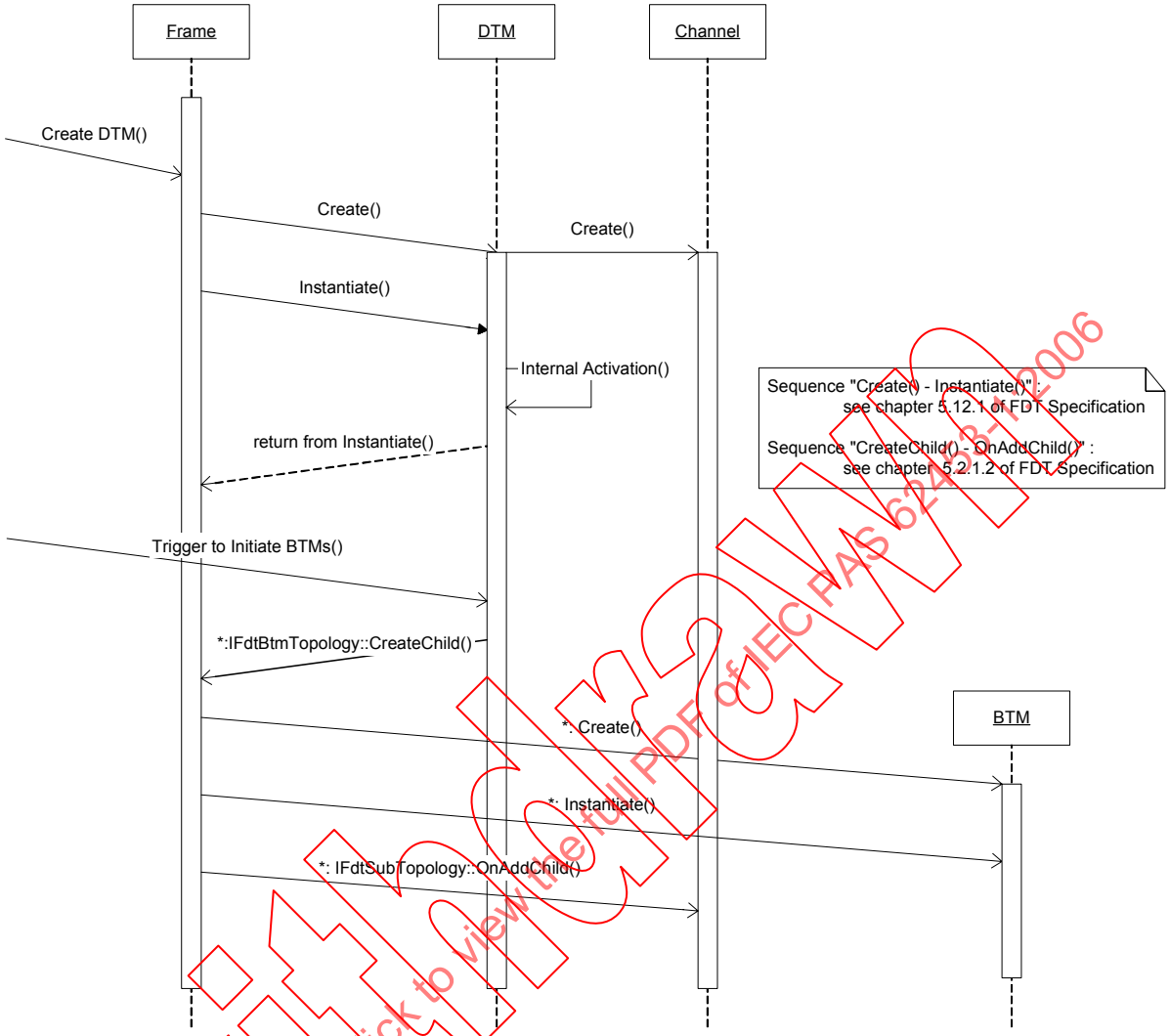

Figure 102 – General sequence of creation and instantiation of blocks

The Frame Application can reject the CreateChild method. In this case, the procedure of creation of the BTM is aborted. It is up to the user to create the BTM.

It does not matter how to trigger BTMs initiation. The general sequence of events does not change. Thus, each DTM and each BTM is handled according to the DTM State Machine.

If the DTM is loaded (transition from Up to Existing Created to Running states), it shall not automatically trigger the BTM creation. The Frame Application should handle the instantiation of the BTMs.

11 Installation issues

11.1 Registry and device information

11.1.1 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 than be detected by a Frame Application or a configuration tool using the Microsoft standard component category manager.

The BTM to DTM assignment follows the same model as the assignment of module Device-DTM to DTM. Vendors are encouraged to define a unique CATID for the protocol between DTM and BTM in order to ensure correct block assignment. The same CATID can be used in different devices if the same BTMs are used.

11.1.2 Component categories

FDT defines the following component categories, see Table 39.

Table 39 – Component categories

CATID description in the registry	SYMBOLIC NAME OF THE CATID	UUID of the CATID	Description
“FDT DTM”	CATID_FDT_DTM	{036D1490-387B-11D4-86E1-00E0987270B9}	Object compatible to FDT major version 1 providing class information via 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 BTM”	CATID_FDT_BTM	{036D1690-387B-11D4-86E1-00E0987270B9}	Object represents a block

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

The following Table 40 shows the valid combination of category ids.

Table 40 – Combinations of categories

SYMBOLIC NAME OF THE CATID	CATID_FDT_DTM	CATID_FDT_DEVICE	CATID_FDT_MODULE
CATID_FDT_DTM		√	√
CATID_FDT_DEVICE			
CATID_FDT_MODULE	√		
CATID_FDT_BTM	√		

For example, class objects must register for categories according to the following list, see Table 41.

Table 41 – Example for DTM registration

Description	Categories
DTM for a device	CATID_FDT_DTM CATID_FDT_DEVICE
DTM for a module of a modular device	CATID_FDT_DTM CATID_FDT_MODULE

It is expected that a DTM will first create any categories it uses and then registers for those categories during installation. Deregistering a server should cause it to be removed from that category but not to deregister the category by itself. See the ICatRegister documentation for additional information.

A DTM may register additional component categories according to COM rules.

11.1.3 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.

11.1.4 Installation issues

It is assumed that the DTM vendor will provide a SETUP.EXE to install the needed components for his DTM. This will not be discussed further. Other than the actual components, the main issue affecting COM software is management of the Windows Registry and Component Categories.

All FDT components must have a version information resource containing at least a version number, so that an installation tool can decide whether it can overwrite an installed component or not.

Furthermore a Frame Application is responsible to install all FDT related XML schemas. A DTM has to use these documents provided by the Frame Application via IFdtContainer::GetXMLSchemaPath().

DTM specific schemas within FDT related XML documents must be declared as an inline definition. During the de-installation of FDT related components, the procedure has to take care about the availability of the FDT related interface description To avoid problems the usage of Microsoft Installer Technology is mandatory. This means all Merge Modules provided by FDT-JIG must be used. E.g. it is not allowed to copy the FDT100.dll directly on the PC. Furthermore all common DLLs (e.g. for usage of ATL, VB runtime or third party controls) must be installed via a Merge Modules if provided by the vendor of the DLL.

Furthermore it is highly recommended not to include the FDT specific Interface description (IDL) within own components.

11.1.5 Microsoft 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.

11.1.6 Building a frame application-database of supported devices

Using component categories a Frame Application is able to detect all installed DTMs. Additional information, e.g. vendor information, list of supported devices, can be determined by instantiating a DTM and using the IDtmInformation and IDtmInformation2 interfaces.

By using this information, a Frame Application is able to build a database with

- all available DTMs,
- all available BTMs,
- DTMs supporting a specific fieldbus,
- supported field devices,
- etc.

Based on this information it is possible to generate a mapping between specific field devices and supporting DTMs. This functionality is described in 4.10.8.

11.1.7 DTM registration

DTMs have to write registry entries whenever the DTM is

- installed
- uninstalled
- modified (e.g. new device types are supported or the DTM was updated (bug fix)).

If the path does not exist, the first installed DTM has to create the path.

A FDT root path is defined for FDT in windows registry.

DTM Registration – Path in registry
[HKEY_LOCAL_MACHINE\SOFTWARE\FDT\DTMCatalogUpdates][3]

Key	Format	Description
ModificationComment	REG_SZ (String Value)	**Mandatory**: Any string, e.g.: Timestamp of modification. Empty string allowed[4].
ModificationFlag	REG_SZ (String – GUID)	**Mandatory**: GUID created by DTM setup during setup runtime

Frame Applications may update an DTM Library and save the last known ModificationFlag entry internally.

A Frame can easily check if there is a need to update the DTM Catalog by comparing last known entry with current entry.

NOTE

It was seen as acceptable to leave the entry in registry without a concept for a cleanup because there may be a number of Frame Applications installed on one PC.

Frame Applications are only allowed to read these keys. DTMs must update ModificationFlag and ModificationComment during setup.

[3] HKEY_LOCAL_MACHINE allows DTMs to write a registry key and string. During DTM setup user has to have Administrator rights anyway, so there is no missing right. While Frame session, a user can have normal rights. A Frame must only read the registry key so the missing right is not a problem.

[4] Instead of setting this attribute to optional , its mandatory and allowed to set an empty string. This avoids having old comment together with new flag.

12 Description of data types, parameters and structures

12.1 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. See Table 42.

Table 42 – FDT specific Ids

Name	Data type	Description
invokeId	FdtUUIDString	An invokeId of a request method of a server object is a unique identifier to be generated by the client via CoCreateGuid() API and is only valid within the calling object. Within a callback method this identifier must be used by the client to identify the appropriate request send to the server object. Association of asynchronous function calls is used for methods like xxxRequest(), OnxxxResponse()
systemTag	BSTR	Unique identifier of a device instance within a project of the Frame Application. The system tag will be set during the initialization of a DTM and has to be used by the DTM for all instance specific function calls to the Frame Application. The DTM must not store the systemTag (e.g. in the instance data set). The DTM may not rely on the fact that it receives the same systemTag during each initialization (call of IDtm::Environment() or IDtm2::Environment2())
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

12.2 Data type definitions

Following helper objects for documentation are defined (see Table 43).

Table 43 – Helper objects for documentation

Name	Data type	Description
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). Due to the definition of the UUID format, the value must NOT be handled in a case sensitive way. That means comparing “C2137dd1-7842-11d4-A3C9-005004DC410F” with “C2137DD1-7842-11d4-A3C9-005004DC410f” will return TRUE
FdtXmlDocument	BSTR	String containing an XML document
FdtXPath	BSTR	String containing an Xpath to an element of an XML document **For FDT 1.2 and FDT 1.2.1 the Xpath has to be the root tag “FDT”** - XML documents are accepted as complete document. Otherwise the DTM informs the Frame Application via the event interface about the occurred of errors **Exceptions:** – IFdtChannel::GetChannelPath(), refer to method description – Interface IFdtTopology, usage as channel path within the specified methods
All boolean parameters	VARINAT_BOOL	TRUE and FALSE according to the definition of VARIANT_TRUE and VARIANT_FALSE

Annex A
(normative)

FDT IDL

NOTE This IDL file should never be modified in any way. The standard marshaller can be used based on a type library generated from this IDL. If you add vendor-specific interfaces to your application (which is allowed), you must generate a SEPARATE vendor-specific IDL file to describe only those interfaces and a separate vendor-specific ProxyStub DLL to marshal only those interfaces.

```
/*****************************************************************************
 *
 * FDT 1.2.1 Interfaces
 *
 *****************************************************************************/

[
  uuid(036D1471-387B-11D4-86E1-00E0987270B9),
  version(1.20100)
]
library Fdt100
{
    importlib("STDOLE2.TLB");

            // FDT Datatypes
            typedef [uuid(036D1472-387B-11D4-86E1-00E0987270B9), version(1.0)] BSTR FdtUUIDString;
            typedef [uuid(036D1473-387B-11D4-86E1-00E0987270B9), version(1.0)] BSTR FdtXmlDocument;
            typedef [uuid(036D1474-387B-11D4-86E1-00E0987270B9), version(1.0)] BSTR FdtXPath;

    // Forward declaration of all required interfaces
    interface IFdtContainer;
            interface IFdtChannelCollection;
            interface IFdtCommunication;

            //
            //  DTM Interfaces
            //  ==============

// IDtm
    [
      odl,
      uuid(036D1481-387B-11D4-86E1-00E0987270B9),
      dual,
      oleautomation
    ]
    interface IDtm : IDispatch {
        [id(0x1)]
        HRESULT Environment(
                    [in] BSTR systemTag,
                    [in] IFdtContainer* container,
                    [out, retval] VARIANT_BOOL* result);
        [id(0x2)]
        HRESULT InitNew(
                    [in] FdtXmlDocument deviceType,
                    [out, retval] VARIANT_BOOL* result);
        [id(0x3)]
        HRESULT Config(
                    [in] FdtXmlDocument userInfo,
                    [out, retval] VARIANT_BOOL* result);
        [id(0x4)]
        HRESULT SetCommunication(
                    [in] IFdtCommunication* communication,
                    [out, retval] VARIANT_BOOL* result);
        [id(0x5)]
        HRESULT PrepareToRelease(
                                        [out, retval] VARIANT_BOOL* result);
        [id(0x6)]
        HRESULT PrepareToReleaseCommunication(
                                        [out, retval] VARIANT_BOOL* result);
        [id(0x7)]
        HRESULT ReleaseCommunication(
                                        [out, retval] VARIANT_BOOL* result);
```

```
        [id(0x8)]
        HRESULT PrepareToDelete(
                                        [out, retval] VARIANT_BOOL* result);
        [id(0x9)]
        HRESULT SetLanguage(
                    [in] long languageId,
                    [out, retval] VARIANT_BOOL* result);
        [id(0xa)]
        HRESULT GetFunctions(
                                        [in] FdtXmlDocument operationState,
                                        [out, retval] FdtXmlDocument* result);
        [id(0xb)]
        HRESULT InvokeFunctionRequest(
                                        [in] FdtUUIDString 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(036D147E-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 childsystemTag,
              [out, retval] VARIANT_BOOL* result);
    [id(0x4)]
    HRESULT ValidateRemoveChild(
              [in] BSTR childsystemTag,
              [out, retval] VARIANT_BOOL* result);
    [id(0x5)]
    HRESULT OnAddChild(
                                  [in] BSTR childsystemTag);
    [id(0x6)]
    HRESULT OnRemoveChild(
                                  [in] BSTR childsystemTag);
  };

// IFdtFunctionBlockData
  [
    odl,
    uuid(036D1475-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
  ]
  interface IFdtFunctionBlockData : IDispatch {
    [id(0x1)]
    HRESULT SelectFBInstance(
              [in] BSTR systemTag,
              [out, retval] VARIANT_BOOL* result);
    [id(0x2)]
    HRESULT GetFBInstanceData(
              [in] BSTR systemTag,
              [out, retval] FdtXmlDocument* result);
  };

          //
          // FDT Channel ActiveX Control interfaces
          // ======================================
          //

// IFdtChannelActiveXControl
  [
    odl,
    uuid(036D148B-387B-11D4-86E1-00E0987270B9),
    dual,
    oleautomation
  ]
  interface IFdtChannelActiveXControl : IDispatch {
    [id(0x1)]
    HRESULT Init(
              [in] FdtUUIDString invokeId,
              [in] IFdtChannel* channel,
              [out, retval] VARIANT_BOOL* result);
    [id(0x2)]
    HRESULT PrepareToRelease(
                                  [out, retval] VARIANT_BOOL* result);
  };

          //
          // Frame Application interfaces
          // ============================
          //

// IDtmEvents
  [
    odl,
    uuid(F15BA42E-BBF1-42ed-8009-7F664A002CFB),
    dual,
    oleautomation
  ]
  interface IDtmEvents : IDispatch {
```

```
    [id(0x1)]
    HRESULT OnParameterChanged(
                [in] BSTR systemTag,
                [in] FdtXmlDocument parameter);
    [id(0x2)]
    HRESULT OnErrorMessage(
                [in] BSTR systemTag,
                [in] BSTR errorMessage);
    [id(0x3)]
    HRESULT OnProgress(
                [in] BSTR systemTag,
                [in] BSTR title,
                [in] short percent,
                                                [in] VARIANT_BOOL show);
    [id(0x4)]
    HRESULT OnUploadFinished(
                                        [in] FdtUUIDString 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);
            };
```

```
            //
            //  BTM Interfaces
            //  ==============

// IBtm
   [
     odl,
     uuid(96341E37-9611-46ba-80ED-A85BD73BF518),
     dual,
     oleautomation
   ]
   interface IBtm : IDtm {
   };

// IBtmInformation
   [
     odl,
     uuid(87DCC81C-9F97-46c2-A483-5A89B155204C),
            dual,
     oleautomation
   ]
   interface IBtmInformation : IDispatch {
      [id(0x1)]
      HRESULT GetInformation(
                                          [out, retval] FdtXmlDocument* result);
   };

// IBtmParameter
   [
     odl,
     uuid(43D592CC-5F8E-4eca-9365-8D4749390C55),
     dual,
     oleautomation
   ]
   interface IBtmParameter : IDispatch {
      [id(0x1)]
      HRESULT GetParameters(
                  [in] FdtXPath parameterPath,
                  [out, retval] FdtXmlDocument* result);
      [id(0x2)]
      HRESULT SetParameters(
                  [in] FdtXPath parameterPath,
                  [in] FdtXmlDocument FdtXmlDocument,
                  [out, retval] VARIANT_BOOL* result);
   };

// IBtmActiveXControl
   [
     odl,
     uuid(0E0418B4-9BC6-4a28-B980-5D3E7F457E4F),
     dual,
     oleautomation
   ]
   interface IBtmActiveXControl : IDispatch {
      [id(0x1)]
      HRESULT Init(
                  [in] FdtUUIDString invokeId,
                  [in] FdtXmlDocument functionCall,
                  [in] IBtm* btm,
                  [out, retval] VARIANT_BOOL* result);
      [id(0x2)]
      HRESULT PrepareToRelease(
                                          [out, retval] VARIANT_BOOL* result);
   };

 // IFdtBtmTopology
   [
     odl,
     uuid(18250F40-73FB-4c22-A0F1-DB2A11B3FE8D),
     dual,
     oleautomation
   ]
   interface IFdtBtmTopology : IDispatch {
      [id(0x1)]
      HRESULT GetParentNodes(
                  [in] BSTR systemTag,
                  [out, retval] FdtXmlDocument* result);
```

```
        [id(0x2)]
        HRESULT GetChildNodes(
                    [in] BSTR systemTag,
                    [in] FdtXPath channelPath,
                    [out, retval] FdtXmlDocument* result);
        [id(0x3)]
        HRESULT CreateChild(
                    [in] FdtXmlDocument deviceType,
                    [in] FdtXPath channelPath,
                    [out, retval] BSTR* result);
        [id(0x4)]
        HRESULT DeleteChild(
                    [in] BSTR systemTag,
                    [in] FdtXPath channelPath,
                    [out, retval] VARIANT_BOOL* result);
                        [id(0x5)]
                        HRESULT GetBtmForSystemTag(
                                                    [in] BSTR systemTag,
                                                    [out,retval] IBtm **result);

                        [id(0x6)]
                        HRESULT GetBtmInfoList(
                                                    [out, retval] FdtXmlDocument* result);
                        [id(0x7)]
                        HRESULT MoveChild(
                                                    [in] BSTR systemTag,
                    [in] FdtXPath destinationchannelPath,
                    [out, retval] VARIANT_BOOL* result);
        [id(0x8)]
        HRESULT ReleaseBtmForSystemTag(
                                                    [in] BSTR systemTag,
                                                    [out, retval] VARIANT_BOOL* result);
    };

            //
            //  FDT 1.2.1 Interfaces
            //  ===============

// IFdtActiveX2
    [
      odl,
      uuid(1922C2DE-4EE7-4085-878A-80AC6C6728AD),
      dual,
      oleautomation
    ]
    interface IFdtActiveX2 : IDispatch
    {
        [id(0x1)]
        HRESULT OpenDialogActiveXControlRequest(
                    [in] BSTR systemTag,
                    [in] FdtXmlDocument functionCall,
                    [out, retval] VARIANT_BOOL* result);

                        [id(0x2)]
                        HRESULT OpenChannelActiveXControlRequest(
                                                    [in] BSTR channelPath,
                                                    [in] FdtXmlDocument functionCall,
                                                    [out, retval] VARIANT_BOOL* result);

                        [id(0x3)]
                        HRESULT CloseChannelActiveXControlRequest(
                                                    [in] FdtUUIDString invokeId,
                                                    [out, retval] VARIANT_BOOL* result);

        [id(0x4)]
        HRESULT OpenDialogChannelActiveXControlRequest(
                    [in] BSTR channelPath,
                    [in] FdtXmlDocument functionCall,
                    [out, retval] VARIANT_BOOL* result);

    };

// IDtm2
    [
      odl,
      uuid(51E1F44B-D6A1-423d-B11F-AD38EDE78047),
      dual,
      oleautomation
```

```
  ]
  interface IDtm2: IDispatch
  {
     [id(0x1)]
     HRESULT Environment2(
                [in] BSTR systemTag,
                [in] IFdtContainer* container,
                                                [in] FdtXmlDocument frameInfo,
                [out, retval] VARIANT_BOOL* result);
                     [id(0x2)]
                     HRESULT SetSystemGuiLabel(
                                                [in] FdtXmlDocument systemGuiLabel,
                                                [out, retval] VARIANT_BOOL* result);
  };

// IDtmRedundancyEvents
  [
    odl,
    uuid(04808A4C-90C3-45a7-B69E-034A2FA8314D),
    dual,
    oleautomation
  ]
  interface IDtmRedundancyEvents: IDispatch
  {
     [id(0x1)]
     HRESULT OnAddedRedundantChild(
                [in] BSTR systemTag,
                [in] FdtXPath channelPath,
                [out, retval] VARIANT_BOOL* result);
     [id(0x2)]
     HRESULT OnRemovedRedundantChild(
                [in] BSTR systemTag,
                [in] FdtXPath channelPath,
                [out, retval] VARIANT_BOOL* result);

  };

 // IDtmEvents2
  [
    odl,
    uuid(494FFD2B-6E58-4e42-80DB-85EBDF6E2CF5),
    dual,
    oleautomation
  ]
          interface IDtmEvents2 : IDispatch
          {
                     [id(0x1)]
                     HRESULT OnStateChanged(
                                                [in] BSTR systemTag,
                                                [in] FdtXmlDocument xmldoc);
          };
          // IFdtChannelActiveXControl2
          [
            odl,
            uuid(73757F49-F3A6-41f7-BEA0-1A3E59D69D5B),
            dual,
            oleautomation
          ]
          interface IFdtChannelActiveXControl2 : IDispatch
          {
                     [id(0x1)]
                     HRESULT Init2(
                                                [in] FdtUUIDString invokeId,
                                                [in] FdtXMLDocument functionCall,
                                                [in] IFdtChannel* channel,
                                                [out, retval] VARIANT_BOOL* result);
          };

// IDtmScanEvents
  [
    odl,
    uuid(515590B9-5177-474a-9310-708A3E785B2B),
    dual,
    oleautomation
  ]
  interface  IDtmScanEvents : IDispatch
```

```
    {
        [id(0x1)]
        HRESULT OnScanResponse(
                                                    [in] FdtUUIDString invokeId,
                                                    [in] FdtXMLDocument response);

        [id(0x2)]
        HRESULT OnScanHardwareResponse (
                                                    [in] FdtUUIDString invokeId,
                                                    [in] FdtXMLDocument response);

            };


// IFdtChannelScan
    [
      odl,
      uuid(64A4310D-F9EE-411d-B4F9-51D3360DF359),
      dual,
      oleautomation
    ]
    interface  IFdtChannelScan : IDispatch
    {
        [id(0x1)]
        HRESULT ScanRequest(
                    [in] FdtUUIDString invokeId,
                    [in] FdtXMLDocument request,
                    [out, retval] VARIANT_BOOL* result);
        [id(0x2)]
        HRESULT CancelAction (
                    [in] FdtUUIDString invokeId,
                    [out, retval] VARIANT_BOOL* result);

    };

// IFdtChannelSubTopology2
    [
      odl,
      uuid(6359FBF1-D373-4202-90E8-E696C37739D4),
      dual,
      oleautomation
    ]
    interface  IFdtChannelSubTopology2 : IDispatch
    {
        [id(0x1)]
        HRESULT SetChildrenAdresses(
                                                    [in] FdtXMLDocument dtmDeviceList,
                    [out, retval] FdtXMLDocument* result);
    };


// IDtmInformation2
    [
      odl,
      uuid(C2934CC6-B72D-4611-890F-1C6531D1F8EB),
      dual,
      oleautomation
    ]
    interface IDtmInformation2 : IDispatch
    {
        [id(0x1)]
        HRESULT GetDeviceIdentificationInformation(
                    [in] FdtXMLDocument request,
                                                    [in] FdtUUIDString protocolId,
                    [out, retval] FdtXMLDocument* response);
    };

            // IDtmHardwareIdentification
            [
                        odl,
                        uuid(9A1DD233-987C-43d1-9424-DA5C1FC6F292),
                        dual,
                        oleautomation
            ]
            interface IDtmHardwareIdentification : IDispatch
            {
                        [id(0x1)]
                        HRESULT ScanHardwareRequest (
                                                    [in] FdtUUIDString invokeID,
```

```
                                                    [in] FdtXMLDocument request,
                                                    [out, retval] VARIANT_BOOL* result);
                [id(0x2)]
                HRESULT CancelAction (
                                                    [in] FdtUUIDString invokeID,
                                                    [out, retval] VARIANT_BOOL* result);
        };

// IFdtCommunicationEvents2
   [
            odl,
     uuid(7A0AEF6A-A9E7-4673-8F45-01B8AC28F55D),
     dual,
     oleautomation
   ]
        interface IFdtCommunicationEvents2 : IDispatch
        {
                [id(0x1)]
                HRESULT OnConnectResponse2(
                                                    [in] FdtUUIDString invokeID,
                                                    [in] FdtXMLDocument parentInformation,
                                                    [in] FdtXMLDocument response);
        };

// IDtmSingleDeviceDataAccess
   [
            odl,
     uuid(D67240E4-664B-44b0-B692-A1D1ED3FB8F8),
     dual,
     oleautomation
   ]
        interface IDtmSingleDeviceDataAccess : IDispatch
        {
                [id(0x1)]
                HRESULT CancelRequest (
                                                    [in] FdtUUIDString invokeId,
                                                    [out, retval] VARIANT_BOOL* result);
                [id(0x2)]
                HRESULT ItemListRequest(
                                                    [in] FdtUUIDString invokeId);
                [id(0x3)]
                HRESULT ReadRequest(
                                                    [in] FdtUUIDString invokeId,
                                                    [in] FdtXmlDocument itemSelectionList);
                [id(0x4)]
                HRESULT WriteRequest(
                                                    [in] FdtUUIDString invokeId,
                                                    [in] FdtXmlDocument itemList);
        };

// IDtmSingleInstanceDataAccess
   [
            odl,
     uuid(84F9A19A-7E38-40b5-A311-60B14F30C258),
     dual,
     oleautomation
   ]
        interface IDtmSingleInstanceDataAccess : IDispatch
        {
                [id(0x1)]
                HRESULT GetItemList(
                                                    [out, retval] FdtXmlDocument* result);
                [id(0x2)]
                HRESULT Read(
                                                    [in] FdtXmlDocument itemSelectionList,
                                                    [out, retval] FdtXmlDocument* result);
                [id(0x3)]
                HRESULT Write(
                                                    [in] FdtXmlDocument itemList,
                                                    [out, retval] FdtXmlDocument* result);
        };

// IDtmSingleDeviceDataAccessEvents
```

```
    [
            odl,
      uuid(2357691C-E69A-4e0a-A8AA-EB7F1D080CEF),
      dual,
      oleautomation
    ]
            interface IDtmSingleDeviceDataAccessEvents : IDispatch
            {
                    [id(0x1)]
                    HRESULT OnItemListResponse(
                                                    [in] FdtUUIDString invokeId,
                                                    [in] FdtXMLDocument response);
                    [id(0x2)]
                    HRESULT OnDeviceItemListChanged(
                                                    [in] BSTR systemTag);
                    [id(0x3)]
                    HRESULT OnReadResponse(
                                                    [in] FdtUUIDString invokeId,
                                                    [in] FdtXMLDocument response);
                    [id(0x4)]
                    HRESULT OnWriteResponse(
                                                    [in] FdtUUIDString invokeId,
                                                    [in] FdtXMLDocument response);
            };

// IDtmSingleInstanceDataAccessEvents
    [
            odl,
      uuid(EBC093F1-4F7A-4208-A209-C172E54AB185),
      dual,
      oleautomation
    ]
            interface IDtmSingleInstanceDataAccessEvents : IDispatch
            {
                    [id(0x1)]
                    HRESULT OnInstanceItemListChanged(
                                                    [in] BSTR systemTag);
            };


};
```

Annex B
(normative)

FDT XML schemas

B.1 FDTDataTypesSchema

The data type schema is used as a global FDT definition. Data types of this schema are reference via the prefix fdt: within the other schemas. Several data defined as attribute and as element to support the XML features for element and group definitions. See Table B.1.

Table B.1 – Definition of general XML attributes

Attribute	Description
alarmType	Identifier for the alarm type to show the association between high and low alarm and high-high and low-low alarm
applicationDomain	Attribute defining the application domain, that applies to provided semanticId. This may be a protocol-specific ID or an other FDT-defined application domain
address	Parameter Addressing information. The format of the value for this parameter is described for each communication protocol in the corresponding section of the specification. For interoperability reasons this parameter is defined as an optional in the XML schema document. The protocol portion of the specification provides the guidelines for address attribute. Most of the protocols specify that the DTM must expose the addressing information to the Frame Application.
binData	Variable containing binary data
bitLength	Length of a binary variable as bit count
bitPosition	Position of a bit within a enumeration (0 based position)
boolValue	Variable for configured static boolean data like alarm value
busCategory	Unique identifier for a supported bus type like Profibus or HART according to the FDT specific unique identifier
busRedundancy	Number of redundant fieldbus interfaces
byteArray	Variable used to transfer binary data. Binary data can be transferred if the attribute is defined as 'bin.hex'. The value has to be set as string at the DOM. This string has to be generated by the DTM developer, because the 'bin.hex' data type of XML shows the problem, that the last byte gets lost at the non-typed contents variable, if the value is set to the nodeTypedValue of the DOM
byteLength	Number of bytes to describe data types like string
channelMode	Type information enumeration for a channel
classificationDomainId	Device classification domain group according to IEC 62390 (Annex G). See tables below
classificationId	Unique identifier for classification of a channel or device according to its primary function. Primarily, attribute 'classificationId' should be selected from the table ' Device classification according to IEC 62390 (Annex G)' and use corresponding 'classificationDomainId' attribute for it. If suitable classification ID does not exist in that table, it should be selected from 'FDT classification ID table' and use it without 'classificationDomainId' attribute. See tables below
communicationError	Fieldbus protocol independent error occurred during communication. This kind of error is used especially if an error occurred during nested communication. The fieldbus specific communication error is part of the fieldbus specific XML schema Communication errors detected by a communication component, e.g. a Communication-DTM, must be handled within the XML document returned to the child component

Attribute	Description
	• All errors returned by a device as result of a fieldbus transaction should be returned by protocol-specific response read or write response elements, typically by using fieldbus specific status and error attributes. • All errors detected by the Communication-DTM must be mapped to one of the fdt:communicationError enumeration values (e.g., abort busy invalidCommunicationReference noConnection noParallelServices noPendingRequest unknownError timeout dtmSpecific notSupportedFeature) and returned as a fdt:CommunciationError element to the child DTM. A DTM must be able to handle both types of error responses for a transaction request sent to the parent component.
dataSetID	Unique identifier of the data set version
dataSetState	State of an instance data set concerning modifications (refer to clause 10.14.1.1Modifications)
dataType	Identifier for the data type of a transferred variable
dataTypeDescriptor	Description of data type
date	Date variable within an element
descriptor	Human readable description within the context of a element
description	A human readable description of the supported and current data set format. Can be used to provide additional information to the user in case of partial level of support
deviceGraphicState	List of possible device states used in the DeviceIcon and DeviceBitmap element. Possible states are: • device: symbolic representation of the device • diagnostic: symbolic representation of device if there is online diagnosis available • oem: symbol representation of device in special operating modes (e.g. OEM service) The Frame Application should display the correct picture according to protocol specific rules
deviceTypeId	Unique identifier for a device type within the name space of the fieldbus specification
deviceTypeInformation	Additional device type information supplied with a device. For example, a PROFIBUS device has to provide its GSD information as human readable string at this attribute. The GSD information must be provided based on the current locale according to the usage of IDtm::SetLanguage(). NOTE The GSD information is accessible via: IDtmParameter::GetParameters() IDtmInformation::GetInformation()
deviceTypeInformationPath	Path to the file containing the information which is provided via the attribute 'deviceTypeInformation'. It is recommended to use this attribute for providing additional protocol specific descriptions of the device type. The use of this attribute is specified in the protocol-specific annex
deviceTypeVariant	Manufactuer-specific device type variant definition string.
deviceTypeVariantInfo	Contains additional information for manufactuer-specific device type variant definition
display	Carries an human readable display string for tasks like documentation
displayFormat	Describes the display format for a display attribute (e.g. "%3.2f " for a

Attribute	Description
	float value)
documentClassification	Specifies the subject of the document
documentLanguageId	Identifier for the language of the document. The language-id is defined as a Windows locale identifier (LCID)
dtmDevTypeID	dtmDevTypeID is a DTM specific unique identifier of the software related to the device type. For the same device type it is value remains unchanged although some identification information in DtmDeviceType element is updated. This can be a result of DTM update. The same dtmDevTypeID value must always be related to the same device as in the previous DTM version (e.g. to provide backward compatibility). **It is strongly recommended to provide this attribute!**
dtmDevTypeIDVersion	Version number of the dtmDevTypeID. It is also used to indicate that the information related to DtmDeviceType is changed. Frame Application can use this information e.g. to update DTM related information in DTM library. **It is strongly recommended to provide this attribute!** An example: When a DTM is DD based, an upgrade of DD will cause the change of dtmDevTypeIDVersion without changing the dtmDevTypeID.
dtmStateMachine	Current state of DTM state machine
errorCode	Status information according to the Profibus specification
file	Contains a document or executable file with an absolute path
help	Human readable help string for a document
Id	Unique identifier for an element. This identifier is used within collections for the direct access of elements. This id must be unique within the namespace of a device instance. The according reference within the XML schemas Is the attribute 'idref'
Idref	Reference to an element specified by the attribute 'id'. The identifier is used within collections for the direct access of elements
Index	Index within an enumerator
label	Human readable label for a document
levelOfSupport	Provides an indication about the level of support of the supported data set version. • full – the data set is fully supported partial – some of the information in the data set cannot be used in the upgrade procedure. Additional information should be provided in the description attribute. Can be used to warn the user that some values can be lost
languageId	Identifier for a supported language or the language a DTM should 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
modifiedInDevice	Flag to provide more information about current state of the instance data set. Although this flag is an optional attribute, usage of the flag is strongly recommended. TRUE, indicates that parameters have been changed in the device but not in the instance data set (e.g.: see use case Online parameterization, IDtmSingleDeviceDataAccess interface definition) 'modifiedInDevice' flag must be set only once in case the data in the device has been changed. In case of successful Upload or Download of complete data set,

Attribute	Description
	'modifiedInDevice' flag must be reset (set to FALSE). Data in the device may also be modified directly by a tool out of the scope of the FDT. In this case, the flag 'modifiedInDevice' should not be set. Data set must be locked before an application is started, which may need to change the flag "modifiedInDevice" (the flag 'modifiedInDevice is part of the instance data set and could not be changed if the data set is not locked)
name	Human readable name within the context of a element
nodeId	DTM specific node identifier. Can be used to speed up the access to a node
number	Number variable like float, integer or other numeric data types
orderCode	Order code of the device variant
parameters	Contains the parameters to be passed to the application, if the file attribute specifies an executable file
path	Path to the icon for a device
physicalLayer	CATID for description of a physical layer of a fieldbus
physicalLayerName	Human readable description for the physical layer
readAccess	Specifies whether the value can be read from a device
reference	Reference to a variable of a structure
semanticId	Semantic identifier for an element. This identifier must be unique at least within the context of an element. By using this attribute e.g. a Frame Application is able to get the information regarding the meaning and usage of a single data structure. The definition regarding the identifier is protocol specific and described in protocol specific annexes Furthermore the following protocol independent semanticIds are defined to cover the network information: FDT.slaveAddress FDT.busAddress FDT.busMasterConfigurationPart A Parameter with one of these semanticId must be the same parameter as the corresponding XML attribute (slaveAddress, busAddress or busMasterConfigurationPart) within the DTMParameterSchema represent
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 clause 10.14.1.2 Persistence states)
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 subtype. For example, the same transmitter can be pre-configured for level or flow measuring
substituteType	Type of a substitute value
systemGuiLabel	Unique human readable identifier of the DTM instance in the context of the Frame Application
tag	Unique identifier for a device, module, or channel
time	Time variable within an element
url	Contains a URL to a document in the Web
vendor	Human readable description of the vendor of component
version	Human readable description of the version of component like "1.0"
writeAccess	Specifies whether the value can be written into a device

Table B.2 – Definition of general XML attributes

Tag	Description
Alarm	Description of an alarm
AlarmEnumerationEntry	Alarm enumeration entry.
AlarmEnumerationEntries	Collection of alarm enumeration entries.
Alarms	Collection of alarms
BinaryVariable	Element containing binary data
BitEnumeratorEntries	Collection of EnumerationEntry
BitEnumeratorVariable	Current EnumeratorEntry and the collection of possible EnumeratorEntries
BitVariable	Selected element of an enumeration
BoolValue	Variable for configured static boolean data
BusCategory	Description of busCategory
BusCategories	Collection of BusCategory
BusRedundancy	Number of redundant fieldbus interfaces
ChannelMode	Type information element for a channel
ChannelModes	List of ChannelMode elements
ChannelInformation	Type information for a FDT channel. This element is recommended within a document returned by a IDtmParameter::GetParameters() call. The BusCategories element should contain the list of supported fieldbus protocols of this channel
ChannelReference	Reference to an object identified by its id
ChannelReferences	Collection of references
ClassificationId	Classification Id. See tables below
ClassificationIds	Collection of ClassificationId elements. See tables below
CommunicationData	Variable used to transfer binary communication data
CommunicationError	Description of a fieldbus protocol independent error occurred during nested communication with error code, the tag of the communication channel's device and optional error description
Current	A current version of the data set used for saving the data
DataSetFormats	Data formats of the persisted data used and supported by the DTM
Deadband	Deadband is the amount of value changes that triggers for example new trend values
DeviceIcon	Icon for a device
DeviceTypeVariant	Contains manufacturer-specific device type variant information. A device type variant is a property of the physical device that has no influence to the software (i.e. flange material, Ex certificate …). However some Frame Applications will use this information for documentation purposes. The DeviceVariant element can occur under DeviceVariants element within context of DTMInformationSchema XMLs or directly under DtmDeviceType elements in context of DTMInitInstanceSchema or DTMParameterSchema XMLs
DeviceTypeVariants	Collection of DeviceVariant elements. The DeviceVariants element should only be used within context of DTMInformationSchema XMLs
Display	Display variable
DtmDeviceType	Description of a device type
DtmVariable	Variable description with name, value, range, etc.
DtmVariables	Collection of DTM variables

Tag	Description
DtmVariableReference	Reference to a DTM variable
EnumeratorEntries	Enumeration element
EnumeratorEntry	Element of an enumeration
EnumeratorVariable	Current EnumeratorEntry and the collection of possible EnumeratorEntries
LanguageId	Contains the languageId (refer to languageId)
LowerRange	Defintion of a lower range element
LowerRawValue	A numeric entry which reflects the real via e.g. PROFIBUS transferred value. The value is mapped to the related range (LowerRangeValue). For example if a PROFIBIUS device maps a range from 10 to 100 mbar to an integer of value 1024-4096 the 'LowerRawValue' contains 1024, the 'UpperRawValue' contains 4096
NumberData	Number variable like float, integer or other numeric data types
PhysicalLayer	Unique identifier for a physical layer of a fieldbus like Profibus PA
Range	Describes the valid range of a process variable
Ranges	Collection of ranges
SemanticInformation	The element provides semantic information for a data object. For each object at least one <SemanticInformation> element with an FDT-defined protocol-specific semanticId must be provided. The DTM must provide a <SemanticInformation> element for all supported fieldbus protocol of the DTM instance
StaticValue	Variable for configured static data like an alarm value
StatusInformation	Current status information of a device or module
StringData	String variable
StructuredElement	Variable as display value or as reference to a variable with data length information
StructuredElements	Collection of structured elements
StructuredVariable	Describes a binary value and its structure information
SubstituteValue	Describes a substitute value which is used in combination of the behavior of disturbed channel values
Supported	A list of the supported data set that can be upgraded by the DTM
SupportedLanguages	Collection of language ids
TimeData	Element of a time date value
Unit	Current unit and the collection of possible units of a process variable
UpperRange	Definition of an upper range element
UpperRawValue	A numeric entry which reflects the real via e.g. PROFIBUS transferred value. The value is mapped to the related range (UpperRangeValue). For example if a PROFIBIUS device maps a range from 10 to 100 mbar to an integer of value 1024-4096 the 'UpperRawValue' contains 4096, the 'UpperRange' contains 4096
Value	Contains the display string for a variable or the variable itself
Variable	Selected element of an enumeration
Variant	Variable with data type and display format
VersionInformation	Description of the version of a component where used. For example: This element provides the version information of the DTM when used in DtmInfo. It is recommended to use this element to describe the physical device information in the DtmDeviceType. This information should correlate to information provided by SW revision or HW revision in IDtmInformation2::GetDeviceIdentificationInformation()

Tag	Description
DtmDocument	Definition of a document, which is provided by a DTM and displayed by the Frame Application
DocumentFile	Defines a file document
DocumentExe	Defines a executable, which is used to show DTM documents
DocumentUrl	Defines a URL to a document in the Web Implementation hint: The file, url and executable documents can be implemented by using the ShellExecute() function
DtmDocuments	List of DTM documents. This tag allows a DTM to publish his documents.
DeviceBitmap	Bitmap for a device in BMP format (70*40 pixels (width*height), 16 colors)

The following tables provide the FDT classification ID (see Table B.3 and Table B.4):

Table B.3 – Device classification ID

classificationID
flow
level
pressure
temperature
valve
positioner
actuator
nc_rc
encoder
speedDrive
hmi
analogInput
analogOuput
digitalInput
digitalOutput
electrochemicalAnalyser
dtmSpecific
universal
analyser
remoteIO
analogCombinedIO
digitalCombinedIO
recorder
controller
angle
limitSwitch
converter
motor

The following Table B.4 defines the mapping between IEC 60947 device classification and the FDT classification attributes.

Table B.4 – Device classification according to IEC 60947, Annex G

Domain ()		Subdomain
classificationDomainId	**IEC domain group name**	**classificationId**
powerDistribution	Power distribution	switchboard
		circuitBreaker
		powerMonitoring
		distributionPanel
motionControl	Motion control	contactor
		protectionStarter
		softStarter
		drive
		axisControl
		motorControlCenter
		motorMonitoring
		positioner
		controlValve
measurement	Detection, measurement	electrical
		density
		flow
		level
		quality
		pressure
		speedOrRotaryFrequency
		radiation
		temperature
		weightMass
		distanceOrPositionOrPresence
operatorInterface	Dialogue / operator interfaces	pushButton
		joystick
		keypad
		pilotLight
		stackLight
		display
		combinedButtonsAndLights
		operatorStation
modulesAndControllers	Logic / universal I/O modules and controllers	generalInput
		generalOuput
		combinedInputOuput
		relay
		timer
		scanner
		programmableController

```
<Schema name="FDTDataTypesSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <!--Definition of Attributes-->
    <AttributeType name="alarmType" dt:type="enumeration" dt:values="highHighAlarm highAlarm lowLowAlarm lowAlarm"/>
    <AttributeType name="binData" dt:type="bin.hex"/>
    <AttributeType name="bitLength" dt:type="ui4"/>
    <AttributeType name="byteArray" dt:type="bin.hex"/>
    <AttributeType name="byteLength" dt:type="ui4"/>
    <AttributeType name="classificationId" dt:type="enumeration" dt:values="flow level pressure temperature valve positioner actuator nc_rc encoder speedDrive hmi analogInput analogOutput digitalInput digitalOutput electrochemicalAnalyser dtmSpecific universal analyser remoteIO analogCombinedIO digitalCombinedIO recorder controller angle limitSwitch converter motor  switchboard circuitBreaker powerMonitoring distributionPanel contactor protectionStarter softStarter drive axisControl motorControlCenter controlValve electrical density quality speedOrRotaryFrequency radiation weightMass distanceOrPositionPresence pushButton joystick keypad pilotLight stackLight display combinedButtonsAndLights operatorStation generalInput generalOutput combinedInputOutput relay timer scanner programmableController"/>
    <AttributeType name="communicationError" dt:type="enumeration" dt:values="abort busy invalidCommunicationReference noConnection noParallelServices noPendingRequest unknownError timeout dtmSpecific notSupportedFeature sequenceTimeExpired"/>
    <AttributeType name="dataSetState" dt:type="enumeration" dt:values="default validModified invalidModified allDataLoaded"/>
    <AttributeType name="dataType" dt:type="enumeration" dt:values="byte float double int unsigned enumerator bitEnumerator index ascii packedAscii password bitString hexString date time dateAndTime duration binary structured dtmSpecific"/>
    <AttributeType name="dataTypeDescriptor" dt:type="string"/>
    <AttributeType name="date" dt:type="date"/>
    <AttributeType name="descriptor" dt:type="string"/>
    <AttributeType name="display" dt:type="string"/>
    <AttributeType name="displayFormat" dt:type="string"/>
    <AttributeType name="errorCode" dt:type="bin.hex"/>
    <AttributeType name="id" dt:type="string"/>
    <AttributeType name="idref" dt:type="string"/>
    <AttributeType name="index" dt:type="ui4"/>
    <AttributeType name="name" dt:type="string"/>
    <AttributeType name="number" dt:type="number"/>
    <AttributeType name="reference" dt:type="string"/>
    <AttributeType name="signalType" dt:type="enumeration" dt:values="input output "/>
    <AttributeType name="staticValue" dt:type="number"/>
    <AttributeType name="statusFlag" dt:type="enumeration" dt:values="ok warning error invalid"/>
    <AttributeType name="storageState" dt:type="enumeration" dt:values="persistent transient"/>
    <AttributeType name="string" dt:type="string"/>
    <AttributeType name="tag" dt:type="string"/>
    <AttributeType name="time" dt:type="dateTime"/>
    <AttributeType name="vendor" dt:type="string"/>
    <AttributeType name="version" dt:type="string"/>
    <AttributeType name="nodeId" dt:type="id"/>
    <AttributeType name="readAccess" dt:type="boolean" default="1"/>
    <AttributeType name="writeAccess" dt:type="boolean" default="0"/>
    <AttributeType name="deviceTypeId" dt:type="ui4"/>
    <AttributeType name="subDeviceType" dt:type="string"/>
    <AttributeType name="deviceTypeInformation" 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"/>
    <AttributeType name="communicationType" dt:type="enumeration" dt:values="supported required"/>
    <AttributeType name="busCategoryName" dt:type="string"/>
    <AttributeType name="help" dt:type="string"/>
    <AttributeType name="label" dt:type="string"/>
    <AttributeType name="file" dt:type="uri"/>
    <AttributeType name="url" dt:type="uri"/>
    <AttributeType name="parameters" dt:type="string"/>
    <AttributeType name="documentLanguageId" dt:type="ui4"/>
    <AttributeType name="documentClassification" dt:type="enumeration" dt:values="help technicalDocumentation orderingInformation miscellaneous"/>
    <AttributeType name="deviceGraphicState" dt:type="enumeration" dt:values="device diagnosis oem"/>
    <AttributeType name="deviceTypeInformationPath" dt:type="uri"/>
    <AttributeType name="systemGuiLabel" dt:type="string"/>
    <AttributeType name="busAddress" dt:type="string"/>
    <AttributeType name="systemTag" dt:type="string"/>
     <AttributeType name="channelMode" dt:type="enumeration" dt:values="communication moduleSlot processValue"/>
     <AttributeType name="physicalLayerName" dt:type="string"/>
     <AttributeType name="physicalLayer" dt:type="uuid"/>
    <AttributeType name="busRedundancy" dt:type="ui4"/>
    <AttributeType name="modifiedInDevice" dt:type="boolean" default="0"/>
```

```
      <AttributeType name="dtmStateMachine" dt:type="enumeration" dt:values="communicationSet goingOnline goingOffline
online"/>
       <AttributeType name="orderCode" dt:type="string"/>
       <AttributeType name="deviceTypeVariant" dt:type="string"/>
       <AttributeType name="deviceTypeVariantInfo" dt:type="string"/>
       <AttributeType name="bitPosition" dt:type="ui4"/>
       <AttributeType name="boolValue" dt:type="boolean"/>
       <AttributeType name="classificationDomainId" dt:type="enumeration" dt:values="powerDistribution motionControl
measurement operatorInterface modulesAndControllers"/>
      <AttributeType name="dataSetID" dt:type="uuid"/>
        <AttributeType name="description" dt:type="string"/>
      <AttributeType name="levelOfSupport" dt:type="enumeration" dt:values="full partial" />
      <AttributeType name="semanticId" dt:type="string"/>
      <AttributeType name="address" dt:type="string"/>
      <AttributeType name="applicationDomain" dt:type="string"/>
      <AttributeType name="dtmDevTypeID" dt:type="uuid"/>
      <AttributeType name="dtmDevTypeIDVersion" dt:type="ui4"/>


      <!--Definition of Elements-->
      <ElementType name="SemanticInformation" content="empty" model="closed">
            <attribute type="nodeId" required="no"/>
            <attribute type="applicationDomain" required="yes"/>
            <attribute type="semanticId" required="yes"/>
            <attribute type="address" required="no"/>
       </ElementType>
      <ElementType name="LowerRawValue" content="empty" model="closed">
            <attribute type="nodeId" required="no"/>
            <attribute type="number" required="yes"/>
      </ElementType>
      <ElementType name="UpperRawValue" content="empty" model="closed">
            <attribute type="nodeId" required="no"/>
            <attribute type="number" required="yes"/>
      </ElementType>
    <ElementType name="Current" content="empty" model="closed">
      <attribute type="nodeId" required="no"/>
      <attribute type="dataSetID" required="yes"/>
      <attribute type="description" required="no"/>
    </ElementType>
    <ElementType name="Supported" content="empty" model="closed">
      <attribute type="nodeId" required="no"/>
      <attribute type="dataSetID" required="yes"/>
      <attribute type="levelOfSupport" required="no" default="partial"/>
      <attribute type="description" required="no"/>
    </ElementType>
    <ElementType name="DataSetFormats" content="eltOnly" model="closed">
      <attribute type="nodeId" required="no"/>
      <element type="Current" minOccurs="1" maxOccurs="1"/>
      <element type="Supported" minOccurs="0" maxOccurs="*"/>
    </ElementType>
    <ElementType name=" ClassificationId" content="empty" model="closed">
      <attribute type="nodeId" required="no"/>
      <attribute type="classificationId" required="yes"/>
    </ElementType>
    <ElementType name=" ClassificationIds" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
      <attribute type="classificationDomainId" required="no"/>
      <element type="ClassificationId" minOccurs="1" maxOccurs="*"/>
    </ElementType>
    <ElementType name="BoolValue" content="empty" model="closed">
      <attribute type="nodeId" required="no"/>
      <attribute type="boolValue" required="no"/>
    </ElementType>
    <ElementType name="Deadband" content="empty" model="closed">
      <attribute type="nodeId" required="no"/>
      <attribute type="number" required="yes"/>
    </ElementType>
    <ElementType name="AlarmEnumerationEntry" content="empty" model="closed">
      <attribute type="nodeId" required="no"/>
      <attribute type="bitPosition" required="yes"/>
      <attribute type="name" required="yes"/>
      <attribute type="boolValue" required="yes"/>
      <attribute type="descriptor" required="no"/>
    </ElementType>
    <ElementType name="AlarmEnumerationEntries" content="eltOnly" model="closed">
      <attribute type="nodeId" required="no"/>
      <element type="AlarmEnumerationEntry" minOccurs="1" maxOccurs="*"/>
    </ElementType>
```

```
    <ElementType name="DeviceTypeVariant" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="deviceTypeVariant" required="yes"/>
        <attribute type="deviceTypeVariantInfo" required="no"/>
        <attribute type="orderCode" required="no"/>
    </ElementType>
    <ElementType name="DeviceTypeVariants" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="DeviceTypeVariant" minOccurs="1" maxOccurs="*"/>
    </ElementType>
<ElementType name="BusRedundancy " content="empty" model="closed">
    <attribute type=" busRedundancy " required="yes"/>
</ElementType>
 <ElementType name="ChannelMode" content="empty" model="closed" >
        <attribute required="no" type="nodeId"/>
        <attribute required="yes" type="channelMode"/>
  </ElementType>
  <ElementType name="ChannelModes" content="eltOnly" model="closed" >
        <attribute required="no" type="nodeId"/>
        <element type="ChannelMode" minOccurs="1" maxOccurs="*"/>
  </ElementType>
  <ElementType name="ChannelInformation" content="eltOnly" model="closed" >
        <attribute required="no" type="nodeId"/>
        <element type="BusCategories" minOccurs="1" maxOccurs="1"/>
        <element maxOccurs="1" minOccurs="1" type=" ChannelModes"/>
  </ElementType>
   <ElementType name="CommunicationTypeEntry" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="communicationType" required="yes"/>
   </ElementType>
   <ElementType name="DeviceIcon" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="deviceGraphicState" required="no"/>
        <attribute type="path" required="yes"/>
   </ElementType>
   <ElementType name="DeviceBitmap" content="empty" model="closed">
            <attribute type="nodeId" required="no"/>
            <attribute type="deviceGraphicState" required="yes"/>
            <attribute type="path" required="yes"/>
   </ElementType>
   <ElementType name="SubstituteType" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="substituteType" required="yes"/>
   </ElementType>
   <ElementType name="LanguageId" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="languageId" required="yes"/>
   </ElementType>
   <ElementType name="SupportedLanguages" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="LanguageId" minOccurs="1" maxOccurs="*"/>
   </ElementType>
   <ElementType name="PhysicalLayer" content="empty" model="closed">
            <attribute type="physicalLayer" required="yes"/>
            <attribute type="physicalLayerName" required="no"/>
   </ElementType>
   <ElementType name="BusCategory" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="busCategory" required="yes"/>
        <attribute type="busCategoryName" required="no"/>
        <element type="CommunicationTypeEntry" minOccurs="0" maxOccurs="*"/>
        <element type="PhysicalLayer" minOccurs="0" maxOccurs="*"/>
   </ElementType>
   <ElementType name="BusCategories" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="BusCategory" minOccurs="1" maxOccurs="*"/>
   </ElementType>
  <!--Definition of a document entry, which specifies a path to a DTM provided document-->
   <ElementType name="DocumentFile" content="empty" model="closed">
        <attribute type="file" required="yes"/>
   </ElementType>
   <ElementType name="DocumentUrl" content="empty" model="closed">
        <attribute type="url" required="yes"/>
   </ElementType>
   <ElementType name="DocumentExe" content="empty" model="closed">
        <attribute type="file" required="yes"/>
        <attribute type="parameters" required="no"/>
   </ElementType>
```

```
<ElementType name="DtmDocument" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="label" required="yes"/>
    <attribute type="help" required="no"/>
   <attribute type="documentClassification" required="yes"/>
    <attribute type="documentLanguageId" required="no"/>
    <group order="one" minOccurs="1" maxOccurs="1">
        <element type="DocumentFile" minOccurs="0" maxOccurs="1"/>
        <element type="DocumentUrl" minOccurs="0" maxOccurs="1"/>
        <element type="DocumentExe" minOccurs="0" maxOccurs="1"/>
    </group>
</ElementType>

<ElementType name="DtmDocuments" content="mixed" model="closed">
    <attribute type="nodeId" required="no"/>
    <group order="many">
        <element type="DtmDocument" minOccurs="1" maxOccurs="*"/>
    </group>
</ElementType>

<ElementType name="DtmDeviceType" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="readAccess" required="no"/>
    <attribute type="writeAccess" required="no"/>
    <attribute type="manufacturerId" required="no"/>
    <attribute type="deviceTypeId" required="no"/>
    <attribute type="subDeviceType" required="no"/>
    <attribute type="deviceTypeInformation" required="no"/>
    <attribute type="deviceTypeInformationPath" required="no"/>
    <attribute type="classificationId" required="no"/>
    <attribute type="dtmDevTypeID" required="no"/>
    <attribute type="dtmDevTypeIDVersion" required="no"/>
    <element type="VersionInformation" minOccurs="1" maxOccurs="1"/>
    <element type="SupportedLanguages" minOccurs="1" maxOccurs="1"/>
    <element type="BusCategories" minOccurs="0" maxOccurs="1"/>
    <element type="DeviceIcon" minOccurs="0" maxOccurs="*"/>
    <element type="DtmDocuments" minOccurs="0" maxOccurs="1"/>
    <element type="DeviceBitmap" minOccurs="0" maxOccurs="*"/>
      <element type="ClassificationIds" minOccurs="0" maxOccurs="1"/>
     <element type="DataSetFormats" minOccurs="0" maxOccurs="1"/>
      <group order="one" minOccurs="0" maxOccurs="1">
        <element type="DeviceTypeVariants"/>
        <element type="DeviceTypeVariant"/>
      </group>
</ElementType>
<!-- Definition of Version Information -->
<ElementType name="VersionInformation" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="readAccess" required="no"/>
    <attribute type="writeAccess" required="no"/>
    <attribute type="name" required="yes"/>
    <attribute type="vendor" required="no"/>
    <attribute type="version" required="no"/>
    <attribute type="date" required="no"/>
    <attribute type="descriptor" required="no"/>
</ElementType>
<ElementType name="NumberData" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="number" required="yes"/>
</ElementType>
<ElementType name="StringData" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="string" required="yes"/>
</ElementType>
<ElementType name="TimeData" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="time" required="yes"/>
</ElementType>
<!-- Definition of Binary Variable -->
<ElementType name="BinaryVariable" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="binData" required="yes"/>
</ElementType>
<!-- Definition of Enumerator Variable -->
<ElementType name="EnumeratorEntry" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="index" required="yes"/>
    <attribute type="name" required="yes"/>
```

```
    <attribute type="descriptor" required="no"/>
</ElementType>
<ElementType name="Variable" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="EnumeratorEntry" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="EnumeratorEntries" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="EnumeratorEntry" minOccurs="1" maxOccurs="*"/>
</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="StringData"/>
        <element type="TimeData"/>
        <element type="ChannelReference"/>
    </group>
</ElementType>
<!-- Definition of UpperRange -->
<ElementType name="UpperRange" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <group order="one" minOccurs="0" maxOccurs="1">
        <element type="NumberData"/>
        <element type="StringData"/>
        <element type="TimeData"/>
        <element type="ChannelReference"/>
    </group>
</ElementType>
<!-- Definition of Ranges -->
<ElementType name="Range" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="LowerRange" minOccurs="1" maxOccurs="1"/>
    <element type="UpperRange" minOccurs="1" maxOccurs="1"/>
    <element type="Unit" minOccurs="0" maxOccurs="1"/>
    <element type="LowerRawValue" minOccurs="0" maxOccurs="1"/>
    <element type=" UpperRawValue " minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="Ranges" content="eltOnly" model="closed">
    <attribute type="readAccess" required="no"/>
    <attribute type="writeAccess" required="no"/>
    <attribute type="nodeId" required="no"/>
    <element type="Range" minOccurs="1" maxOccurs="*"/>
</ElementType>
<!-- Definition of Channel References -->
<ElementType name="ChannelReference" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="idref" required="yes"/>
```

```
        <element type="ChannelInformation" maxOccurs="1" minOccurs="0" />
    </ElementType>
    <ElementType name="ChannelReferences" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="ChannelReference" minOccurs="1" maxOccurs="*"/>
    </ElementType>
    <!-- Definition of Alarms -->
    <ElementType name="StaticValue" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="staticValue" required="yes"/>
    </ElementType>
    <ElementType name="Alarm" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="alarmType" required="yes"/>
        <element type="Unit" minOccurs="0" maxOccurs="1"/>
        <group order="one" minOccurs="0" maxOccurs="1">
            <element type="StaticValue"/>
     <element type="NumberData"/>
     <element type="AlarmEnumerationEntries"/>
            <element type="ChannelReferences"/>
        </group>
    </ElementType>
    <ElementType name="Alarms" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="Alarm" minOccurs="1" maxOccurs="*"/>
    </ElementType>
    <!-- DtmVariable -->
    <ElementType name="Display" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="string" required="yes"/>
    </ElementType>
    <ElementType name="DtmVariableReference" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="reference" required="yes"/>
    </ElementType>
    <!-- StructuredVariable -->
    <ElementType name="StructuredElement" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="bitLength" required="yes"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="Display"/>
            <element type="DtmVariableReference"/>
        </group>
    </ElementType>
    <ElementType name="StructuredElements" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="StructuredElement" minOccurs="1" maxOccurs="*"/>
    </ElementType>
    <ElementType name="StructuredVariable" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="BinaryVariable" minOccurs="1" maxOccurs="1"/>
        <element type="StructuredElements" minOccurs="1" maxOccurs="1"/>
        <attribute type="dataTypeDescriptor" required="no"/>
    </ElementType>
    <ElementType name="Variant" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="dataType" required="yes"/>
        <attribute type="byteLength" required="no"/>
        <attribute type="displayFormat" required="no"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="StringData"/>
            <!-- for types: ascii, packedAscii, password, bitString, hexString -->
            <element type="NumberData"/>
            <!-- for types: float, double, int, unsigned, index -->
            <element type="TimeData"/>
            <!-- for types: date, time, dateAndTime, duration -->
            <element type="EnumeratorVariable"/>
            <!-- for type: enumerator -->
            <element type="BitEnumeratorVariable"/>
            <!-- for type: bitEnumerator -->
            <element type="BinaryVariable"/>
            <!-- for types: binary, dtmSpecific -->
            <element type="StructuredVariable"/>
            <!-- for type: structured -->
        </group>
    </ElementType>
    <!-- SubstituteValue -->
    <ElementType name="SubstituteValue" content="eltOnly" model="closed">
```

```
        <attribute type="nodeId" required="no"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="SubstituteType"/>
            <element type="Variant"/>
        </group>
    </ElementType>
    <!-- Value -->
    <ElementType name="Value" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="readAccess" required="no"/>
        <attribute type="writeAccess" required="no"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="Display"/>
            <element type="Variant"/>
        </group>
    </ElementType>
    <!-- DTMVariableStatus -->
    <ElementType name="StatusInformation" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="readAccess" required="no"/>
        <attribute type="writeAccess" required="no"/>
        <element type="EnumeratorEntry" minOccurs="1" maxOccurs="*"/>
    </ElementType>
    <!-- DtmVariable -->
    <ElementType name="DtmVariable" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="SemanticInformation" minOccurs="0" maxOccurs="*"/>
        <attribute type="name" required="yes"/>
        <attribute type="descriptor" required="no"/>
        <element type="Value" minOccurs="1" maxOccurs="1"/>
        <element type="Unit" minOccurs="0" maxOccurs="1"/>
        <element type="Ranges" minOccurs="0" maxOccurs="1"/>
        <attribute type="statusFlag" required="no"/>
        <element type="StatusInformation" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <ElementType name="DtmVariables" content="mixed" model="closed">
        <attribute type="nodeId" required="no"/>
        <element type="SemanticInformation" minOccurs="0" maxOccurs="*"/>
        <attribute type="name" required="yes"/>
        <attribute type="descriptor" required="no"/>
        <group order="many">
            <element type="DtmVariables" minOccurs="0" maxOccurs="*"/>
            <element type="DtmVariable" minOccurs="0" maxOccurs="*"/>
        </group>
    </ElementType>
    <!-- Communication Data -->
    <ElementType name="CommunicationData" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="byteArray" required="yes"/>
    </ElementType>
    <!-- Definition of FDT specic communication errors for nested communication-->
    <ElementType name="CommunicationError" content="mixed" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="communicationError" required="yes"/>
        <attribute type="tag" required="yes"/>
        <attribute type="errorCode" required="no"/>
        <attribute type="descriptor" required="no"/>
    </ElementType>
</Schema>
```


B.2 FDTApplicationIdSchema

The application id schema is used as a global FDT definition. The application id of this schema is reference via the prefix fdtappid: within the other schemas. The appearance and the functionality of a DTM user interface is controlled by the entry of the element applicationId, functionId, and operationPhase.

Attribute	Description
applicationId	Identification of the current application context (The enumeration refers to the DTM Realization Elements)

Tag	Description
ApplicationId	FDT global application id coded as an enumeration
FDTApplicationIds	Collection of application id
FDT	Root tag

```
<Schema name="FDTApplicationIdSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
    <!--Definition of Attributes-->
    <AttributeType name="applicationId" dt:type="enumeration" dt:values="fdtAdjustSetValue fdtAssetManagement fdtAuditTrail fdtConfiguration fdtDiagnosis fdtForce fdtManagement fdtObserve fdtOfflineCompare fdtOfflineParameterize fdtOnlineCompare fdtOnlineParameterize fdtIdentify fdtCalibration fdtMainOperation fdtNetworkManagement"/>
    <!-- ApplicationId specifies the standard user interface called -->
    <ElementType name="ApplicationId" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="applicationId" required="yes"/>
    </ElementType>
    <!-- FDTApplicationIds: a list of application -->
    <ElementType name="FDTApplicationIds" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="ApplicationId" minOccurs="0" maxOccurs="*"/>
    </ElementType>
    <!-- main FDT element -->
    <ElementType name="FDT" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="FDTApplicationIds" minOccurs="1" maxOccurs="1"/>
    </ElementType>
</Schema>
```

Example:

See usage within DTMFunctionsSchema

B.3 FDTUserInformationSchema

Used at: IDtm::Config()

The user information schema is used as a global FDT definition. It informs the DTM during initialization about the role and the rights of the current user.

Attribute	Description
administrator	Flag describing if the user has administrative right (default to “FALSE”)
loginLocation	Description of the location the user logged in
loginTime	Time of last login
oemService	Flag describing if the user has OEM service rights (default to “FALSE”)
projectName	Unique identifier for the project within the name space of the Frame Application
sessionDescription	Description of the user session (may be given by the user)
userLevel	User level specifying users rights
userName	Name of the current user

Tag	Description
FDTUserInformation	Description of the user role
FDT	Root tag

```
<Schema name="FDTUserInformationSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes"  xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
    <!--Definition of Attributes-->
```

```
    <AttributeType name="administrator" dt:type="boolean" default="0"/>
    <AttributeType name="loginLocation" dt:type="string"/>
    <AttributeType name="loginTime" dt:type="dateTime"/>
    <AttributeType name="oemService" dt:type="boolean" default="0"/>
    <AttributeType name="projectName" dt:type="string"/>
    <AttributeType name="sessionDescription" dt:type="string"/>
    <AttributeType name="userLevel" dt:type="enumeration" dt:values="observer operator maintenance
planningEngineer"/>
    <AttributeType name="userName" dt:type="string"/>
    <!-- FDTUserInformation -->
    <!--the contents of an FDTUserInformation instance lies in the responsibility of the frame-application only -->
    <ElementType name="FDTUserInformation" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="projectName" required="yes"/>
        <!-- project name -->
        <attribute type="userName" required="yes"/>
        <!-- user name -->
        <attribute type="userLevel" required="yes"/>
        <!-- user level as defined within the FDT sepcification -->
        <attribute type="oemService" required="no"/>
        <!-- oemService set to True enables OEM access -->
        <attribute type="administrator" required="no"/>
        <!-- administrator set to True enables administrative access -->
        <attribute type="loginTime" required="no"/>
        <!-- time and date of login for this session -->
        <attribute type="loginLocation" required="no"/>
        <!-- station description at which the user loged 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>
```

B.4 DTMInformationSchema

Used at: IDtmInformation::GetInformation()

The XML document provides DTM specific information.

Attribute	Description
major	Major part of the FDT-Specification version on which the implementation of a DTM is based. For FDT-Specification 1.2.1.0 it must be set to '1'
minor	Minor part of the FDT-Specification version on which the implementation of a DTM is based. For FDT-Specification 1.2.1.0 it must be set to '2'
release	Release part of the FDT-Specification version on which the implementation of a DTM is based. For FDT-Specification 1.2.1.0 it must be set to '1'. It is highly recommended to use this attribute
build	Build part of the FDT-Specification version on which the implementation of a DTM is based. For FDT-Specification 1.2.1.0 it must be set to '0'. It is highly recommended to use this attribute