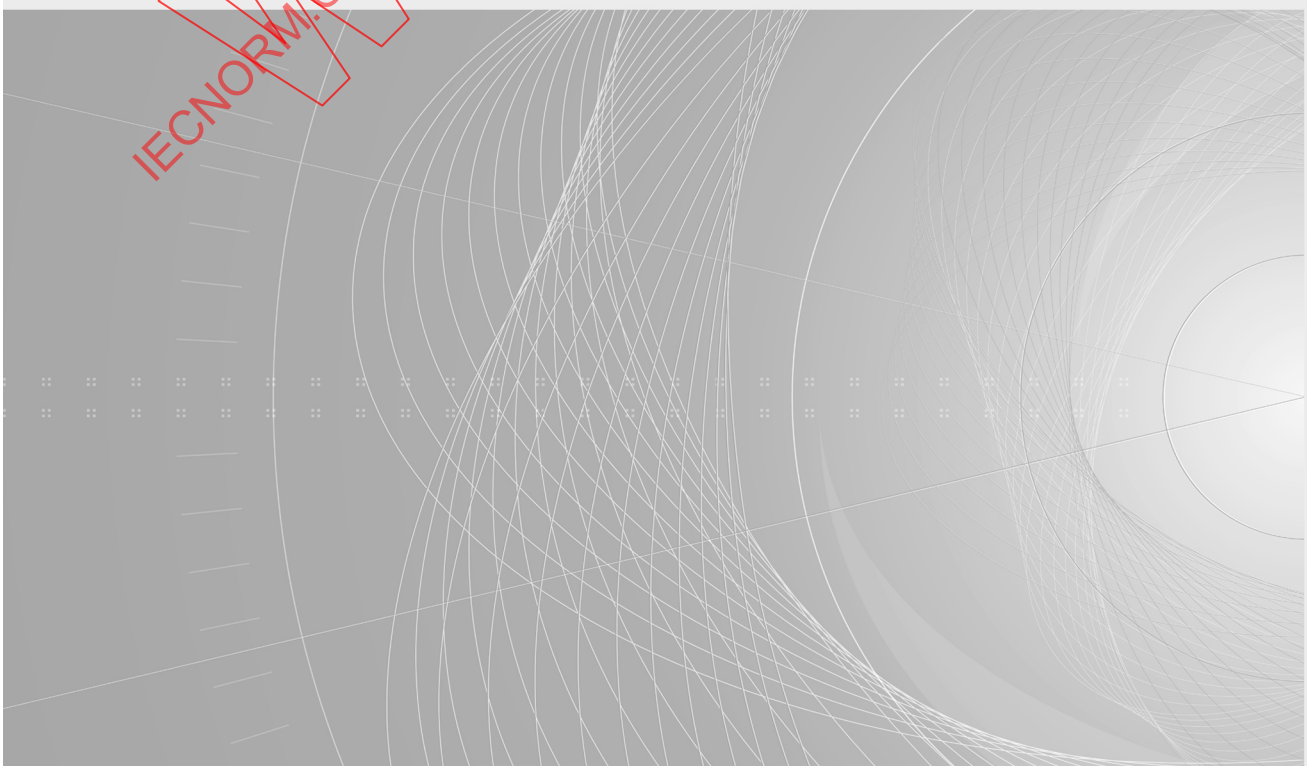
IEC/TR 62453-41

Edition 1.0 2009-08

TECHNICAL REPORT

colour inside

Field device tool (FDT) interface specification – Part 41: Object model integration profile – Common object model



IEC/TR 62453-41:2009(E)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: www.iec.ch

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

- Catalogue of IEC publications: www.iec.ch/searchpub
The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub
Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv
If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:
Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

IEC/TR 62453-41

Edition 1.0 2009-08

TECHNICAL REPORT

colour inside

Field device tool (FDT) interface specification – Part 41: Object model integration profile – Common object model

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE XH

ICS 25.040.40; 35.100.05; 35.110

ISBN 978-2-88910-717-9

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

FIELD DEVICE TOOL (FDT) INTERFACE SPECIFICATION –

Part 41: Object model integration profile – Common object model

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.

The main task of IEC technical committees is to prepare International Standards. However, a technical committee may propose the publication of a technical report when it has collected data of a different kind from that which is normally published as an International Standard, for example "state of the art".

IEC/TR 62453-41, which is a technical report, has been prepared by subcommittee 65E: Devices and integration in enterprise systems, of IEC technical committee 65: Industrial-process measurement, control and automation:

This part, in conjunction with the other parts of the first edition of the IEC 62453 series cancels and replaces IEC/PAS 62453-1, IEC/PAS 62453-2, IEC/PAS 62453-3, IEC/PAS 62453-4 and IEC/PAS 62453-5 published in 2006, and constitutes a technical revision.

The text of this technical report is based on the following documents:

Enquiry draft	Report on voting
65E/64/DTR	65E/113/RVC

Full information on the voting for the approval of this technical report can be found in the report on voting indicated in the above table.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts of the IEC 62453 series, under the general title *Field Device Tool (FDT) interface specification,* can be found on the IEC website.

The committee has decided that the contents of this publication will remain unchanged until the maintenance result date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

A bilingual version of this publication may be issued at a later date.

IMPORTANT – The "colour inside" logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this publication using a colour printer.

INTRODUCTION

This part of IEC 62453 is an interface specification for developers of FDT (Field Device Tool) components for function control and data access within a client/server architecture. The specification is a result of an analysis and design process to develop standard interfaces to facilitate the development of servers and clients by multiple vendors that need to interoperate seamlessly.

With the integration of fieldbusses into control systems, there are a few other tasks which need to be performed. In addition to fieldbus- and device-specific tools, there is a need to integrate these 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.

Figure 1 shows how IEC/TR 62453-41 is incorporated in the structure of the IEC 62453 series.

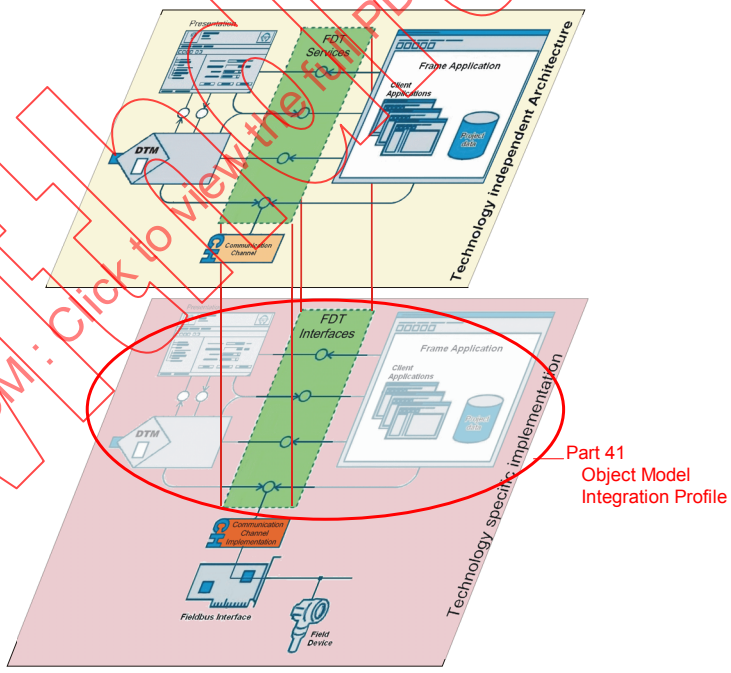


Figure 1 – Part 41 of the IEC 62453 series

FIELD DEVICE TOOL (FDT) INTERFACE SPECIFICATION –

Part 41: Object model integration profile – Common object model

1 Scope

This part of IEC 62453, which is a Technical report, defines how the common FDT principles are implemented based on the MS COM technology, including the object behavior and object interaction via COM interfaces.

This part specifies the technology specific implementation of the protocol specific functionality and communication services.

This part of IEC 62453 is informative, however when this part is applied its requirements shall be implemented as specified.

This part specifies FDT version 1.2.1.

2 Normative references

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

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

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

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

3 Terms, definitions, symbols, abbreviated terms and conventions

3.1 Terms and definitions

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

3.1.1
ActiveX®
GUI component technology based on the Microsoft Component Object Model (COM/DCOM)

NOTE Former standard was OLE controls (OCX).

3.1.2
asynchronous function
a non-blocking function, the calling process continues execution while the function is executed in the background

3.1.3
CLSID
UUID for a COM class

3.1.4
ProgID
human readable ID for a COM class

3.1.5
synchronous function
a blocking function, the calling process stops execution while the function is executed

3.2 Abbreviations

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

API	Application Programming Interface
BTM	Block Type Manager
CLSID	Class ID
DCOM	Distributed COM
DLL	Dynamic Linked Library
DOM	Document Object Model
DTM	Device Type Manager
FA	Frame Application
FDT	Field Device Tool
GUI	Graphical User Interface
GUID	Globally Unique Identifier (a UUID)
HART®	Highway Addressable Remote Transducer
IID	Interface ID
LCID	Locale ID
MIDL	Microsoft Interface Definition Language
MSDN®	Microsoft Developer Network
OCX	OLE Controls
OLE	Object Linking and Embedding
ProgID	Programmatic ID
XDR	XML Data Reduced
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations

3.3 Conventions

The conventions for UML notation used in this document are defined in IEC 62453-1.

4 Implementation concept

4.1 Technological orientation

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 (originally: OLE for Process Control) definition.

Within IEC/TR 62453-41 implementation of FDT only user 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 implementation of FDT´s client/server architecture defined in this Technical report is based on Microsoft COM.

This part of IEC 62453 specifies COM interfaces (what the interfaces are), not the implementation (not the "how" of the implementation) of those interfaces. It specifies the behavior that the interfaces are expected to provide to client applications that use them. The FDT-specification neither specifies the implementation of DTMs nor the implementation of Frame Applications.

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

4.2 Implementation of abstract FDT object model

4.2.1 General

The FDT objects are implemented as COM objects. The general expectation is, that these objects may be implemented as inproc as well as outproc servers. The Frame Application is responsible to organize the execution of the objects in a distributed system (based on a vendor-specific implementation).

4.2.2 FDT Frame Application (FA)

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 2).

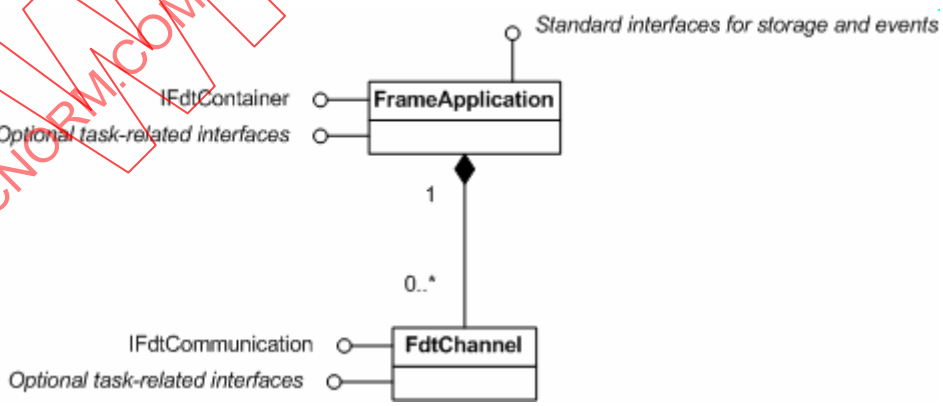


Figure 2 – Frame Application interfaces

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.

4.2.3 Device Type Manager (DTM)

4.2.3.1 DTM interfaces

The services of a DTM are provided by means of COM interfaces, see Figure 3.

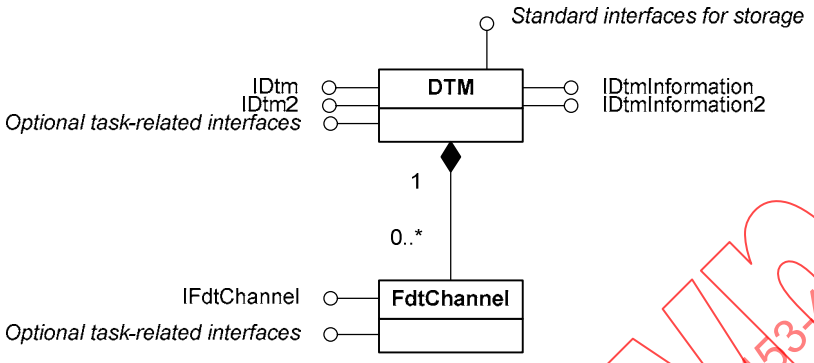


Figure 3 – DTM interfaces

The interfaces IDtm and IDtmInformation have to be implemented by each DTM (see 5.2). These interfaces provide the services 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 corresponding device. Each interface is described in detail, see the appropriate subclause.

4.2.3.2 Block Type Manager (BTM)

The BTM implements similar interfaces to those specified for a DTM. Block specific schemas replace device related XML schemas to provide block information. For example, BtmInformationSchema replaces DTMInformationSchema.

4.2.4 Presentation object

A presentation object as defined by this specification may be an ActiveX object (allows integration into GUI of FA) or a standalone program, that may be integrated by a DTM.

IEC/TR 62453-61 provides a style guide for a common look and feel of ActiveX user interfaces.

4.2.5 FDT-Channel object

The FDT-Channel 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 shall implement further interfaces for the communication via this channel (Communication Channel). Each interface of the FDT-Channel object is described in detail, see the appropriate subclause.

As defined in IEC 62453-2, Communication Channels represent the gateway from the FDT-specific to the Frame Application-specific communication. At least the interface IFdtCommunication shall be implemented for Communication Channels.

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.

4.3 Object interaction

4.3.1 Parameter interchange via XML

Data exchange between objects is implemented based on transport of XML documents (XDR format) via COM interfaces. One example for this data exchange is the parameter interchange via XML is to provide a way to exchange information between Frame Application and DTMs (see Figure 4). Typically, in process control systems, multiple use cases 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. It is recommended that data is stored locally in the fashion that makes the most sense. XML provides an extendable standard to connect FDT components. The data exchange is done via XML documents. Within these documents XML tags are used to delimit pieces of data. XML leaves the interpretation of the data to the application that reads it. To get a common understanding of the exchanged data FDT uses XML schemas for validation. For the data access are standardized tools like the DOM (W3C's document object model) available.

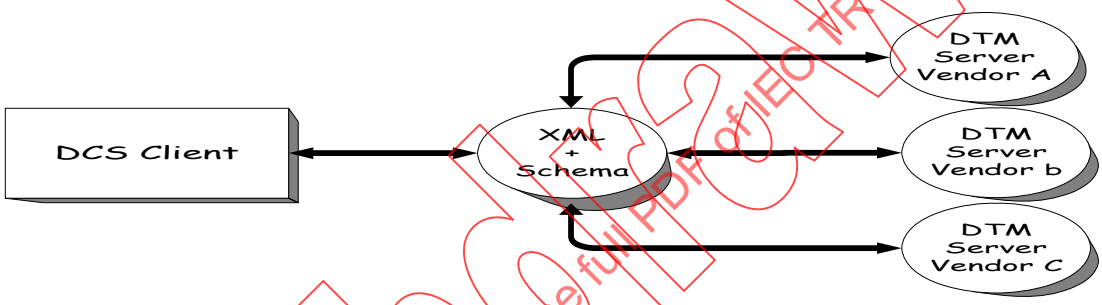


Figure 4 – FDT Client/server relationship via XML

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 accessible in VC++ , Visual Basic and VBScript.

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

It is recommended that the FDT developer always work with the DOM because

- the XML parser generates the DOM from the transferred XML data,
- the schemas ensure 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.

The validation of a XML document ensures that the content of an attribute is valid according to the XML schemas.

- Non string data types (Ui4,enumeration,…): Empty or invalid attribute values will be detected during the validation of the document.
- Data types string and bin.hex: Empty values are possible. No parsing error is generated.

To avoid interoperability problems developers should consider following basic rules.

- Receiving XML documents:
  To ensure a robust implementation, developers should be aware that XML documents with empty attributes of type ‘string’ or ‘bin.hex’ could be received. FDT components should be capable to handle this in a proper way.
- Providing XML documents:
  Empty optional attributes should be avoided generally, because they are a potential cause of interoperability problems and a waste of resources. Also optional XML elements should be removed if they contain no information. Exception: Empty ‘string’ attributes can make sense in some cases (as shown in the example below).

If a Frame Application can not provide information about login location and session description, the corresponding attributes should be removed.

NOTE

If a Frame Application can not provide attribute values, the attributes should not be provided:

Negative Example:
```
<FDT    xmlns="x-schema:FDTUserInformationSchema.xml"    xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
        <FDTUserInformation projectName="Project1" userName="ThisUser" userLevel="maintenance"
                            loginLocation="" sessionDescription=""/>
</FDT>
```

Positive Example:
```
<FDT    xmlns="x-schema:FDTUserInformationSchema.xml"    xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
        <FDTUserInformation projectName="Project1" userName="ThisUser" userLevel="maintenance"/>
</FDT>
```

If the session description is generally available, but the user has not typed in something, then it makes sense to use an empty attribute. Acceptable Example:

```
<FDT    xmlns="x-schema:FDTUserInformationSchema.xml"    xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
        <FDTUserInformation projectName="Project1" userName="ThisUser" userLevel="maintenance"
                            sessionDescription=""/>
</FDT>
```

4.3.2 Examples of usage

Parameter interchange between DTM and Frame Application is done via XML document. Object oriented access to data is provided when using XML parser that generates an in-memory representation of the XML data (e.g. a DOM). Instance data to be stored (persistence) can also be handled as XML document to simplify the DTM development and to have a homogeneous data handling within a DTM. But also if the data are stored as XML the content of this data is only known by DTM (see Figure 5).

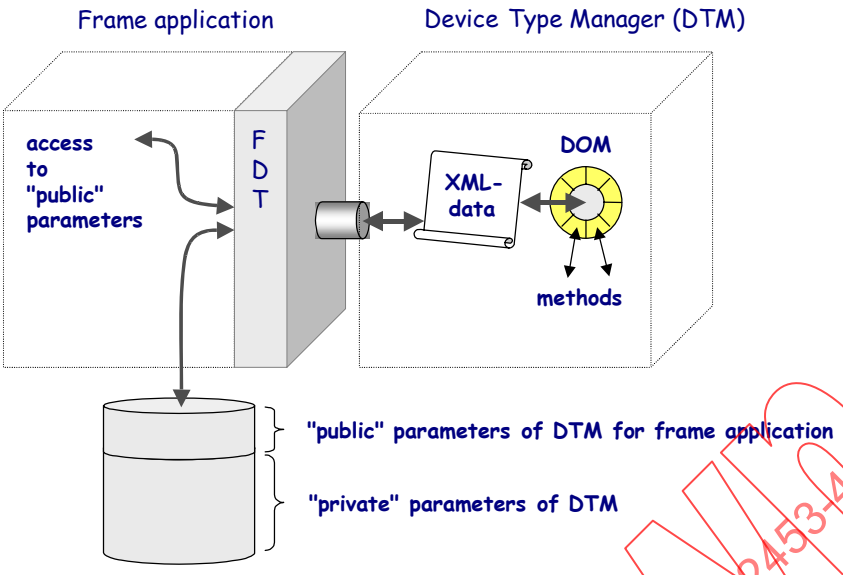


Figure 5 – Data access and storage

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 6).

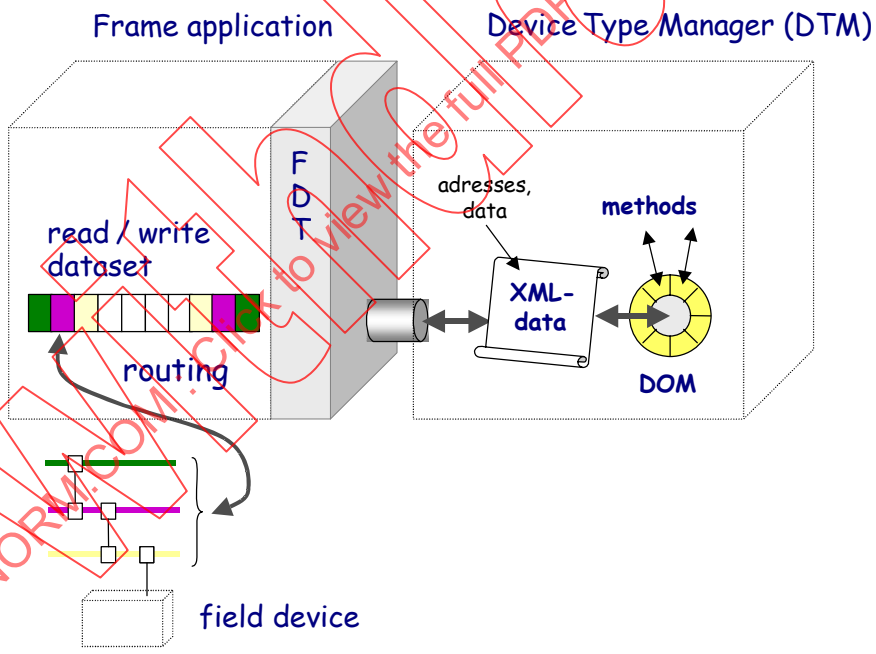


Figure 6 – Communication

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 7). A default style sheet is supported by the Frame Application.

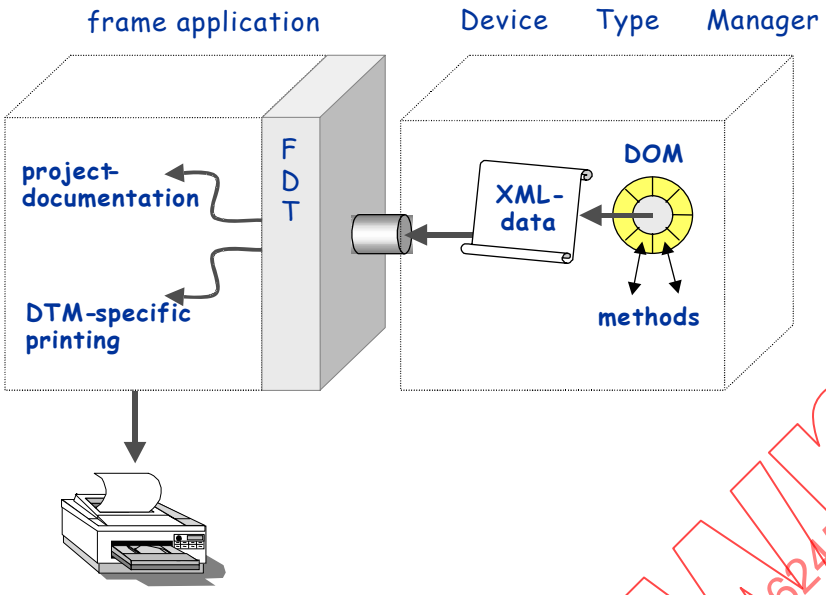


Figure 7 – Documentation

In case of failsafe field devices the instance data that are stored in the controller (e.g. PLC) after upload from the field device have to be verified by the DTM (see Figure 8). Interchange format of this data is also an XML document.

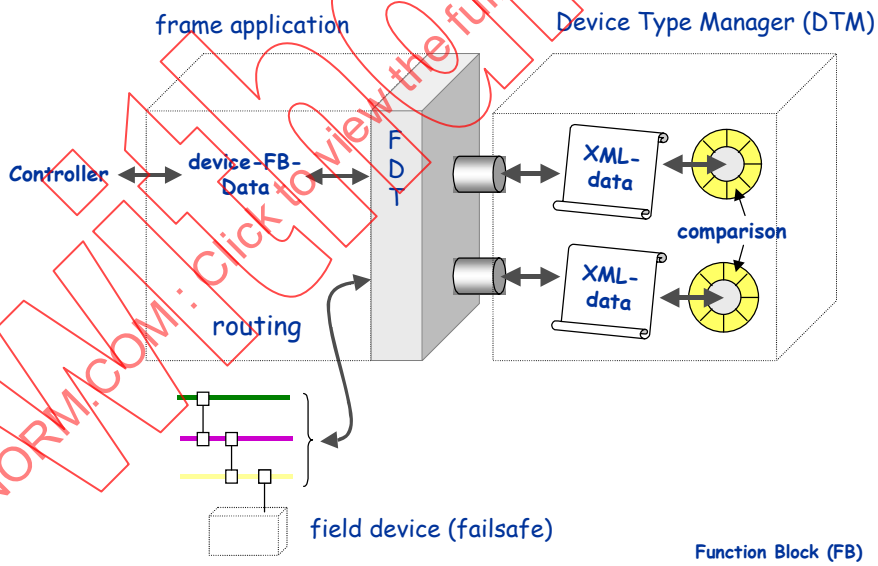


Figure 8 – Parameter verification in case of failsafe devices

4.4 Implementation of DTM data persistence and synchronization

4.4.1 Persistence overview

The synchronization of the DTM data set 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 shall 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. Some DTMs do not support reload of a DTM object by calling IPersistXXX::Load several times. With an IPersistXXX:Save request a DTM shall store its private data within the storage provided by the Frame Application. A DTM object for a new instance shall be initialized if the IPersistXXX:InitNew method is called by the Frame Application.

DTMs using additional own data storage shall provide all data which are necessary for commissioning via the IPersistXXX interface. Private data not provided via the IPersistXXX interface shall 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 shall be able to handle both.

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

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).

4.4.2 Persistence interfaces

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

4.5 DTM state machine

The following state machine shows the different states of a DTM (see Figure 9). The state machine is based on the general state machine as defined in IEC 62453-2. It is extended to accommodate the specific needs of COM based implementation.

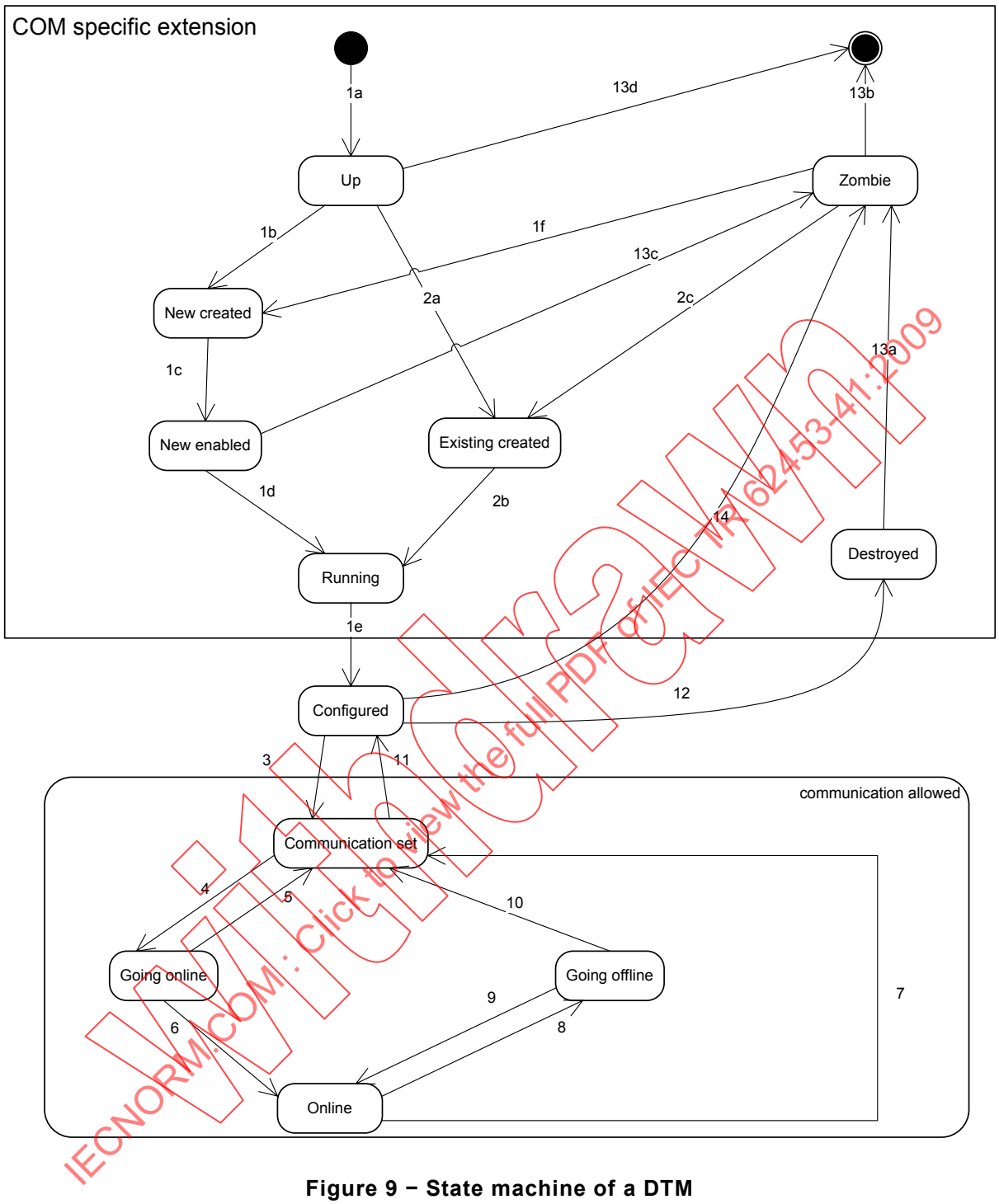


Figure 9 − State machine of a DTM

Table 1 provides a description of the transitions.

Table 1 – Definition of DTM state machine

ID	Start state	End state	Trigger	Condition	Action
1a	<non existent>	Up	CoCreateInstance()		
1b	Up	New created	IPersistXXX:InitNew()		
1c	New created	New enabled	IDtm:Environment() or IDtm2:Environment2()		
1d	New enabled	Running	IDtm:InitNew()		

ID	Start state	End state	Trigger	Condition	Action
1e	Running	Configured	IDtm:Config()		
1f	zombie	New created	IPersistXXX:InitNew()		
2a	Up	Existing created	IPersistXXX:Load()		
2b	Existing created	Running	IDtm:Environment() or IDtm2:Environment2()		
2c	Zombie	Existing created	IPersistXXX:Load()		
3	Configured	Communication set (not connected)	IDtm: SetCommunication()		
4	Communication set (not connected)	Going online (connecting)	Trigger of Online service of the DTM		IDtmCommunication: ConnectRequest()
5	Going online (connecting)	Communication set (not connected)	IDtmCommunicationEvent: OnConnectResponse() or IDtmCommunicationEvent2: OnConnectResponse2()	Connection could not be established, negative response	
6	Going online (connecting)	Online	IDtmCommunicationEvent: OnConnectResponse() or IDtmCommunicationEvent2: OnConnectResponse2()	Connection was established, positive response	
7	Online (connected)	Communication set (not connected)	IDtmCommunicationEvent: OnAbort()		
8	Online (connected)	Going offline (disconnecting)	Online service of DTM is finished or IDtm: PrepareToReleaseCommunication()		IDtmCommunication: DisconnectRequest()
9	Going offline (disconnecting)	Online (connected)	IDtmCommunicationEvent: OnDisconnectResponse()	Connection could not be terminated	
10	Going offline (disconnecting	Communication set (not connected)	IDtmCommunicationEvent: OnDisconnectResponse()	Connection was terminated	If PreparedTo-Release-Communication was called (see transition 8): IDtmEvents:On-PreparedToRelease-Communication()
11	Communication set (not connected)	Configured	IDtm: ReleaseCommunication()		
12	Configured	Destroyed	IDtm:PrepareToDelete()		
13a	Destroyed	Zombie	IDtm:PrepareToRelease()		IDtmEvents: OnPreparedToRelease()
13b	Zombie	<NonExisting>	Release()		
13c	new enabled	Zombie	IDtm:PrepareToRelease()		IDtmEvents: OnPreparedToRelease()
13d	Up	<NonExisting>	Release()		
14	Configured	Zombie	IDtm:PrepareToRelease()		IDtmEvents: OnPreparedToRelease()

5 General concepts

5.1 General

This clause 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.

5.2 Overview of task related FDT interfaces

All FDT interfaces are task related. Each object shall implement a mandatory set of interfaces expected by all other objects. By implementing optional FDT interfaces an object is able to support additional functionality, for example 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 2).

Table 2 – Task related DTM interfaces

Device Type Manager	Availability	User interface	ActiveX control user interface	Device with online data	Gateway DTM	Communication DTM for PC/Fieldus adapter
IPersistXXX	Mandatory					
IDtm	Mandatory					
IDtm2	Mandatory					
IDtmActiveXInformation	Optional		Mandatory			
IDtmApplication	Optional	Mandatory				
IDtmChannel	Optional				Mandatory	Mandatory
IDtmDocumentation	Mandatory					
IDtmDiagnosis	Mandatory					
IDtmImportExport	Optional					
IDtmInformation	Mandatory					
IDtmInformation2	Mandatory					
IDtmOnlineDiagnosis	Mandatory					
IDtmOnlineParameter	Optional			Mandatory		
IDtmParameter	Mandatory					
IFdtCommunicationEvents	Optional			Mandatory	Mandatory	
IFdtCommunicationEvents2	Optional			Mandatory	Mandatory	
IDtmHardwareIdentification	Optional					
IDtmSingleDeviceDataAccess	Optional			Mandatory		
IDtmSingleInstanceDataAccess	Mandatory					
IFdtEvents	Mandatory					

The mandatory interfaces of a DTM ActiveX are shown in Table 3.

Table 3 – 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 4).

Table 4 – Task related FDT-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 5.

Table 5 – Task related Channel ActiveX interfaces

FDT Channel ActiveX control	Availability
IfdtChannelActiveXControl	Mandatory
IfdtChannelActiveXControl2	Mandatory

The mandatory interfaces of a BTM are shown in Table 6.

Table 6 – 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

BTM	Availability
IBtmParameter	Mandatory
IFdtCommunicationEvents	Mandatory
IFdtCommunicationEvents2	Mandatory
IDtmHardwareIdentification	Mandatory
IDtmSingleDeviceDataAccess	Mandatory
IDtmSingleInstanceDataAccess	Mandatory
IFdtEvents	Mandatory

The mandatory interfaces of a BTM ActiveX are shown in Table 7.

Table 7 – Task related BTM ActiveX interfaces

BTM ActiveX control	Availability
IBtmActiveXControl	Mandatory

Depending on the functionality of a Frame Application, not all interfaces defined for a Frame Application shall be supported (see Table 8).

Table 8 – 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.

5.3 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) shall 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.

5.4 Dual interfaces

All interfaces defined within the FDT specification are implemented as dual interfaces. This decision was made to support C++, Visual Basic , Java and other COM compliant development languages. The functionality of an object is implemented in separate task oriented interfaces, so that only the default interface is accessible via the dispatch interface. This prevents 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.

5.5 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 document 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.

5.6 Asynchronous versus synchronous behavior

In general each function call is synchronous. Within FDT there are two special cases of asynchronous behavior.

- After starting the user interface of a DTM, the DTM works asynchronous to the Frame Application. Asynchronous in this case means that the user works with the DTM and the Frame Application is the server for communication and data access. This state ends by a notification to the Frame Application, when the DTM closes the user interface.
- While a DTM has opened its user interface, the DTM uses the asynchronous behavior at the communication interface. The time a communication function call needs to return depends on the system topology and the bus protocol and would block an application for seconds. Dividing a communication function call to a request and a response function causes a non-blocking behavior without the pain of multi-threading implementation. The DTM sends its request or several requests without being disturbed by incoming responses. When a response is available, the DTM gets a notification and can receive the response from the communication component. Due to this mechanism, on one hand a DTM should not implement a timeout control and on the other hand the communication has to provide a response for each request. Only a response can contain the timeout information.

5.7 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.

5.8 Implementation of DTM, DTM device type and hardware identification information

5.8.1 Device identification

Figure 10 shows how the identity information provided by the DTM and the identity information provided as scan result is converted and used for comparison.

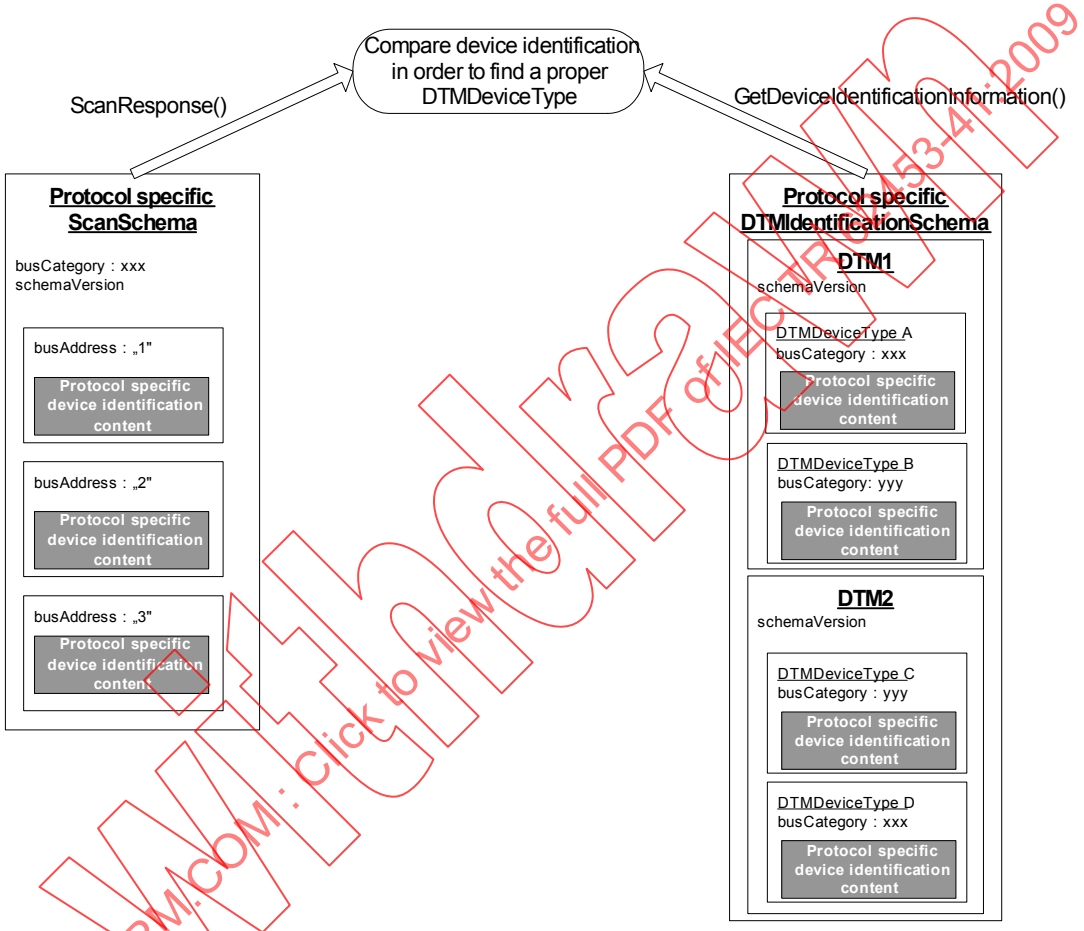


Figure 10 − Device identification

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 shall be converted to a fieldbus independent format using a fieldbus specific XSL transformation.

Figure 11 shows, how protocol specific schemas are integrated in the FDT specification and used by Frame Application and DTMs (example HART® is shown):

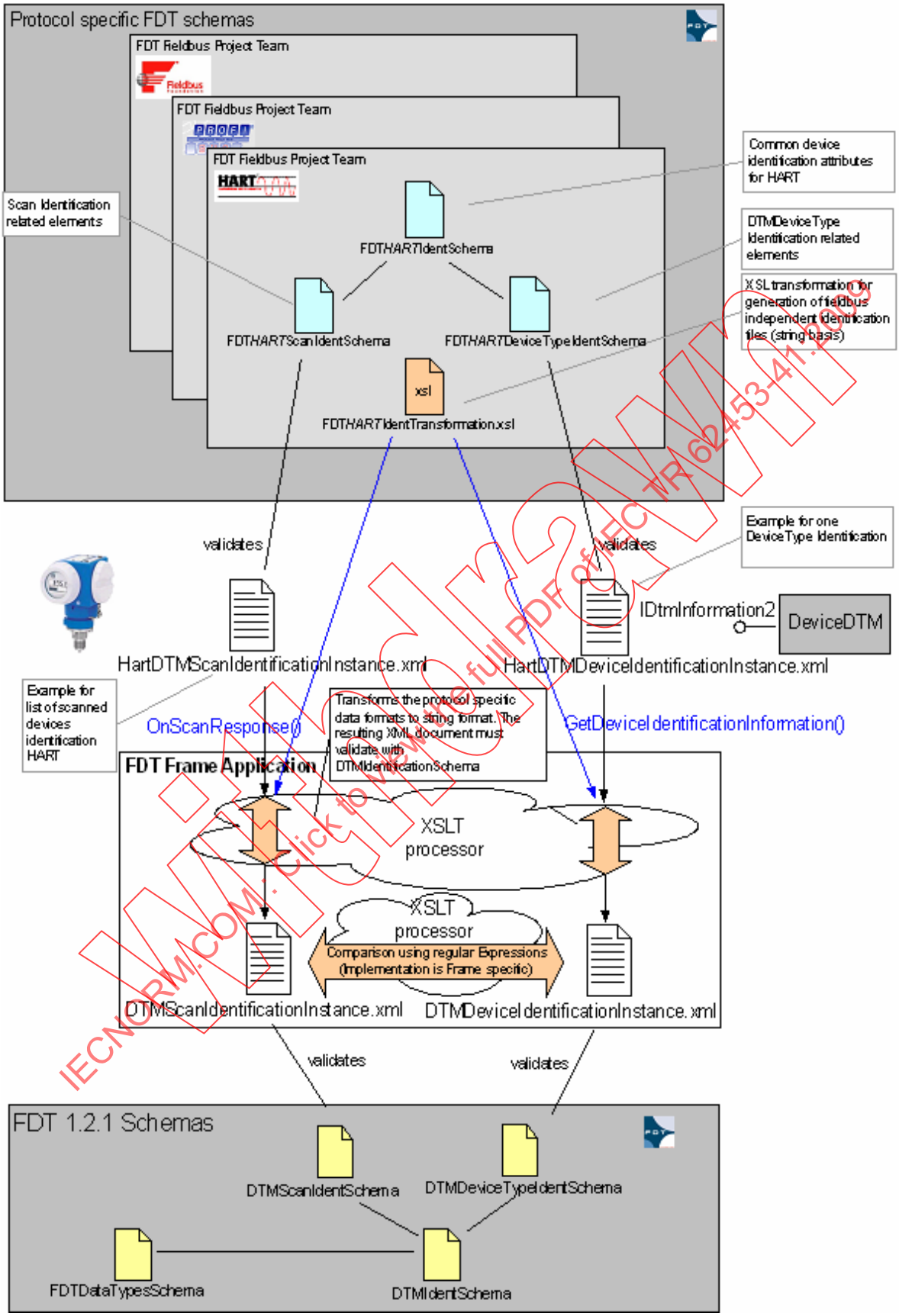


Figure 11 − Structural overview

5.8.2 Protocol specific transformation style sheet (xsl)

As shown in the structural overview Figure 11, 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 shall 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 shall implement a pattern matching according to the regular expression syntax defined in 5.8.5.

5.8.3 Semantic identification information

Table 9 lists identification elements, which have to be provided by the scan and DTM identification mechanism. After xsl transformation, the following values shall be available in string format with the elements listed below in DTMScanIdentSchema and DTMDeviceTypeIdentSchema.

Table 9 − 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	(p)	(p)
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	?
IdDeviceTag	Tag name, which set in the physical device or blok	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 shall always be combined with manufacturerID and deviceTypeID.	x	-
IdDTMSupportLevel	Enum: genericSupport, profileSupport, blockspecificProfileSupport, specificSupport(=default)	-	x

Key
x shall be provided
- is not to be provided
? optional
(p) optional, but may be defined mandatory for a specific protocol (see parts 3xy)

If a semantic element cannot be defined for a fieldbus protocol, the value shall be set to 'NOT_APPLICABLE'.

5.8.4 Device assignment

The comparison of scan result with DTM device identification information and the device assignment is done by the Frame Application based on its 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 shall match to the value within the scan document.

5.8.5 Regular expression specification

If the element of the DTM device identification document contains a pattern, the Frame Application shall use regular expressions to compare scan and DTM device identification information, see Table 10.

Table 10 − 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 shall be used: "\\+", "\\a", "<{.*?}>.*?</\\0>".
$	At the end of a regular expression, this character matches the end of the input. Example: [0-9]$ matches a digit at the end of the input.
|	Alternation operator: separates two expressions, exactly one of which matches (for example, T|the matches "The" or "the")
!	Negation operator: the expression following ! does not match the input. Example: a!b matches "a" not followed by "b"

5.9 Implementation of slave redundancy

5.9.1 General

Implementation of slave redundancy is defined in IEC 62453-2.

5.9.2 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.

6 Implementation of FDT services: FDT interfaces

6.1 Overview of the FDT interfaces

The FDT interface specification includes the following:

- 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 shall 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 shall implement.

6.2 FDT objects

6.2.1 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 Frame Application 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 Figure 12 and Figure 13. After the ActiveX is created by the Frame Application, the DTM cooperates with the ActiveX object.

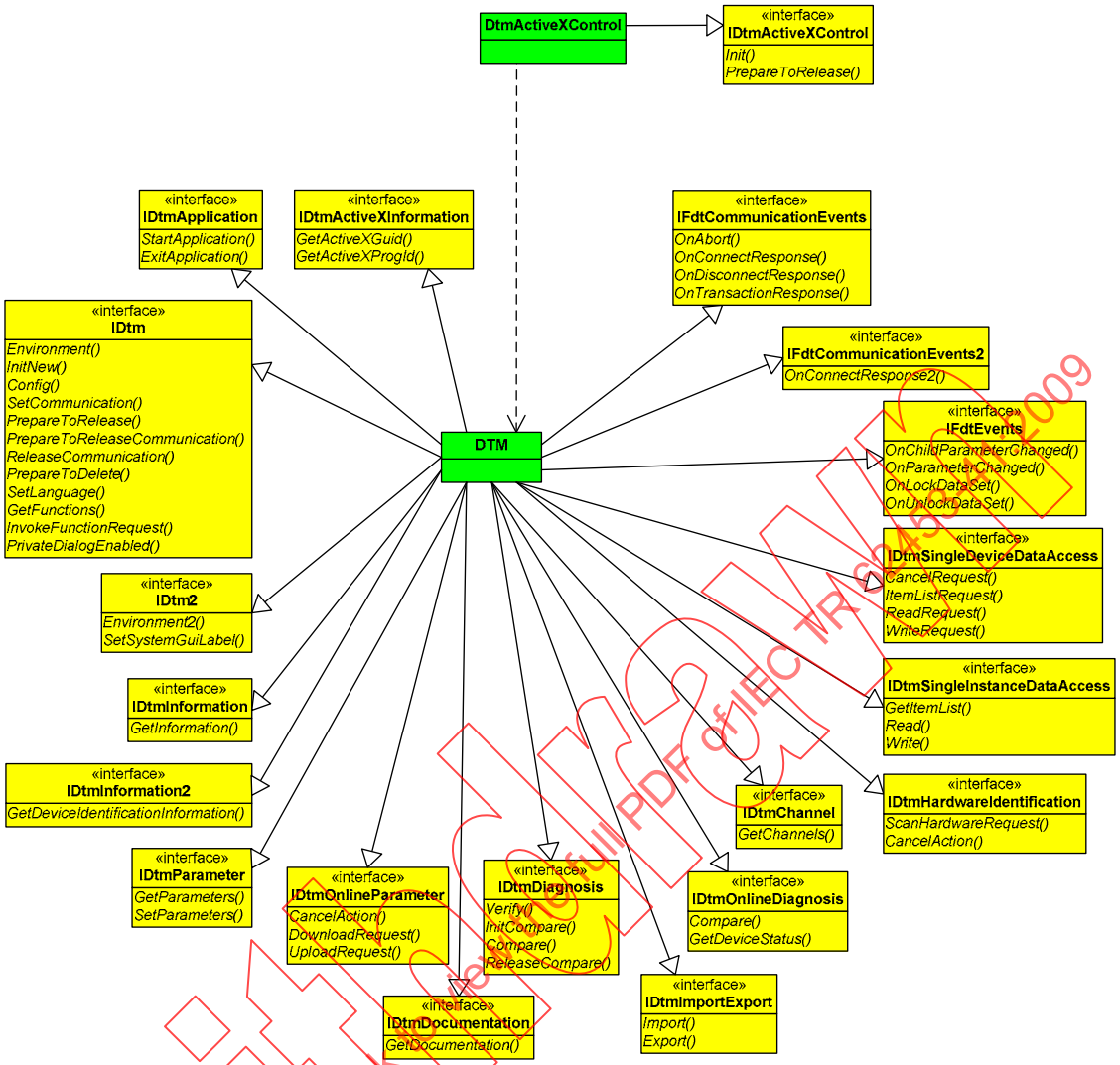


Figure 12 – Interfaces of FDT objects – DTM and DtmActiveXControl

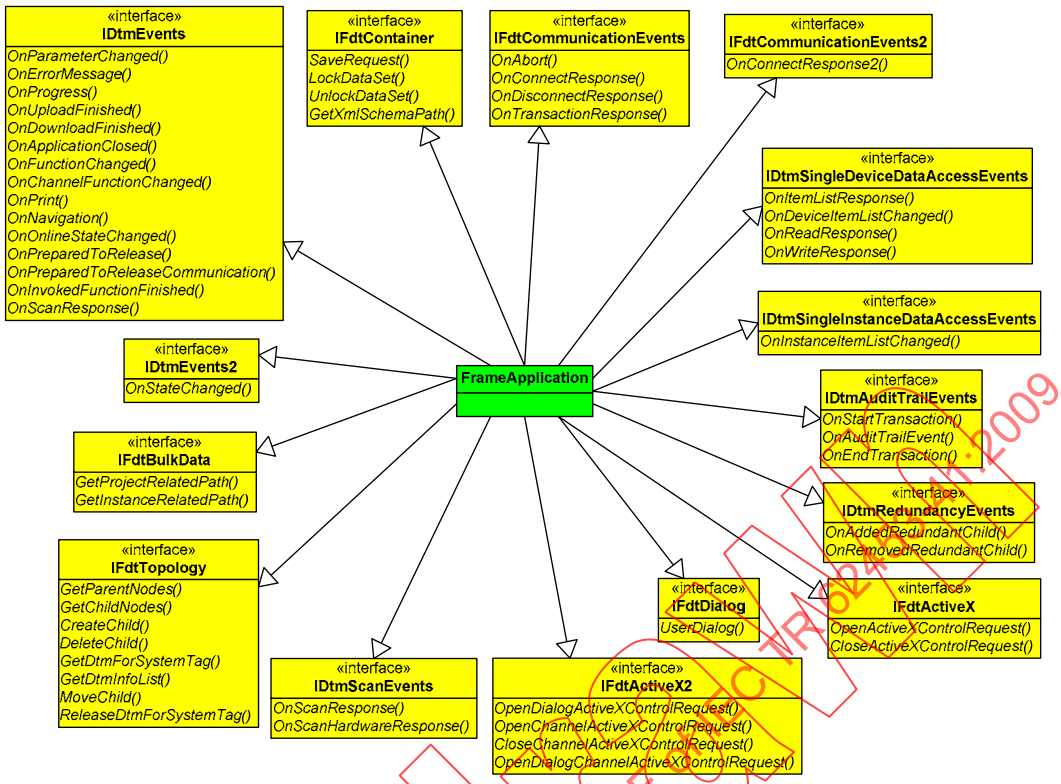


Figure 13 – Interfaces of FDT object – Frame Application

The interfaces provided by a channel are shown in Figure 14. The DTM accesses communication by sending requests to the interface IFdtCommunication and receives responses by the interface IFdtCommunicationEvents. A hierarchy of channels and DTMs is used to provide nested communication.

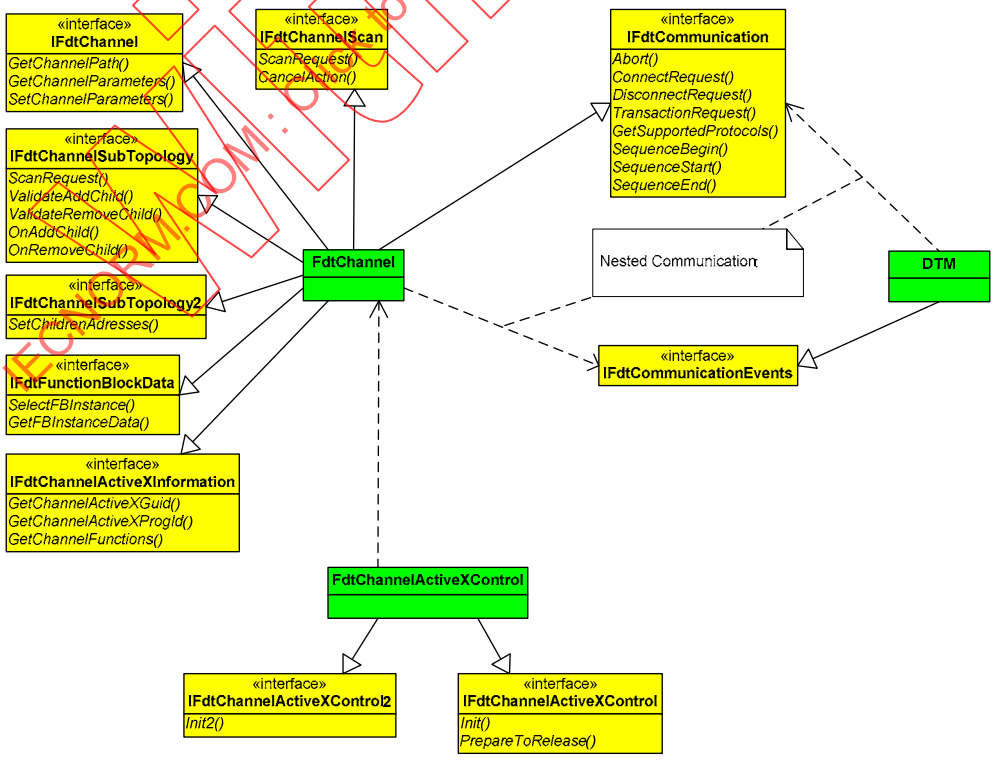
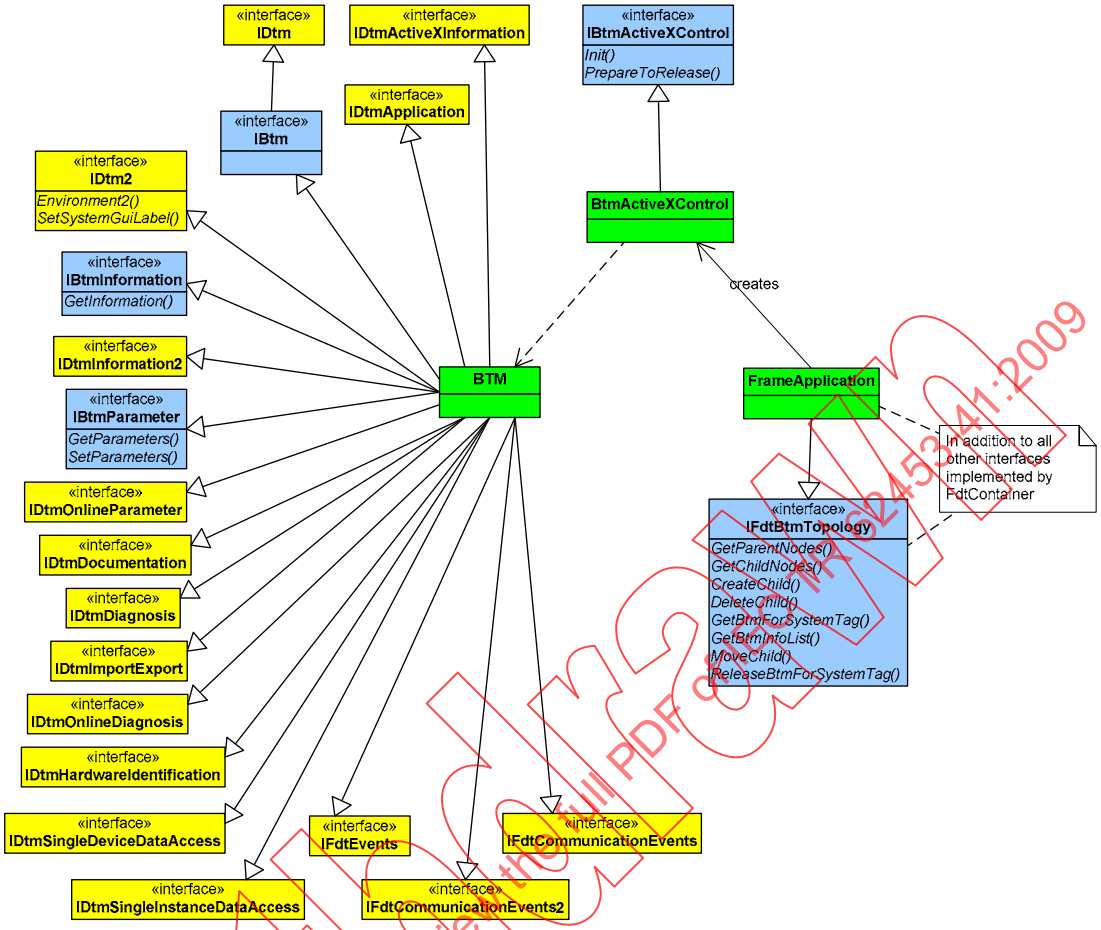


Figure 14 – FDT objects – FDT-Channel

The interfaces related to a BTM are shown in Figure 15.



Key

indicates interfaces specific for use with BTMs.

Figure 15 – FDT objects – BTM and BtmActiveXControl

6.2.2 Avalability of interface methods

The availability of interface methods of the different objects may depend on the state of a DTM instance. This dependency is defined in Table 11 and Table 12.

The table below defines the interfaces of a DTM which can be used by a Frame Application at the shown states, see Table 11.

Table 11 − 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()	√											(√)

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

Interface / Method	up	new created	new enabled	existing created	running	configured	going offline	online	going online	destroyed	communication set	zombie
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 shall 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.

The table below defines the interfaces of a Frame Application which can be used by a DTM at the shown states, see Table 12. A Frame Application shall be aware that after IDtm:Environment a DTM complying to older versions of FDT can call any Frame Application interface method.

Table 12 − 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()						√	√	√	√		√	

Interface / Method	up	new created	new enabled	existing created	running	configured	going offline	online	going online	destroyed	communication set	zombie
OnReadResponse()								√	√		√	
OnWriteResponse()								√	√		√	
IDtmSingleInstanceDataAccessEvents						√	√	√	√		√	

6.3 Device Type Manager

6.3.1 Interface IDtm

6.3.1.1 General

This interface is the main interface of a DTM according to line one of Table G.1. 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.

6.3.1.2 Config

HRESULT Config(

[in] FdtXmlDocument userInfo,

[out, retval] VARIANT_BOOL* result);

Description

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

The method is part of the implementation of the Initialize service as defined in IEC 62453-2.

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 6.3.1.4).

6.3.1.3 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.

The method is part of the implementation of the initialize service as defined in IEC 62453-2.

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.

Comments

The systemTag is independent of communication tags (e.g. IEC 61784 CPF 9 device tag).

6.3.1.4 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.

The method is one implementation of the GetFunctions service as defined in IEC 62453-2.

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

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

Comments

The XML document provided by IDtm::GetFunctions() can contain functions (<Function> or <StandardFunction> elements) and groups of functions (<Functions>). Both can separately be enabled/disabled or shown/hidden, described by a <Status> element.

Because the FDT standard does not define status inheritance, the group (<Functions>) status is not inherited by the children (<Function> or <StandardFunction>). The DTM shall take care about the status of sub-functions.

For example, if the DTM disables a group of functions (<Functions>), it shall also disable all functions (<Function>) below that group if this is the intended behavior. If the DTM does not disable sub-functions, the Frame Application can still make use of them.

If the DTM has set the attribute 'isPrintable' of a <Function> element to true, the Frame Application should offer printing, even if attribute 'isEnabled' of the same <Function> element is set to false.

Microsoft Windows supports that menu functions may be called from the keyboard via their mnemonic access characters. An ampersand ('&') in a menu item string is normally translated into an underline character and used as the mnemonic access character for that menu item.

Because of different FDT Frame Application specific presentations (e.g. drop-down menu, listbox or combo box) mnemonic access characters should not be used within the XML document provided by IDtm::GetFunctions().

6.3.1.5 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.

The method is part of the implementation of the initialize service as defined in IEC 62453-2.

Parameters	Description

deviceType	XML document containing the manufacturer specific data like unique identifier for a sub-device type specified by DTMInitSchema.

Return value

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

Behavior

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

Comments

None

6.3.1.6 InvokeFunctionRequest

HRESULT InvokeFunctionRequest(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument functionCall,

[out, retval] VARIANT_BOOL* result);

Description

Starts a function of a DTM.

The method is one implementation of the InvokeFunction service as defined in IEC 62453-2.

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

See IEC 62453-2.

Comments

None

6.3.1.7 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.

The method is one implementation of the ClearInstanceData service as defined in IEC 62453-2.

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, for example 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 shall ensure, that all DTM instances 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

6.3.1.8 PrepareToRelease

HRESULT PrepareToRelease(

[out, retval] VARIANT_BOOL* result);

Description

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

The method is one implementation of the terminate service as defined in IEC 62453-2.

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 also shall 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 decision of a DTM, whether to store transient data or not. In order to trigger storing of the data, IFdtContainer::SaveRequest() shall be called.

NOTE The DTM has to implement a proper behavior concerning subclause 4.4 to give a Frame Application the information about the storing state of DTM related data (refer to FDT data types' attribute 'storageState').

6.3.1.9 PrepareToReleaseCommunication

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

Description

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

The method is part of the implementation of the EnableCommunication service as defined in IEC 62453-2.

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. A DTM has to fire progress events to inform Frame Application about ongoing progress if IDtmEvents::OnPrepareToReleaseCommunication() notification takes a longer time, for example, if still some communication calls have not returned.

See also 6.9.1.14 OnProgress and 6.3.1.12 SetCommunication

6.3.1.10 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.

The method is implementation of the PrivateDialogEnabled service, as defined in IEC 62453-2.

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

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.

If private dialogs are disabled, any private dialogs shall be prevented by the DTM. Also the DTM should inform user via IFdtDialog::UserDialog() if a specific functionality can not be performed because private dialogs are not allowed. The message should be something like “Due to application context request function is not available (opening of corresponding window is not allowed)”.

Comments

According to this formulation, examples for private dialogs are

- message boxes (e.g. standard message box),
- file or printer selection dialogs (e.g. provided by Microsoft Common Controls library),
- (default) web browsers (e.g. Internet Explorer),
- (default) mail clients (e.g. MS Outlook),
- help file view (e.g. HLP or CHM files),
- manual viewer (e.g. PDF or RTF),
- splash screens,
- external stand-alone applications or
- any other windows.

If the dialogs are opened under control of the Frame Application, they are defined not to be private dialogs:

- dialogs opened by IFdtDialog::UserDialog(),
- ActiveX controls opened by IFdtActiveX::OpenActiveXControlRequest(),
- applications started by IDtmApplication::StartApplication() and

- windows opened by the Frame Application due to a <Document> entry in the XMLdocument received from IDtm::GetFunctions().

6.3.1.11 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.

The method is one implementation of the ReleaseLinkedCoummunicationChannel service as defined in IEC 62453-2.

Return value

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

Behavior

It is recommended that the DTM sets 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 6.9.1.12, OnPreparedToReleaseCommunication().

6.3.1.12 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.

The method is implementation of the services SetLinkedCommunicatinChannel and EnableCommunication as defined in IEC 62453-2.

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 that the link is established correctly or in case of communication problems.

Comments

To ensure a proper behavior, it is recommended that the Frame Application implements following rules.

- For each DTM, which acts as a root concerning the chain of interfaces for nested communication, the method IDtm::SetCommunication() shall 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 Figure 9, DTM State Machine)
- To set up the chain for nested communication, the Frame Application shall 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 6.3.1.9, IDtm::PrepareToReleaseCommunication(), the Frame Application shall act vice versa.

6.3.1.13 SetLanguage

HRESULT SetLanguage(

[in] long languageId,

[out, retval] VARIANT_BOOL* result);

Description

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

The method is implementation of the SetLanguage service as defined in IEC 62453-2.

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

Return value

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

Behavior

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

Comments

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

6.3.2 Interface IDtm2

6.3.2.1 General

This interface is the main interface of a DTM according to line two of Table G.1 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 according to column 2 of Table G.1 has to use IDtm2, if the DTM supports this interface. Instead of calling IDtm::Environment() such a Frame Application shall then use the IDtm2::Environment2() method.

6.3.2.2 Environment2

HRESULT Environment2(

[in] BSTR systemTag,

[in] IFdtContainer* container,

[in] FdtXmlDocument frameInfo,

[out, retval] VARIANT_BOOL* result);

Description

Is called by a Frame Application according to Table G.1 column two and 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.

The method is part of the implementation of the initialize service as defined in IEC 62453-2.

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 according to Table G.1 column 2 and higher to initialize a DTM according to Table G.1 row 2 and 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 shall 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. IEC 61784 CRF 9 device tag).

6.3.2.3 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.

The method is implementation of the SetSystemGuiLabel service as defined in IEC 62453-2.

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, for example for a message box or a user interface which is part of the DTM and can not be embedded within a Frame Application. The Frame Application sets a unique human readable identifier of the DTM instance in the context of the Frame Application which ensures a unique identification between the device and for example a message box of a DTM. The DTM shall use this system label for all kinds of windows that will be opened by the DTM themselves. In special cases 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 shall not be stored by the DTM. It is recommended to update the labels of all open windows when SetSystemGuiLabel() is called.

6.3.3 Interface IDtmActiveXInformation

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

6.3.3.1 GetActiveXGuid

HRESULT GetActiveXGuid(

[in] FdtXmlDocument functionCall,

[out, retval] FdtUUIDString* result);

Description

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

The method is part of the implementation of the GetGuiInformation service as defined in IEC 62453-2.

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

6.3.3.2 GetActiveXProgId

HRESULT GetActiveXProgId(

[in] FdtXmlDocument functionCall,

[out, retval] BSTR* result);

Description

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

The method is part of the implementation of the GetGuiInformation service as defined in IEC 62453-2.

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

6.3.4 Interface IDtmApplication

6.3.4.1 General

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.

6.3.4.2 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.

The method is part of the implementation of the ClosePresentation service as defined in IEC 62453-2.

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 shut down procedure for the user interface. During this shut down it has to unlock its instance data set and release the online connection to its device if necessary. Finally, it has to notify the Frame Application via IDtmEvents::OnApplicationClosed(). This notification will cause the related releases on Frame Application's side. The DTM itself is not terminated.

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

The invoke id is used by a Frame Application for the association at the callback interface if the application is terminated. (See IdtmEvents::OnApplicationClosed.)

Comments

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

6.3.4.3 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.

The method is implementation of the OpenPresentation service as defined in IEC 62453-2.

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() always brings 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

6.3.5 Interface IDtmChannel

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

6.3.5.1 GetChannels

HRESULT GetChannels(

[out, retval] IFdtChannelCollection** result);

Description

Returns the FDT-Channel objects of a DTM.

The method is implementation of the GetChannels service as defined in IEC 62453-2.

Return value

Return value	Description
result	Collection of IFdtChannel of the requested channel objects.

Behavior

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

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

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

Comments

The IFdtChannelCollection allows access to FDT-Channel objects of a DTM. The interface definition follows the Microsoft COM standards for providing access to a group of objects that is known as a collection interface.

The IFdtChannelCollection provides a Count property that returns the number of items in the collection, an Item property that returns an item from the collection based on an index or a key, and a _NewEnum property that returns an enumerator for the collection.

The DTM shall declare its available channels with <ChannelReference> elements in XML returned in IDtmParameter::GetParamter(). The item property of IFdtChannelCollection shall accept keys that correspond to the idRef attribute in <ChannelReference> elements and numeric index values between 1 and count property value.

The item property shall accept the key values as variant data type VT_BSTR and (VT_BSTR | VT_BYREF), because different type of programming languages pass the strings differently (e.g. Visual Basic 6 uses VT_BSTR | VT_BYREF).

6.3.6 Interface IDtmDocumentation

6.3.6.1 General

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

6.3.6.2 GetDocumentation

HRESULT GetDocumentation(

[in] FdtXmlDocument functionCall,

[out, retval] FdtXmlDocument* result);

Description

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

The method is implementation of the GetDocumentation service as defined in IEC 62453-2.

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 technical report 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 to DTMDocumentationStyle.xsl. Refer to Annex D.

6.3.7 Interface IDtmDiagnosis

6.3.7.1 General

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

6.3.7.2 Compare

HRESULT Compare(

[out, retval] VARIANT_BOOL* result);

Description

Returns TRUE if the complete data sets are equal.

The method is part of the implementation of the CompareDataValueSet service as defined in IEC 62453-2.

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.

6.3.7.3 InitCompare

HRESULT InitCompare(

[in] BSTR systemTag,

[out, retval] VARIANT_BOOL* result);

Description

Initializes a DTM for comparison of two device instances.

The method is part of the implementation of the CompareDataValueSet service as defined in IEC 62453-2.

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() shall be closed using ReleaseCompare().

6.3.7.4 ReleaseCompare

HRESULT ReleaseCompare(

[out, retval] VARIANT_BOOL* result);

Description

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

The method is part of the implementation of the CompareDataValueSet service as defined in IEC 62453-2.

Return value

Return value	Description
TRUE	Compare sequence closed and external DTM reference released.
FALSE	A comparison is in progress (e.g. a 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 shall 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.

6.3.7.5 Verify

HRESULT Verify(

[out, retval] VARIANT_BOOL* result);

Description

Returns TRUE if the complete data set is valid.

The method is implementation of the verify service as defined in IEC 62453-2.

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.

6.3.8 Interface IDtmImportExport

6.3.8.1 General

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

6.3.8.2 Export

HRESULT Export(

[in] IStream* stream,

[out, retval] VARIANT_BOOL* result);

Description

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

The method is implementation of the export service as defined in IEC 62453-2.

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

6.3.8.3 Import

HRESULT Import(

[in] IStream* stream,

[out, retval] VARIANT_BOOL* result);

Description

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

The method is implementation of the import service as defined in IEC 62453-2.

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

6.3.9 Interface IDtmInformation

6.3.9.1 General

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.

6.3.9.2 GetInformation

HRESULT GetInformation(

[out, retval] FdtXmlDocument* result);

Description

This method is one implementation of the service GetTypeInformation as defined in IEC 62453-2.

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

Return value

Return value	Description
Result	XML document containing static DTM information specified by the DTMInformationSchema.

Behavior

Comments

Frame Applications should handle the identification-related information available from each <DtmDeviceType> and its <VersionInformation> element as a unique identification information of supported device type.

For DTM developers it is recommended that the identification information consist at least from the following attributes:

Element <DtmDeviceType>

- 'manufacturerId' (see attribute definition)
- 'deviceTypeId' (see attribute definition)

and <VersionInformation> element within <DtmDeviceType>

- 'name' (name of the device type)
- 'vendor' (vendor of the device type)
- 'version' (version of the device type, for example firmware version)

For a DTM it is recommended that

- above listed attributes are provided (if applicable);
- 'manufacturerId', 'deviceTypeId' and 'name' are used to uniquely identify the <DtmDeviceType> element. Changing any of these attributes for the same element in newer DTM software version is not allowed;
- the ‘name’ attribute shall be unique within the namespace of a DTM. It shall contain enough information to distinguish between different device types and sub-device types. This information will be used as a display string within a Frame Application specific device catalogue;

- supporting of a new device version, which could not be supported by existing <DtmDeviceType> elements, shall lead to new <DtmDeviceType> element with the different 'name' attribute;
- a new DTM software version shall support all the device types of the previous version.

Example:

DTM V1.0

ManufacturerX	DeviceX	V1

DTM V1.1

ManufacturerX	DeviceX	V1
ManufacturerX	DeviceXR2	V2

Frame Application vendors should be aware that there are DTMs on the market, that do not follow these recommendations. DTM can expose more than one <DtmDeviceType> element with the same name, but different values for 'manufacturerId', 'deviceTypeId', 'subDeviceType' and 'version'. Only in this case the Frame Application should consider the listed attributes as additional identifiers. This also could be used to distinguish between an update of an existing device type and a creation of an additional device type entry in the device catalogue.

The information should be used in same way for FDT1.2.1 DTMs in order to avoid problems in FDT1.2 based Frame Application.

6.3.10 Interface IDtmInformation2

6.3.10.1 General

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

6.3.10.2 GetDeviceIdentificationInformation()

HRESULT GetDeviceIdentificiationInformation(

[in] FdtXmlDocument typeRequest,

[in] FdtUUIDString protocolId,

[out, retval] FdtXmlDocument* result);

Description

This method implements service GetDeviceIdentificationInformation as defined in IEC 62453-2.

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 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 fieldbus specific schema. (FDTxxxDeviceTypeIdentSchema where xxx is the protocol name.) If method was called at a communiction DTM, then XML document according DTMDeviceTypeIdentSchema is returned and shall not be transformed.

Behavior

The response contains a protocol specific document, which can be validated by Frame Application against the protocol specific schemas.

Comments

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

See usage of information returned by IDtmInformation2::GetDeviceIdentificiationInformation() in 7.4.

6.3.11 Interface IDtmOnlineDiagnosis

6.3.11.1 General

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

6.3.11.2 Compare

HRESULT Compare(

[out, retval] FdtXmlDocument* result);

Description

Returns an XML document containing the result of the compare.

The method is the implementation of the CompareValueDataSetWithDevice service as defined in IEC 62453-2.

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 corresponding 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 shall return ‘noComparableData’ as value of the attribute ‘statusFlag’.

Comments

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

6.3.11.3 GetDeviceStatus

HRESULT GetDeviceStatus(

[out, retval] FdtXmlDocument* result);

Description

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

The method is part of the implementation of the DeviceStatus service as defined in IEC 62453-2.

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 shall work without a user interface to allow the check of complete networks.

Comments

A BTM shall return the status of the related block.

6.3.12 Interface IDtmOnlineParameter

6.3.12.1 General

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

6.3.12.2 CancelAction

HRESULT CancelAction(

[in] FdtUUIDString invokeId,

[out, retval] VARIANT_BOOL* result);

Description

Cancels an active parameter-upload or download.

The method is part of the implementation of the services WriteDataToDevice and ReadDataFromDevice as defined in IEC 62453-2.

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 shall not fire the IDtmEvents::OnDownloadFinished() or IDtmEvents::OnUploadFinished() events.

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

Comments

None

6.3.12.3 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.

The method is part of the implementation of the WriteDataToDevice service as defined in IEC 62453-2.

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

6.3.12.4 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.

The method is part of the implementation of the ReadDataFromDevice service as defined in IEC 62453-2.

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 sent during commissioning.

Comments

None

6.3.13 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.

6.3.13.1 GetParameters

HRESULT GetParameters(

[in] FdtXPath parameterPath,

[out, retval] FdtXmlDocument* result);

Description

Returns an XML document with the device specific parameters.

The method is implementation of the services GetActiveTypeInfo, InstanceDataInformation, InstanceDataRead, NetworkManangementInfoRead and GetChannels as defined in IEC 62453-2.

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

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

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

Comments

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

See also 7.14.1.

6.3.13.2 SetParameters

HRESULT SetParameters(

[in] FdtXPath parameterPath,

[in] FdtXmlDocument xmlDocument,

[out, retval] VARIANT_BOOL* result);

Description

Sets changes of device specific parameters.

The method is implementation of the services InstanceDataWrite and NetworkManangementInfoWrite as defined in IEC 62453-2.

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 shall verify complete document according to the business rules before accepting the requested changes.

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

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

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

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

Comments

See also 7.14.1.

6.3.14 Interface IFdtCommunicationEvents

6.3.14.1 General

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

6.3.14.2 OnAbort

HRESULT OnAbort(
[in] FdtUUIDString communicationReference);

Description

Notification that the connection identified by the communicationReference has been aborted.

The method is the implementation of the AbortRequest service as defined in IEC 62453-2.

Parameters	Description
communicationReference	Unique identifier for the connection.

Return value

None

Behavior

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

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

Comments

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

6.3.14.3 OnConnectResponse

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

Description

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

The method is part of the implementation of the Connect service as defined in IEC 62453-2.

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

Return value

None

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

6.3.14.4 OnDisconnectResponse

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

Description

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

The method is part of the implementation of the disconnect service as defined in IEC 62453-2.

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

Return value

None

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

Comments

None

6.3.14.5 OnTransactionResponse

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

Description

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

The method is part of the implementation of the transaction service as defined in IEC 62453-2.

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

Return value

None

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

6.3.15 Interface IFdtCommunicationEvents2

6.3.15.1 General

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 shall then use the IFdtCommunicationEvents2::OnConnectResponse2() method.

6.3.15.2 OnConnectResponse2

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

Description

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

The method is part of the implementation of the Connect service as defined in IEC 62453-2.

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, for example like FDTHARTCommunicationSchema or FDTProfibusCommunicationSchema.

Return value

None

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

6.3.16 Interface IFdtEvents

6.3.16.1 General

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

6.3.16.2 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.

The method is implementation of the OnChildInstanceDataChanged service as defined in IEC 62453-2.

Parameters	Description
systemTag	Identifier of the device instance.

Return value

None

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 corresponding parent DTMs (primary parent, secondary parents, see IFdtTopology::GetParentNodes()), the Frame Application shall implement the following behavior.

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

6.3.16.3 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.

The method is implementation of the OnLockInstanceData service as defined in IEC 62453-2.

Parameters	Description
systemTag	Identifier of the device instance.
userName	Human readable name of the user who has locked the data set.

Return value

None

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.

6.3.16.4 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.

The method is implementation of the OnInstanceDataChanged service as defined in IEC 62453-2.

Parameters	Description
systemTag	Identifier of the device instance.
Parameter	XML document containing the changed parameters.

Return value

None

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

6.3.16.5 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.

The method is implementation of the OnUnlockInstanceData service as defined in IEC 62453-2.

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 shall 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

6.3.17 Interface IDtmHardwareIdentification

6.3.17.1 General

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 to request manufacturer specific identification information from a field device by Gateway DTMs or Device DTMs.

6.3.17.2 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.

The method is part of the implementation of the HardwareInformation service as defined in IEC 62453-2.

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 shall call IDtmScanEvents::OnScanHardwareResponse() when operation has finished.

Comments

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

6.3.17.3 CancelAction

HRESULT CancelAction(

[in] FdtUUIDString invokeID,

[out, retval] VARIANT_BOOL* result);

Description

Cancels active hardware check request.

The method is part of the implementation of the HardwareInformation service as defined in IEC 62453-2.

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 can't cancel the operation then it returns FALSE and will fire the IDtmScanEvents::OnScanHardwareResponse() event when operation has finished.

Comments

None

6.3.18 Interface IDtmSingleDeviceDataAccess

6.3.18.1 General

This interface allows a Frame Application 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 shall be prepared for multiple asynchronous requests in parallel. The requests shall be processed in the order received.

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

6.3.18.2 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.

The method is part of the implementation of the services DeviceDataRead, DeviceDataWrite and DeviceDataInformation as defined in IEC 62453-2.

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 shall not fire the response event for the canceled operation.

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

Comments

None

6.3.18.3 ItemListRequest

HRESULT ItemListRequest(

[in] FdtUUIDString invokeId)

Description

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

The method is part of the implementation of the DeviceDataInformation service as defined in IEC 62453-2.

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

Comments

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

6.3.18.4 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.

The method is part of the implementation of the DeviceDataRead service as defined in IEC 62453-2.

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

Return value

None

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 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 shall not change the data of the instance data set.

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

6.3.18.5 WriteRequest

```
HRESULT WriteRequest(
        [in] FdtUUIDString invokeId,
        [in] FdtXmlDocument itemList);
```

Description

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

The method is part of the implementation of the DeviceDataWrite service as defined in IEC 62453-2.

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

Return value

None

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 shall 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 shall be provided.

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

6.3.19 Interface IDtmSingleInstanceDataAccess

6.3.19.1 General

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

6.3.19.2 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.

The method is part of the implementation of the InstanceDataInformation service as defined in IEC 62453-2.

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 FDT-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 IDtmSingleInstanceDataAccessEvents::OnInstanceItemListChanged().

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

6.3.19.3 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.

The method is part of the implementation of the InstanceDataRead service as defined in IEC 62453-2.

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

6.3.19.4 Write

HRESULT Write(

[in] FdtXmlDocument itemList,

[out, retval] FdtXmlDocument* result);

Description

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

The method is part of the implementation of the InstanceDataWrite service as defined in IEC 62453-2.

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

6.4 DTM ActiveXControl

6.4.1 Interface IDtmActiveXControl

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

6.4.2 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 corresponding DTM.

The method is implementation of an technology-specific service.

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 corresponding 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

6.4.3 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().

The method is implementation of an technology-specific service.

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

6.5 FDT Channel

6.5.1 Interface IFdtChannel

6.5.1.1 General

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

6.5.1.2 GetChannelParameters

HRESULT GetChannelParameters(

[in] FdtXPath parameterPath,

[in] FdtUUIDString protocolId,

[out, retval] FdtXmlDocument* result);

Description

Returns an XML document with fieldbus dependent channel specific parameters.

The method is part of the implementation of the ReadChannelInformation service as defined in IEC 62453-2.

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 a document based on the FDTBasicChannelParameterSchema instead of an empty document.

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

6.5.1.3 GetChannelPath

HRESULT GetChannelPath(

[out, retval] FdtXPath* result);

Description

Returns an identifier for a channel.

The method is part of the implementation of the ReadChannelInformation service as defined in IEC 62453-2.

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 shall 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>

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 shall 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 shall 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 corresponding 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).

6.5.1.4 SetChannelParameters

HRESULT SetChannelParameters(

[in] FdtXPath parameterPath,

[in] FdtUUIDString protocolId,

[in] FdtXmlDocument xmlDocument,

[out, retval] VARIANT_BOOL* result);

Description

Sets changes of channel specific parameters.

The method is the implementation of the WriteChannelInformation service as defined in IEC 62453-2.

Parameters	Description
parameterPath	FdtXPath within the XML document.
protocolId	UUID of a fieldbusprotocol. This UUID shall be one of the UUIDs returned by GetSupportedProtocols() and specified by DTMProtocolsSchema.
xmlDocument	XML document specified by a schema, for example 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 7.14.1.

6.5.2 Interface IFdtChannelActiveXInformation

Usually an FdtChannel does not have a user-interface, but in case of Communication Channels it may be required to have one. Depending on the fieldbus protocol and the

capability of the Communication Channel it might be necessary to implement a user interface to configure the communication itself.

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

6.5.2.1 GetChannelActiveXGuid

HRESULT GetChannelActiveXGuid(

[in] FdtXmlDocument,

[out, retval] FdtUUIDString* result);

Description

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

The method is one implementation of the GetGuiInformation service as defined in IEC 62453-2.

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

6.5.2.2 GetChannelActiveXProgId

HRESULT GetChannelActiveXProgId(

[in] FdtXmlDocument functionCall,

[out, retval] BSTR* result);

Description

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

The method is one implementation of the GetGuiInformation service as defined in IEC 62453-2.

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

6.5.2.3 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 FDT-Channel object.

The method is part of the implementation of the GetFunctions service as defined in IEC 62453-2.

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 FDT-Channel object functionality, defined by the applicationIDs and specific functionality which is not within the scope of FDT. These data are available as soon as the FDT-Channel object is instantiated but the information may change if it is instance specific.

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

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

The asynchronous behavior is described in the appropriate subclauses.

Comments

None

6.5.3 Interface IFdtCommunication

6.5.3.1 General

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 shall 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.

6.5.3.2 Abort

HRESULT Abort(

[in] FdtXmlDocument fieldbusFrame);

Description

Aborts a communication link to a device without any response.

The method is implementation of the AbortRequest service as defined in IEC 62453-2.

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

Return value

None

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

6.5.3.3 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.

The method is part of the implementation of the connect service as defined in IEC 62453-2.

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, for example 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 definition of use cases and scenarios in IEC 62453-2.

6.5.3.4 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.

The method is part of the implementation of the disconnect service as defined in IEC 62453-2.

Parameters	Description
invokeId	Unique identifier for the request.
fieldbusFrame	Fieldbus-protocol-specific information how to release the connection specified by a fieldbus specific schema, for example 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 definition of use cases and scenarios in IEC 62453-2.

6.5.3.5 GetSupportedProtocols

HRESULT GetSupportedProtocols(

[out, retval] FdtXmlDocument * result);

Description

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

The method is implementation of the GetSupportedProtocols service as defined in IEC 62453-2.

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 childes.

Comments

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

6.5.3.6 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.

The method is part of the implementation of the SequenceDefine service as defined in IEC 62453-2.

Parameters	Description
fieldbusFrame	Fieldbus-protocol-specific information describing the sequence. The structure is specified by a fieldbus specific schema, for example 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 requests. This can be a sequence containing several TransactionRequest() calls on one connection. The collection of pending requests is closed by IFdtCommunication::SequenceEnd().

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

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

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

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

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

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

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

For each TransactionRequest there shall 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.

6.5.3.7 SequenceEnd

HRESULT SequenceEnd(

[in] FdtXmlDocument fieldbusFrame,

[out, retval] VARIANT_BOOL* result);

Description

Closes the communication block started with SequenceBegin().

The method is part of the implementation of the SequenceDefine service as defined in IEC 62453-2.

Parameters	Description
fieldbusFrame	Fieldbus-protocol-specific information closing a sequence of transaction calls. The structure is specified by a fieldbus specific schema, for example 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.

6.5.3.8 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 provided by the corresponding transaction response function calls. (For each recorded transaction request a corresponding transaction response is received.)

The method is part of the implementation of the SequenceStart service as defined in IEC 62453-2.

Parameters	Description
fieldbusFrame	Fieldbus-protocol-specific information starting a sequence. The structure is specified by a fieldbus specific schema, for examplbe 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.

6.5.3.9 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.

The method is part of the implementation of the transaction service as defined in IEC 62453-2.

Parameters	Description
invokeId	Unique identifier for the request.
fieldbusFrame	Fieldbus-protocol-specific information to be transferred, specified by a fieldbus specific schema, for example 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 part of IEC 62453.

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

result. In case of more than one pending request the association of request and response is done by the passed invoke id. The client is responsible to pass a unique invoke id for the specified communication link.

The response will be provided by OnTransactionResponse().

If a TransactionRequest() is called as part of a sequence 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 commands 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 the same device.

Developers of Device DTM's should consider that the used communication infrastructure creates delays in the communication. So the Device DTM's should limit the number of communication requests. Also the Device DTM shall 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 shall 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.

6.5.4 Interface IFdtChannelSubTopology

6.5.4.1 General

This interface shall 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 respective DTM can validate the configured connections.

6.5.4.2 OnAddChild

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

Description

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

The method is implementation of the ChildAdded service as defined in IEC 62453-2.

Parameters	Description
childSystemTag	SystemTag of the newly added child instance.

Return value

None

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, for example project related data, this method shall not be called.

6.5.4.3 OnRemoveChild

HRESULT OnRemoveChild(

[in] BSTR childSystemTag);

Description

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

The method is implementation of the ChildeRemoved service as defined in IEC 62453-2.

Parameters	Description
childSystemTag	SystemTag of the child DTM which was removed.

Return value

None

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, for example closing a project, this method shall not be called.

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

6.5.4.4 ScanRequest

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

Description

The method requests one asynchronous scan of the sub topology.

The method is part of the implementation of the scan service as defined in IEC 62453-2.

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.

6.5.4.5 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.

The method is implementation of the ValidateAddChild service as defined in IEC 62453-2.

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 6.5.4.2

6.5.4.6 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.

The method is implementation of the ValidateRemoveChild service as defined in IEC 62453-2.

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 shall include a check for active connection and return FALSE if a connection is active via this channel.

6.5.5 Interface IFdtChannelSubTopology2

This mandatory interface is implemented by a Communication Channel and extends the interface IFdtChannelSubTopology by new methods.

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

6.5.5.1 SetChildrenAddresses

HRESULT SetChildrenAddresses(

[in] FdtXmlDocument dtmDeviceList,

[out, retval] FdtXmlDocument* result);

Description

Requests bus address setting for specified device list.

The method is implementation of the SetChildrenAddresses service as defined in IEC 62453-2.

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() may be used to request address selection from the user.

Setting the bus address on a Device DTM via the IDtmParameter interface is intended to provide the Device DTM with information. The Device DTM shall not use this information for execution of communication transactions, that set the address in the field device.

6.5.6 Interface IFdtChannelScan

6.5.6.1 General

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 FDT-Channel object can rely on that the methods of this interface are used instead of the replaced methods of the IFdtChannelSubTopology interface.

6.5.6.2 ScanRequest

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

Description

Requests the asynchronous scan of the sub topology.

The method is part of the implementation of the scan service as defined in IEC 62453-2.

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.

6.5.6.3 CancelAction

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

Description

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

The method is part of the implementation of the scan service as defined in IEC 62453-2.

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.

6.5.7 Interface IFdtFunctionBlockData

6.5.7.1 General

This interface shall 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.

6.5.7.2 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.

The method is a technology-specific service.

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 FDT-Channel object routes upward in the topology to the bus master driver that contains the corresponding proxy FB of the DTM, reads the individual device parameter set directly out of the bus master and passes them to the DTM.

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

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

Comments

None

6.5.7.3 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.

The method is a technology-specific service.

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 parameter of devices. The user has to select the proxy FB of the corresponding failsafe device for the DTM. The bus-master DTM stores the assignment of the proxy FB to the calling DTM-instance.

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

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

Comments

None

6.6 Channel ActiveXControl

6.6.1 Interface IFdtChannelActiveXControl

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

6.6.1.1 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 corresponding FdtChannel.

The method is a technology-specific service.

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 corresponding 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

6.6.1.2 PrepareToRelease

HRESULT PrepareToRelease(

[out, retval] VARIANT_BOOL* result);

Description

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

The method is a technology-specific service.

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 corresponding 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

6.6.2 Interface IFdtChannelActiveXControl2

6.6.2.1 General

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

6.6.2.2 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 corresponding FdtChannel.

The method is a technology-specific service.

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 corresponding 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 shall use this method instead of the former method.

6.7 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

6.7.1 Interface IBtm

6.7.1.1 General

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.

6.7.1.2 Config

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

6.7.1.3 Environment

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

6.7.1.4 GetFunctions

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

6.7.1.5 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().

6.7.1.6 InvokeFunctionRequest

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

6.7.1.7 PrepareToDelete

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

6.7.1.8 PrepareToRelease

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

6.7.1.9 PrepareToReleaseCommunication

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

6.7.1.10 PrivateDialogEnabled

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

6.7.1.11 ReleaseCommunication

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

6.7.1.12 SetCommunication

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

6.7.1.13 SetLanguage

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

6.7.2 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 Type Manager and the Device Type Manager objects.

6.7.2.1 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().

6.7.3 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 Type Manager and the Device Type Manager objects.

6.7.3.1 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().

6.7.3.2 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().

6.8 BTM ActiveXControl

6.8.1 General

This section describes the handling of ActiveX controls that are provided by BTMs.

6.8.2 Interface IBtmActiveXControl

6.8.2.1 General

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

6.8.2.2 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 corresponding BTM.

The method is a technology-specific service.

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 corresponding 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

6.8.2.3 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.

6.9 Frame Application

6.9.1 Interface IDtmEvents

6.9.1.1 General

This interface is the callback-interface for the DTMs.

6.9.1.2 OnApplicationClosed

HRESULT OnApplicationClosed(

[in] FdtUUIDString invokeId);

Description

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

The method is part of the implementation of the ClosePresentation service as defined in IEC 62453-2.

Parameters	Description
invokeId	Identifier of the closed application.

Return value

None

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

6.9.1.3 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.

The method is part of the implementation of the WriteDataToDevice service as defined in IEC 62453-2.

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

Return value

None

Behavior

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

Comments

None

6.9.1.4 OnErrorMessage

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

Description

Notification by a DTM about errors during a function call.

The method is implementation of the OnErrorMessage service as defined in IEC 62453-2.

Parameters	Description
systemTag	Identifier of the device instance.
errorMessage	Human readable error message.

Return value

None

Behavior

The method is necessary if the DTM works without a user interface. If a DTM works without user interface it is not allowed to display any error message within own dialog windows.

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

Comments

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

6.9.1.5 OnFunctionChanged

HRESULT OnFunctionChanged(
[in] BSTR systemTag);

Description

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

The method is part of the implementation of the OnFunctionChanged service as defined in IEC 62453-2.

Parameters	Description
systemTag	Identifier of the device instance.

Return value

None

Behavior

The method is used if the additional functionality depends on the configuration of a device. Via this method the DTM informs the Frame Application to update its menus or function calls which reference on this extended functionality. The Frame Application gets the actual available functionality via IDtm::GetFunctions().

Call of this method is mandatory for a DTM whenever status or number of functions has changed.

Comments

None

6.9.1.6 OnChannelFunctionChanged

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

Description

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

The method is part of the implementation of the OnFunctionChanged service as defined in IEC 62453-2.

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

Return value

None

Behavior

Via this method the DTM can inform the Frame Application to update its channel related menus which refer to these functions. The Frame Application gets the actual available functionality via IFdtChannelActiveXInformation::GetChannelFunctions().

Comments

None

6.9.1.7 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.

The method is part of the implementation of the InvokeFunction service as defined in IEC 62453-2.

Parameters	Description
invokeId	Identifier of the closed application.
success	TRUE if the operation is successfully finished.

Return value

None

Behavior

Notification by a DTM, whether the asynchronously invoked function call IDtm::InvokeFunctionRequest() is successfully finished.

Comments

None

6.9.1.8 OnNavigation

HRESULT OnNavigation(

[in] BSTR systemTag);

Description

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

The method is a technology specific service.

Parameters	Description
systemTag	Identifier of the device instance.

Return value

None

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

6.9.1.9 OnOnlineStateChanged

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

Description

A DTM sends an notification about its online state.

The method is part of the implementation of the OnOnlineStatusChanged service as defined in IEC 62453-2.

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

None

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.

6.9.1.10 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.

The method is implementation of the InstanceDataChanged service as defined in IEC 62453-2.

Parameters	Description
systemTag	Identifier of the device instance
parameter	XML document containing the changed parameters.

Return value

None

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 corresponding parent DTM via IFdtEvents::OnChildParameterChanged().

Comments

This notification could also be used by a Frame Application to trigger an update, for example 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.

6.9.1.11 OnPreparedToRelease

HRESULT OnPreparedToRelease(
[in] BSTR systemTag);

Description

A DTM sends a notification that it can be released.

The method is part of the implementation of the terminate service as defined in IEC 62453-2.

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

6.9.1.12 OnPreparedToReleaseCommunication

HRESULT OnPreparedToReleaseCommunication(
[in] BSTR systemTag);

Description

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

The method is part of the implementation of the EnableCommunication service as defined in IEC 62453-2.

Parameters	Description
systemTag	Identifier of the device instance.

Return value

None

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

6.9.1.13 OnPrint

HRESULT OnPrint(

[in] BSTR systemTag,

[in] FdtXmlDocument functionCall);

Description

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

The method is a technology specific service.

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

None

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

6.9.1.14 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 of handling of a function call.

The method is implementation of the OnProgress service as defined in IEC 62453-2.

Parameters	Description
systemTag	Identifier of the device instance.
title	Description of the running process.

Parameters	Description
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 shall be closed within the Frame Application.

Return value

None

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 for example the title.

It's up to the Frame Application how to handle the information.

Comments

None

6.9.1.15 OnScanResponse

HRESULT OnScanResponse(

[in] FdtUUIDString invokeId,

[in] FdtXmlDocument response);

Description

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

The method is part of the implementation of the scan service as defined in IEC 62453-2.

Parameters	Description
invokeId	Unique identifier for the request.
Response	XML document containing the result of the topology scan specified by the DTMTopologyScanSchema.

Return value

None

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.

6.9.1.16 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.

The method is part of the implementation of the ReadDataFromDevice service as defined in IEC 62453-2.

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

None

Behavior

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

Comments

None

6.9.2 Interface IDtmEvents2

6.9.2.1 General

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 shall call IDtmEvents2, if the Frame Application supports this interface. Instead of calling IDtmEvents::OnOnlineStateChanged() such a DTM then shall use the IDtmEvents2::OnStateChanged() method.

6.9.2.2 OnStateChanged

HRESULT OnStateChanged(

[in] BSTR systemTag,

[in] FdtXmlDocument xmldoc);

Description

A DTM sends a notification about change in its state.

The method is implementation of the OnOnlineStatusChanged service as defined in IEC 62453-2.

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

None

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, for example ConnectResponse failed or OnAbort, the fdt:CommunicationError information shall be provided within the XML document.

Comments

None

6.9.3 Interface IDtmScanEvents

6.9.3.1 General

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

6.9.3.2 OnScanResponse

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

Description

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

The method is part of the implementation of the scan service as defined in IEC 62453-2.

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 fieldbus specific schema (FDTxxxScanIdentSchema) . The fieldbus specific document shall be transformed using protocol specific xsl to get a protocol independent document, which shall 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.

6.9.3.3 OnScanHardwareResponse

HRESULT OnScanHardwareResponse(

[in] FdtUUIDString invokeID,

[in] FdtXmlDocument response);

Description

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

The method is part of the implementation of the HardwareInformation service as defined in IEC 62453-2.

Parameters	Description
invokeID	Identifier of the request.
Response	XML document containing the result of the scan hardware request specified by a fieldbus specific schema (FDTxxxScanIdentSchema) if request was called at a Gateway or Device DTM. The fieldbus specific document containing additional manufacturer specific extensions and shall 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 shall not be transformed (ID entries which can not be filled are left empty).

Return value

None

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

6.9.4 Interface IDtmAuditTrailEvents

6.9.4.1 General

This interface shall 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.

6.9.4.2 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.

The method is part of the implementation of the RecordAuditTrailEvent service as defined in IEC 62453-2.

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

Return value

None

Behavior

A DTM shall 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.

6.9.4.3 OnEndTransaction

HRESULT OnEndTransaction(

[in] BSTR systemTag,

[out, retval] VARIANT_BOOL* result);

Description

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

The method is part of the implementation of the RecordAuditTrailEvent service as defined in IEC 62453-2.

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.

6.9.4.4 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.

The method is part of the implementation of the RecordAuditTrailEvent service as defined in IEC 62453-2.

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

6.9.5 Interface IFdtActiveX

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

6.9.5.1 CloseActiveXControlRequest

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

Description

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

The method is part of the implementation of the ClosePresentationRequest service as defined in IEC 62453-2.

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

6.9.5.2 OpenActiveXControlRequest

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

Description

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

The method is part of the implementation of the OpenPresentationRequest service as defined in IEC 62453-2.

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 shall 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.

6.9.6 Interface IFdtActiveX2

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

6.9.6.1 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.

The method is part of the implementation of the OpenPresentationRequest service as defined in IEC 62453-2.

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 shall 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 shall 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 V 1.2 – May 2003 - Addendum, chapter 2.30.4 – A closer look at message loops).

6.9.6.2 OpenDialogChannelActiveXControlRequest

HRESULT OpenDialogChannelActiveXControlRequest(

[in] BSTR channelPath,

[in] FdtXmlDocument functionCall,

[out, retval] VARIANT_BOOL* result);

Description

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

The method is part of the implementation of the OpenPresentationRequest service as defined in IEC 62453-2.

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 shall 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 shall 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 – May 2003 - Addendum, chapter 2.30.4 – A closer look at message loops, paragraph DTM's point of view).

6.9.6.3 CloseChannelActiveXControlRequest

HRESULT CloseChannelActiveXControlRequest(

[in] FdtUUIDString invokeId,

[out, retval] VARIANT_BOOL* result);

Description

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

The method is part of the implementation of the ClosePresentationRequest service as defined in IEC 62453-2.

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 FDT-Channel object. This is done via IFdtChannelActiveXControl::PrepareToRelease().

Comments

None

6.9.6.4 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 FDT_Channel object (identified by the channelPath).

The method is part of the implementation of the OpenPresentationRequest service as defined in IEC 62453-2.

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 shall 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 Application creates the Channel ActiveX and assigns it to the FDT-Channel object (referenced by channelPath) by calling IFdtChannelActiveXControl::Init().

6.9.7 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 shall be no impact on the instance specific configuration data set of a DTM. The instance data set of a DTM shall be always consistent to the configuration data stored via the standard IPersistXXX interface. It is 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.

6.9.7.1 GetInstanceRelatedPath

HRESULT GetInstanceRelatedPath(

[in] BSTR systemTag,

[out, retval] BSTR* result);

Description

Returns the instance related path for bulk data.

The method is part of the implementation of the GetPrivateDtmStorageInfo service as defined in IEC 62453-2.

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 a unique path for each instance. It shall be an absolute path to allow a DTM the direct access.

Comments

A DTM shall work without any side effects if a path is not available.

A DTM shall 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 consistency of data.

6.9.7.2 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).

The method is part of the implementation of the GetPrivateDtmStorageInfo service as defined in IEC 62453-2.

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 to provide a unique path for each DTM type within a project. It shall be an absolute path to allow a DTM the direct access.

Comments

A DTM shall work without any side effects if a path is not available.

If the DTM holds any references between project and instance related data it shall clean up these data if IDtm::PrepareToDelete() is called

There is no FDT specific locking mechanism, so the DTM is responsible for consistency of data.

6.9.8 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.

6.9.8.1 GetXmlSchemaPath

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

Description

Returns a path where the default schemas are stored.

The method is a technology specific service.

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

6.9.8.2 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.

The method is part of the implementation of the LockInstanceData service as defined in IEC 62453-2.

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

Comments

Within a single user system the method returns always TRUE.

Examples for lock conditions are

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

6.9.8.3 SaveRequest

HRESULT SaveRequest(

[in] BSTR systemTag,

[out, retval] VARIANT_BOOL* result);

Description

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

The method is part of the implementation of the SaveInstanceData service as defined in IEC 62453-2.

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 gets 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 shall reject the save request if the DTM has no write access rights.

Storing the data triggers IDtmEvents::OnParameterChange() which can lead to actions in the system and IFdtEvents::OnChildParameterChange() notifications which might start a new DTM (or a whole line). This can lead to performance issues.

Therefore a DTM should limit the calls to IDtmEvents::SaveRequest().

For exmple, DTM should not fire SaveRequest() for every single change in a user interface, instead it should only call SaveRequest() once when the user interface is closed.

The Frame Application decides whether and when the DTM related instance data set will be stored into the data base. The Frame Application requests the DTM to store by calling IPersistXXX::Save() on the DTM. After executing the save procedure successfully, the DTM data set is in storage state “persistent”.

6.9.8.4 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.

The method is part of the implementation of the UnlockInstanceData service as defined in IEC 62453-2.

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

Comments

Within a single user system the method returns always TRUE.

6.9.9 Interface IFdtDialog

6.9.9.1 General

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

6.9.9.2 UserDialog

HRESULT UserDialog(

[in] BSTR systemTag,

[in] FdtXmlDocument userMessage,

[out, retval] FdtXmlDocument* result);

Description

Call the Frame Application to display a message.

The method is implementation of the UserDialog service as defined in IEC 62453-2.

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 shall ensure displaying the user dialog and the user interface of the DTM at the same workplace.

Comments

None

6.9.10 Interface IFdtTopology

6.9.10.1 General

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 corresponding DTM can validate the connections.

6.9.10.2 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.

The method is the implementation of the CreateChild service for a DTM as defined in IEC 62453-2.

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 7.13.1. It is also in the responsibility of the Frame Application to insert the created DTM into the topology.

Comments

None

6.9.10.3 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.

The method is the implementation of the DeleteChild service for a DTM as defined in IEC 62453-2.

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

6.9.10.4 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.

The method is the implementation of the GetChildNodes service for a DTM as defined in IEC 62453-2.

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.

6.9.10.5 GetDtmForSystemTag

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

Description

Return the associated DTM according the given system tag.

The method is the implementation of the GetDtm service for a DTM as defined in IEC 62453-2.

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 shall 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

6.9.10.6 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.

The method is the implementation of the GetDtmInfoList service for DTM related information as defined in IEC 62453-2.

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. For example the list could contain only information concerning HART® devices even if PROFIBUS devices are installed.

6.9.10.7 GetParentNodes

HRESULT GetParentNodes(

[in] BSTR systemTag,

[out, retval] FdtXmlDocument* result);

Description

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

The method is the implementation of the ParentNodes service for DTMs as defined in IEC 62453-2.

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.

6.9.10.8 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.

The method is the implementation of the MoveChild service for a DTM as defined in IEC 62453-2.

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

6.9.10.9 ReleaseDtmForSystemTag

HRESULT ReleaseDtmForSystemTag(

[in] BSTR systemTag,

[out, retval] VARIANT_BOOL* result);

Description

Release the associated DTM according the given system tag.

The method is the implementation of the ReleaseDtm service for a DTM as defined in IEC 62453-2.

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

6.9.11 Interface IDtmRedundancyEvents

6.9.11.1 General

This Frame Application interface provides access for parent components handling redundant slaves. A Frame Application not implementing this interface is not able to display redundancy information within its topology information (for redundancy refer 5.9)

6.9.11.2 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.

The method is implementation of the OnAddedRedundantChild service as defined in IEC 62453-2.

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 shall not call ValidateAddChild() or OnAddChild() on this channel.

Comments

None

6.9.11.3 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.

The method is implementation of the OnRemovedRedundantChild service as defined in IEC 62453-2.

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 shall not call ValidateRemoveChild() or OnRemoveChild() on this channel.

Comments

None

6.9.12 Interface IDtmSingleDeviceDataAccessEvents

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

6.9.12.1 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.

The method is part of the implementation of the DeviceDataInformation service as defined in IEC 62453-2.

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

None

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 FDT-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 IDtmSingleDeviceDataAccessEvents::OnDeviceItemListChanged().

6.9.12.2 OnDeviceItemListChanged

HRESULT OnDeviceItemListChanged(

[in] BSTR systemTag);

Description

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

The method is part of the implementation of the DeviceDataInformation service as defined in IEC 62453-2.

Parameters	Description
systemTag	Identifier of the device instance.

Return value

None

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.

OnDeviceItemListChanged should not be fired in case of pending responses.

Comments

None

6.9.12.3 OnReadResponse

HRESULT OnReadResponse(
[in] FdtUUIDString invokeId,
[in] FdtXmlDocument response);

Description

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

The method is part of the implementation of the DeviceDataRead service as defined in IEC 62453-2.

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

Return value

None

Behavior

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

Comments

None

6.9.12.4 OnWriteResponse

HRESULT OnWriteResponse(
[in] FdtUUIDString invokeId,
[in] FdtXmlDocument response);

Description

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

The method is part of the implementation of the DeviceDataWrite service as defined in IEC 62453-2.

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

None

Behavior

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

Comments

None

6.9.13 Interface IDtmSingleInstanceDataAccessEvents

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

6.9.13.1 OnInstanceItemListChanged

HRESULT OnInstanceItemListChanged(
[in] BSTR systemTag);

Description

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

The method is part of the implementation of the InstanceDataInformation service as defined in IEC 62453-2.

Parameters	Description
systemTag	Identifier of the device instance.

Return value

None

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.

OnInstanceItemListChanged should not be fired in case of pending responses.

Comments

None

6.9.14 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 Type Manager 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.

6.9.14.1 CreateChild

The method is the implementation of the CreateChild service for a BTM as defined in IEC 62453-2.

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().

6.9.14.2 DeleteChild

The method is the implementation of the DeleteChild service for a BTM as defined in IEC 62453-2.

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

6.9.14.3 GetChildNodes

The method is the implementation of the GetChildNodes service for a BTM as defined in IEC 62453-2.

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

6.9.14.4 GetBtmForSystemTag

The method is the implementation of the GetDtm service for a BTM as defined in IEC 62453-2.

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

6.9.14.5 GetBtmInfoList

The method is the implementation of the GetDtmInfoList service for BTM related information as defined in IEC 62453-2.

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

6.9.14.6 GetParentNodes

The method is the implementation of the ParentNodes service for BTMs as defined in IEC 62453-2.

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

6.9.14.7 MoveChild

The method is the implementation of the MoveChild service for a BTM as defined in IEC 62453-2.

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

6.9.14.8 ReleaseBtmForSystemTag

The method is the implementation of the ReleaseDtm service for a BTM as defined in IEC 62453-2.

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

7 FDT sequence charts

7.1 DTM peer to peer communication

7.1.1 General

For a DTM each connection is established as a peer-to-peer connection. This subclause describes the communication function calls from a DTM developer's point of view.

7.1.2 Establish a peer-to-peer connection between DTM and device

The connection is established by method calls to the IFdtCommunication pointer, which was provided to the DTM (see Figure 16).

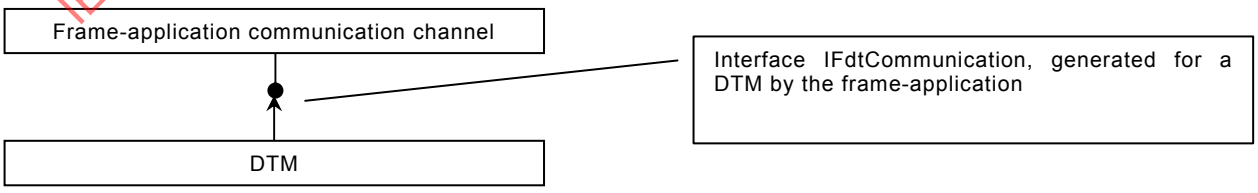
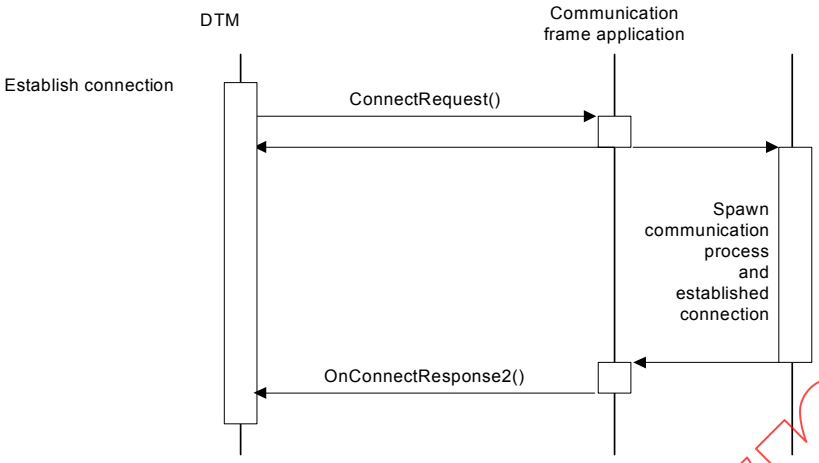


Figure 16 − Peer to peer connection between DTM and device

7.1.3 Asynchronous connect for a peer-to-peer connection

The connect request is handled asynchronously (see Figure 17).



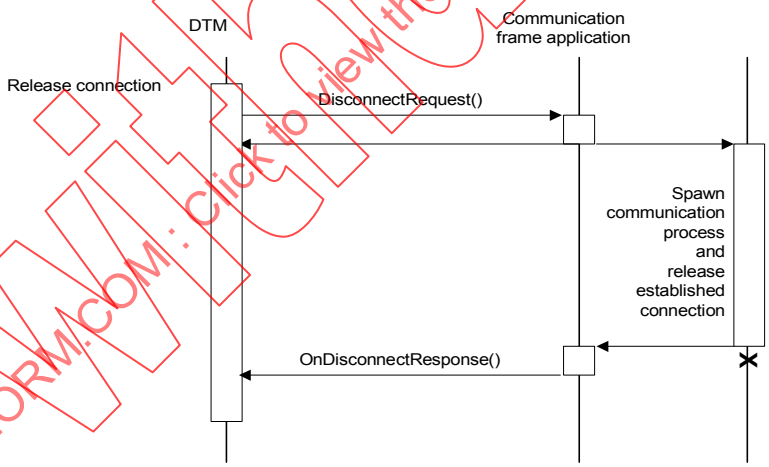
Used methods:

IFdtCommunication::ConnectRequest()
IFdtCommunicationEvents2::OnConnectResponse2()

Figure 17 – Asynchronous connect (peer to peer)

7.1.4 Asynchronous disconnect for a peer-to-peer connection

Also the disconnect is handled in an asynchronous way (see Figure 18).



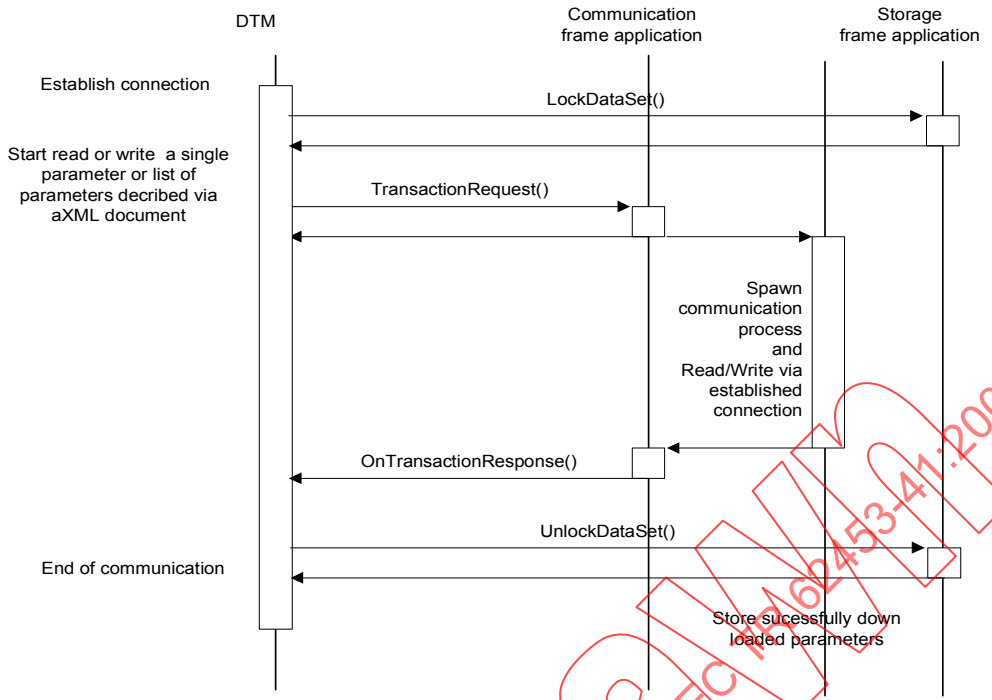
Used methods:

IFdtCommunication::DisconnectRequest()
IFdtCommunicationEvents::OnDisconnectResponse()

Figure 18 – Asynchronous disconnect (peer to peer)

7.1.5 Asynchronous transaction for a peer-to-peer connection

Transaction requests are handled asynchronously (see Figure 19).



Used methods:

IFdtCommunication::TransactionRequest()

IFdtCommunicationEvents::()

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

Figure 19 – Asynchronous transaction (peer to peer)

7.2 Nested communication

7.2.1 General

This subclause is important for DTM developers who support a device with gateway functionality (e.g. remote I/O). This clause 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 shall 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 20 shows the system topology for the following examples:

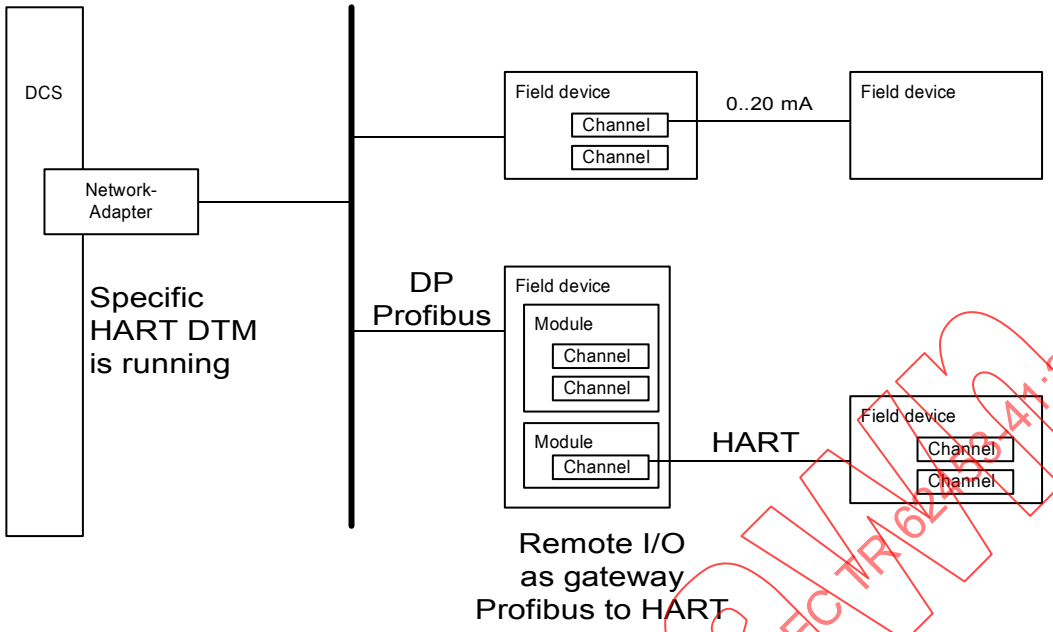


Figure 20 – System-topology

A connection from DCS to field device in such a hierarchical communication system is called system connection in the following clauses and subclauses.

7.2.2 Generate system topology

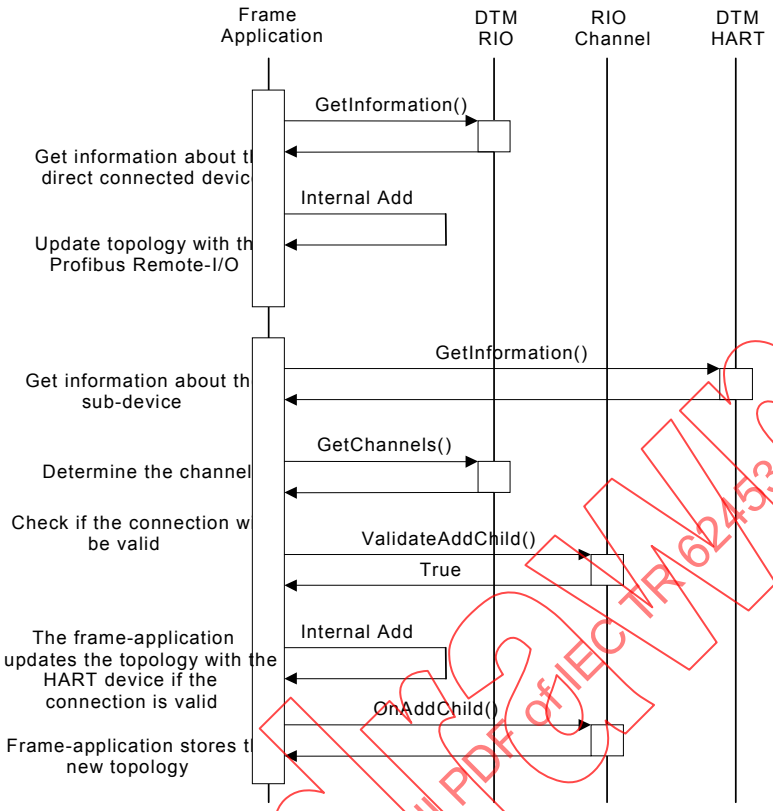
7.2.2.1 Topology

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.

7.2.2.2 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 21).



Used methods:

IDtmInformation::GetInformation()

IDtmChannel::GetChannels()

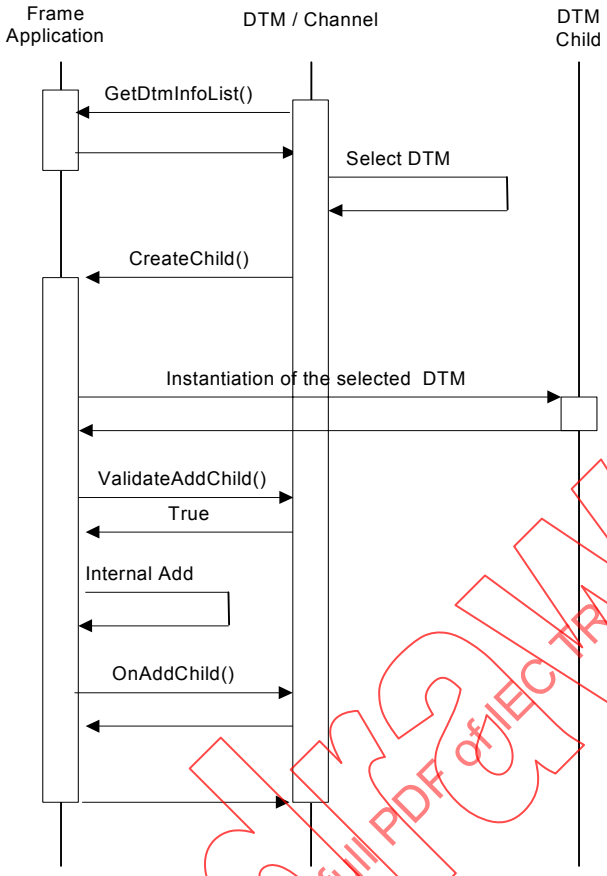
IFdtChannelSubTopology::ValidateAddChild()

IFdtChannelSubTopology::OnAddChild()

Figure 21 – Generation of system topology by Frame Application

7.2.2.3 DTMs point of view

This sequence shows the generation of the topology triggered by a DTM (see Figure 22).



Used methods:

IFdtTopology::GetDtmInfoList()

IFdtTopology::CreateChild()

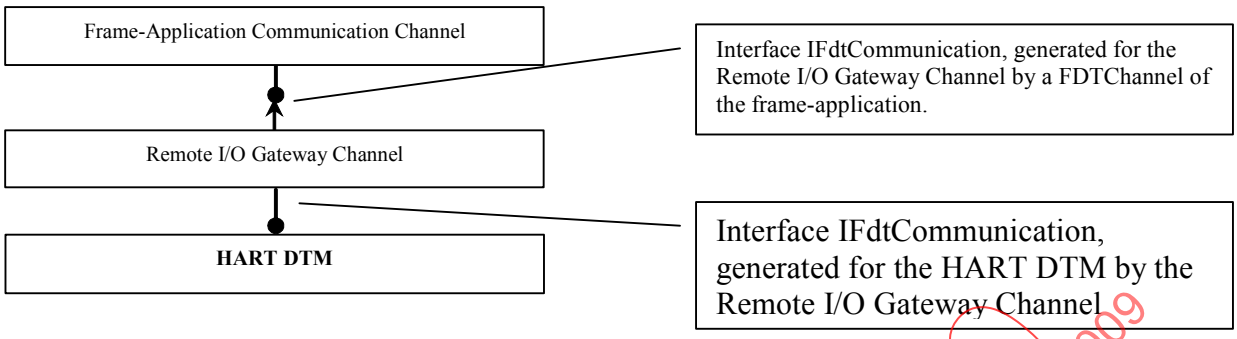
IFdtChannelSubTopology::ValidateAddChild()

IFdtChannelSubTopology::OnAddChild()

Figure 22 – Generation of system topology – Participation of DTM

7.2.3 Establish a system connection between DTM and device

Figure 23 shows possible communication hierarchy in FDT.



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

Used methods:

IDtm::SetCommunication()

IFdtTopology::GetChildNodes()

IFdtCommunication::ConnectRequest()

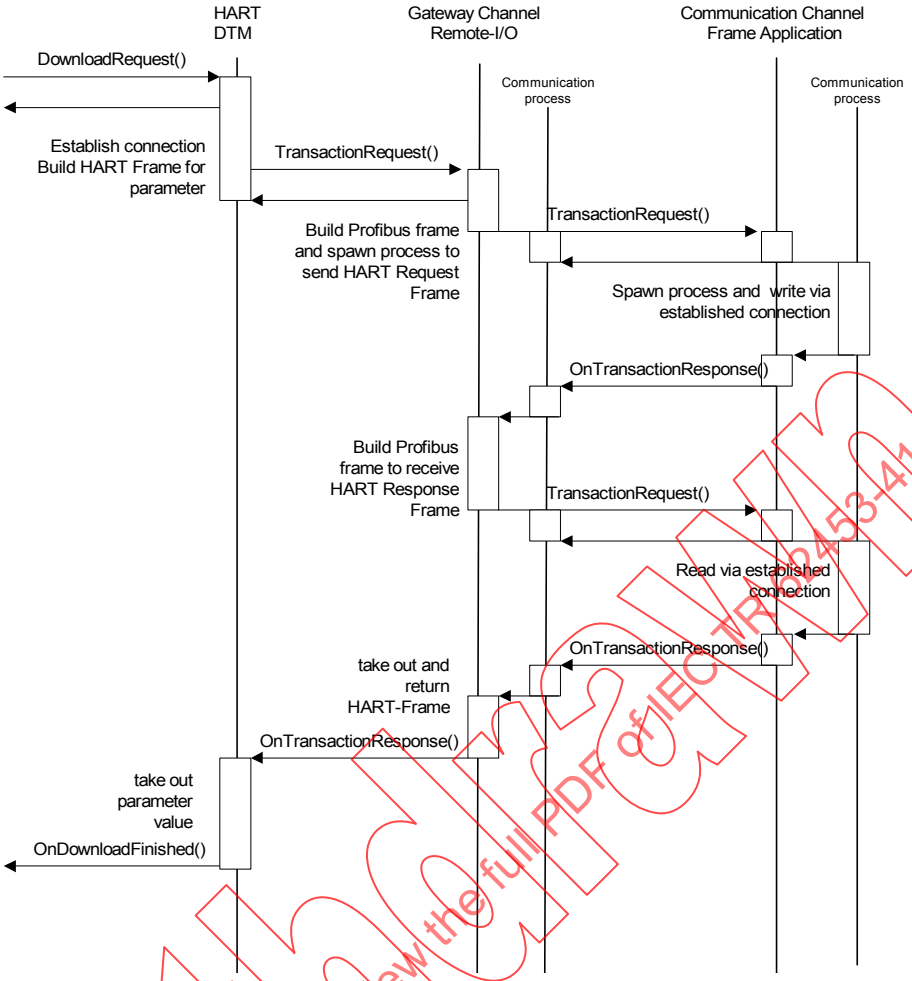
IFdtCommunicationEvents2::OnConnectResponse2()

Figure 23 – System connection (across communication hierarchy)

7.2.4 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 (see Figure 24).



Used methods:

IFdtCommunication::TransactionRequest()

IFdtCommunicationEvents::OnTransactionResponse()

IDtmOnlineParameter::DownloadRequest()

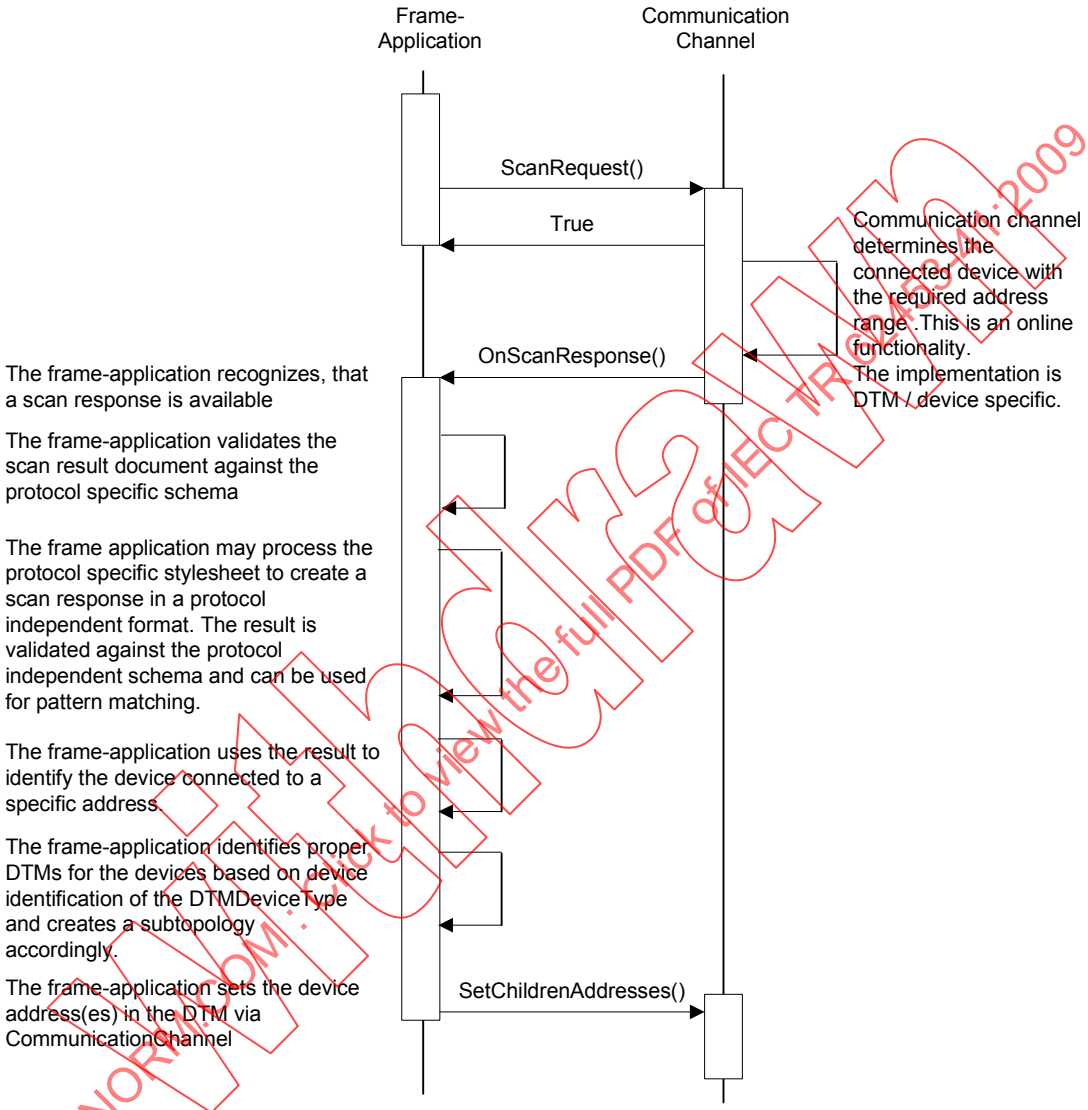
IDtmEvents::OnDownloadFinished()

Figure 24 – Asynchronous transactions (system connection)

7.3 Topology scan

7.3.1 Scan network

Sequence diagram: scan network topology (see Figure 25).



Used methods:

IFdtChannelScan::ScanRequest()

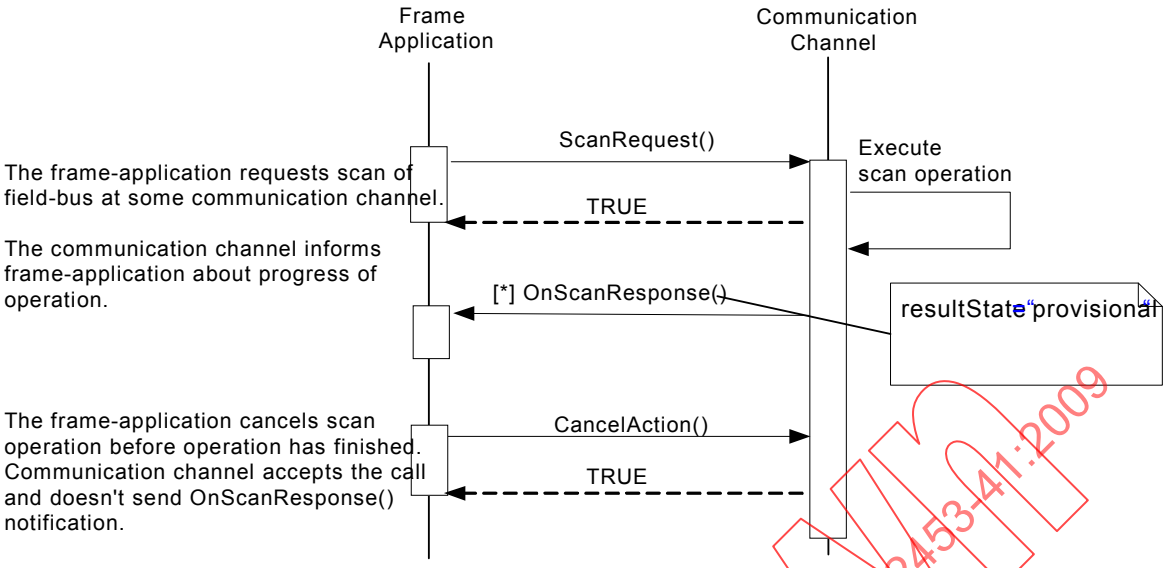
IDtmScanEvents::OnScanResponse()

IFdtChannelSubTopology2::SetChildrenAddresses()

Figure 25 − Scan network topology

7.3.2 Cancel topology scan

In this scenario Frame Application cancels an active scan request (see Figure 26).



Used methods:

IFdtChannelScan::ScanRequest()

IFdtChannelScan:CancelAction()

IDtmScanEvents::OnScanResponse()

Figure 26 – Cancel topology scan

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.

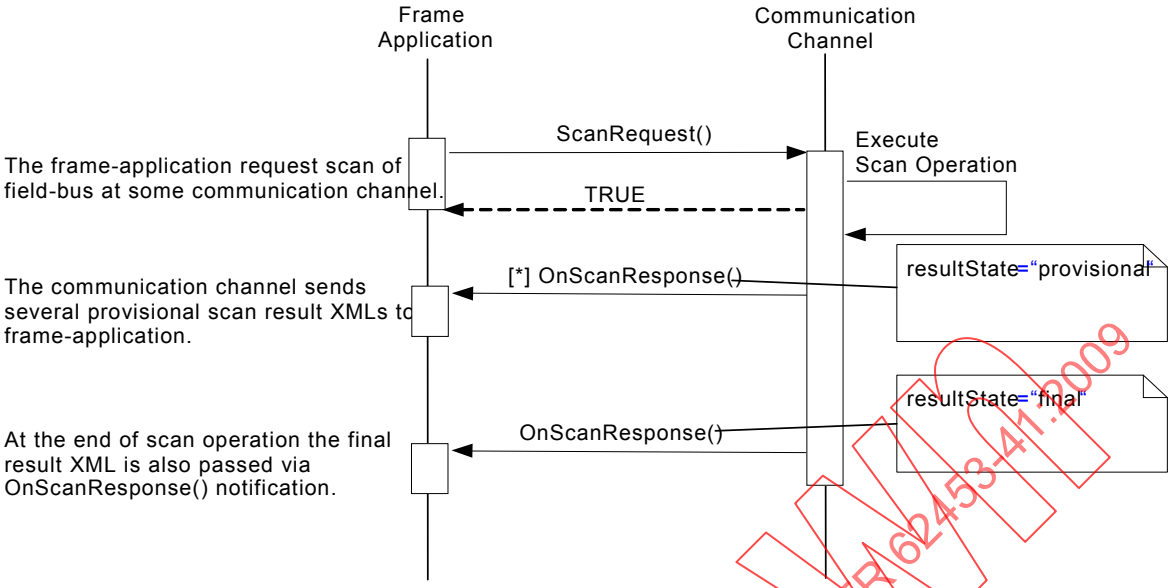
If the final response is not returned in expected time, user or Frame Application can cancel this action. No further OnScanResponse() should be called.

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 shall be fired by Communication Channel while scanning is performed.

7.3.3 Provisional scan result notifications

In this scenario Communication Channel sends provisional topology scan result XMLs to the Frame Application (see Figure 27).



Used methods:

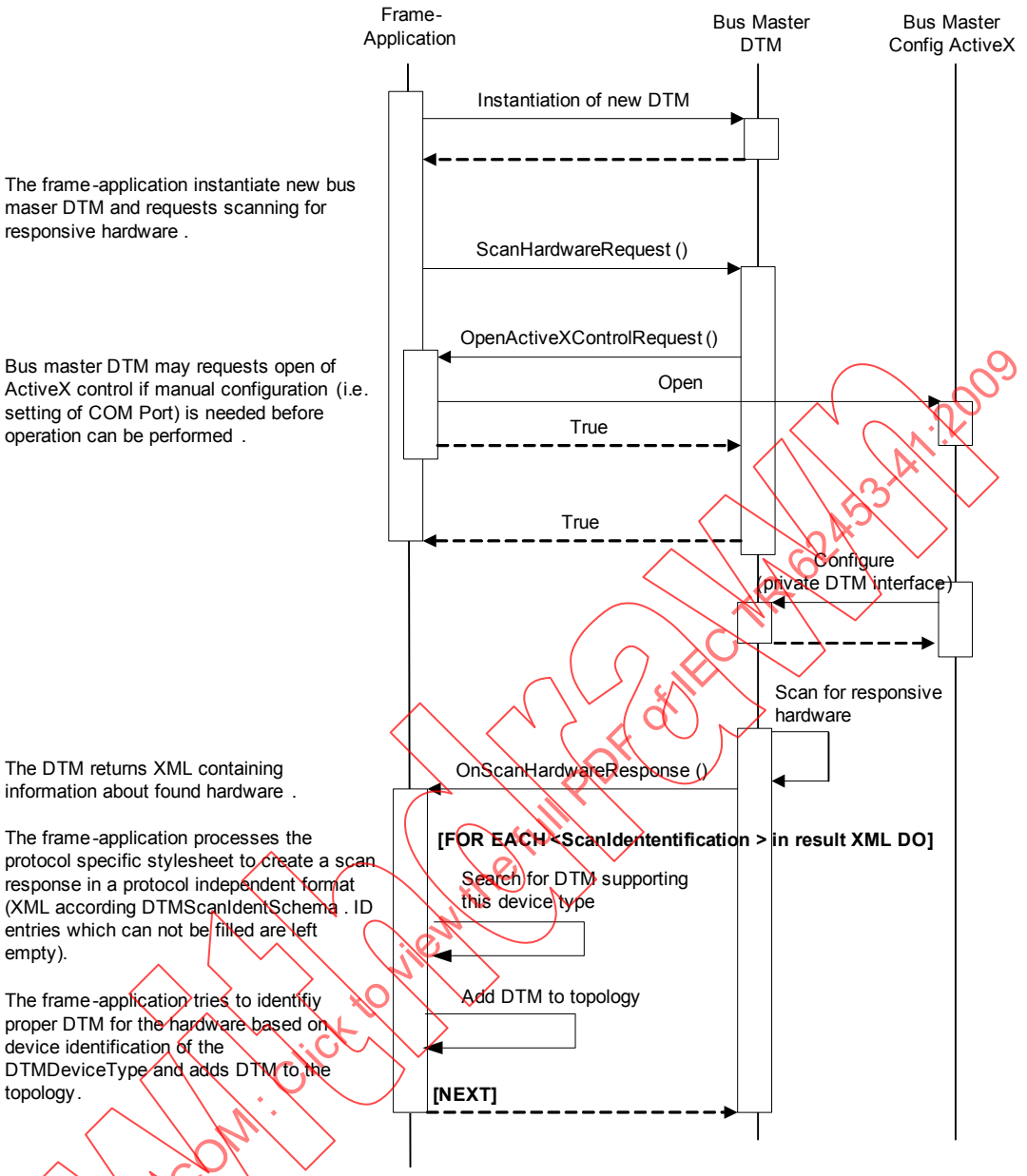
IFdtChannelScan::ScanRequest()

IDtmScanEvents::OnScanResponse()

Figure 27 – Provisional topology scan

7.3.4 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 28).



Used methods:

IDtmHardwareIdentification::ScanHardwareRequest()

IDtmScanEvents::OnScanHardwareResponse()

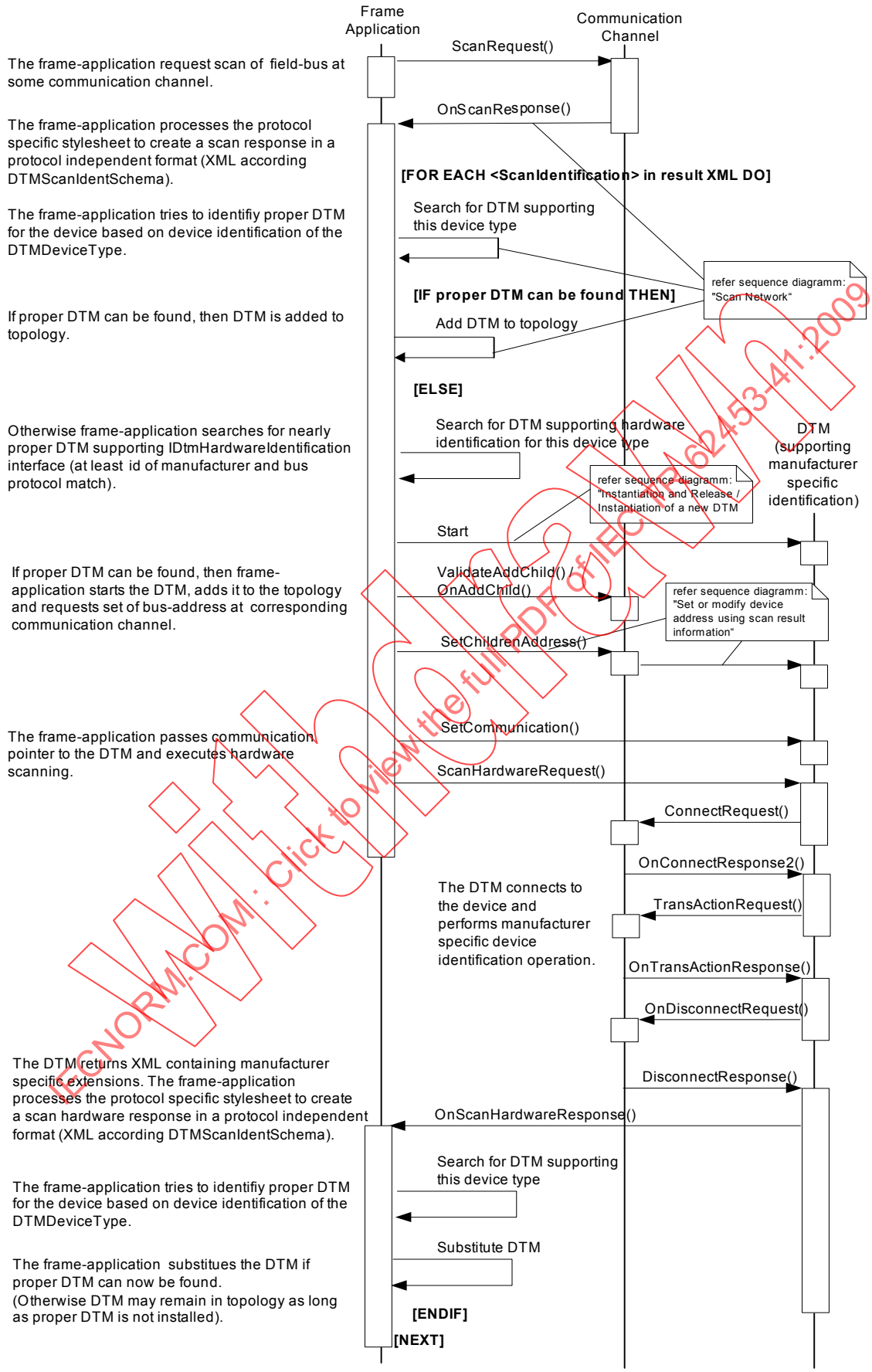
IFdtActiveX::OpenActiveXControlRequest()

(or IFdtActiveX2:OpenDialogActiveXControlRequest())

Figure 28 – Scan for communication hardware

7.3.5 Manufacturer specific device identification

In this scenario a Frame Application scans an existing field-bus network and uses DTM implementing IDtmHardwareIdentification interface to identify devices for which manufacturer specific operation shall be performed (see Figure 29).



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()

Figure 29 – Manufacturer specific device identification

7.4 Registration of protocol specific FDT schemas

Protocol specific schemas are stored in a sub-path of the FDT schema path. The name of this path is identical with the FDT fieldbus category id of the protocol (e.g. '036D1498-387B-11D4-86E1-00E0987270B9' for HART protocol)".

In order to add new protocol support to the FDT specification and to existing Frame Application installations, a CommDTM installation shall provide protocol specific schemas.

There are two cases:

a) Protocol specific schemas accepted and published by FDT Group. These schemas may be released independent on a FDT specification release.

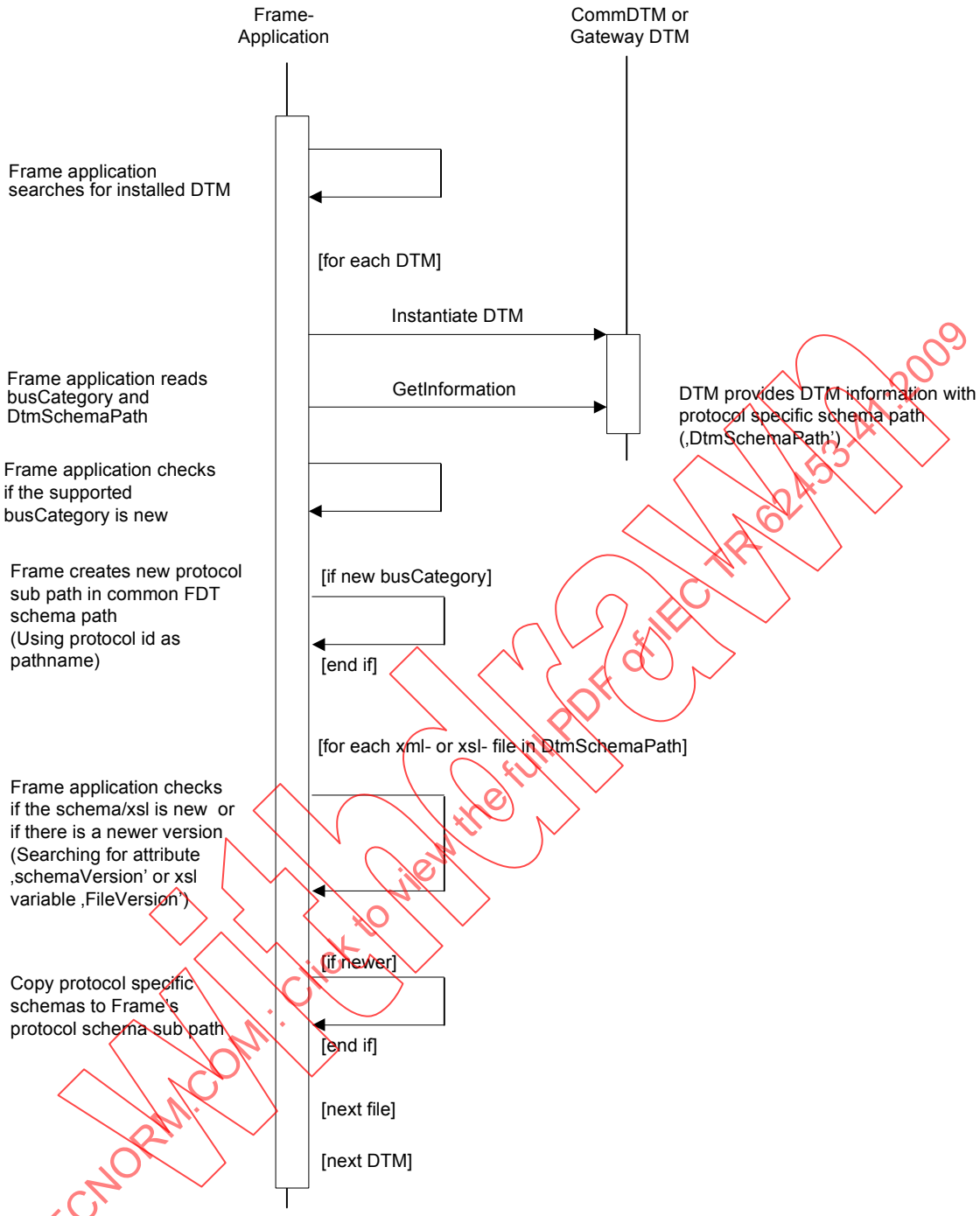
b) Schemas of a proprietary protocol. In this case all CommDTMs and Device DTMs are provided in one consistent package. Consistency of all related data shall be ensured internally by DTM.

The following structure explains an overall workflow (see Figure 30):

- 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 the sub schema path of a Frame Application and copies the schemas if not, or if a higher version is offered.
- Frame Application is responsible to synchronize the schema files over all subdirectories.



Used methods:

IDtmInformation::GetInformation()

Figure 30 − Add protocol specific schemas to Frame Applications schema sub path

After updating the schema cache, the Frame Application gets additional protocol specific device identification information from a DTM by calling IDtmInformation2::GetDeviceIdentificationInformation (see Figure 31).

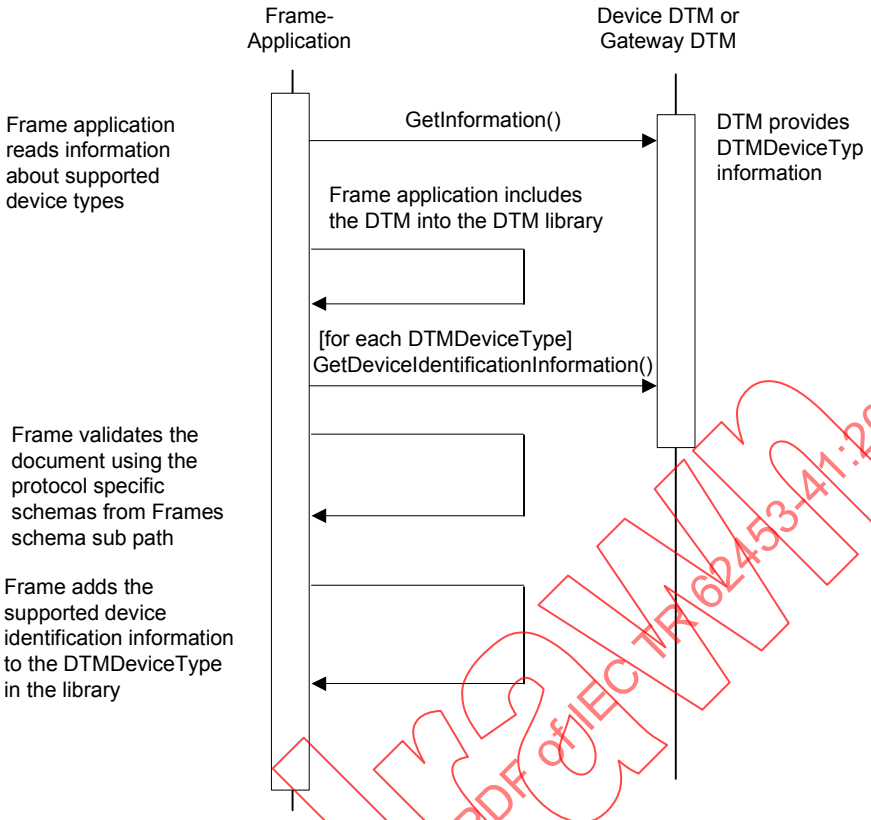


Figure 31 – Frame Application reads protocol specific device identification information of DTMDeviceTypes

7.5 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 (see Parts 3yx for protocol specific definitions).

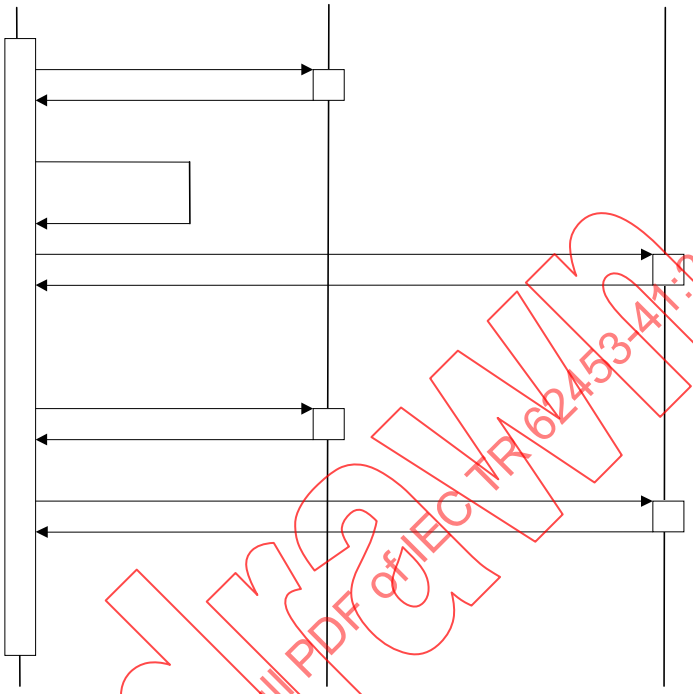
Each DTM shall at least fill in the device specific parameters and all parameters which can be changed by its application.

All other entries may be filled up with substitute values like zeroes. 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 32). For that purpose, it collects the <busMasterConfigurationPart> of each network participant and calculates the bus parameters of the corresponding master device.

The master configuration tool can be part of the DTM of the master device or like in the following example part of the Frame Application. If a DTM for a master device exists, the master configuration will be downloaded by IDtmOnlineParameter::DownloadRequest().



Used methods:

IDtmParameter::GetParameters()

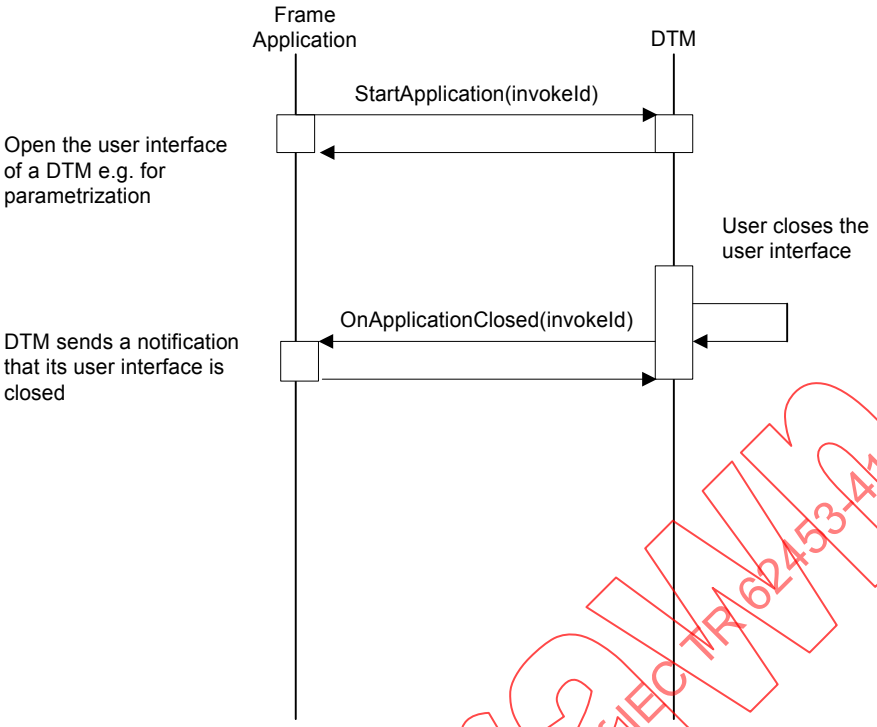
IDtmParameter::SetParameters()

IFdtTopology::GetChildNodes()

Figure 32 – Bus master configuration

7.6 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 sequence chart in Figure 33 shows how the Frame Application starts an application and how it handles the asynchronous behavior of the user interface via the invoke id. The same mechanism is used for ActiveX controls. The association between user interface and invoke id can always be used to synchronize DTM and Frame Application, independent whether the user closes the user-interface or it is closed by the Frame Application via ExitApplication() or PrepareToRelease().



Used methods:

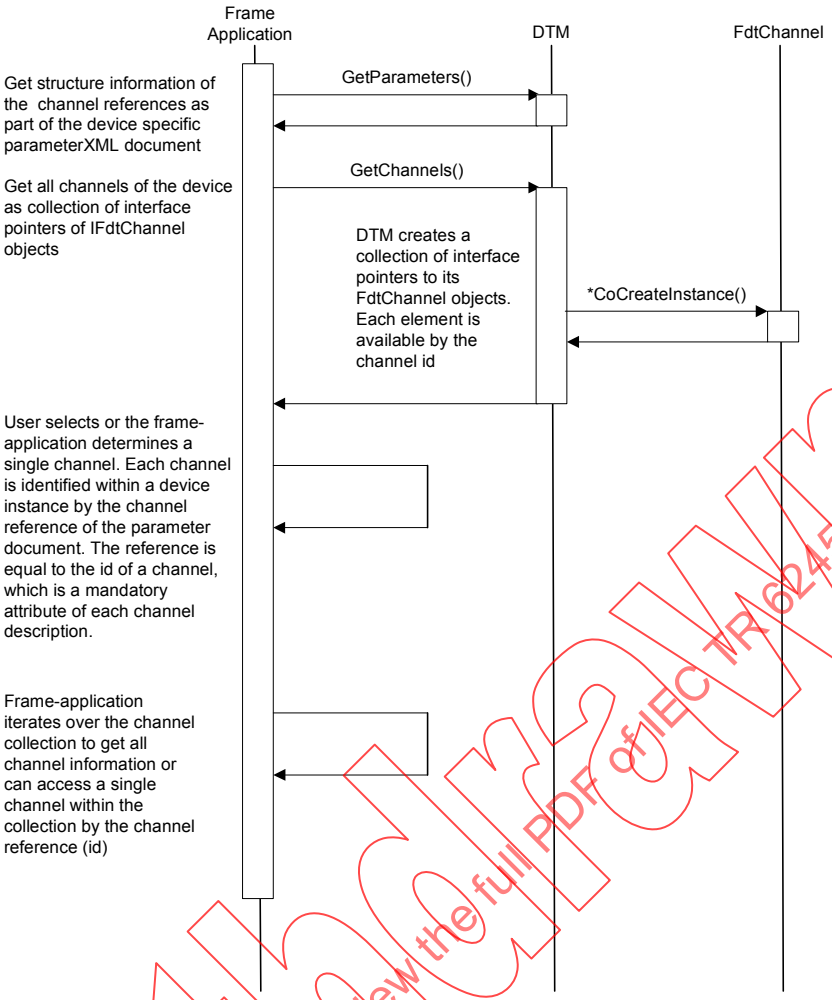
IDtmApplication::StartApplication()

IDtmEvents::OnApplicationClosed()

Figure 33 – Starting and releasing applications

7.7 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 FDT-Channel objects. These objects carry all address information for the configuration of a fieldbus master or a connected fieldbus controller. How to access such a FDT-Channel object is fieldbus independent (see Figure 34). But the information which is accessible as an XML document depends on the specific fieldbus protocol.



Used methods:

Standard Microsoft

IDtmParameter::GetParameters()

IDtmChannel::GetChannels()

Figure 34 – Channel access

7.8 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 35).

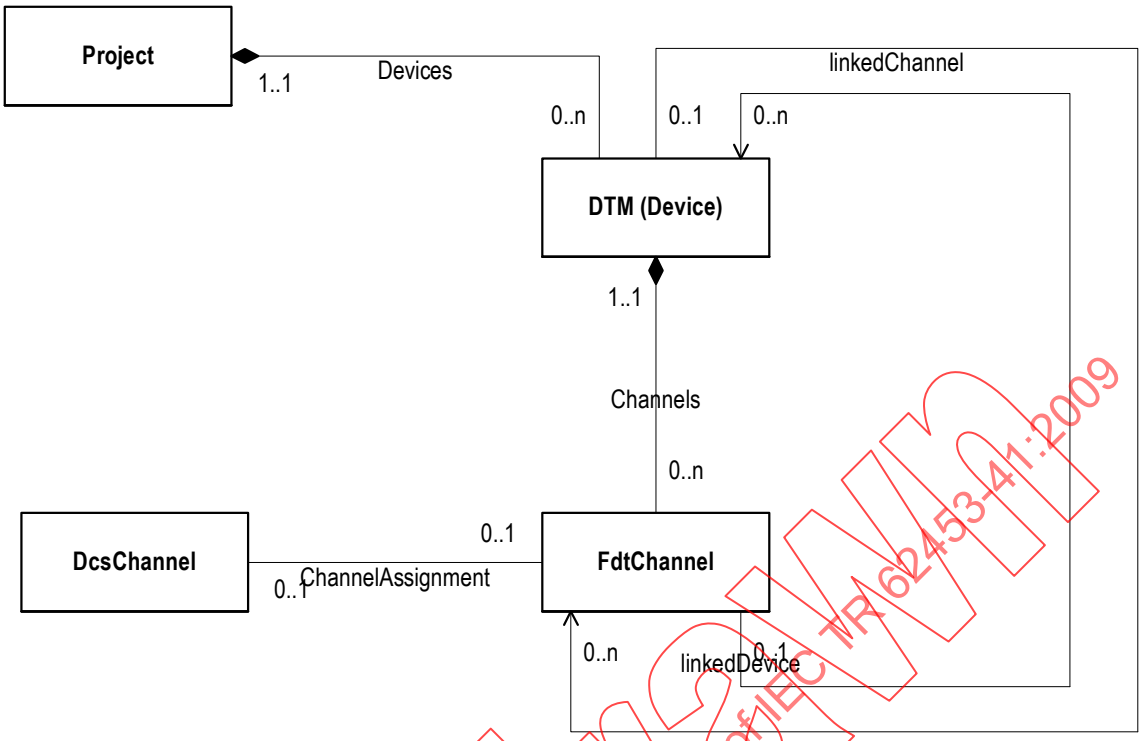
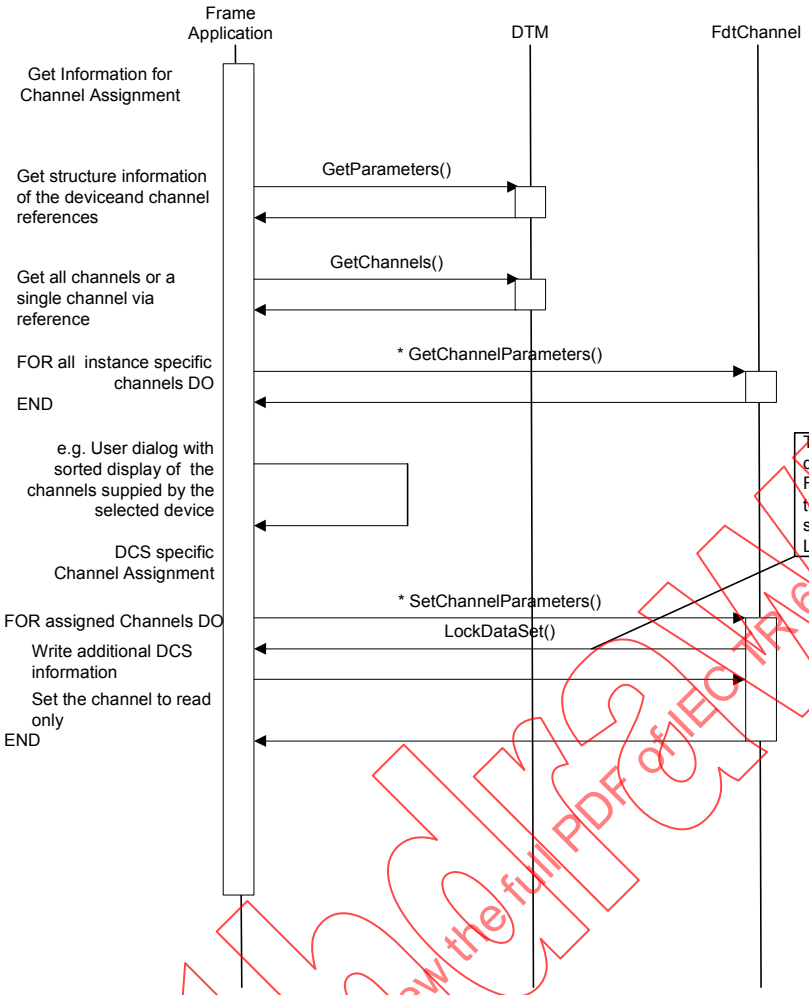


Figure 35 – DCS channel assignment single DTM

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 36).

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.



Used methods:

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

IDtmParameter::GetParameters()

IDtmChannel::GetChannels()

IFdtChannel::GetChannelParameters()

IFdtChannel::SetChannelParameters()

Figure 36 – Sequence of channel assignement for a single DTM

If a DTM instance represents a complete device, all information for channel assignment is available at this DTM.

In case of a modular device like remote I/Os which is represented by one DTM, the internal structure is also available via the parameter interface. From the channel assignment point of view this information allows the Frame Application to generate a structured presentation, so that there is no difference for a user whether he works with a single DTM or with several DTMs which represent the parts of the modular device.

Figure 37 shows a sub structure with DTMs for modular devices especially for remote I/Os with modules of different vendor. The connection of the Device DTM and its Module DTMs is established via the standard topology methods. At this sub structure the Device DTM is the gateway between the fieldbus and the proprietary backplane bus. So the communication can be realized by the mechanisms of nested communication.

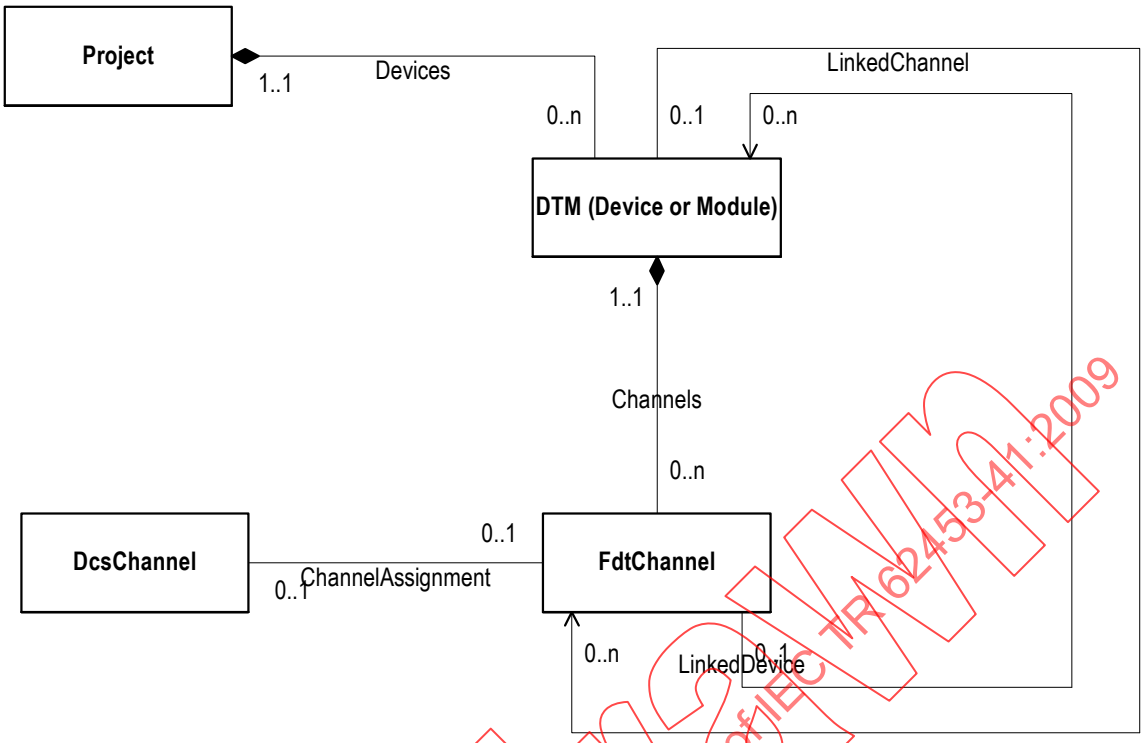
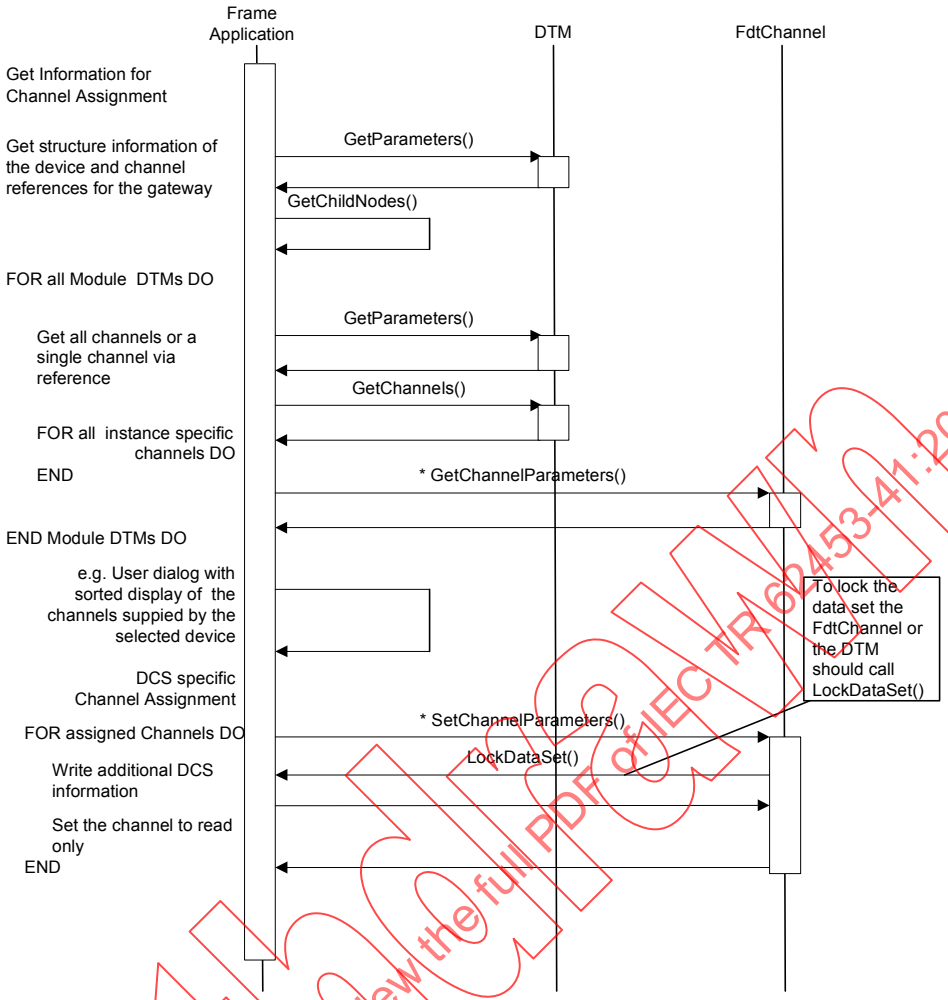


Figure 37 – Modular DTM structure

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

So if a DTM instance represents only a part of a device, the information for channel assignment has to be collected by the Frame Application

In case of a modular device like remote I/Os, the gateway is signed as main DTM and is the start point to collect the information of the sub DTMs which at least belong to the same device. So the internal structure is represented by the topology. From the channel assignment point of view the channel information together with the topology allows the Frame Application to generate a structured presentation, so that there is no difference for a user whether he works with a single DTM or with several DTMs which represent the parts of the modular device (see Figure 38).



Used methods:

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

IDtmParameter::GetParameters()

IDtmChannel::GetChannels()

IFdtChannel::GetChannelParameters()

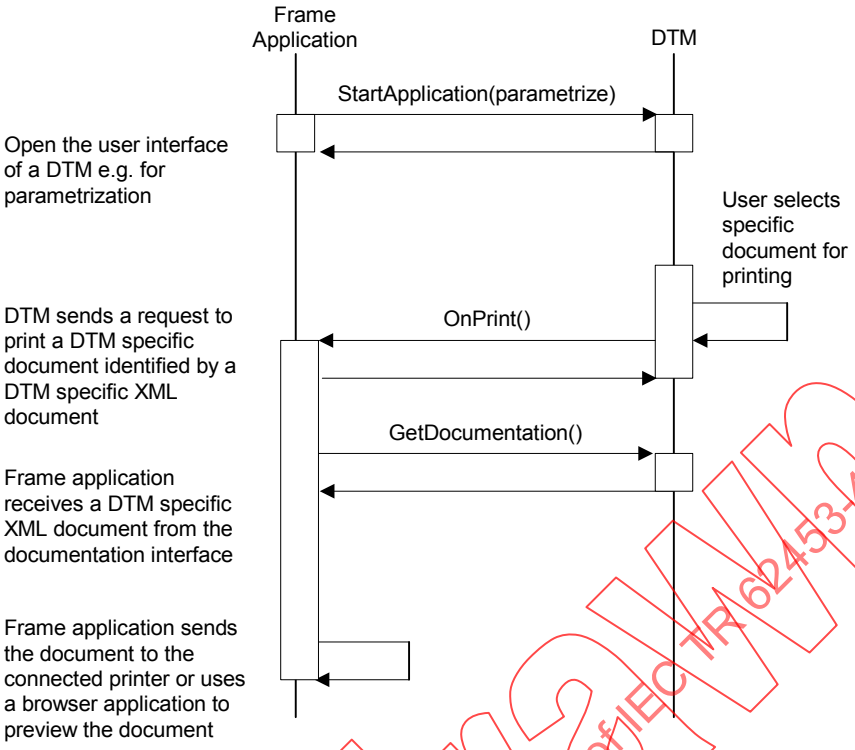
IFdtChannel::SetChannelParameters()

IFdtTopology::GetChildNodes()

Figure 38 – Channel assignment for modular DTMs

7.9 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 39). Beneath the documents defined by application id of a Frame Application a DTM can have device- or task-specific documents. These documents are not necessary for the integration itself but are mandatory for special environments like failsafe.



Used methods:

IDtmApplication::StartApplication()

IDtmEvents::OnPrint()

IDtmDocumentation::GetDocumentation()

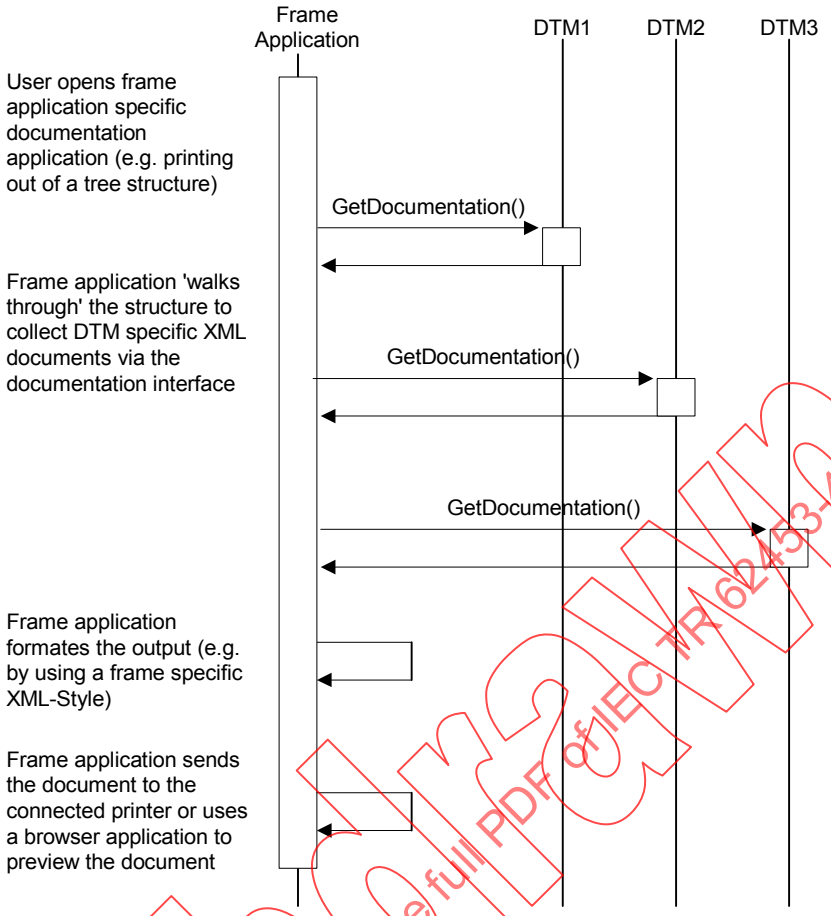
Figure 39 – Printing of DTM specific documents

Only functions that are exposed in the XML document provided by IDtm:GetFunctions() are printable. If a DTM requests printing of a function by call to IDtmEvents::OnPrint(), it has to provide a <Function> element with the appropriate functionId.

7.10 Printing of Frame Application specific documents

7.10.1 General

IDtmDocumentation allows the Frame Application to ask a DTM for context specific documents identified by the information passed via an XML document (see Figure 40).



Used methods:

IDtmDocumentation:: GetDocumentation()

Figure 40 – Printing of Frame Application specific documents

7.10.2 Processing a document

Figure 41 shows the general flow of events for processing of a document.

1) Frame Applications calls IDtmDocumentation.GetDocumentation() at DTM to request the documentation for a specific function (e.g. offline or online parameterization, configuration etc.).
2) DTM returns an XML document according DTMDocumentationSchema that contains the requested documentation.
3) Frame Application transforms the XML document into a human readable format, for example an HTML or PDF document. The transformation uses the information included in <DocumentVariables>, <DocumentVariable> and <GraphicalReference> XML elements.
4) Frame Application sends the generated documentation to a printer or displays it to the user.

Figure 41 – Report generation (Frame Application style)

Optionally, a DTM can define a DTM specific style sheet (XSL) that can be used by the Frame Application for transformation of the XML document to a device vendor specific HTML document as shown in Figure 42.

1) Frame Applications calls IDtmDocumentation.GetDocumentation() at DTM to request the documentation for a specific function (e.g. offline or online parameterization, configuration etc.).
2) DTM returns XML according DTMDocumentationSchema that contains requested documentation + specific style sheet embedded in the XML document <DTMStyleForCompleteDocument>.
3) Frame Application extracts DTM specific style sheet from the XML document <DTMStyleForCompleteDocument>.
4) Frame Application uses the DTM specific style sheet for transformation of the returned XML document to a HTML document.
5) Frame Application sends the generated HTML documentation to a printer or displays it to the user.

Figure 42 – Report generation (device vendor specific style)

7.10.3 Rules for use of DTM specific style sheets

7.10.3.1 Use of XML elements

The use of DTM specific style sheets is optional for a Frame Application. Some Frame Applications may ignore the embedded style sheets if consistent plant wide documentation or non HTML reports shall be generated.

DTM also shall provide complete information in <DocumentVariables>, <DocumentVariable> and <GraphicalReference> XML elements even if a specific style sheet is embedded in the returned XML document. That means providing the information in <DTMSpecificXMLData> XML element only is not sufficient. This XML element may contain additional information that is only used if the vendor specific style is applied.

7.10.3.2 Transformation to HTML

The embedded DTM specific style sheet shall transform the complete XML document to HTML format.

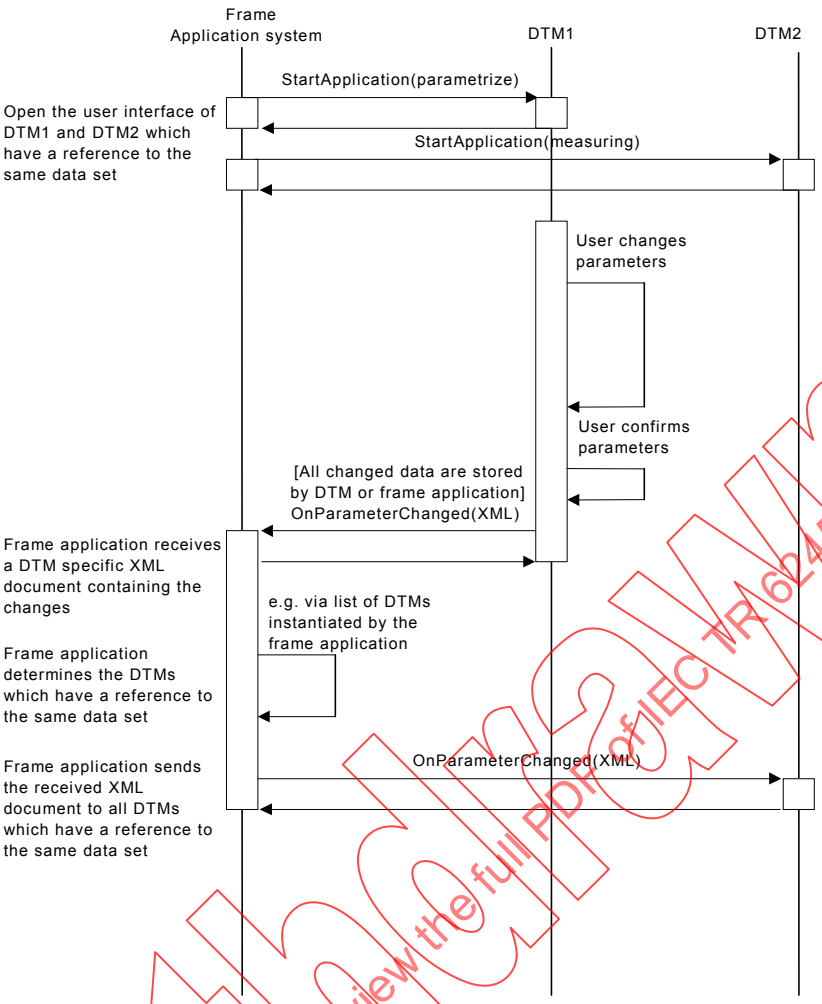
7.10.3.3 XML transformation language

The embedded DTM specific style sheet shall use the W3C XSL transformation version 1 syntax (refer to XSL Transformations (XSLT), Version 1.0, W3C Recommendation 16 November 1999).

7.11 Propagation of changes

Within a multi-user environment it is common, that more than one DTM have access to the same data set. Such an environment may be a Frame Application executed as distributed system on several PCs (Frame Application System).

To synchronize DTMs which are started by several users on different workplaces FDT provides a notification mechanism via OnParameterChanged() (see Figure 43). 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 shall have a common agreement about the contents and schema of the XML document send for propagation of changes. The Frame Application only passes the document to all DTMs which have a reference to the same data set.



Used methods:

IDtmApplication::StartApplication()

IDtmEvents::OnParameterChanged()

IFdtEvents::OnParameterChanged()

Figure 43 – Propagation of changes

7.12 Locking

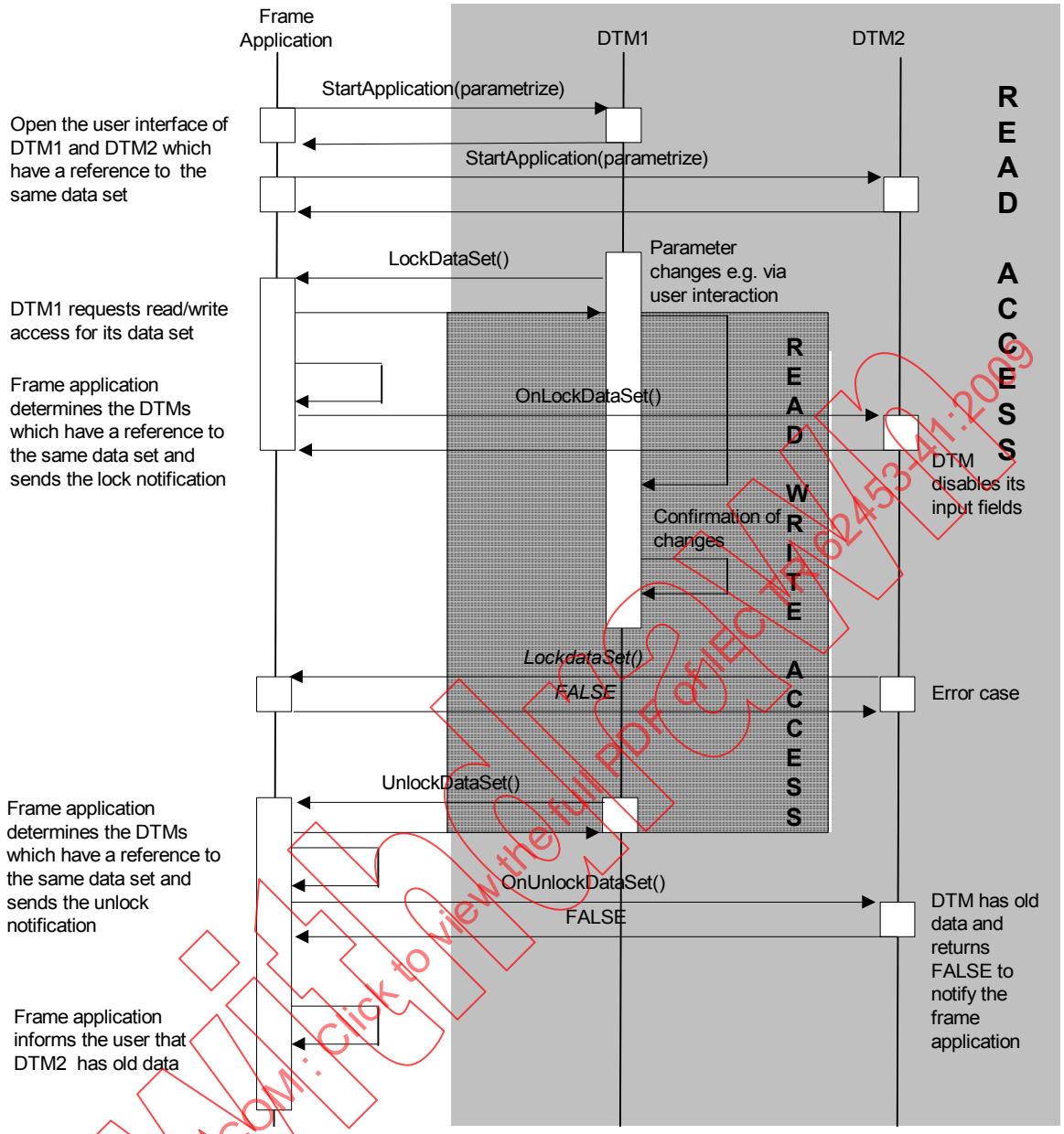
Within a multi-user environment it is common, that more than one DTM have access to the same data set. To synchronize DTMs which are started by several users on different workplaces FDT provides a locking mechanism. Target for this event mechanism is that only one DTM has read/write access to the data set. All other DTMs have read access only.

For this reason a DTM shall lock its 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 shall unlock its 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 shall try to lock the dataset. If successful all user input fields can be enabled, in the other case user input fields shall 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 shall 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 its dataset after instantiation and unlocks during, for example a IDtm::PrepareToRelease() this DTM is not suitable for use within a multi-user environment.

7.12.1 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 shall be implemented to support multi-user environments (see Figure 44).



Used methods:

IDtmApplication::StartApplication()

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

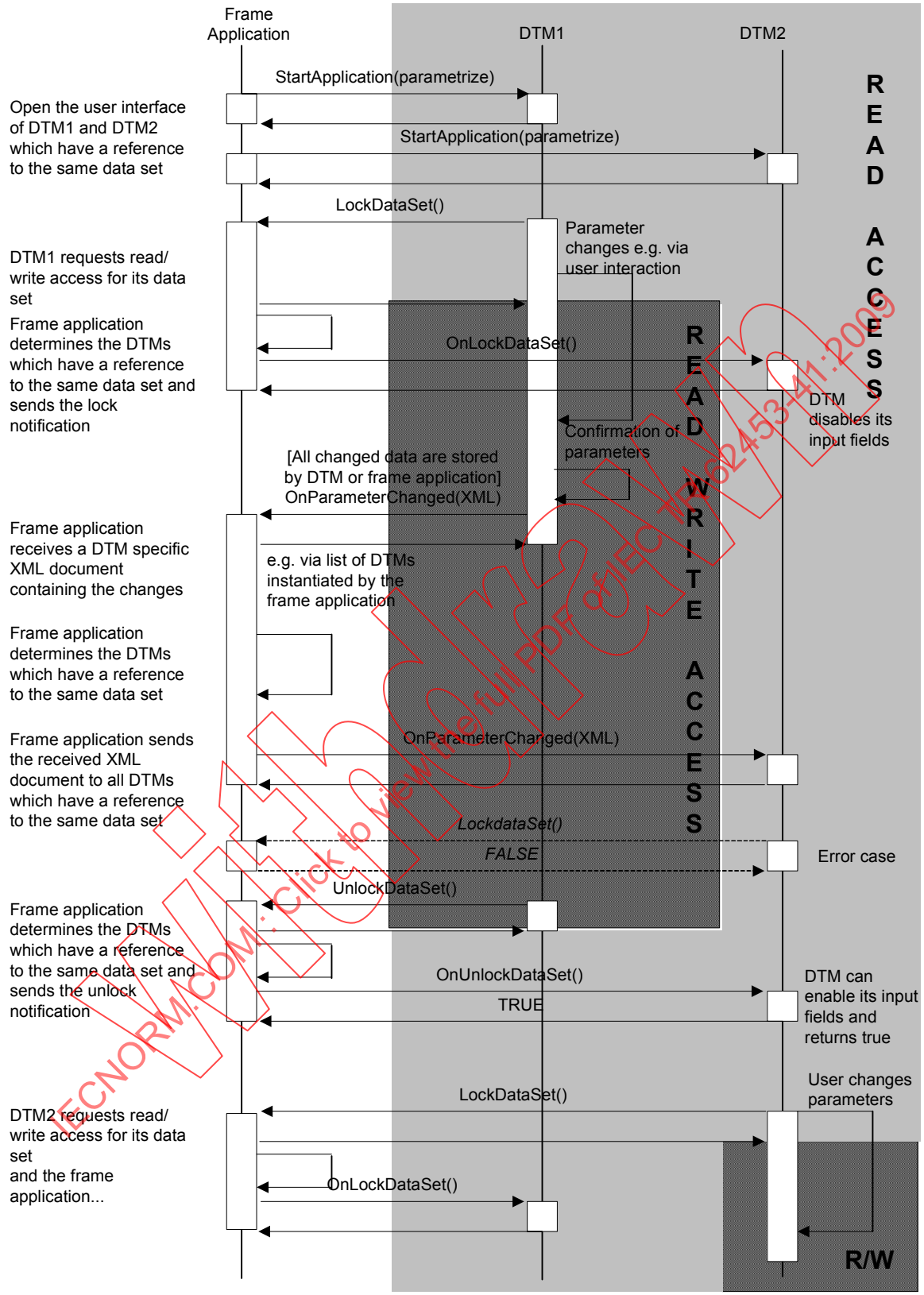
IFdtEvents::OnLockDataSet()

IFdtEvents::OnUnlockDataSet()

Figure 44 – Locking for non-synchronized DTMs

7.12.2 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 45).



Used methods:

IDtmApplication::StartApplication()

IFdtContainer::LockDataSet()

IFdtContainer::UnlockDataSet()

IDtmEvents::OnParameterChanged()

IFdtEvents::OnParameterChanged()

IFdtEvents::OnLockDataSet()

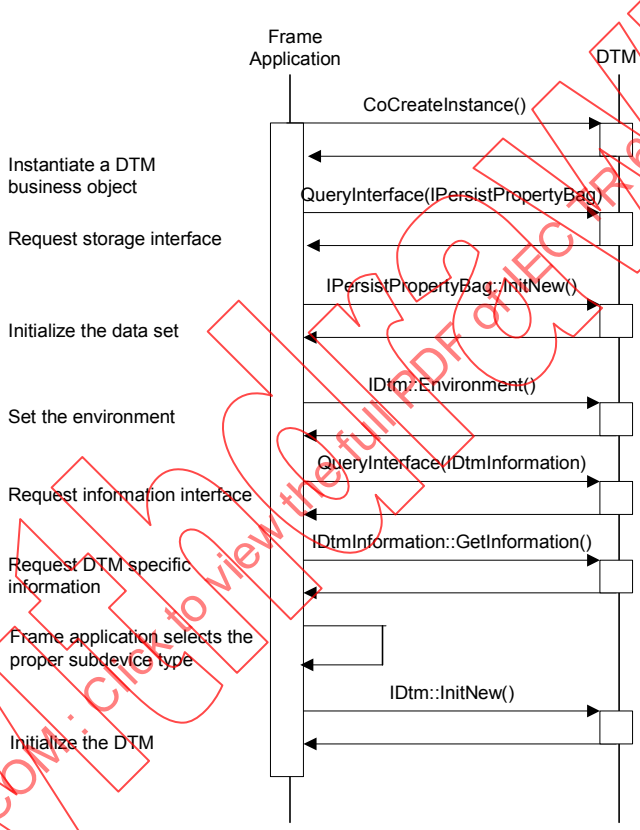
IFdtEvents::OnUnlockDataSet()

Figure 45 – Locking for synchronized DTMs

7.13 Instantiation and release

7.13.1 Instantiation of a new DTM

After creation of a DTM business object the Frame Application shall request the IPersistXXX interface pointer and call IPersistXXX::InitNew() (see Figure 46). Within this method the DTM business object shall initialize its default device parameters.



Used methods:

Standard Microsoft

IDtm::Environment()

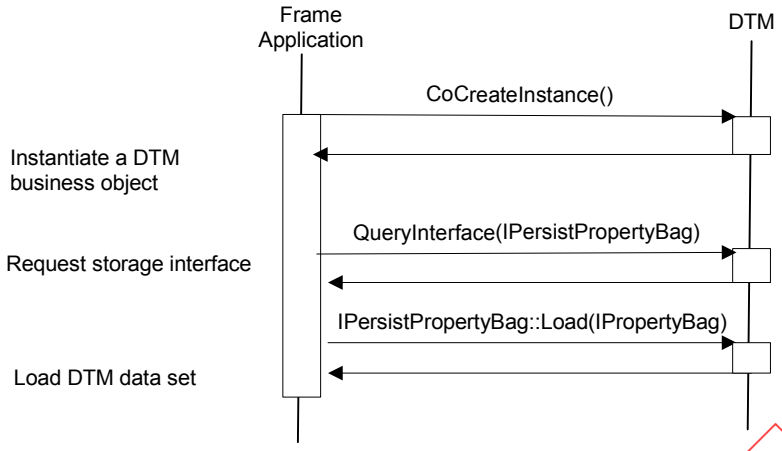
IDtm::InitNew()

IDtmInformation::GetInformation()

Figure 46 – Instantiation of a new DTM

7.13.2 Instantiation of an existing DTM

After creation of a DTM business object for an existing instance the Frame Application shall request the IPersistXXX interface pointer and call IPersistXXX::Load() with a reference to the stream object of the appropriate field device instance (see Figure 47). Within this method the DTM business object shall initialize its device parameters based on the information of the stream object.



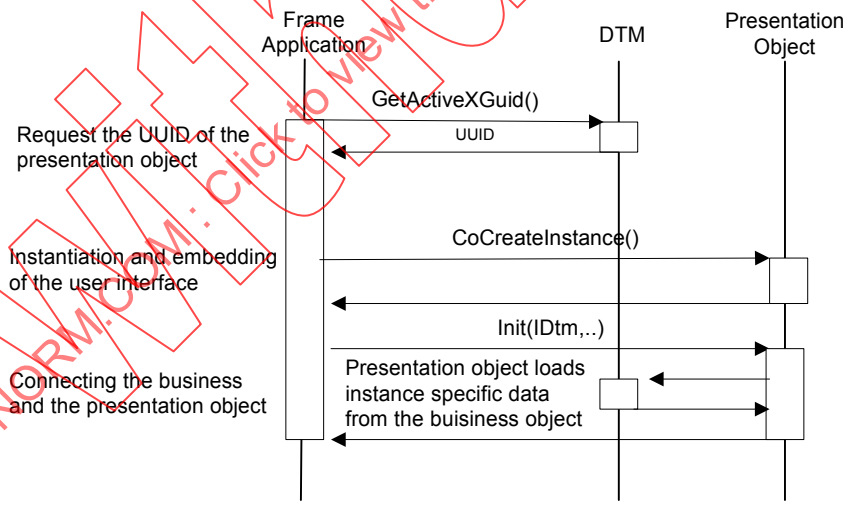
Used methods:

Standard Microsoft

Figure 47 – Instantiation of an existing DTM

7.13.3 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 48). Within the Init() method the presentation object shall initialize its device parameters based on the information of the business object.



Used methods:

Standard Microsoft

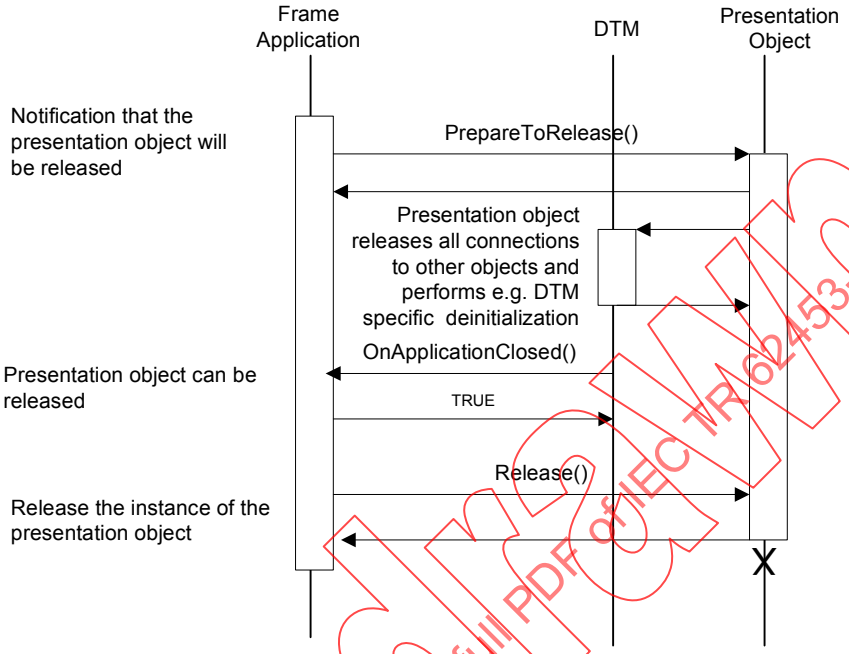
IDtmActiveXControl::Init()

IDtmActiveXInformation::GetActiveXGuid()

Figure 48 – Instantiation of a DTM user interface

7.13.4 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 49). After the IDtmActiveXControl::PrepareToRelease() method the presentation object shall release all its connections to other components and can call DTM specific de-initialization methods.



Used methods:

Standard Microsoft

IDtmActiveXControl::PrepareToRelease()

IDtmEvents::OnApplicationClosed()

Figure 49 – Release of a DTM user interface

7.14 Persistent storage of a DTM

7.14.1 State machine of instance data

7.14.1.1 Modifications

This state machine reflects the possible states of an instance data set concerning modifications (see Figure 50).

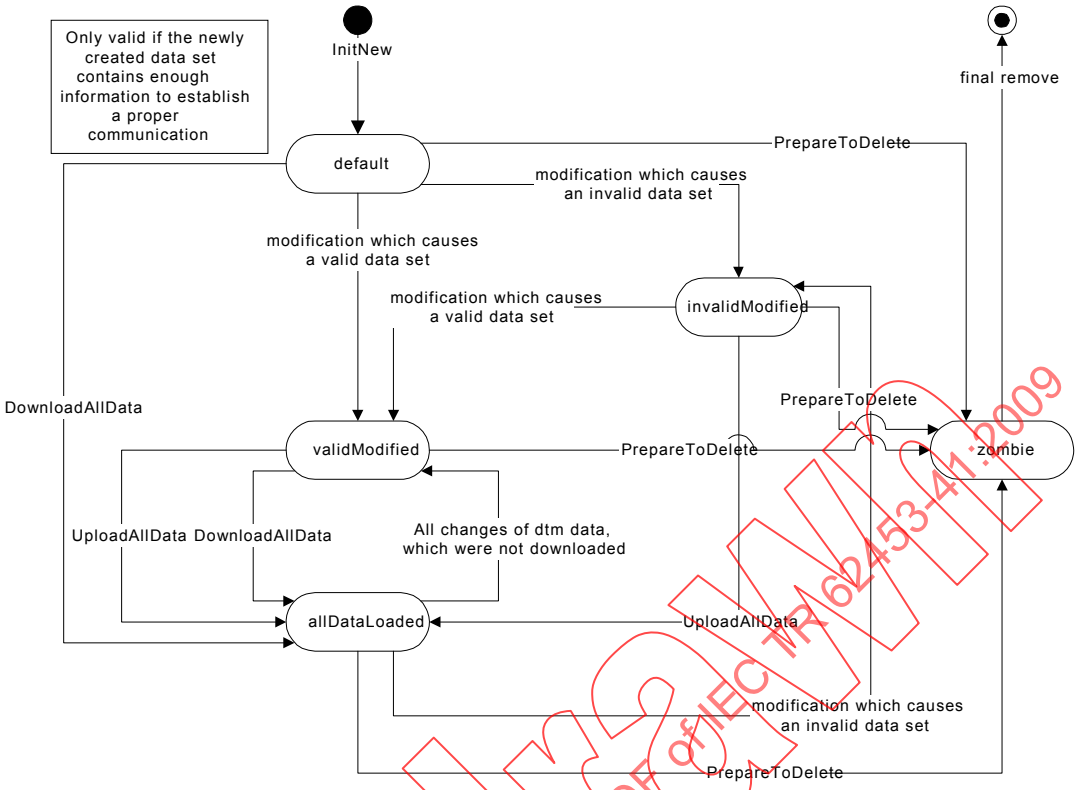


Figure 50 – State machine of instance data set

The meaning of the different states can be seen in Table 13.

Table 13 – 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 means not, that the data within the device are equal to the data found within the related instance data set. Due to the fact that a user can use tools out of the scope of FDT, the FDT specification cannot guarantee such a state.
zombie	The instance data set is prepared to delete, no access to this data is allowed.

7.14.1.2 Persistence states

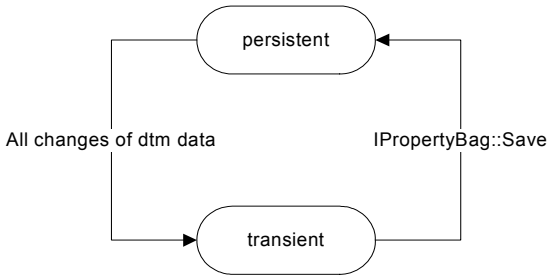


Figure 51 – Persistence states of a data set

This state machine reflects the possible states of an instance data set concerning the persistence of data (see Figure 51). Description of the states can be found in Table 14.

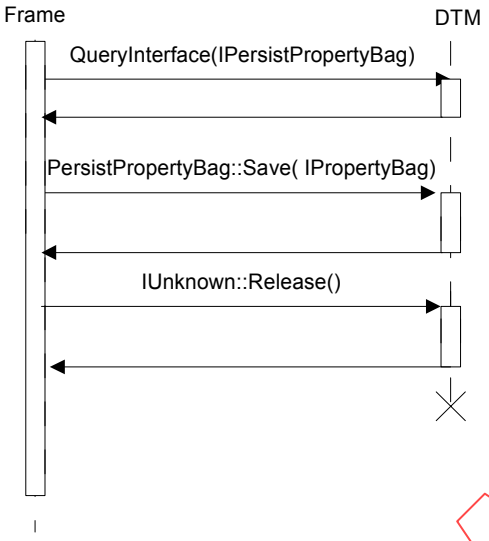
Table 14 – Description of persistent states

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.

7.14.2 Saving instance data of a DTM

To save the private data of a DTM object the Frame Application shall 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 52). Within this method the DTM business object shall save all its device parameters necessary to re-establish its complete state based on the information of the stream / property bag object within a IPersistXXX::Load() call.

After saving private data of a DTM the Frame Application is able to release the DTM business object.



Used methods:

Standard Microsoft

Figure 52 – Saving instance data of a DTM

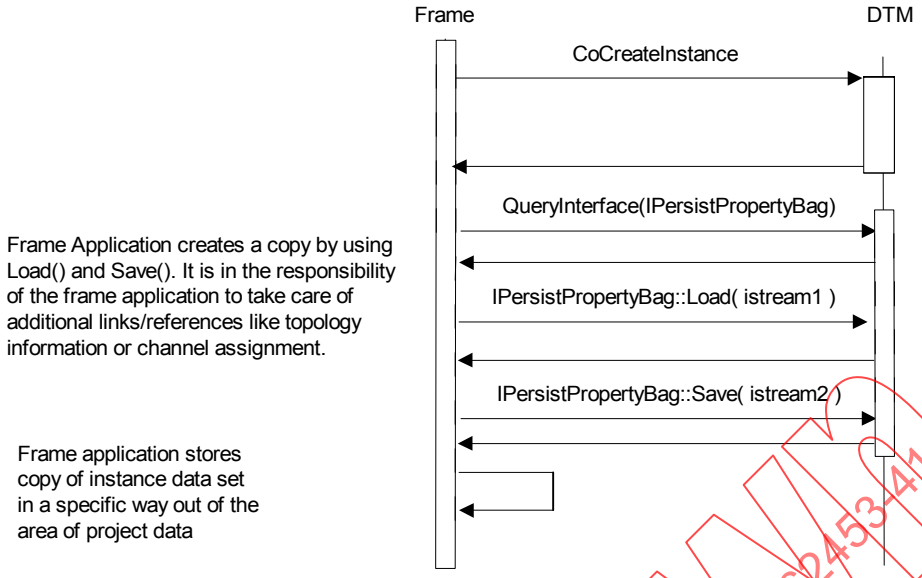
7.14.3 Reload of a DTM object for another instance

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 shall instantiate a new DTM COM object.

7.14.4 Copy and versioning of a DTM instance

After creation of a DTM object the Frame Application shall 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 53). Within this method the DTM business object shall initialize its 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 shall call IPersistXXX::Save() with a reference to a new stream / property bag object.

It is up to the Frame Application to handle the Frame Application specific versioning aspects and to manage the different instance data sets for a device.



Used methods:

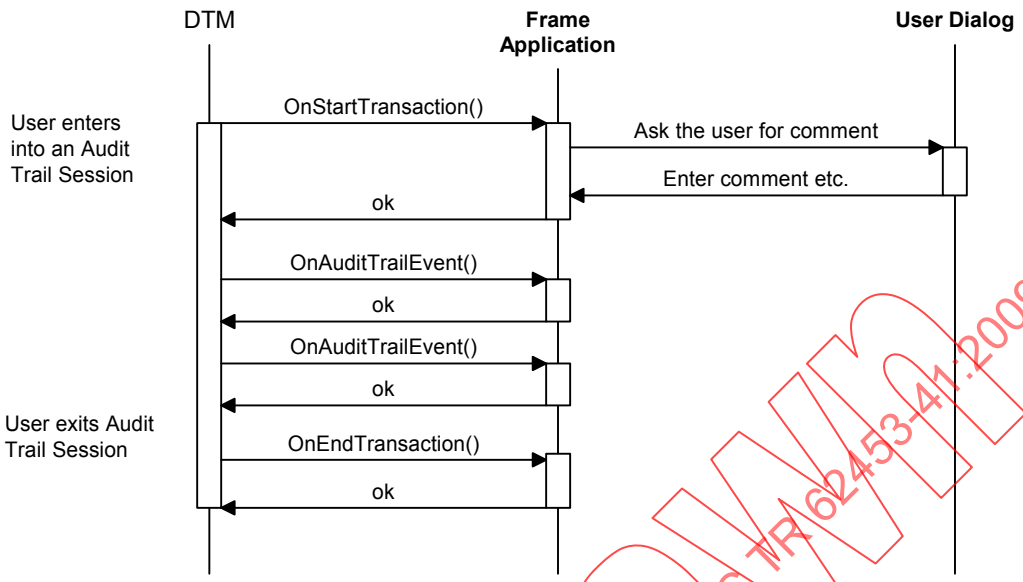
Standard Microsoft

Figure 53 – Copy and versioning of a DTM instance

7.15 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 54).



Used methods:

IDtmAuditTrailEvents::OnStartTransaction()
IDtmAuditTrailEvents::OnAuditTrailEvent()
IDtmAuditTrailEvents::OnEndTransaction()

Figure 54 – Audit trail

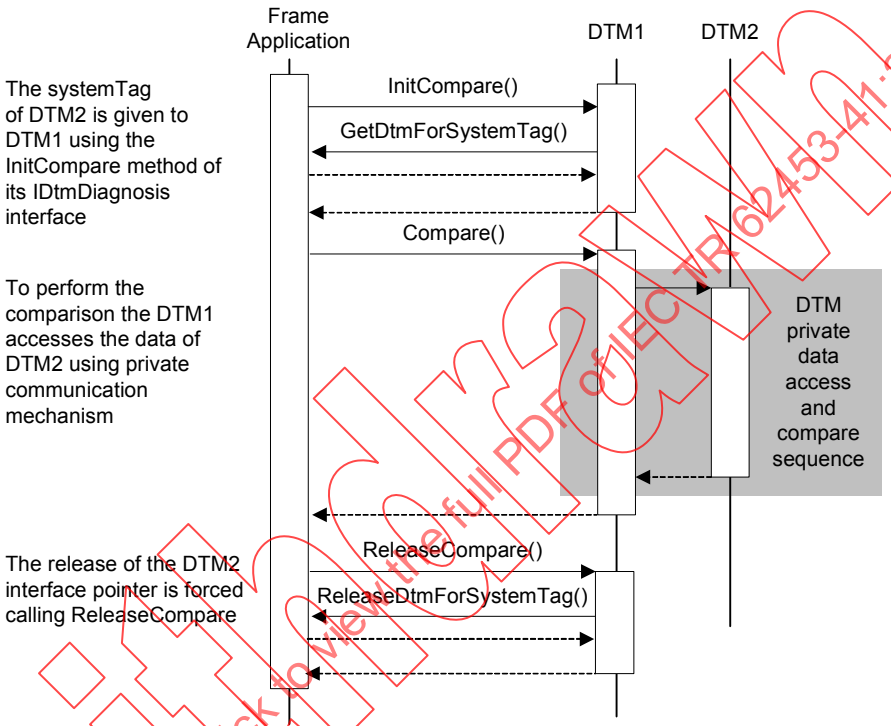
7.16 Comparison of two instance data sets

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

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

7.16.1 Comparison without user interface

The sequence chart for the comparison of two instance data sets is shown in Figure 55:



Used methods:

IDtmDiagnosis::InitCompare()

IDtmDiagnosis::Compare()

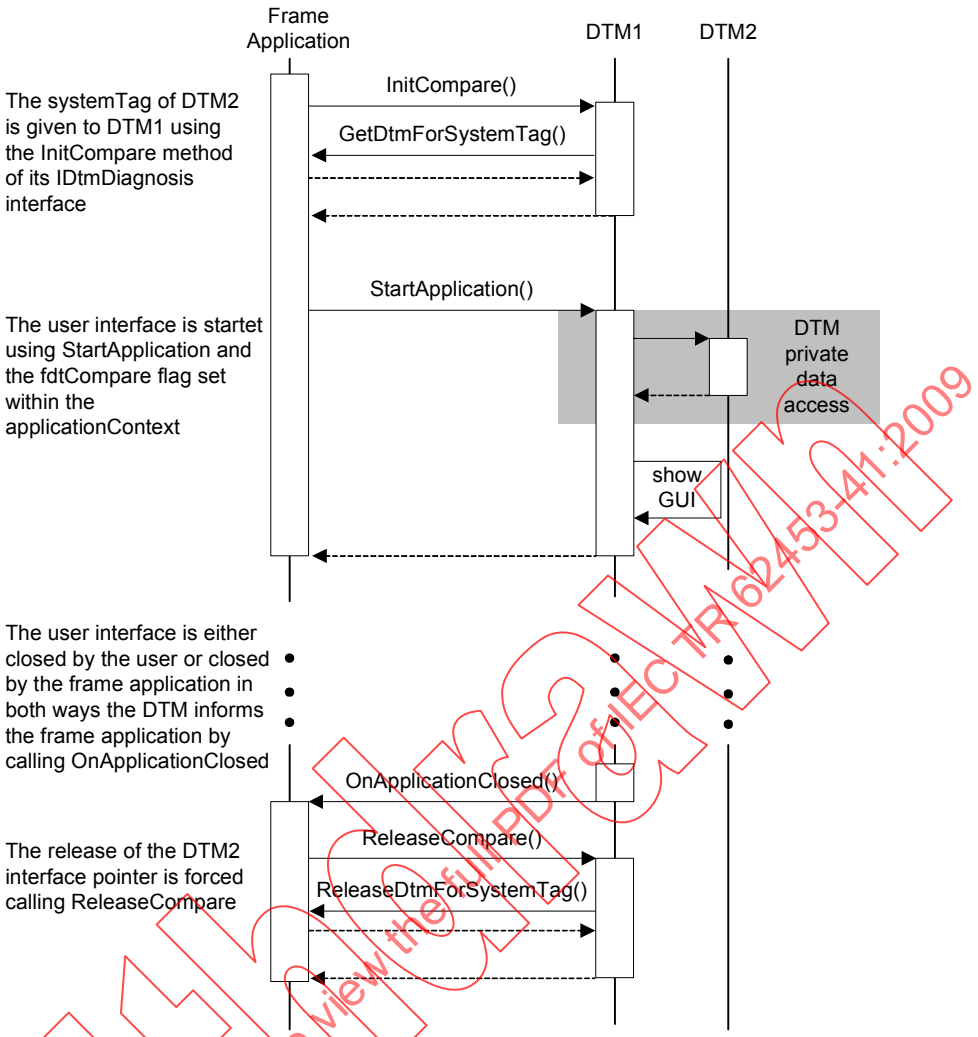
IFdtTopology::GetDtmForSystemTag()

IFdtTopology::ReleaseDtmForSystemTag()

Figure 55 – Comparison without user interface

7.16.2 Comparison with user interface

To invoke a user interface for comparison, the sequence shown in Figure 56 is to be performed.



Equivalent to StartApplication() (as shown in the sequence chart) ActiveX® controls can be used.

Used methods:

IDtmDiagnosis::InitCompare()

IDtmDiagnosis::ReleaseCompare()

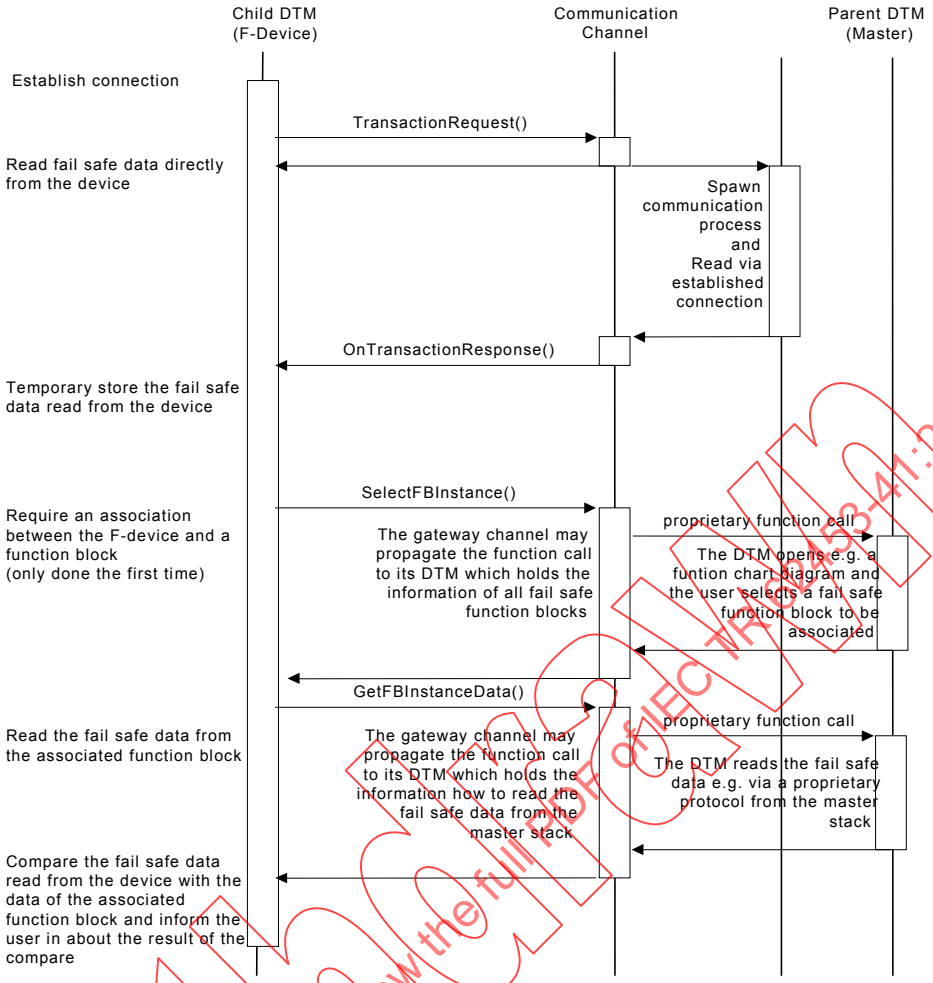
IDtmApplication::StartApplication()

IDtmEvents::OnApplicationClosed()

Figure 56 – Comparison with user interface

7.17 Failsafe data access

The sequence chart in Figure 57 shows how to access failsafe data from a device and its associated function block via two independent communication links.



Used methods:

IFdtCommunication::TransActionRequest()

IFdtCommunicationEvents::OnTransactionResponse()

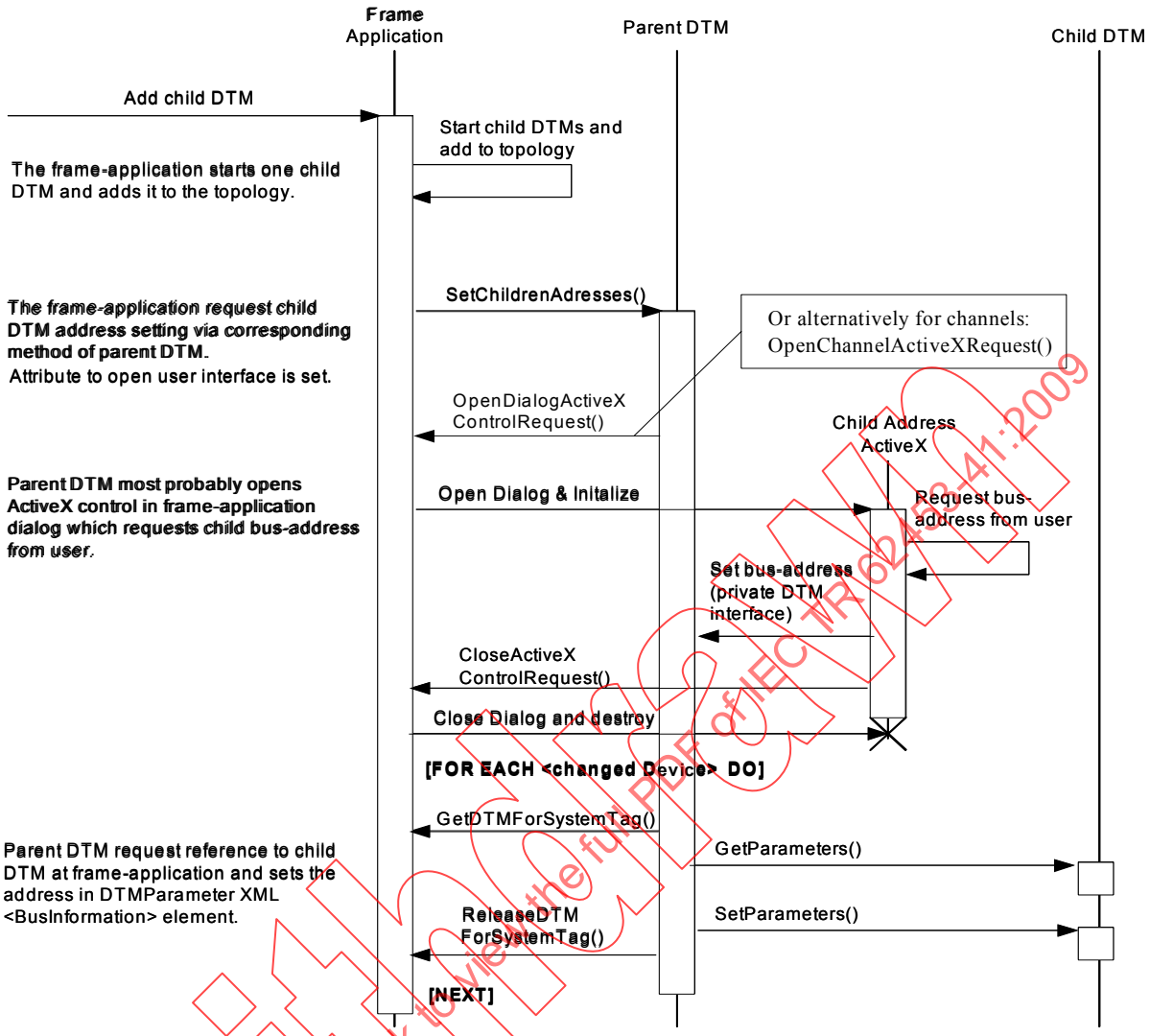
IFdtFunctionBlockData::SelectFBInstance()

IFdtFunctionBlockData::GetFBInstanceData()

Figure 57 − Failsafe data access

7.18 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 58) 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.



Used methods:

IFdtChannelSubTopology2::SetChildrenAddresses()

IFdtActiveX2:OpenDialogActiveXControlRequest()

IFdtActiveX::CloseActiveXControlRequest()

IFdtTopology::GetDtmForSystemTag()

IFdtTopology::ReleaseDtmForSystemTag()

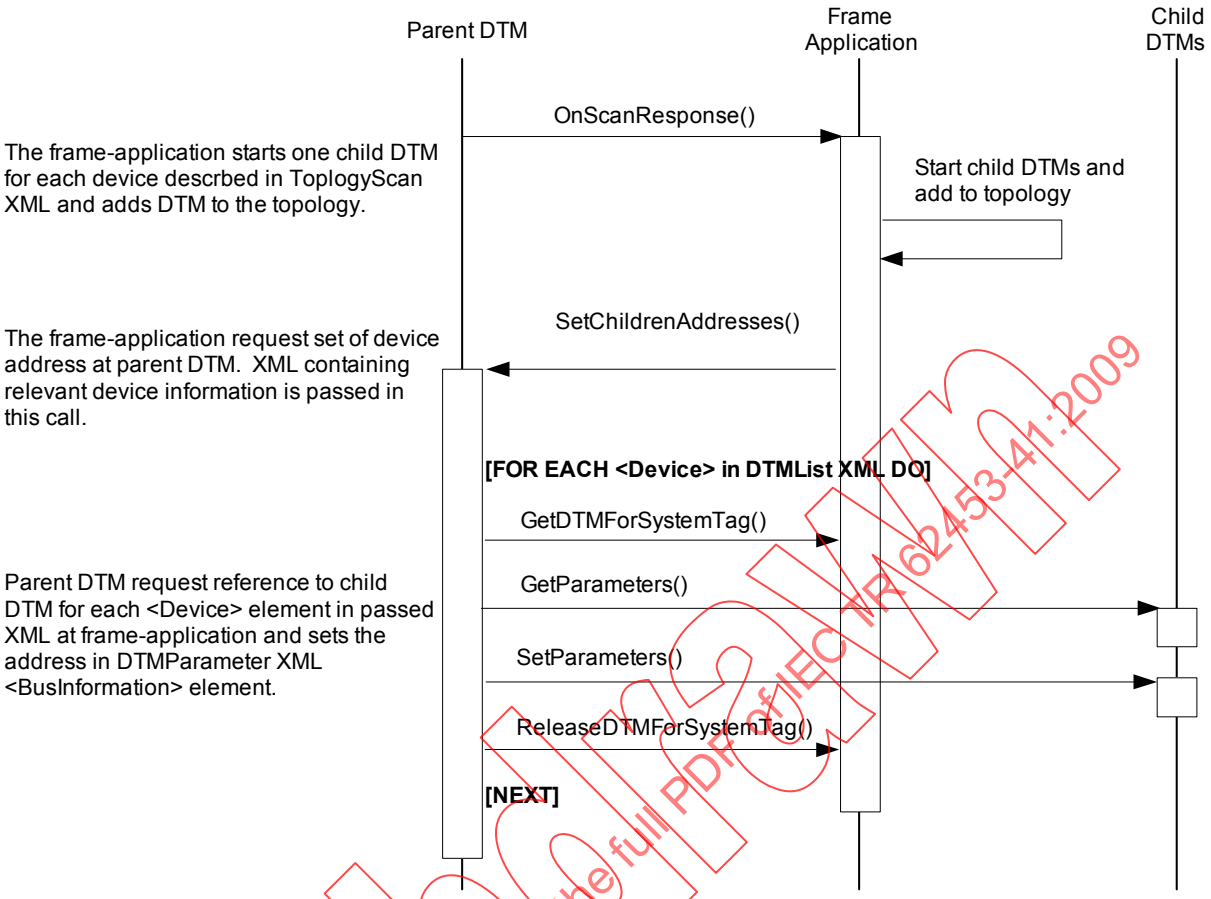
IDtmParameter::SetParameters()

IDtmParameter::GetParameters()

Figure 58 − Set or modify device address with user interface

7.19 Set or modify known device addresses without user interface

In this scenario, the Frame Application requests a set of device addresses at DTM, for example after a scanning or import operation (see Figure 59).



Used methods:

IFdtChannelSubTopology2::SetChildrenAddresses()

IDtmEvents::OnScanResponse()

IFdtTopology::GetDtmForSystemTag()

IFdtTopology::ReleaseDtmForSystemTag()

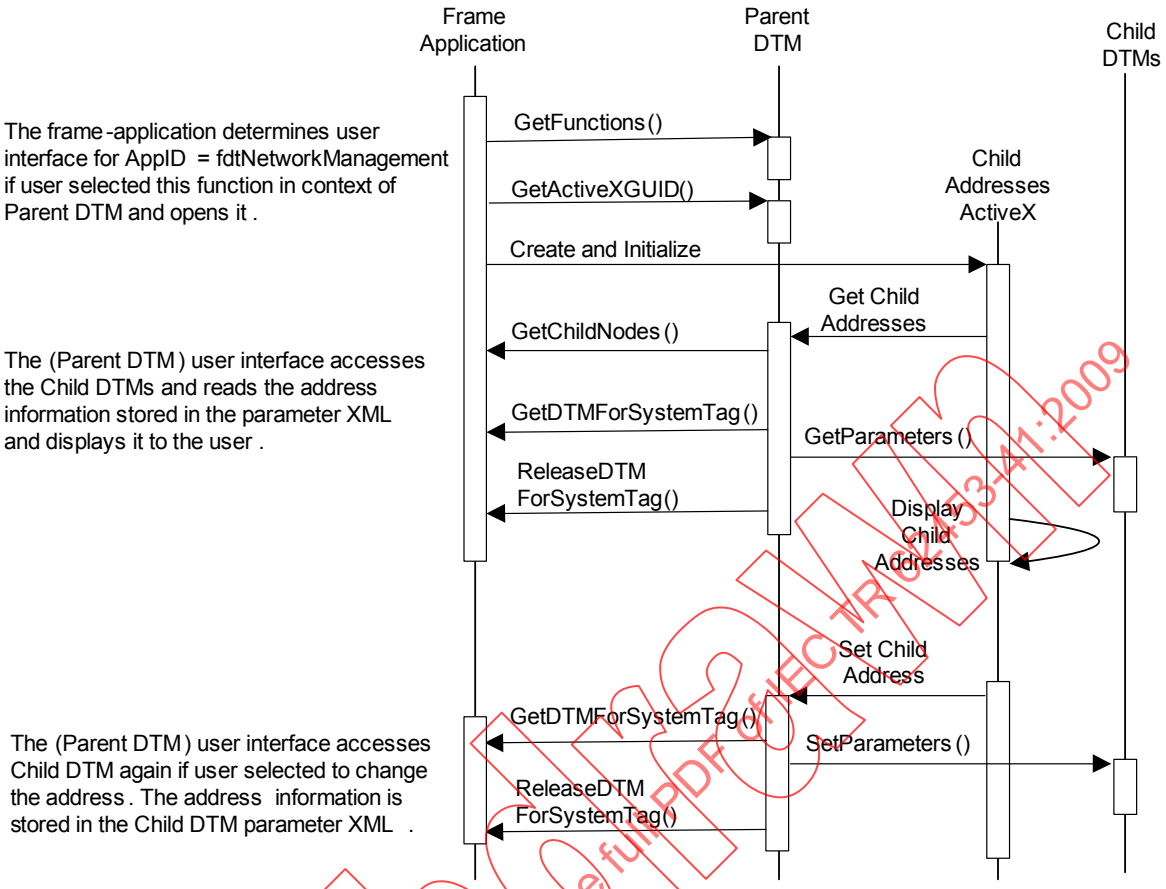
IDtmParameter::SetParameters()

IDtmParameter::GetParameters()

Figure 59 – Set or modify known device addresses without user interface

7.20 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 60) for example is started when a user selects the corresponding menu entry in context of a Communication DTM or a Gateway DTM.



Used methods:

IDtm::GetFunctions()

IDtmActiveXInformation::GetActiveXGuid()

IFdtTopology::GetChildNodes()

IFdtTopology::GetDtmForSystemTag()

IFdtTopology::ReleaseDtmForSystemTag()

IDtmParameter::SetParameters()

IDtmParameter::GetParameters()

Figure 60 – Display or modify all child device addresses with user interface

7.21 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 approach shown in Figure 61 is recommended:

- the infrastructure (filter, service queue) for such services is initiated by a protocol-specific transaction request of the DTM, dependant on the protocol, this service may be acknowledged or not;
- the device initiated data transfer is transported by transaction responses to the initiating request (they carry the same 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.

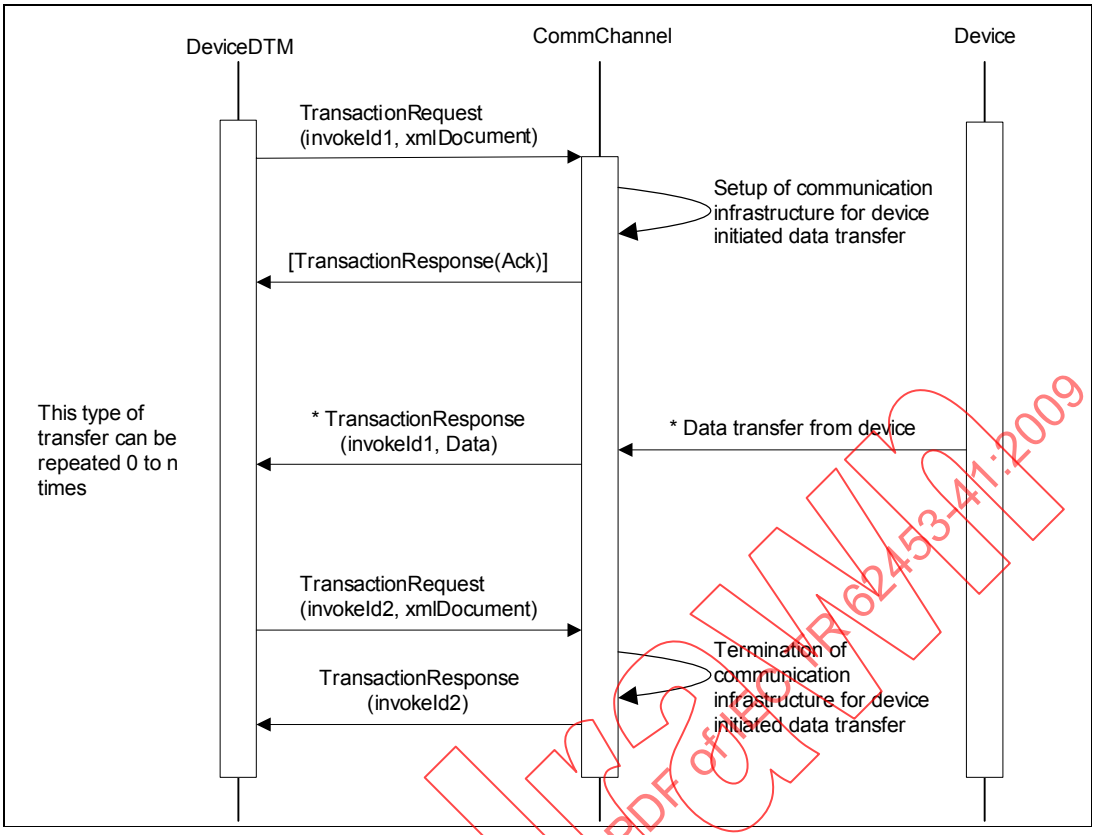


Figure 61 – Device initiated data transfer

Device initiated data transfer needs management by the communication infrastructure. That is why it is necessary to unambiguously identify the request to initialize and terminate this type of behavior.

The TransactionRequests to initiate or terminate device initiated data transfer will transport protocol specific XML-documents.

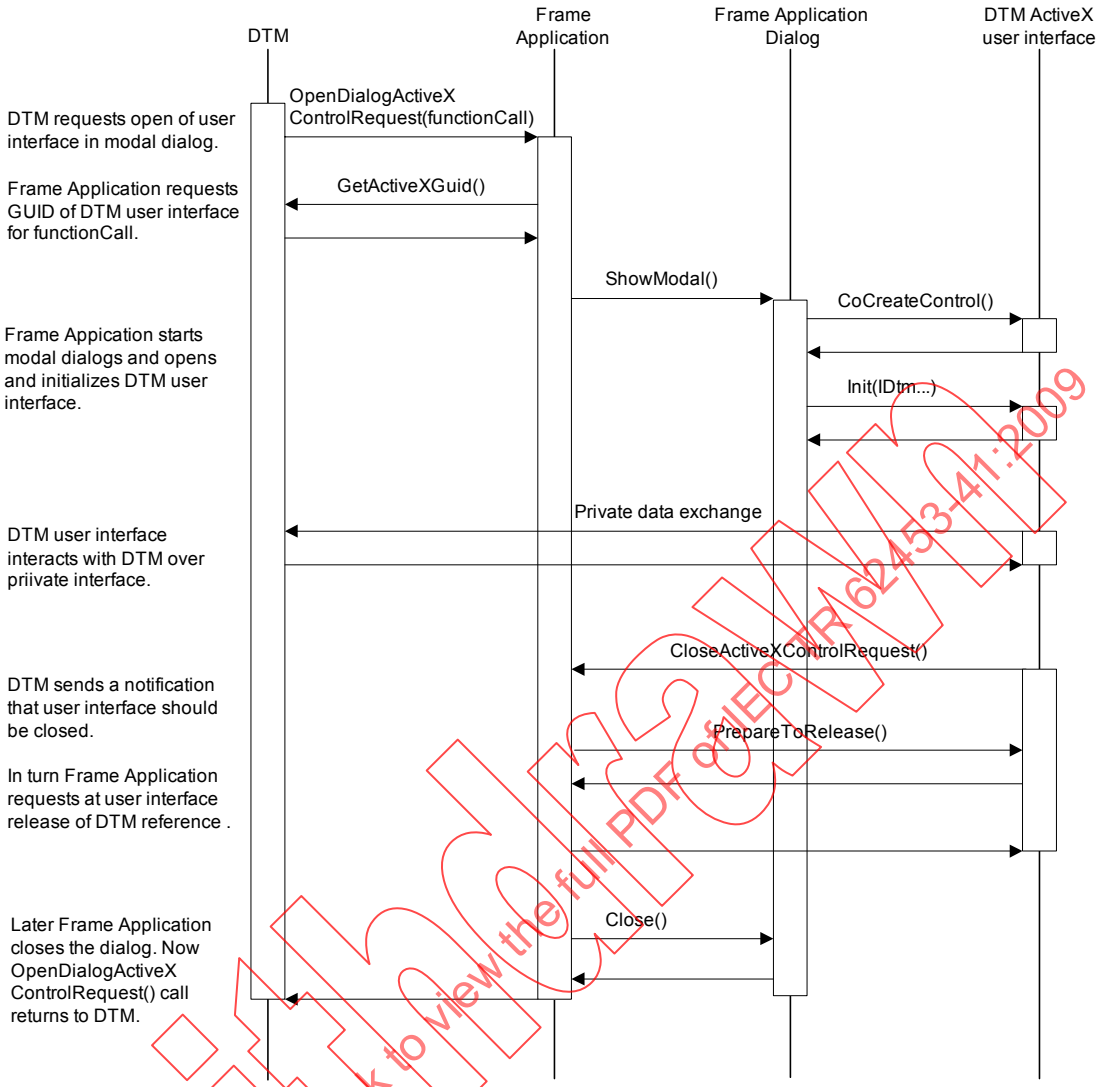
The terminating TransactionRequest will contain the information necessary to terminate the device initiated data transfer. This may include the invokeId of the initiating TransactionRequest.

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.

7.22 Starting and releasing DTM user interface in modal dialog

DTM requests start of user interface over IFdtActiveX2 interface implemented by Frame Application (see Figure 62). This starts modal dialog and opens DTM user interface in it. DTM user interface exchanges data with DTM over private interface and request closing of dialog after work is finished. Frame Application closes the dialog and DTM user interface. Now IFdtActiveX2:OpenDialogActiveXControlRequest() call returns to the DTM.



Used methods:

Standard Microsoft®

IDtmActiveXInformation::GetActiveXGuid()

IFdtActiveX2:OpenDialogActiveXControlRequest()

IFdtActiveX:CloseActiveXControlRequest()

IDtmActiveXControl:Init()

IDtmActiveXControl:PerpareToRelease()

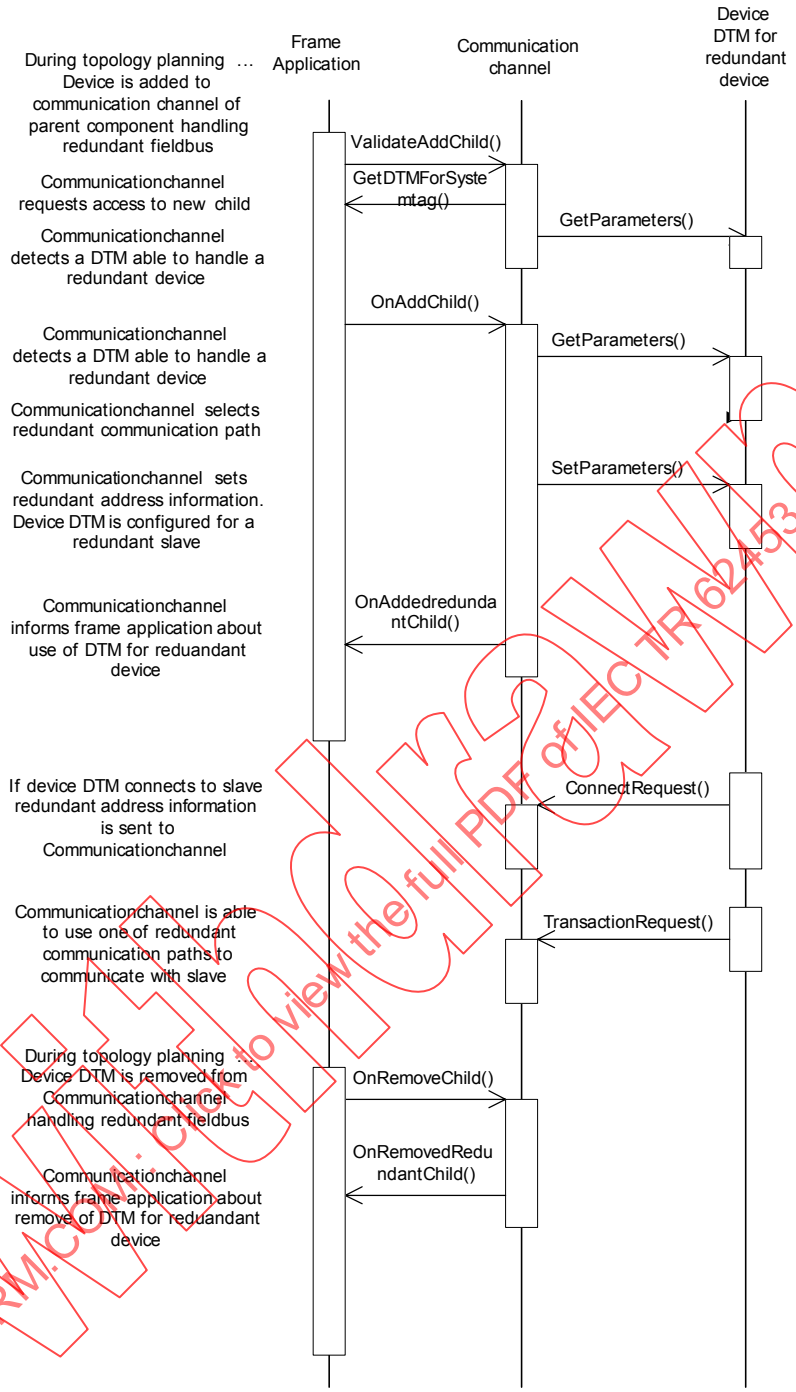
Figure 62 – Modal DTM user interface

7.23 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 63). 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 its 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 its child DTMs.



Used methods:

IDtmRedundancy::OnAddedRedundantChild()

IDtmRedundancyEvents::OnRemovedRedundantChild()

Figure 63 − Handling of a redundant slave

7.24 Initialization of a Channel ActiveX control

7.24.1 General

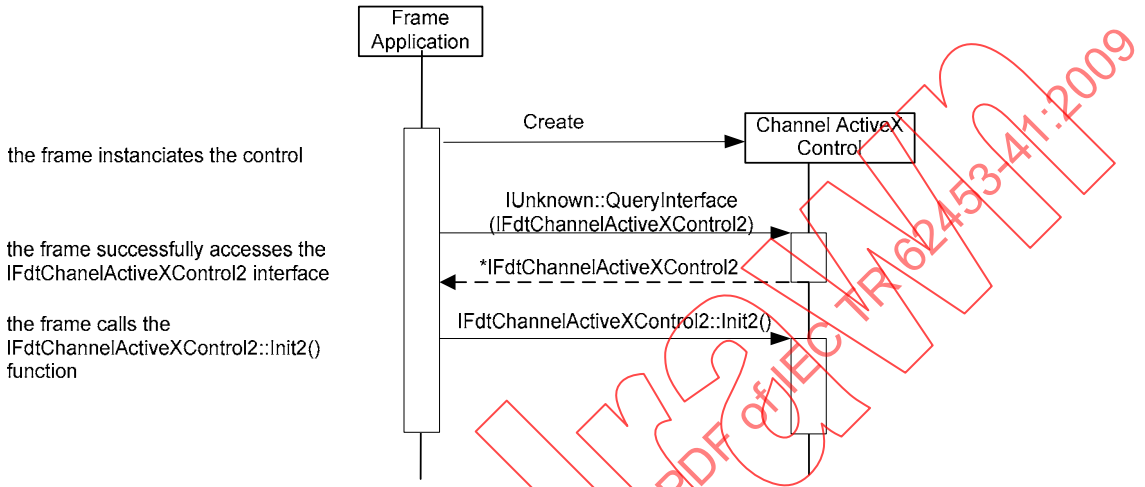
To initialize a Channel ActiveX® control conformant to this specification, the Frame Application has first to check, if the interface IFdtChannelActiveXControl2 exists.

If it exists the Frame Application has to call the IFdtChannelActiveXControl2::Init2() function at this interface.

If it does not exist, the Frame Application calls the IFdtChannelActiveXControl::Init() function.

7.24.2 Supports IFdtChannelActiveXcontrol2

Figure 64 describes the sequence of the Channel ActiveX® control initialization for the case that the Channel ActiveX® control provides the IFdtChannelActiveXControl2 interface.



Used methods:

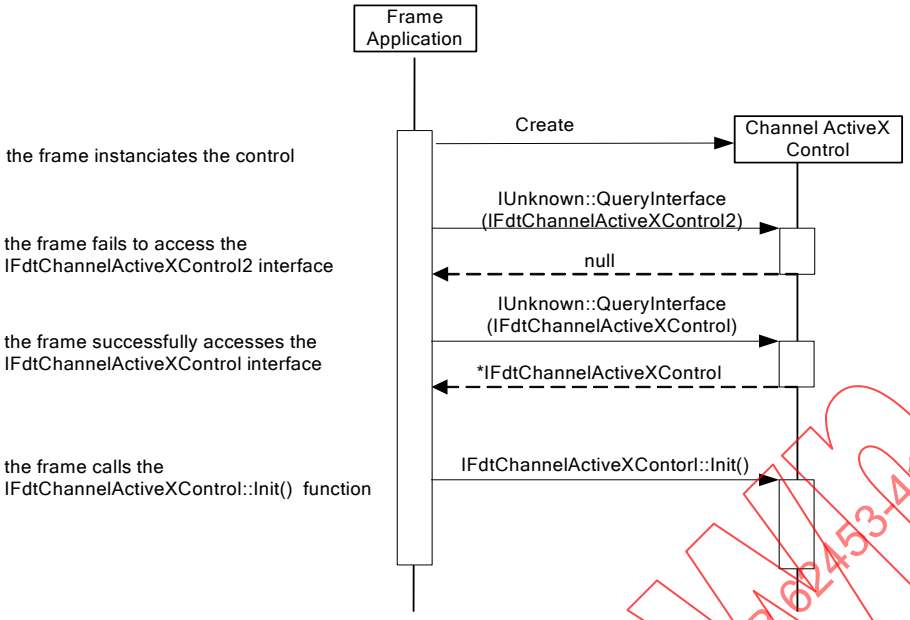
IUnkown::QueryInterface

IFdtChannelActiveXControl2::Init2()

Figure 64 – Init of Channel ActiveX with IFdtChannelActiveXControl2

7.24.3 Does not support IFdtChannelActiveXControl2

Figure 65 describes the sequence of the Channel ActiveX® control initialization for the case that the Channel ActiveX® control does not provide the IFdtChannelActiveXControl2 interface.



Used methods:

IUnkown::QueryInterface

IFdtChannelActiveXControl::Init()

Figure 65 – Init of Channel ActiveX® without IFdtChannelActiveXControl2

7.25 DTM upgrade

7.25.1 General

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 a 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 shall be done by the DTM when the data is loaded.

7.25.2 Saving data from a DTM to be upgraded

The first step is to store the information from the DTM that will be upgraded.

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 66):

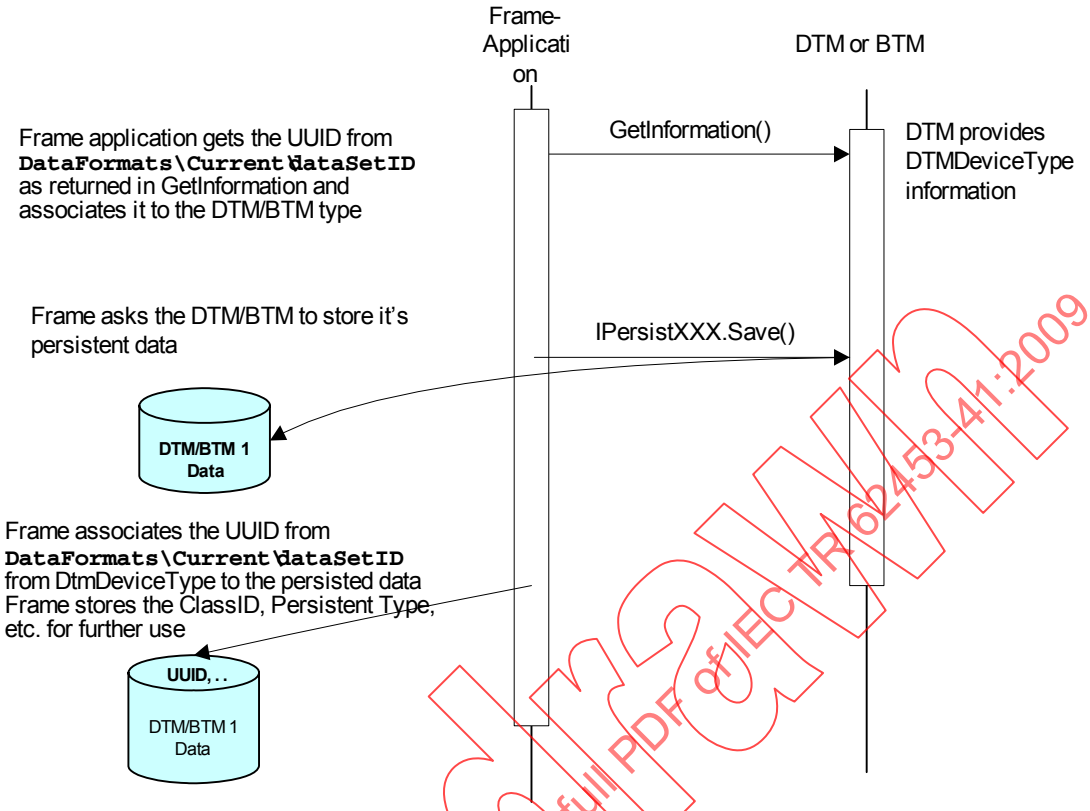


Figure 66 – Saving data from a DTM to be upgraded

7.25.3 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 67).

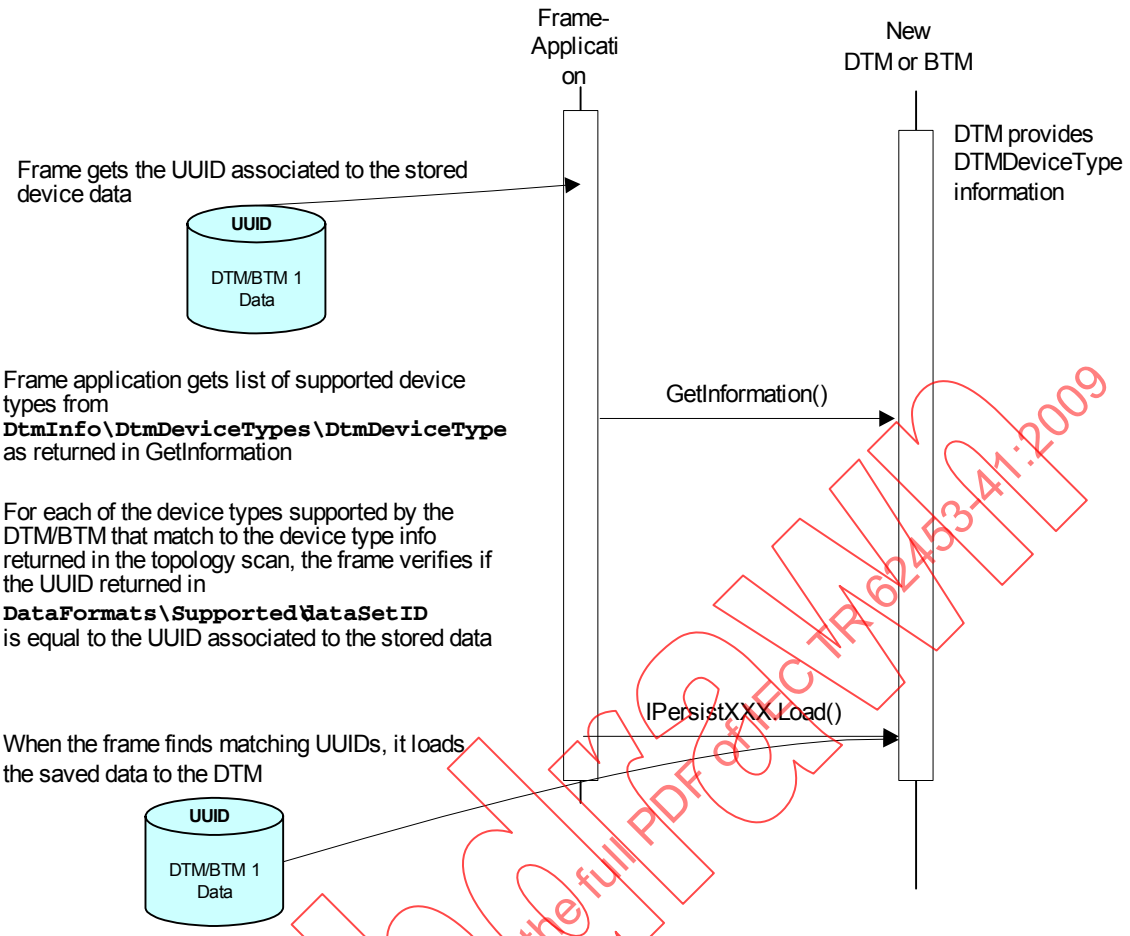
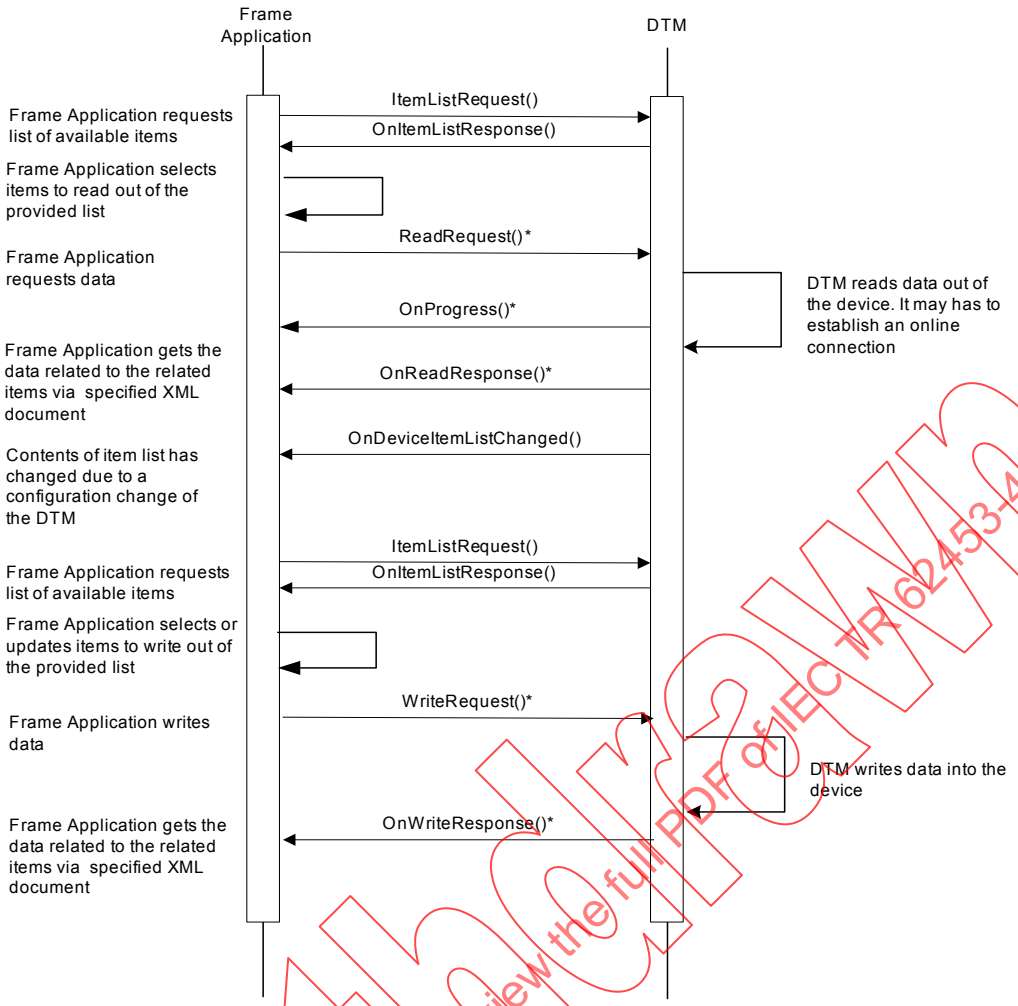


Figure 67 – Loading data in the replacement DTM

7.26 Usage of IDtmSingleDeviceDataAccess::ReadRequest / Write Request

This sequence chart (see Figure 68) gives an example regarding handling of ReadRequest() and WriteRequest().



Used methods:

IDtmSingleDeviceDataAccess::ItemListRequest()

IDtmSingleDeviceDataAccessEvents::OnItemListResponse()

IDtmSingleDeviceDataAccess::ReadRequest()

IDtmSingleDeviceDataAccess::WriteRequest()

IDtmSingleDeviceDataAccessEvents::OnReadResponse()

IDtmSingleDeviceDataAccessEvents::OnWriteResponse()

IDtmEvents::OnProgress()

Figure 68 – Usage of IDtmSingleDeviceDataAccess

7.27 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 shall 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,
- 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 69). 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”);

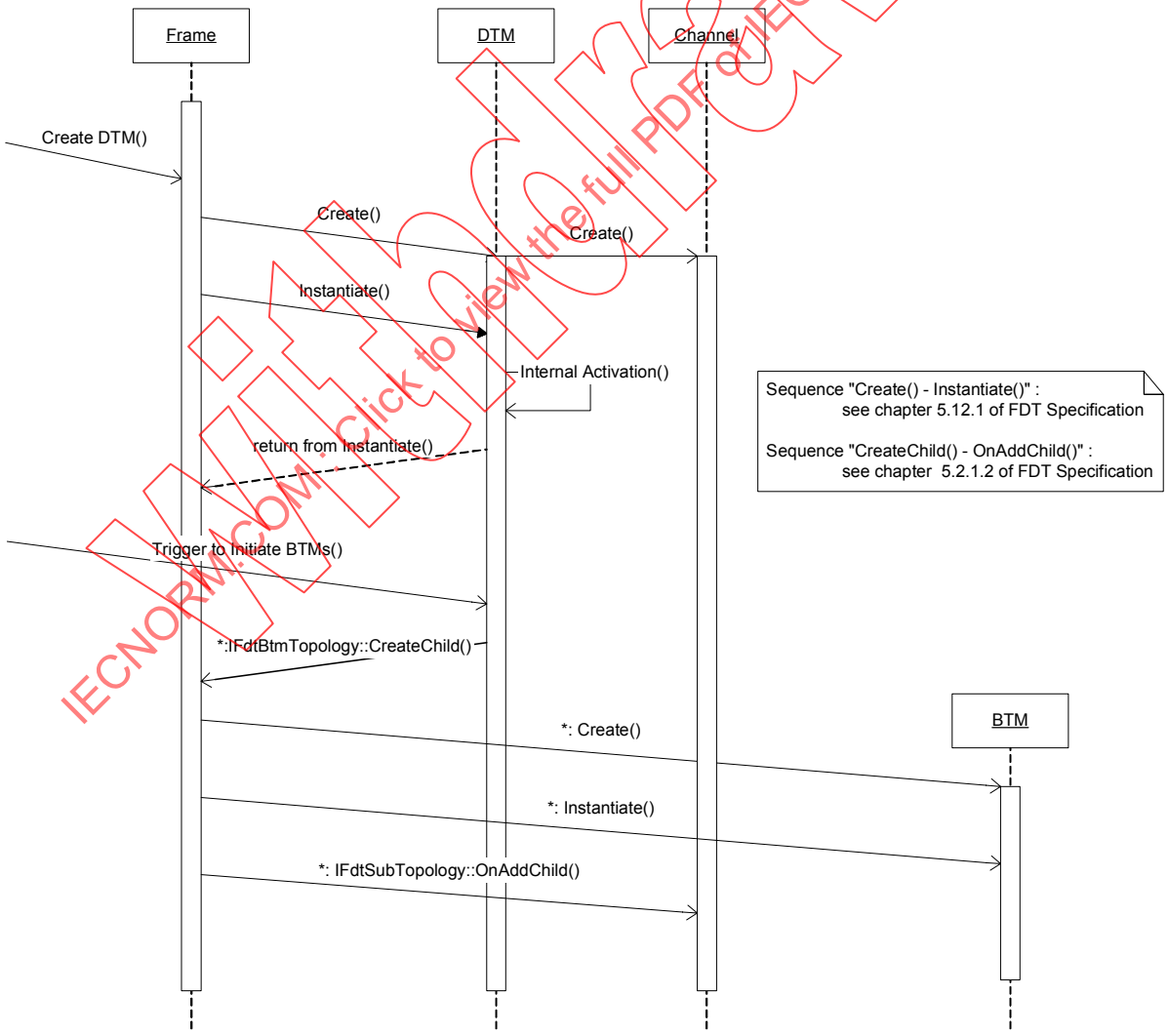


Figure 69 − General sequence of creation and instantiation of blocks

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 state 'Up' to state 'Existing Created' to state 'Running'), it shall not automatically trigger the BTM creation. The Frame Application should handle the instantiation of the BTMs.

8 Installation issues

8.1 Registry and device information

8.1.1 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 shall be registered in the Windows® registry using FDT-specific COM-component category entries. These class objects of a DTM can then be detected by a Frame Application or a configuration tool using the Microsoft® standard component category manager.

The BTM to DTM assignment follows the same model as the assignment of module Device DTM to DTM. Vendors are encouraged to define a unique CATID for the protocol between DTM and BTM in order to ensure correct block assignment. The same CATID can be used in different devices if the same BTMs are used.

8.1.2 Component categories

FDT defines the following component categories (see Table 15).

Table 15 – 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.

Table 16 shows the valid combination of category ids:

Table 16 – Combinations of categories

SYMBOLIC NAME OF THE CATID	CATID_FDT_DTM	CATID_FDT_DEVICE	CATID_FDT_MODULE	CATID_FDT_BTM
CATID_FDT_DTM		√	√	√
CATID_FDT_DEVICE	√			
CATID_FDT_MODULE	√			
CATID_FDT_BTM	√			

For example, class objects shall register for categories according to the following list (see Table 17):

Table 17 – 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. For more information refer to documentation of ICatRegister in MSDN®.

A DTM may register additional component categories according to COM rules.

8.1.3 Registry entries

Each object (device, module, channel, class, presentation) within a DTM shall provide all COM required registry entries within HKEY_CLASSES_ROOT and shall support self-registration.

8.1.4 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 shall 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 shall 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 Group shall be used. For example, 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) shall be installed via 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.

8.1.5 Microsoft's standard component categories manager

Using the Microsoft® standard component categories manager a Frame Application is able to query a list of all available DTMs or a list of DTMs with a set of specific categories by using the interface method ICatInformation::EnumClassesOfCategories.

8.1.6 Building a Frame Application-database of supported devices

Using component categories a Frame Application is able to detect all installed DTMs. Additional information, for example 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 5.8.1.

8.1.7 DTM registration

DTMs have to write registry entries whenever the DTM is

- installed,
- uninstalled,
- modified, (e.g. new device types are supported or the DTM was updated (bug fix)).

If the path doesn't exist, the first installed DTM has to create the path.

A FDT root path is defined for FDT in windows registry:

DTM registration – Path in registry
[HKEY_LOCAL_MACHINE\SOFTWARE\FDT\DTMCatalogUpdates][1]

[1] HKEY_LOCAL_MACHINE allows DTMs to write a registry key and string. During DTM setup user has to have administrator permissions anyway, so there is no missing permission for writing to the registry. During other use case sessions, a user can have normal permissions. Then a Frame Application only reads the registry keys, so the missing write permission is not a problem.

Key	Format	Description
ModificationComment	REG_SZ (String Value)	**Mandatory**: Any string, e.g.: Timestamp of modification. Empty string allowed[a].
ModificationFlag	REG_SZ (String – GUID)	**Mandatory**: GUID created by DTM setup during setup runtime.
[a] Instead of setting this attribute to optional, it's mandatory and allowed to set an empty string. This avoids having old comment together with new flag.		

Frame Applications may update a DTM Library and save the last known ModificationFlag entry internally.

A Frame Application 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 shall update ModificationFlag and ModificationComment during setup.

8.2 Paths and file information

8.2.1 Path information provided by a DTM

There are several possibilities where a DTM has to provide paths to files on the local file system:

- icons and bitmaps (<DeviceIcon>,<BlockIcon>, <DeviceBitmap>)
- documentation (<DocumentFile>, <DocumentExe>)
- protocol-specific schemas (<DtmSchemaPath>)
- GSD files (<deviceTypeInformationPath>)

Usually, these files will be copied on the local file system during DTM installation. The DTM has to provide the full path (including the drive).

8.2.2 Paths and persistency

Installation on different PCs may lead to different path information.

Since a DTM instance data set could be copied from one PC to another PC, a DTM shall not rely on path information, which is stored in its instance data set.

General rule: A DTM should never store path information in its instance data set.

This rule is also valid for paths that are provided by the Frame Application and used by the DTM (e.g. IFdtBulkData:GetInstanceRelatedPath, IFdtBulkData:GetProjectRelatedPath and IFdtContainer:GetXmlSchemaPath)

Example:

Installation path on PC-A "c:\programs\manufacturer\" leads to the Icon path:

```
<fdt:DeviceIcon path="file://c:/programs/manufacturer/icons/device.ico" />
```

Installation path on PC-B: "d:\programme\manufacturer\" leads to the Icon path:

```
<fdt:DeviceIcon path="file://d:/programme/manufacturer/icons/device.ico" />
```

8.2.3 Multi-user systems

Some Frame Applications provide multi-user capability, which means that the system is distributed over several PCs. In such a system, the Frame Application should not retrieve path information from one PC and expect that the file is also available on all other PCs in the same path.

General rule: Path information provided by FDT interfaces is only valid on the local PC.

9 Description of data types, parameters and structures

9.1 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.

Table 18 – 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 shall 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 shall 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.

9.2 Data type definitions

For basic datatypes following mapping is defined:

Table 19 – Basic data types

Data type	Mapping to COM data types
ClassIdentifier	CLSID / ProgID
LanguageID	LCID
ObjectReference	IUnknown pointer
UUID	UUID
URI	URI

The following helper objects for documentation are defined (see Table 20):

Table 20 – Helper objects for documentation

Name	Data type	Description
FdtUUIDString	BSTR	String containing a unique identifier according to the Microsoft® standard UUID. The format shall be e.g. “C2137DD1-7842-11d4-A3C9-005004DC410F” (without bracket). Due to the definition of the UUID format, the value shall 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

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 shall 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(036D147F-387B-11D4-86E1-00E0987270B9),
     dual,
  oleautomation
 ]
 interface IDtmInformation: IDispatch {
   [id(0x1)]
   HRESULT GetInformation(
                [out, retval] FdtXmlDocument* result);
 };

// IDtmOnlineDiagnosis
 [
  odl,
  uuid(036D1477-387B-11D4-86E1-00E0987270B9),
  dual,
  oleautomation
 ]
 interface IDtmOnlineDiagnosis: IDispatch {
   [id(0x1)]
   HRESULT Compare(
                [out, retval] FdtXmlDocument* result);
```

```
    [id(0x2)]
    HRESULT GetDeviceStatus(
                    [out, retval] FdtXmlDocument* result);
  };

 // IDtmOnlineParameter
  [
   odl,
   uuid(036D1483-387B-11D4-86E1-00E0987270B9),
   dual,
   oleautomation
  ]
  interface IDtmOnlineParameter: IDispatch {
     [id(0x1)]
     HRESULT CancelAction(
                        [in] FdtUUIDString invokeId,
                        [out,retval] VARIANT_BOOL* result);
    [id(0x2)]
    HRESULT DownloadRequest(
          [in] FdtUUIDString invokeId,
          [in] FdtXPath parameterPath,
          [out,retval] VARIANT_BOOL* result);
    [id(0x3)]
    HRESULT UploadRequest(
          [in] FdtUUIDString invokeId,
          [in] FdtXPath parameterPath,
          [out,retval] VARIANT_BOOL* result);
  };

  // IDtmParameter
  [
   odl,
   uuid(036D147D-387B-11D4-86E1-00E0987270B9),
   dual,
   oleautomation
  ]
  interface IDtmParameter: IDispatch {
    [id(0x1)]
    HRESULT GetParameters(
          [in] FdtXPath parameterPath,
          [out, retval] FdtXmlDocument* result);
    [id(0x2)]
    HRESULT SetParameters(
          [in] FdtXPath parameterPath,
          [in] FdtXmlDocument FdtXmlDocument,
          [out, retval] VARIANT_BOOL* result);
  };

      //
      // DTM event interfaces
      // ====================

 // IFdtCommunicationEvents
  [
      odl,
   uuid(036D1485-387B-11D4-86E1-00E0987270B9),
   dual,
   oleautomation
  ]
  interface IFdtCommunicationEvents: IDispatch {
     [id(0x1)]
    HRESULT OnAbort(
                    [in] FdtUUIDString communicationReference );
    [id(0x2)]
    HRESULT OnConnectResponse(
                    [in] FdtUUIDString invokeId,
                    [in] FdtXmlDocument response);
    [id(0x3)]
    HRESULT OnDisconnectResponse(
                    [in] FdtUUIDString invokeId,
                    [in] FdtXmlDocument response);
    [id(0x4)]
    HRESULT OnTransactionResponse(
                    [in] FdtUUIDString invokeId,
                    [in] FdtXmlDocument response);

  };
```

```
// IFdtEvents
  [
  odl,
  uuid(036D1478-387B-11D4-86E1-00E0987270B9),
  dual,
  oleautomation
 ]
 interface IFdtEvents: IDispatch {
    [id(0x1)]
    HRESULT OnChildParameterChanged(
                    [in] BSTR systemTag);
   [id(0x2)]
   HRESULT OnParameterChanged(
          [in] BSTR systemTag,
          [in] FdtXmlDocument parameter);
   [id(0x3)]
   HRESULT OnLockDataSet(
          [in] BSTR systemTag,
          [in] BSTR userName);
   [id(0x4)]
   HRESULT OnUnlockDataSet(
          [in] BSTR systemTag,
          [in] BSTR userName,
          [out, retval] VARIANT_BOOL* result);
 };


    //
    // DTM ActiveX Control interfaces
    // ==============================
    //

// IDtmActiveXControl
 [
  odl,
  uuid(036D1486-387B-11D4-86E1-00E0987270B9),
  dual,
  oleautomation
 ]
 interface IDtmActiveXControl: IDispatch {
   [id(0x1)]
   HRESULT Init(
          [in] FdtUUIDString invokeId,
          [in] FdtXmlDocument functionCall,
          [in] IDtm* dtm,
          [out, retval] VARIANT_BOOL* result);
   [id(0x2)]
   HRESULT PrepareToRelease(
                  [out, retval] VARIANT_BOOL* result);
 };

    //
    // FDT Channel interfaces
    // ======================
    //

// IFdtChannel
 [
  odl,
  uuid(036D1488-387B-11D4-86E1-00E0987270B9),
  dual,
  oleautomation
 ]
 interface IFdtChannel: IDispatch {
    [id(0x1)]
   HRESULT GetChannelPath(
                  [out, retval] FdtXPath* result);
   [id(0x2)]
   HRESULT GetChannelParameters(
          [in] FdtXPath parameterPath,
          [in] FdtUUIDString protocolId,
          [out, retval] FdtXmlDocument* result);
   [id(0x3)]
   HRESULT SetChannelParameters(
          [in] FdtXPath parameterPath,
          [in] FdtUUIDString protocolId,
          [in] FdtXmlDocument XmlDocument,
          [out, retval] VARIANT_BOOL* result);
```

```
};

// IFdtChannelActiveXInformation
[
 odl,
 uuid(F2BD2970-13FA-470c-A28C-6A5969A04037),
 dual,
 oleautomation
]
interface IFdtChannelActiveXInformation: IDispatch {
  [id(0x1)]
  HRESULT GetChannelActiveXGuid(
        [in] FdtXmlDocument functionCall,
        [out, retval] FdtUUIDString* result);
  [id(0x2)]
  HRESULT GetChannelActiveXProgId(
        [in] FdtXmlDocument functionCall,
        [out, retval] BSTR* result);
  [id(0x3)]
  HRESULT GetChannelFunctions(
                [in] FdtXmlDocument operationState,
                [out, retval] FdtXmlDocument* result);

};

// IFdtCommunication
[
 odl,
 uuid(039ECFC4-9CA8-44e6-944D-B37F288A34D8),
 dual,
 oleautomation
]
interface IFdtCommunication: IDispatch {
  [id(0x1)]
  HRESULT Abort(
            [in] FdtXmlDocument fieldbusFrame);

  [id(0x2)]
  HRESULT ConnectRequest(
        [in] IFdtCommunicationEvents* callBack,
        [in] FdtUUIDString invokeId,
        [in] FdtUUIDString protocolId,
        [in] FdtXmlDocument fieldbusFrame,
        [out, retval] VARIANT_BOOL* result);
  [id(0x3)]
  HRESULT DisconnectRequest(
        [in] FdtUUIDString invokeId,
        [in] FdtXmlDocument fieldbusFrame,
        [out, retval] VARIANT_BOOL* result);
  [id(0x4)]
  HRESULT TransactionRequest(
        [in] FdtUUIDString invokeId,
        [in] FdtXmlDocument fieldbusFrame,
        [out, retval] VARIANT_BOOL* result);
  [id(0x5)]
  HRESULT GetSupportedProtocols(
            [out, retval] FdtXmlDocument *result );
  [id(0x6)]
  HRESULT SequenceBegin(
        [in] FdtXmlDocument fieldbusFrame,
        [out, retval] VARIANT_BOOL* result);
  [id(0x7)]
  HRESULT SequenceStart(
            [in] FdtXmlDocument fieldbusFrame,
               [out, retval] VARIANT_BOOL* result);
  [id(0x8)]
  HRESULT SequenceEnd(
            [in] FdtXmlDocument fieldbusFrame,
               [out, retval] VARIANT_BOOL* result);

};

// IFdtChannelSubTopology
[
 odl,
 uuid(036D1484-387B-11D4-86E1-00E0987270B9),
 dual,
```

```
  oleautomation
 ]
 interface IFdtChannelSubTopology: IDispatch {
  [id(0x1)]
  HRESULT ScanRequest(
        [in] FdtUUIDString invokeId,
        [out, retval] VARIANT_BOOL* result);
  [id(0x2)]
  HRESULT ValidateAddChild(
        [in] BSTR 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)

Mapping of services to interface methods

B.1 General

This annex describes the mapping of IEC 62453-2 services to interface, methods and XML information.

IEC 62453-2 does not define whether it is optional or mandatory to implement the defined services. This definition is provided by Table 2, Table 4, Table 6 and Table 8.

IEC 62453-2 does not define whether the services are implemented as synchronous or asynchronous calls. This implementation specification (Part 41) uses the asynchronous model, if it is likely that execution of service takes a longer time. This part defines separated request, callback and canceling interfaces / methods for this kind of services. Synchronous services are implemented by a single interface / method.

B.2 DTM services

Table B.1 – General services

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
PrivateDialogEnabled	*Request/Response:*	IDtm.PrivateDialogEnabled
SetLanguage	*Request/Response:*	IDtm.SetLanguage
SetSystemGuiLabel	*Request/Response:*	IDtm.SetSystemGuiLabel

Table B.2 – DTM service related to installation

IEC 62453-2	IEC/TR 62453-41
Not defined.	Registration of DTM at the system (today's ProgID and maybe CategoryID)
Not defined.	Modification flag

Table B.3 – DTM service related to DTM Information

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
GetTypeInformation	*Request/Response:*	IDtmInformation.GetInformation IBtmInformation.GetInformation
GetIdentificationInformation	*Request/Response:*	IDtmInformation2.GetDeviceIdentificiationInformation
HardwareInformation	*Request:* *Response*: *Cancel*:	IDtmHardwareIdentification.ScanHardwareRequest IDtmScanEvents.ScanHardwareResponse IDtmHardwaeIdentification:CancelAction
GetActiveTypeInfo	*Request/Response:*	IDtmParameter.GetParameters: - <*DtmDeviceType*> section : <*InternalTopology*> section IBtmParameter.GetParameters: - <*BtmBlockType*> section

Table B.4 – DTM services related to DTM state machine

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
Initialize	*Request/Response:*	IPersist*xxx*.InitNew
		IPersist*xxx*.Load
		IDtm.InitNew
		IBtm.InitNew
		IDtm.Environment
		IBtm.Environment
		IDtm2.Environment2
		IDtm.Config
		IBtm.Config
SetLinked CommunicationChannel EnableCommunication	*Request/Response:*	The services SetCommunicationChannel and EnableCommunication = TRUE are combined to IDtm.SetCommunication. / IBtm.SetCommunication (SetCommunication sets the channel and enables the communication) EnableCommunication = FALSE maps to IDtm.PrepareToReleaseCommunication / IBtm.PrepareToReleaseCommunication and OnPreparedToReleaseCommunication
ReleaseLinked CommunicationChannel	*Request/Response:*	IDtm.ReleaseCommunication
		IBtm.ReleaseCommunication
ClearInstanceData	*Request/Response:*	IDtm.PrepareToDelete
		IBtm.PrepareToDelete
Terminate	*Request/Response:*	IDtm.PrepareToRelease
		IDtmEvents.OnPreparedToRelease
		IBtm.PrepareToRelease

Table B.5 – DTM services related to function

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
GetFunctions	*Request/Response:*	IDtm.GetFunctions
		IBtm.GetFunctions
	Event	IDtmEvents.OnFunctionChanged
InvokeFunction	*Request:*	IDtm.InvokeFunction
		IBtm.InvokeFunction
	Response:	IDtmEvents.OnInvokedFunctionFinished
GetGuiInformation	*Request/Response:*	IDtmActiveXInformation.GetActiveXGuid
		IDtmActiveXInformation.GetActiveXProgId
OpenPresentation	*Request/Response:*	IDtmApplication.StartApplication
ClosePresentation	*Request:*	IDtmApplication.ExitApplication
	Response:	IDtmEvents.OnApplicationClosed

Table B.6 – DTM services related to documentation

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
GetDocumentation	*Request/Response:*	IDtmDocumentation.GetDocumentation

Table B.7 – DTM services to access the instance data

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
InstanceDataInformation	*Request/Response:*	IDtmSingleInstanceDataAccess.GetItemList
		IDtmParameter.GetParameters / IBtmParameter. GetParameters <*ExportedVarialbes*> section
	Event	IDtmSingleInstanceDataAccessEvents.OnInstanceItemListChanged
		IDtmEvents.OnParmenterChanged
InstanceDataRead	*Request/Response:*	IDtmSingleInstanceDataAccess.Read that enables to read single and multiple parameters
		IDtmParameter.GetParameters / IBtmParameter. GetParameters <*ExportedVarialbes*> section within the returned XML that enables to read all parameters at once.
InstanceDataWrite	*Request/Response:*	IDtmSingleInstanceDataAccess.Read that enables to write single and multiple parameters
		IDtmParameter.GetParameters / IBtmParameter. SetParameters <*ExportedVarialbes*> section within the returned XML that enables to write all parameters at once.

Table B.8 – DTM services to access diagnosis

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
Verify	*Request/Response:*	IDtmDiagnosis.Verify
CompareDataVaueSet	*Request/Response:*	IDtmDiagnosis.InitCompare
		IDtmDiagnosis.Compare
		IDtmDiagnosis.ReleaseCompare

Table B.9 – DTM services to access to device data

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
DeviceDataInformation	*Request:*	IDtmSingleDeviceDataAccess.ItemListRequest
	Response	IDtmSingleDeviceDataAccessEvebts.OnDeviceItemListResponse
	Cancel:	IDtmSingleDeviceDataAccess.CancelRequest
	Event	IDtmSingleDeviceDataAccessEvents.OnDeviceItemListChanged
DeviceDataRead	*Request:*	IDtmSingleDeviceDataAccess.ReadRequest
	Response:	IDtmSingleDeviceDataAccessEvebts.OnReadResponse
	Cancel:	IDtmSingleDeviceDataAccess.CancelRequest
DeviceDataWrite	*Request:*	IDtmSingleDeviceDataAccess.WirteRequest
	Response:	IDtmSingleDeviceDataAccessEvebts.OnWirteResponse
	Cancel:	IDtmSingleDeviceDataAccess.CancelRequest

Table B.10 – DTM services related to network management information

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
NetworkManagementInfo Read	*Request/Response:*	IDtmParameter.GetParameters – section <*BusInformation*>
NetworkManagementInfo Write	*Request/Response:*	IDtmParameter.SetParameters – section <*BusInformation*>

Table B.11 – DTM services related to online operation

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
DeviceStatus	*Request/Response:*	IDtmOnlineDiagnosis.GetDeviceStatus
CompareValueDataSetWith DeviceDataSet	*Request/Response:*	IDtmOnlineDiagnosis.Compare
WriteDataToDevice	*Request:*	IDtmOnlineParameter.DowloadRequest
	Response:	IDtmEvents.OnDownLoadFinished
	Cancel:	IDtmOnlineParameter.CancelAction
ReadDataFromDevice	*Request:*	IDtmOnlineParameter.UploadRequest
	Response:	IDtmEvents.OnUploadLoadFinished
	Cancel:	IDtmOnlineParameter.CancelAction

Table B.12 – DTM services related to FDT-Channel objects

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
GetChannels	*Request/Response:*	IDtmChannel.GetChannels GetParameters <ChannelReferences> sections

Table B.13 – DTM services related to import and export

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
Export	*Request/Response:*	IDtmImportExport.Export for DTM using private data storage IPersistStreamXXX.Save for DTM not using private data storage
Import	*Request/Response:*	IDtmImportExport.Import for DTM using private data storage IPersistStreamXXX.Load for DTM not using private data storage

Table B.14 – DTM services related to data synchronization

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
OnLockInstanceData	*Request/Response:*	IFdtEvents.OnLockDataSet
OnUnlockInstanceData	*Request/Response:*	IFdtEvents.OnUnLockDataSet
OnInstanceDataChanged	*Request/Response:*	IFdtEvents.OnParameterChange
OnChildInstanceData Changed	*Request/Response:*	IFdtEvents.OnChildParameterChanged

B.3 Presentation object services

The interactions and state control between Frame Application, Presentation object and DTM is technology dependent. The IEC 62453 series leaves it to the technology specific parts

(IEC/TR 62453-4z) to define necessary services. This document defines the following interfaces / methods for state control of ActiveX®:

- IDtmActiveXControl.Init / PrepareToRelease
- IFdtChannelActiveX.Init / PrepareToRelease
- IBtmActiveXControl.Init / PrepareToRelease

B.4 General channel services

Table B.15 – General channel service

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
ReadChannelInformation	*Request/Response:*	IFdtChannel.GetChannelPath IFdtChannel.GetChannelParameters – minimum set of information included in all protocol specific channel parameter schemas (information defined by FDTBasicChannelSchema).
WriteChannelInformation	*Request/Response:*	IFdtChannel.SetChannelParameters – minimum set of information included in all protocol specific channel parameter schemas (information defined by FDTBasicChannelSchema)

B.5 Process channel services

Table B.16 – Channel services for IO related information

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
ReadChannelData	*Request/Response:*	IFdtChannel.GetChannelParameters – protocol dependent information defined in corresponding FDT Annex.
WriteChannelData	*Request/Response:*	IFdtChannel.SetChannelParameters - protocol dependent information defined in corresponding FDT Annex.

B.6 Communication Channel Services

Table B.17 – Channel services related to communication

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
GetSupportedProtocols	*Request/Response:*	IFdtCommunication:GetSupportedProtocols
Connect	*Request:* *Response*:	IFdtCommunication:ConnectRequest IFdtCommunicationEvens.OnConnectRespones IFdtCommunicationEvents2:OnConnectResponse2
Disconnect	*Request:* *Response*:	IFdtCommunication.DisconnectRequest IFdtCommunicationEvens.OnDisconnectRespones
AbortRequest	*Request/Response:*	IFdtCommunication.Abort
AbortIndication	*Event:*	IFdtCommunicationEvent.OnAbort
Transaction	*Request:* *Response*:	IFdtCommunication.TransactionRequest IFdtCommunicationEvens:OnTransActionRespones
SequenceDefine	*Request/Response:*	IFdtCommunication.SequenceBegin IFdtCommunication.SequenceEnd
SequenceStart	*Request/Response:*	IFdtCommunication.SequenceStart
Not defined.	*Request/Response:*	IFdtFunctionBlockData.GetFBInstanceData IFdtFunctionBlockData.SelectFBInstance

Table B.18 – Channel services related sub-topology management

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
ValidateAddChild	*Request/Response:*	IFdtChannelSubTopology.ValidateAddChild
ChildAdded	*Request/Response:*	IFdtChannelSubTopology.OnAddChild
ValidateRemoveChild	*Request/Response:*	IFdtChannelSubTopology.ValidateRemoveChild
ChildRemoved	*Request/Response:*	IFdtChannelSubTopology.OnRemoveChild
SetChildrenAddresses	*Request/Response:*	IFdtChannelSubTopology2.SetChildrenAddresses

Table B.19 – Channel services related to functions

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
GetFunctions	*Request/Response:*	IFdtChannelFunctions.GetFunctions
	Event	IDtmEvents.OnChannelFunctionChanged
GetGuiInformation	*Request/Response:*	IFdtChannelActiveXInformation.GetActiveXGuid
		IFdtChannelActiveXInformation.GetActiveXProgId

Table B.20 – Channel services related to scan

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
Scan	*Request:*	IFdtChannelScan.ScanRequest
		IFdtChannelSubTopology.ScanRequest
	Response:	IDtmScanEvents.OnScanResponse
		IDtmEvents::OnScanResponse()
	Cancel:	IFdtChannelScan.CancelAction.

B.7 Frame Application Services

Table B.21 – FA services related to general event

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
OnErrorMessage	*Request/Response:*	IDtmEvents:OnErrorMessage
OnProgress	*Request/Response:*	IDtmEvents:OnProgress
OnOnlineStatusChanged	*Request/Response:*	IDtmEvents2:OnStateChanged
OnFunctionChanged	*Request/Response:*	IDtmEvents:OnFunctionChanged
		IDtmEvents.OnChannelFunctionChanged
Not defined.	*Request/Response:*	IDtmEvents:OnNavigation
Not defined.	*Request/Response:*	IDtmEvents:OnPrint

Table B.22 – FA services related to topology management

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
GetDtmInfoList	*Request/Response:*	IFdtTopology:GetDtmInfoList IFdtBtmTopology.GetBtmInfoList
CreateChild	*Request/Response:*	IFdtTopology.CreateChild IFdtBtmTopology.CreateChild
DeleteChild	*Request/Response:*	IFdtTopology.DeleteChild IFdtBtmTopology.DeleteChild
MoveChild	*Request/Response:*	IFdtTopology.MoveChild IFdtBtmTopology.MoveChild
GetChildNodes	*Request/Response:*	IFdtTopology:GetChildNodes IFdtBtmTopology.GetChildNodes
GetParentNodes	*Request/Response:*	IFdtTopology:GetParentNodes IFdtBtmTopology.GetParentNodes
GetDtm	*Request/Response:*	IFdtTopology:GetDtmForSystemTag IFdtBtmTopology.GetBtmForSystemTag
ReleaseDtm	*Request/Response:*	IFdtTopology:ReleaseDtmForSystemTag IFdtBtmTopology.ReleaseBtmForSystemTag

Table B.23 – FA services related to redundancy

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
OnAddedRedundantChild	*Request/Response:*	IDtmRedundancyEvents:OnAddedRedundantChild
OnRemovedRedundantChild	*Request/Response:*	IDtmRedundancyEvents:OnRemovedRedundantChild

Table B.24 – FA services related to storage of DTM data

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
LoadInstanceData	*Request/Response:*	IStream.Read / IPropertyBag.Read (DTM gets access to these Frame Application interfaces/methods via IPersistStreamInit.Load or IPersistPropertyBag.Load which are called at DTM start up).
SaveInstanceData	*Request/Response:*	IStream.Write / IPropertyBag.Write (DTM gets access to these Frame Application interfaces/methods on demand by calling IFdtContainer.SaveRequest. The reference is then hand over via IPersistStreamInit.Save or IPersistPropertyBag.Save).
GetPrivateDtmStorageInfo	*Request/Response:*	IFdtBulkData.GetProjectRelatedPath IFdtBulkData.GetInstanceRelatedPath

Table B.25 – FA services related to DTM data synchronization

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
LockInstanceData	*Request/Response:*	IFdtContainer.LockDataSet
UnlockInstanceData	*Request/Response:*	IFdtContainer.UnLockDataSet
InstanceDataChanged	*Request/Response:*	IDtmEvents.OnParameterChanged

Table B.26 − FA related to Presentation

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
OpenPresentationRequest	*Request/Response:*	IFdtActiveX.OpenAcitveXControlRequest IFdtActiveX2.OpenChannelAcitveXControlRequest IFdtActiveX2.OpenDialogAcitveXControlRequest IFdtActiveX2.OpenDialogChannelAcitveXControlRequest The Init methods in IDtmActiveXControl, IBtmActiveXControl, and IFdtChannelActiveXControl are used to pass the InvokeID that enables to close the Active by calling IFdtActiveX.CloseAcitveXControlRequest or IFdtActiveX2.CloseChannelAcitveXControlRequest
ClosePresentationRequest	*Request/Response:*	IFdtActiveX.CloseAcitveXControlRequest IFdtActiveX2.CloseChannelAcitveXControlRequest
UserDialog	*Request/Response:*	IFdtDialog.UserDialog

Table B.27 − FA services related to audit trail

IEC 62453-2 service	IEC/TR 62453-41 interface / method	
RecordAuditTrailEvent	*Request/Response:*	IFdtAuditTrai.OnAuditTrailEvent IFdtAuditTrai.OnStartTransaction IFdtAuditTrai.OnEndTransAction

Annex C
(normative)

FDT XML schemas

C.1 General

The schemas defined here are technology specific implementations of the data types as defined in IEC 62453-2. The relation between data types and schema elements is shown by using the same names.

In order to adapt to this specific implementation technology, small modifications have been applied during mapping. These modifications (e.g. differences in naming) have been documented in this Annex.

NOTE Implementers should verify the correct implementation of data types according to this document.

C.2 FDTDataTypesSchema

The data type schema is used as a global FDT definition. Data types of this schema are referenced 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.

Table C.1 – Description 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 can 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 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 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 IEC 61784 CPF 3 or IEC 61784 CPF 9 according to the FDT specific unique identifier This attribute is an implementation of data type ‘fdt:protocolId’.
busCategoryName	Human readable string for the bus type. Examples are: • DPV0 -> 'Profibus DP/V0' • DPV1 -> 'Profibus DP/V1' • HART -> ‘HART’ The categories are defined in the protocol specific parts of IEC 62453. This attribute is an implementation of data type ‘fdt:protocolIdName’.
busRedundancy	Number of redundant fieldbus interfaces

Attribute	Description
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 it is recommended to select the value of attribute 'classificationId' 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, it is recommended to select the ID from 'FDT classification ID table' and use it without 'classificationDomainId' attribute.See definition of data type classificationId in IEC 62453-2.
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, shall be handled within the XML document returned to the child component It is recommended that for all errors returned by a device as result of a fieldbus transaction to 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 shall 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 shall be able to handle both types of error responses for a transaction request sent to the parent component.
communicationType	Indicates the type of the protocol support: • 'supported': Bus protocol which can be provided by the DTM at a channel. • 'required': Bus protocol which will be expected by the DTM from the parent DTM NOTE The actual supported bus protocols depend on the configuration of the DTM. This data type shows the possible supported, not the actual available protocols.
dataSetID	Unique identifier of the data set version
dataSetState	State of an instance data set concerning modifications (refer to 7.14.1.1)
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 an 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

Attribute	Description
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) It is recommmended that the Frame Application displays 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 IEC 61784 CPF 3 device has to provide its GSD information as human readable string at this attribute. The GSD information shall 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 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 its value remains unchanged although some identification information in DtmDeviceType element is updated. This can be a result of DTM update. The same dtmDevTypeID value shall 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 IEC 61784 CPF 3 specification
file	Contains a document or executable file with a absolute path.
help	Human readable help string for a document

Attribute	Description
Id	Unique identifier for an element. This identifier is used within collections for the direct access of elements. This id shall be unique within the namespace of a device instance. The corresponding 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 can 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 can 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 instance data set (E.g.: see use case Online parameterization, IDtmSingleDeviceDataAccess interface definition) 'modifiedInDevice' flag shall 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, 'modifiedInDevice' flag shall be reset (set to FALSE). Data in the device can also be modified directly by a tool out of the scope of the FDT. In this case, it is recommended not to set the flag 'modifiedInDevice'. Data set shall 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 can not be changed if the data set is not locked)
name	Human readable name within the context of an 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

Attribute	Description
semanticId	Semantic identifier for an element. This identifier shall be unique at least within the context of an element. By using this attribute, for example 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 shall be the same parameter as the corresponding XML attribute (slaveAddress, busAddress or busMasterConfigurationPart) within in the DTMParameterSchema represents.
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 7.14.2 and 7.14.1.2)
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 shall be passed to the DTM during initialization via IDtm::Init()to advise a pre configuration for the requested sub type. For example, the same transmitter can be pre configured for Level or flow measuring.
substituteType	Type of an substitute value
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 C.2 – Description of general XML elements

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. It is expected that the BusCategories element contains 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 defintion of data type ClassificationId in IEC 62453-2
ClassificationIds	Collection of ClassificationId elements
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
CommunicationTypeEntry	Enumeration element for the communication type.
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 manufactuer 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. It is expected that the DeviceVariants element only is used within context of DTMInformationSchema documents.
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, for example IEC 61784 CPF 3 transferred value. The value is mapped to the related range (LowerRangeValue). For example, if a IEC 61784 CPF 3 device maps a range from 10 mbar to 100 mbar to an integer of value 1 024 to 4 096 the 'LowerRawValue' contains 1 024, the 'UpperRawValue' contains 4 096.
NumberData	Number variable like float, integer or other numeric data types
PhysicalLayer	Unique identifier for a physical layer of a fieldbus like IEC 61784 CPF 3 (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 shall be provided. The DTM shall 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 is 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, for example IEC 61784 CPF 3 transferred value. The value is mapped to the related range (UpperRangeValue). For example, if a IEC 61784 CPF 9 device maps a range from 10 mbar to 100 mbar to an integer of value 1 024 to 4 096 the 'UpperRawValue' contains 4 096, the 'UpperRange' contains 4 096.
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

Tag	Description
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. It is expected that this information correlates to information provided by SW revision or HW revision in IDtmInformation2::GetDeviceIdentificationInformation().
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 C.3 for a list of valid classification IDs and Table C.4 for a mapping to IEC 60947).

Table C.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

classificationID
controller
angle
limitSwitch
converter
motor

Table C.4 defines the mapping between IEC 62390 device classification and the FDT classification attributes.

Table C.4 – Device classification according to IEC 62390 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

Domain ()		Subdomain
classificationDomainId	**IEC domain group name**	**classificationId**
		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>
```

C.3 FDTApplicationIdSchema

The application id schema is used as a global FDT definition (see Table C.5 and Table C.6). 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.

Table C.5 – Description of applicationId attribute

Attribute	Description
applicationId	Identification of the current application context (The enumeration refers to the DTM Realization Elements)

Table C.6 – Description of applicationId 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

C.4 FDTUserInformationSchema

Used at: IDtm::Config()

The user information schema is used as a global FDT definition (see Table C.7 and Table C.8). It informs the DTM during initialization about the role and the rights of the current user.

Table C.7 – Description of user information attributes

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 (can be given by the user)
userLevel	User level specifying users rights
userName	Name of the current user

Table C.8 – Description of user information elements

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>
```

C.5 DTMInformationSchema

Used at: IDtmInformation::GetInformation()

The XML document provides DTM specific information (see Table C.9 and Table C.10).

Table C.9 – Description of DTM information attributes

Attribute	Description
major	Major part of the FDT specification version on which the implementation of a DTM is based on. For FDT specification, 1.2.1.0 it shall be set to '1'
minor	Minor part of the FDT specification version on which the implementation of a DTM is based on. For FDT specification 1.2.1.0 it shall be set to '2'
release	release part of the FDT specification version on which the implementation of a DTM is based on. For FDT specification 1.2.1.0 it shall be set to '1'. It is highly recommended to use this attribute.
build	build part of the FDT specification version on which the implementation of a DTM is based on. For FDT specification 1.2.1.0 it shall be set to '0'. It is highly recommended to use this attribute.

Table C.10 – Description of DTM information elements

Tag	Description
DtmDeviceTypes	Collection of device types
DtmInfo	Describes the DTM itself
DtmSchemaPath	File system path with protocol specific schemas and XSLT files (path including a trailing backslash). It is expected that this path contains no other files. In order to avoid validation errors, it is recommended to avoid using XML-specific characters (like '&') in this path.
DtmSchemaPaths	Collection of DTM schema paths
FDTVersion	Definition of the element which specifies the version information on which the implementation of a DTM is based on
FDT	Root tag


```
<Schema name="DTMInformationSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
    <!--Definition of Attributes-->
    <AttributeType name="major" dt:type="number"/>
    <AttributeType name="minor" dt:type="number"/>
    <AttributeType name="deviceTypeInformation" dt:type="string"/>
    <AttributeType name="release" dt:type="number"/>
    <AttributeType name="build" dt:type="number"/>
    <!--Definition of Elements-->
    <ElementType name="FDTVersion" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="major" required="yes"/>
        <attribute type="minor" required="yes"/>
        <attribute type="release" required="no"/>
        <attribute type="build" required="no"/>
    </ElementType>
    <ElementType name="DtmDeviceTypes" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
```

```
        <element type="fdt:DtmDeviceType" minOccurs="1" maxOccurs="*"/>
    </ElementType>
    <ElementType name="DtmSchemaPath" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:busCategory" required="yes"/>
        <attribute type="fdt:path" required="yes"/>
    </ElementType>
    <ElementType name="DtmSchemaPaths" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="DtmSchemaPath " minOccurs="1" maxOccurs="*"/>
    </ElementType>
    <ElementType name="DtmInfo" content="eltOnly" model="closed" order="seq">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="FDTVersion" minOccurs="1" maxOccurs="1"/>
        <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1"/>
        <element type="DtmSchemaPaths" minOccurs="0" maxOccurs="1"/>
        <element type="DtmDeviceTypes" minOccurs="1" maxOccurs="*"/>
    </ElementType>
    <ElementType name="FDT" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="DtmInfo" minOccurs="1" maxOccurs="1"/>
    </ElementType>
</Schema>
```

Examples:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" >
    <DtmInfo>
        <FDTVersion major="1" minor="2" release="1" build="1"/>
        <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
        <DtmSchemaPaths>
            <DtmSchemaPath                    fdt:busCategory="12345678-ABCD-7890-86E1-00E0987270B9"
fdt:path="file://c:/mySchemaPath/" />
        </DtmSchemaPaths>
        <DtmDeviceTypes>
            <fdt:DtmDeviceType>
                <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
                <fdt:SupportedLanguages>
                    <fdt:LanguageId languageId="1131"/>
                </fdt:SupportedLanguages>
                <fdt:BusCategories>
                    <fdt:BusCategory                busCategory="036D1499-387B-11D4-86E1-00E0987270B9"
busCategoryName="Profibus DP/V1">
                        <fdt:CommunicationTypeEntry communicationType="required"/>
                    </fdt:BusCategory>
                    <fdt:BusCategory                busCategory="12345678-ABCD-7890-86E1-00E0987270B9"
busCategoryName="Private Bus">
                        <fdt:CommunicationTypeEntry communicationType="supported"/>
                    </fdt:BusCategory>
                    <fdt:BusCategory busCategory="036D1498-387B-11D4-86E1-00E0987270B9"
busCategoryName="HART">
                        <fdt:CommunicationTypeEntry communicationType="supported"/>
                        <fdt:PhysicalLayer          physicalLayer="036D1590-387B-11D4-86E1-00E0987270B9"
physicalLayerName="IEC61158-2"/>
                    </fdt:BusCategory>
                </fdt:BusCategories>
                <fdt:DeviceIcon path="file://c:/programs/manufacturer/icons/device.ico" />
                <fdt:DeviceIcon deviceGraphicState="device" path="file://c:/dev.ico" />
                <fdt:DeviceIcon deviceGraphicState="diagnosis" path="file://c:/devdiag.ico" />
                <fdt:DtmDocuments>
                    <fdt:DtmDocument label="Online Help" help="Help for Dtm" documentClassification="help"
documentLanguageId="1033">
                        <fdt:DocumentFile file="c:\mydtm\helpEN.chm"/>
                    </fdt:DtmDocument>
                    <fdt:DtmDocument label="Online Hilfe" help="Hilfe für DTM" documentClassification="help"
documentLanguageId="1031">
                        <fdt:DocumentFile file="c:\mydtm\helpDE.chm"/>
                    </fdt:DtmDocument>

                    <fdt:DtmDocument label="Ordering Information" help="Order our Products"
documentClassification="orderingInformation">
                        <fdt:DocumentUrl url="http://www.manufacturer.com"/>
                    </fdt:DtmDocument>
```

```
                <fdt:DtmDocument label="Product Database" help="Find a Product"
documentClassification="miscellaneous" documentLanguageId="1033">
                    <fdt:DocumentExe file="c:\programs\manufacturer\productFinder.exe" parameters="-L1033"
/>
                </fdt:DtmDocument>

            </fdt:DtmDocuments>
            <fdt:DeviceBitmap deviceGraphicState="device" path="file://c:/dev.bmp" />
            <fdt:DeviceBitmap deviceGraphicState="diagnosis" path="file://c:/devdiag.bmp" />
            <fdt:DataSetFormats>
                <fdt:Current dataSetID="036D1497-387B-11D4-86E1-00E0987270B9" description="The data set
in encrypted according to the client recommendation"/>
                <fdt:Supported dataSetID="036D1497-387B-11D4-86E1-00E0987270B9" levelOfSupport="full"
description="The data set is fully supported"/>
                <fdt:Supported dataSetID="036D1497-387B-11D4-86E1-00E0987270BA" levelOfSupport="full"
description="The default value for fault action is not supported in this version"/>
            </fdt:DataSetFormats>
        <fdt:DeviceTypeVariants>
            <fdt:DeviceTypeVariant deviceTypeVariant="PM255-EX" deviceTypeVariantInfo="PM255-EX Info"
orderCode="ExampleOrderCode1"/>
            <fdt:DeviceTypeVariant deviceTypeVariant="PM255-TX" deviceTypeVariantInfo="PM255-TX Info"
orderCode="ExampleOrderCode2"/>
            <fdt:DeviceTypeVariant deviceTypeVariant="PM255-ZX" deviceTypeVariantInfo="PM255-ZX Info"
orderCode="ExampleOrderCode3"/>
            <fdt:DeviceTypeVariant deviceTypeVariant="PM255-XX" deviceTypeVariantInfo="PM255-XX Info"
orderCode="ExampleOrderCode4"/>
        </fdt:DeviceTypeVariants>
        </fdt:DtmDeviceType>
    </DtmDeviceTypes>
</DtmInfo>
</FDT>
```

Example for IEC 61784 CPF 3:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" >
    <DtmInfo>
        <FDTVersion major="1" minor="2" release="1" build="1"/>
        <fdt:VersionInformation name="myName" vendor="myVendor" version="1.0" date="2000-08-05"/>
        <DtmDeviceTypes>
            <fdt:DtmDeviceType manufacturerId="12" deviceTypeId="3456" deviceTypeInformation="
#Profibus_DP
GSD_Revision          = 2
Vendor_Name           = &amp;amp;myVendor&amp;amp;
Model_Name                        = &amp;amp;myName&amp;amp;
Revision                          = &amp;amp;1.0&amp;amp;
Ident_Number                      = 0x0D80
Protocol_Ident                    = 0
Station_Type                      = 0
Bitmap_Device = &amp;amp;Src0D80n&amp;amp;
FMS_supp                                = 0
Hardware_Release                        = &amp;amp;1.0&amp;amp;
Software_Release                  = &amp;amp;1.0&amp;amp;
31.25_supp                              = 1
45.45_supp                              = 1
93.75_supp                              = 1
187.5_supp                              = 1
MaxTsdr_31.25                     = 100
MaxTsdr_45.45           = 250
MaxTsdr_93.75           = 1000
MaxTsdr_187.5           = 1000
Redundancy              = 0
Repeater_Ctrl_Sig       = 0
24V_Pins              = 0
Freeze_Mode_supp            = 0
Sync_Mode_supp          = 0
Auto_Baud_supp          = 0
Set_Slave_Add_supp      = 1
Min_Slave_Intervall     = 250
Modular_Station         = 1
Max_Module              = 6
Max_Input_Len           = 30
Max_Output_Len          = 6
Max_Data_Len          = 36
Max_Diag_Data_Len           = 14
Slave_Family          = 12
```

```
User_Prm_Data_Len       = 0
;Ext_User_Prm_Data_Const(0)     = 0x00,0x00,0x00

;Empty module
Module  = &amp;amp;EMPTY_MODULE&amp;amp;                         0x00
EndModule
"
                deviceTypeInformationPath="file://c:/myDtm/myGsdFile.gsd"
            >
                <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
                <fdt:SupportedLanguages>
                    <fdt:LanguageId languageId="1131"/>
                </fdt:SupportedLanguages>
                <fdt:BusCategories>
                    <fdt:BusCategory                    busCategory="036D1499-387B-11D4-86E1-00E0987270B9"
busCategoryName="Profibus DP/V1">
                            <fdt:CommunicationTypeEntry communicationType="required"/>
                    </fdt:BusCategory>
                </fdt:BusCategories>
                <fdt:DeviceIcon path="file://c:/programs/manufacturer/icons/device.ico" />
                <fdt:DeviceIcon deviceGraphicState="device" path="file://c:/myDtm/Src0D80n.ico" />
                <fdt:DeviceIcon deviceGraphicState="diagnosis" path="file://c:/myDtm/Src0D80ndiag.ico" />
                <fdt:DtmDocuments>
                    <fdt:DtmDocument label="Online Help" help="Help for Dtm" documentClassification="help"
documentLanguageId="1033">
                        <fdt:DocumentFile file="c:\mydtm\helpEN.chm"/>
                    </fdt:DtmDocument>
                    <fdt:DtmDocument label="Online Hilfe" help="Hilfe für DTM" documentClassification="help"
documentLanguageId="1031">
                        <fdt:DocumentFile file="c:\mydtm\helpDE.chm"/>
                    </fdt:DtmDocument>

                    <fdt:DtmDocument label="Ordering Information" help="Order our Products"
documentClassification="orderingInformation">
                        <fdt:DocumentUrl url="http://www.manufacturer.com"/>
                    </fdt:DtmDocument>

                    <fdt:DtmDocument label="Product Database" help="Find a Product"
documentClassification="miscellaneous" documentLanguageId="1033">
                        <fdt:DocumentExe file="c:\programs\manufacturer\productFinder.exe" parameters="-L1033"
/>
                    </fdt:DtmDocument>

                </fdt:DtmDocuments>
                <fdt:DeviceBitmap deviceGraphicState="device" path="file://c:/myDtm/Src0D80n.bmp" />
                <fdt:DeviceBitmap deviceGraphicState="diagnosis" path="file://c:/myDtm/Src0D80diag.bmp" />
                <fdt:DataSetFormats>
                    <fdt:Current dataSetID="036D1497-387B-11D4-86E1-00E0987270B9" description="The data set
in encrypted according to the client recommendation"/>
                    <fdt:Supported dataSetID="036D1497-387B-11D4-86E1-00E0987270B9" levelOfSupport="full"
description="The data set is fully supported"/>
                    <fdt:Supported dataSetID="036D1497-387B-11D4-86E1-00E0987270BA" levelOfSupport="full"
description="The default value for fault action is not supported in this version"/>
                </fdt:DataSetFormats>
            <fdt:DeviceTypeVariants>
                <fdt:DeviceTypeVariant deviceTypeVariant="PM255-EX" deviceTypeVariantInfo="PM255-EX Info"
orderCode="ExampleOrderCode1"/>
                <fdt:DeviceTypeVariant deviceTypeVariant="PM255-TX" deviceTypeVariantInfo="PM255-TX Info"
orderCode="ExampleOrderCode2"/>
                <fdt:DeviceTypeVariant deviceTypeVariant="PM255-ZX" deviceTypeVariantInfo="PM255-ZX Info"
orderCode="ExampleOrderCode3"/>
                <fdt:DeviceTypeVariant deviceTypeVariant="PM255-XX" deviceTypeVariantInfo="PM255-XX Info"
orderCode="ExampleOrderCode4"/>
            </fdt:DeviceTypeVariants>
            </fdt:DtmDeviceType>
        </DtmDeviceTypes>
    </DtmInfo>
</FDT>
```

C.6 DTMFunctionCallSchema

Used at:

IDtmActiveXInformation::GetActiveXGuid()

IDtmActiveXInformation::GetActiveXProgId()

IDtmApplication::StartApplication()

IDtmDocumentation::GetDocumentation()

IFdtChannelActiveXInformation::GetChannelActiveXGuid()

IFdtChannelActiveXInformation::GetChannelActiveXProgId

The XML document provides information to specifiy a specific function call (see Table C.11).

Table C.11 – Description of function call attributes

Tag	Description
FDTFunctionCall	Definition of the element which specifies a specific function call
FDT	Root tag

```
<Schema name="DTMFunctionCallSchema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
xmlns:appId="x-schema:FDTApplicationIdSchema.xml" xmlns:func="x-schema:DTMFunctionsSchema.xml"
xmlns:ops="x-schema:FDTOperationPhaseSchema.xml">
    <!--Definition of Elements-->
    <ElementType name="FDTFunctionCall" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="func:functionId" required="yes"/>
        <element type="appId:ApplicationId" minOccurs="0" maxOccurs="1"/>
        <attribute type="ops:operationPhase" required="no"/>
    </ElementType>
    <ElementType name="FDT" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="FDTFunctionCall" minOccurs="1" maxOccurs="1"/>
    </ElementType>
</Schema>
```

Example:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMFunctionCallSchema.xml" xmlns:func="x-schema:DTMFunctionsSchema.xml"
xmlns:appId="x-schema:FDTApplicationIdSchema.xml" xmlns:ops="x-
schema:FDTOperationPhaseSchema.xml" >
    <FDTFunctionCall func:functionId="32" ops:operationPhase ="runtime">
        <appId:ApplicationId applicationId="fdtObserve"/>
    </FDTFunctionCall>
</FDT>
```

C.7 DTMParameterSchema

Used at:

IDtmParameter::GetParameters ()

IDtmParameter::SetParameters()

The XML document provides all instance specific information about a device. For a description of the document contents see Table C.12 and Table C.13.

Table C.12 – Description of parameter document attributes

Attribute	Description
busAddress	Network participant number of a device according to fieldbus protocols like IEC 61784 CPF 3
busMasterConfigurationPart	Bus parameters are needed to allow an interaction between DTMs and a master configuration tool. To provide a standard access to this bus specific data, it is stored as a binary stream which contains the device

Attribute	Description
	specific bus information according to the fieldbus specification. Each DTM shall a least fill in the device specific parameters and all parameters which can be changed by its application. All other entries can be filled up with substitute values like zeros. The substituted values of the structure will be set by the environment's master configuration tool according to the requirements of the complete bus. Independent of the values filled in, it is very important that the structure generated by the DTM adheres to the definitions of the fieldbus specification. After all network participants have written their instance data, the master configuration tool can commission the fieldbus. For that purpose it collects the configuration part of each network participant and calculates the bus parameters of the corresponding master device.
configurationData	Additional configuration data as binary stream according to the fieldbus specification
moduleId	Unique identifier for a module within the name space of the device instance
moduleTypeId	Unique identifier for a module type within the name space of the device type
redundant	Specifies whether a device or module is redundant
slaveAddress	Universal slave address e.g. polling address for IEC 61784 CPF 9 communication
slot	Unique identifier for the slot of a module within the name space of the device instance

Table C.13 – Description of parameter document elements

Tag	Description
BusInformation	This element is an implementation of data type 'fdt:NetworkInfo' as defined in IEC 62453-2. It is recommended that Device DTMs with more than one required protocol includes busCategory in BusInformation. If a 1.2.1. DTM is connected to a 1.2 Communication Channel, a 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 channel and a 1.2 Frame Application has to consider: • The DTM can support only one protocol, because the old schema supports only one BusInformation element
DtmDevice	Description of a device instance It is recommended that Device DTMs with more than one required protocol include BusInformation for each protocol.
ExportedVariables	Collection of DTM variables for common access. This means that the Frame Application or other DTMs are allowed to get access to the data within this section
FDT	Root tag
InternalChannel	An internal channel is the connection point for an internal module within the internal topology
InternalTopology	Description of the internal topology of a modular device build as a none software modular DTM
MasterSlaveBus	Description of bus parameters for communication and configuration This element is an implementation of data type 'fdt:DeviceAddress' as defined in IEC 62453-2.
Module	Description of a hardware or software module of a device

Tag	Description
Modules	Collection of modules
SlaveAddress	Universal slave address. This element is an implementation of data type 'fdt:DeviceAddress' as defined in IEC 62453-2.
UserDefinedBus	Description bus parameters for the integration of proprietary bus systems . This element is an implementation of data type 'fdt:UserDefinedBus' as defined in IEC 62453-2. It is specified as open element so that protocol specific elements may by integrated into this element.

If a DTM can be used by an redundant aware parent component as DTM for a redundant slave, the parameter document shall provide additional redundancy information:

- the BusInformation element shall contain a BusRedundancy element with the number of supported redundant fieldbus interfaces of the slave.

It is recommend that with the address, set by calling SetParameter(), the parent component provides complete redundant address information. Then the DTM is able to detect if its appropriate slave is used as redundant slave. Only in this case, specific redundancy handling is to be activated within the DTM, for example during communication requests or configuration dialogs. Therefore, it is recommended to set all fieldbus addresses using SlaveAddress elements by the parent component.

```
<Schema name="DTMParameterSchema" xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
    <!--Definition of Attributes-->
    <AttributeType name="busAddress" dt:type="ui4"/>
    <AttributeType name="busMasterConfigurationPart" dt:type="bin.hex"/>
    <AttributeType name="configurationData" dt:type="bin.hex"/>
    <AttributeType name="moduleId" dt:type="ui4"/>
    <AttributeType name="moduleTypeId" dt:type="ui4"/>
    <AttributeType name="redundant" dt:type="boolean"/>
    <AttributeType name="slaveAddress" dt:type="ui4"/>
    <AttributeType name="slot" dt:type="ui4"/>
    <!--Definition of Elements-->
    <ElementType name="SlaveAddress" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:readAccess" required="no"/>
        <attribute type="fdt:writeAccess" required="no"/>
        <attribute type="slaveAddress" required="yes"/>
    </ElementType>
    <ElementType name="MasterSlaveBus" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:readAccess" required="no"/>
        <attribute type="fdt:writeAccess" required="no"/>
        <attribute type="busAddress" required="yes"/>
        <attribute type="busMasterConfigurationPart" required="yes"/>
    </ElementType>
    <ElementType name="UserDefinedBus" content="mixed" model="open"/>
    <ElementType name="BusInformation" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
    <attribute type="fdt:busCategory" required="no"/>
        <element type="fdt:BusRedundancy" maxOccurs="1" minOccurs="0"/>
        <group order="one" minOccurs="0" maxOccurs="*">
            <element type="SlaveAddress"/>
            <element type="MasterSlaveBus"/>
            <element type="UserDefinedBus"/>
        </group>
    </ElementType>
    <ElementType name="ExportedVariables" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="fdt:DtmVariables" minOccurs="1" maxOccurs="1"/>
    </ElementType>
    <ElementType name="Module" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:readAccess" required="no"/>
        <attribute type="fdt:writeAccess" required="no"/>
        <attribute type="moduleId" required="yes"/>
        <attribute type="moduleTypeId" required="no"/>
        <attribute type="slot" required="no"/>
        <attribute type="redundant" required="no"/>
```

```
        <attribute type="configurationData" required="no"/>
        <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1"/>
        <element type="fdt:ChannelReferences" minOccurs="0" maxOccurs="1"/>
        <element type="ExportedVariables" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <ElementType name="InternalChannel" content="eltOnly" model="closed">
        <attribute type="fdt:readAccess" required="no"/>
        <attribute type="fdt:writeAccess" required="no"/>
        <attribute type="fdt:nodeId" required="no"/>
        <element type="Module" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <ElementType name="InternalTopology" content="eltOnly" model="closed">
        <attribute type="fdt:readAccess" required="no"/>
        <attribute type="fdt:writeAccess" required="no"/>
        <attribute type="fdt:nodeId" required="no"/>
        <element type="BusInformation" minOccurs="0" maxOccurs="1"/>
        <element type="InternalChannel" minOccurs="1" maxOccurs="*"/>
    </ElementType>
    <ElementType name="DtmDevice" content="eltOnly" model="closed" order="seq">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:readAccess" required="no"/>
        <attribute type="fdt:writeAccess" required="no"/>
        <attribute type="fdt:tag" required="yes"/>
        <attribute type="redundant" required="no"/>
        <element type="fdt:ChannelReferences" minOccurs="0" maxOccurs="1"/>
        <element type="BusInformation" minOccurs="0" maxOccurs="*"/>
        <element type="InternalTopology" minOccurs="0" maxOccurs="1"/>
        <element type="ExportedVariables" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <ElementType name="FDT" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:storageState" required="yes"/>
        <attribute type="fdt:dataSetState" required="yes"/>
        <attribute type="fdt:modifiedInDevice" required="no"/>
        <element type="fdt:DtmDeviceType" minOccurs="1" maxOccurs="1"/>
        <element type="DtmDevice" minOccurs="1" maxOccurs="1"/>
    </ElementType>
</Schema>
```

Example for a simple device:

```
<?xml version="1.0"?>
<FDT    xmlns="x-schema:DTMParameterSchema.xml"    xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default ">
    <fdt:DtmDeviceType >
        <fdt:VersionInformation  name="Transmitter"  vendor="Vendor  name"  version="1.0"  date="2000-08-
05"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="00PGH10EC001">
        <fdt:ChannelReferences>
            <fdt:ChannelReference idref="temperature"/>
        </fdt:ChannelReferences>
        <ExportedVariables>
            <fdt:DtmVariables name="Parameter" descriptor="root of parameters">
                <fdt:DtmVariable name="Cell" descriptor="Measuring point data">
                    <fdt:Value>
                        <fdt:Display string="PT100"/>
                    </fdt:Value>
                    <fdt:Unit>
                        <fdt:EnumeratorVariable>
                            <fdt:Variable>
                                <fdt:EnumeratorEntry index="1" name="C"/>
                            </fdt:Variable>
                            <fdt:EnumeratorEntries>
                                <fdt:EnumeratorEntry index="1" name="C"/>
                                <fdt:EnumeratorEntry index="2" name="K"/>
                                <fdt:EnumeratorEntry index="3" name="F"/>
                            </fdt:EnumeratorEntries>
```

```
                    </fdt:EnumeratorVariable>
                </fdt:Unit>
            </fdt:DtmVariable>
            <fdt:DtmVariables name="Curve" descriptor="characteristic curve interpolation points">
                <fdt:DtmVariable name="point1">
                    <fdt:Value>
                        <fdt:Display string="1.1"/>
                    </fdt:Value>
                </fdt:DtmVariable>
                <fdt:DtmVariable name="point2">
                    <fdt:Value>
                        <fdt:Display string="1.2"/>
                    </fdt:Value>
                </fdt:DtmVariable>
                <fdt:DtmVariable name="point3">
                    <fdt:Value>
                        <fdt:Display string="1.3"/>
                    </fdt:Value>
                </fdt:DtmVariable>
            </fdt:DtmVariables>
        </fdt:DtmVariables>
    </ExportedVariables>
</DtmDevice>
</FDT>
```

Example for a redundant slave:

```
<?xml version="1.0"?>
<FDT fdt:dataSetState="default " fdt:storageState="persistent" xmlns="x-schema:DTMParameterSchema.xml"
xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
    <fdt:DtmDeviceType readAccess="1" writeAccess="0">
        <fdt:VersionInformation date="2000-08-05" name="myname" readAccess="1" vendor="myVendor"
version="1.0" writeAccess="0"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9">
            </fdt:BusCategory>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9">
            </fdt:BusCategory>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="myTag">
        <fdt:ChannelReferences>
            <fdt:ChannelReference idref="n1"/>
            <fdt:ChannelReference idref="n25"/>
        </fdt:ChannelReferences>
        <BusInformation>
            <fdt:BusRedundancy busRedundancy="2"/>
            <SlaveAddress fdt:readAccess="1" fdt:writeAccess="0" slaveAddress="123"/>
            <SlaveAddress fdt:readAccess="1" fdt:writeAccess="0" slaveAddress="1234"/>
        </BusInformation>
        <ExportedVariables>
            <fdt:DtmVariables descriptor="root of parameters" name="ParaRoot">
                <fdt:DtmVariable descriptor="display only parameter" name="Simple">
                    <fdt:Value readAccess="1" writeAccess="0">
                        <fdt:Display string="12.345"/>
                    </fdt:Value>
                </fdt:DtmVariable>
            </fdt:DtmVariables>
        </ExportedVariables>
    </DtmDevice>
</FDT>
```

Example for the usage of channel mode:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default ">
    <fdt:DtmDeviceType readAccess="1" writeAccess="0">
```

```
        <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="myTag">
        <fdt:ChannelReferences>
            <fdt:ChannelReference idref="n1"/>
            <fdt:ChannelReference idref="n25"/>
        </fdt:ChannelReferences>
        <InternalTopology>
            <InternalChannel>
                <Module moduleId="1">
                    <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-
05"/>
                    <fdt:ChannelReferences>
                        <fdt:ChannelReference idref="m25.n1">
                            <fdt:ChannelInformation>
                                <fdt:BusCategories>
                                    <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
                                </fdt:BusCategories>
                                <fdt:ChannelModes>
                                    <fdt:ChannelMode channelMode="processValue" />
                                </fdt:ChannelModes>
                            </fdt:ChannelInformation>
                        </fdt:ChannelReference>
                        <fdt:ChannelReference idref="m25.n25">
                            <fdt:ChannelInformation>
                                <fdt:BusCategories>
                                    <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
                                    <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
                                </fdt:BusCategories>
                                <fdt:ChannelModes>
                                    <fdt:ChannelMode channelMode="communication" />
                                    <fdt:ChannelMode channelMode="processValue" />
                                </fdt:ChannelModes>
                            </fdt:ChannelInformation>
                        </fdt:ChannelReference>
                    </fdt:ChannelReferences>
                </Module>
            </InternalChannel>
        </InternalTopology>
        <ExportedVariables>
            <fdt:DtmVariables name="ParaRoot" descriptor="root of parameters">
                <fdt:DtmVariable name="Simple" descriptor="display only parameter">
                    <fdt:Value>
                        <fdt:Display string="12.345"/>
                    </fdt:Value>
                </fdt:DtmVariable>
                <fdt:DtmVariables name="SubStructur" descriptor="sub structure of parameters">
                    <fdt:DtmVariable name="Integer" descriptor="editable integer format">
                        <fdt:Value>
                            <fdt:Variant dataType="int" byteLength="2">
                                <fdt:NumberData number="4711"/>
                            </fdt:Variant>
                        </fdt:Value>
                    </fdt:DtmVariable>
                </fdt:DtmVariables>
            </fdt:DtmVariables>
        </ExportedVariables>
    </DtmDevice>
</FDT>
```

Example for a modular device:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default">
    <fdt:DtmDeviceType>
        <fdt:VersionInformation name="RemoteI/O" vendor="Vendor name" version="1.0" date="2000-08-05"
descriptor="Bus coupler"/>
```

```
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="00PGH10EC001">
        <fdt:ChannelReferences>
            <fdt:ChannelReference idref="Binary input"/>
        </fdt:ChannelReferences>
        <InternalTopology>
            <InternalChannel>
                <Module moduleId="1">
                    <fdt:VersionInformation name="AI4" vendor="Vendor name" version="1.0" date="2000-08-
05" descriptor="Analog Input 4 Channels"/>
                    <fdt:ChannelReferences>
                        <fdt:ChannelReference idref="AI4.C1"/>
                        <fdt:ChannelReference idref="AI4.C2"/>
                        <fdt:ChannelReference idref="AI4.C3"/>
                        <fdt:ChannelReference idref="AI4.C4"/>
                    </fdt:ChannelReferences>
                </Module>
            </InternalChannel>
            <InternalChannel>
                <Module moduleId="8">
                    <fdt:VersionInformation name="AO4" vendor="Vendor name" version="1.0" date="2000-08-
05" descriptor="Analog Output 4 Channels"/>
                    <fdt:ChannelReferences>
                        <fdt:ChannelReference idref="AO4.C1"/>
                        <fdt:ChannelReference idref="AO4.C2"/>
                        <fdt:ChannelReference idref="AO4.C3"/>
                        <fdt:ChannelReference idref="AO4.C4"/>
                    </fdt:ChannelReferences>
                    <ExportedVariables>
                        <fdt:DtmVariables name="Parameter" descriptor="root of parameters">
                            <fdt:DtmVariable name="Substitute Value" descriptor="Module type specific
parameter">
                                <fdt:Value>
                                    <fdt:Display string="false"/>
                                </fdt:Value>
                            </fdt:DtmVariable>
                        </fdt:DtmVariables>
                    </ExportedVariables>
                </Module>
            </InternalChannel>
        </InternalTopology>
    </DtmDevice>
</FDT>
```

Example for a modular device with one DTM for the bus coupler and a DTM for each module type:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default ">
    <fdt:DtmDeviceType >
        <fdt:VersionInformation name="RemoteI/O" vendor="Vendor name" version="1.0" date="2000-08-05"
descriptor="Bus coupler"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="00PGH10EC001">
        <fdt:ChannelReferences>
```

```
            <fdt:ChannelReference idref="Binary input"/>
        </fdt:ChannelReferences>
    </DtmDevice>
</FDT>


<?xml version="1.0"?>
<FDT     xmlns="x-schema:DTMParameterSchema.xml"     xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default ">
    <fdt:DtmDeviceType >
        <fdt:VersionInformation   name="AI4"   vendor="Vendor   name"   version="1.0"   date="2000-08-05"
descriptor="Analog Input 4 Channels"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="00PGH10EC001">
        <fdt:ChannelReferences>
            <fdt:ChannelReference idref="AI4.C1"/>
            <fdt:ChannelReference idref="AI4.C2"/>
            <fdt:ChannelReference idref="AI4.C3"/>
            <fdt:ChannelReference idref="AI4.C4"/>
        </fdt:ChannelReferences>
    </DtmDevice>
</FDT>


<?xml version="1.0"?>
<FDT     xmlns="x-schema:DTMParameterSchema.xml"     xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default ">
    <fdt:DtmDeviceType >
        <fdt:VersionInformation   name="AO4"   vendor="Vendor   name"   version="1.0"   date="2000-08-05"
descriptor="Analog Output 4 Channels"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
            <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="00PGH10EC001">
        <fdt:ChannelReferences>
            <fdt:ChannelReference idref="AO4.C1"/>
            <fdt:ChannelReference idref="AO4.C2"/>
            <fdt:ChannelReference idref="AO4.C3"/>
            <fdt:ChannelReference idref="AO4.C4"/>
        </fdt:ChannelReferences>
        <ExportedVariables>
            <fdt:DtmVariables name="Parameter" descriptor="root of parameters">
                <fdt:DtmVariable name="Substitute Value" descriptor="Module type specific parameter">
                    <fdt:Value>
                        <fdt:Display string="false"/>
                    </fdt:Value>
                </fdt:DtmVariable>
            </fdt:DtmVariables>
        </ExportedVariables>
    </DtmDevice>
</FDT>
```

Example for a completed datatype structure:

```
<?xml version="1.0"?>
<FDT     xmlns="x-schema:DTMParameterSchema.xml"     xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default ">
    <fdt:DtmDeviceType>
```

```
        <fdt:VersionInformation name="Transmitter" vendor="Vendor name" version="1.0" date="2000-08-
05"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="00PGH10EC001">
        <ExportedVariables>
            <fdt:DtmVariables name="Parameter" descriptor="root of parameters">
                <fdt:DtmVariable name="Float">
                    <fdt:Value>
                        <fdt:Display string="2.34"/>
                    </fdt:Value>
                </fdt:DtmVariable>
                <fdt:DtmVariable name="Integer">
                    <fdt:Value>
                        <fdt:Variant dataType="int">
                            <fdt:NumberData number="2"/>
                        </fdt:Variant>
                    </fdt:Value>
                </fdt:DtmVariable>
                <fdt:DtmVariable name="Integer2">
                    <fdt:Value>
                        <fdt:Variant dataType="int">
                            <fdt:NumberData number="3"/>
                        </fdt:Variant>
                    </fdt:Value>
                </fdt:DtmVariable>
                <fdt:DtmVariable name="Structured1">
                    <fdt:Value>
                        <fdt:Variant dataType="structured">
                            <fdt:StructuredVariable>
                                <fdt:BinaryVariable binData="28FC215403"/>
                                <fdt:StructuredElements>
                                    <fdt:StructuredElement bitLength="4">
                                        <fdt:DtmVariableReference reference="Integer"/>
                                    </fdt:StructuredElement>
                                    <fdt:StructuredElement bitLength="32">
                                        <fdt:DtmVariableReference reference="Float"/>
                                    </fdt:StructuredElement>
                                    <fdt:StructuredElement bitLength="4">
                                        <fdt:DtmVariableReference reference="Integer2"/>
                                    </fdt:StructuredElement>
                                </fdt:StructuredElements>
                            </fdt:StructuredVariable>
                        </fdt:Variant>
                    </fdt:Value>
                </fdt:DtmVariable>
            </fdt:DtmVariables>
        </ExportedVariables>
    </DtmDevice>
</FDT>
```


Example for device type variant

```
<FDT xmlns="x-schema:DTMParameterSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
fdt:storageState="persistent" fdt:dataSetState="default ">
    <fdt:DtmDeviceType readAccess="1" writeAccess="0">
        <fdt:VersionInformation name="Name" vendor="Vendor" version="1.0" date="2000-08-05"/>
        <fdt:SupportedLanguages>
            <fdt:LanguageId languageId="1311"/>
        </fdt:SupportedLanguages>
        <fdt:BusCategories>
            <fdt:BusCategory busCategory="036D1497-387B-11D4-86E1-00E0987270B9"/>
        </fdt:BusCategories>
    <fdt:DeviceTypeVariant deviceTypeVariant="PM255-EX" deviceTypeVariantInfo="PM255-EX Info"
orderCode="ExampleOrderCode"/>
```

```
    </fdt:DtmDeviceType>
    <DtmDevice fdt:tag="myTag">
        ...
    </DtmDevice>
</FDT>
```

C.8 DTMDocumentationSchema

Used at:

IDtmDocumentation::GetDocumentation()

The XML document contains the instance specific information according to the request. The context (function id, application id and operation phase) is described via the passed XML document of type DTMFunctionCallSchema (see Table C.14 and Table C.15).

Table C.14 – Description of documentation attributes

Attribute	Description
path	Path to an object that has to be embedded within the document like bitmaps or other graphical elements
title	Human readable title for the documentation

Table C.15 – Description of documentation elements

Tag	Description
DocumentVariable	Human readable variable description with name, value, range, etc
DocumentVariables	Collection of document variables
DTMSpecificXMLData	Optional additional information which shall be described by a private style
DTMStyleForCompleteDocument	Optional style information which has to be provided by a DTM if it returns documents which cannot be described by the FDT standard style
FDT	Root tag
GraphicReference	Reference to an object that has to be embedded within the document like bitmaps or other graphical elements

```
<Schema name="DTMDocumentationSchema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml" xmlns:dtmi="x-
schema:DTMInformationSchema.xml" >
    <!--Definition of Attributes-->
    <AttributeType name="path" dt:type="uri"/>
    <AttributeType name="title" dt:type="string"/>
    <!--Definition of Elements-->
    <ElementType name="GraphicReference" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="title" required="yes"/>
        <attribute type="fdt:descriptor" required="no"/>
        <attribute type="path" required="yes"/>
    </ElementType>
    <ElementType name="DocumentVariable" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:name" required="yes"/>
        <attribute type="fdt:descriptor" required="no"/>
        <attribute type="fdt:display" required="yes"/>
        <element type="fdt:Unit" minOccurs="0" maxOccurs="1"/>
        <element type="fdt:Ranges" minOccurs="0" maxOccurs="1"/>
        <attribute type="fdt:statusFlag" required="no"/>
        <element type="fdt:StatusInformation" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <ElementType name="DocumentVariables" content="mixed" model="closed">
```