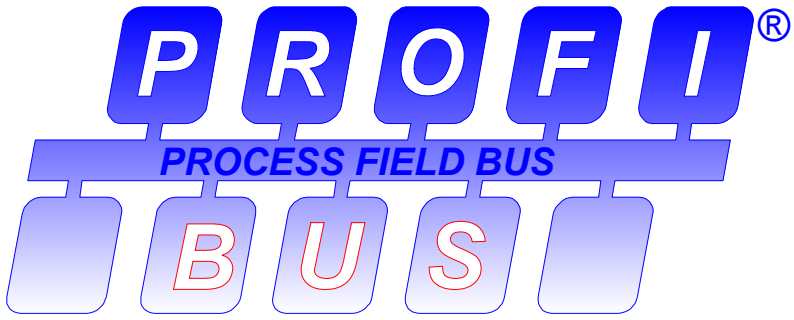
PROFIBUS



PROFIBUS Guideline

Specification for PROFIBUS Device Description and Device Integration

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

Volume 3: FDT V 1.2, Mai 2001
Addendum

PROFIBUS Guideline, Order No. 2.162_Add

Specification for PROFIBUS

Device Description and Device Integration

Volume 1: GSD V3.0
Volume 2: EDDL V1.0

Volume 3: FDT V 1.2, Mai 2001
Addendum

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

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

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

1 Abstract

This document contains additional information concerning the FDT Specification 1.2. All changes are backward compatible and will be part of the next release of FDT.

Even if these changes are backward compatible, it is recommended to check existing FDT components concerning the mentioned changes.

2 Changes

2.1 Chapter 2.7.1, Persistence Overview

Remarks:

- In order to simplify the DTM development, it is up to a DTM to implement one of the defined persistent interfaces (IPersistStreamInit or IPersistPropertyBag) according to the Microsoft standard. The frame-application must be able to handle both.
- References to DTMs do not belong to the instance data of a DTM. A DTM must not store any references to other DTMs. A DTM can get information concerning its parents or childs via IFdtTopology::GetParentNodes() and IFdtTopology::GetChildNotes().

2.2 Chapter 2.10, Abstract FDT Device Model

Change	Reason
Additional description to clarify the meaning of objects	Enhanced description

Object	Description
DTM	Each device is represented by a DTM object. This object is the starting point for navigation to the finer parts of the device like modules and channels. A device can be subdivided into modules and sub modules. A module is either a hardware module or a software module. Hardware modules are plugged into physical slots of the device. For example, an I/O module can be plugged into a Remote I/O device. A module can also be a software module. Software modules represent static or configurable substructures of a device like a closed loop module. A DTM may represent different types of devices, e.g. field devices, communication devices and gateway devices. • A Device-DTM represents a 'normal' field device that uses a Communication-Channel to communicate with the related field device • A Communication-DTM represents a communication device that

Object	Description
	provides communication capabilities via Communication-Channels (in sense of FDT) but does not needs any communication capabilities of a parent DTM • A Gateway-DTM is a Communication-DTM that provides communication capabilities via Communication-Channels (in sense of FDT) and requires communication capabilities of a parent DTM
FdtChannel	FdtChannel-Object could behave in 2 ways: As a 'Communication-Channel' and a 'Process-Channel'. • Communication-Channel is an object that provides access to communication infrastructure. It is used by Device-DTM or Gateway-DTM as a service provider for communication. • Process-Channel represents a single process value, its optional state information, and its assigned ranges and alarms. A channel either belongs to a device or a module DTM. An FdtChannel-Object could implement both behaviors.

2.3 Chapter 3.2.2, DTM State Machine

Change	Reason
Call of PrepareToRelease() moved from 'new created' to 'new enabled'	Within the state 'new created', the interface pointer IFdtContainer is unknown by the DTM. So, the DTM is not able to call OnPreparedToRelease()
Call of IDtmInformation(), IDtmActiveXInformation() and IFdtChannelActiveXInformation() removed from state 'existing created'	Within the state 'existing created', the interface pointer IFdtContainer is unknown by the DTM. So, the DTM is not able to determine the location of the XML Schemas which is required to parse XML documents
New entry IDtmImportExport	Missing
Remark concerning state transitions in case of errors	Clear definition was missed
Allow 'Save' within state 'online'	Allow saving of online related data
Allow 'PrepareToReleaseCommunication' within state 'communication set' 'going online' and 'going offline'	Simplification of handling concerning state transition between 'online' and 'communication set'
Allow 'GetSupportedProtocols' within state 'configured'	Allow to get the information concerning supported protocols also in offline state

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

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

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

Interface / Method	up	new created	new enabled	existing created	running	configured	going offline	online	going online	destroyed	communication set	zombie
DisconnectRequest							√	√	√		√	
TransactionRequest							√	√	√		√	
GetSupportedProtocols						√	√	√	√		√	
SequenceBegin							√	√	√		√	
SequenceStart							√	√	√		√	
SequenceEnd							√	√	√		√	
IFdtFunctionBlockData						√	√	√	√		√	

2.4 Chapter 3.3.1.3, GetFunctions

Change	Reason
Additional description concerning the usage of ‘notSupported’ as value for attribute ‘operationPhase’	Enhanced description

Comments:

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

2.5 Chapter 3.3.1.7, PrepareToRelease

Change	Reason
Additional hint concerning the usage of SaveRequest() within this method	Enhanced description

Comments

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

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

2.6 Chapter 3.3.1.8, PrepareToReleaseCommunication

Change	Reason
Additional comment	Enhanced description

Comments

The method returns FALSE if communication call is active and cannot be terminated.
The method returns TRUE if DTM accepts shutdown of communication. DTM has to fire progress events to inform frame application about ongoing progress if IDtmEvents::OnPrepareToReleaseCommunication() notification may take a longer time, for example if still some communication calls have not returned.

See also 3.7.1.13, OnProgress and 3.3.1.11 SetCommunication

2.7 Chapter 3.3.1.9, PrivateDialogEnabled

Change	Reason
More exact definition of the parameter 'enabled'. Additional 'Behavior' description concerning the usage of private dialogs	Enhanced description

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

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

.

2.8 Chapter 3.3.1.10, ReleaseCommunication

Change	Reason
Additional reference to chapter 3.7.1.11	Enhanced description

Comments

See also 3.7.1.11 OnPreparedToReleaseCommunication

2.9 Chapter 3.3.1.11, SetCommunication

Comments

To ensure a proper behavior, the frame application should implement the following rules:

- For each DTM, which acts as a root concerning the chain of interfaces for nested communication, the method SetCommunication() must be called with a NULL pointer as parameter ‘communication’. This leads to the transition from ‘configured’ to ‘communication set’ concerning the DTM state machine (refer to chapter 3.2.2, DTM State Machine)
- To set up the chain for nested communication, the frame application should 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 3.3.1.8 PrepareToReleaseCommunication, the frame application should act vice versa

2.10 Chapter 3.3.1.12, SetLanguage

Change	Reason
Additional comment concerning the coding of LCID	Enhanced description
Additional comment concerning the output format of data	Clarification

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)

Comments

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

2.11 Chapter 3.3.12.2, SetParameters

Change	Reason
Additional comment	Clarification concerning the term ‘transient’

Behavior

Transient data will become persistent e.g. by calling IFdtContainer::SaveRequest.

2.12 Chapter 3.3.13.1, OnChildParameterChanged

Change	Reason
Additional comment	Give a hint concerning the update of frame application related user interfaces

Comments

The frame-application has to ensure, that the parent DTM instance which will be selected to perform IFdtEvents::OnChildParameterChanged(), is in any case capable to lock its data set.

2.13 Chapter 3.3.13.3, OnParameterChanged

Change	Reason
Correction concerning description of behavior	Missing hint, that the DTM must store any changed data before calling IDtmEvents::OnParameterChanged(). See also 3.7.1.9
Additional comment	Give a hint concerning the update of frame application related user interfaces

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

This notification could also be used by a frame application to trigger an update e.g. to visualize the topology information.

2.14 Chapter 3.5.1.1, GetChannelParameters

Change	Reason
Clarification concerning the handling of ‘pure’ communication channels	Support of channels without any process data related information not clear

Behavior

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

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

2.15 Chapter 3.5.1.3, SetChannelParameters

Change	Reason
Additional comment	Missing hint concerning transient data

Behavior

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

Comments

See also chapter: State machine of instance data

2.16 Chapter 3.5.3, Interface IFdtCommunication

Change	Reason
Additional comment concerning the usage of request and response functions	Enhanced description

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.

2.17 Chapter 3.5.5.1, OnAddChild

Change	Reason
Additional comment concerning the usage	Enhanced description

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

2.18 Chapter 3.5.5.2, OnRemoveChild

Change	Reason
Additional comment concerning the usage	Enhanced description

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

2.19 Chapter 3.7.1.3, OnErrorMessage

Change	Reason
Correction concerning description	OnErrorMessage will also be used in combination with synchronous function calls
Additional comment concerning the usage	Enhanced description

Description

Notification by a DTM about errors during a function call

Comment

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.

2.20 Chapter 3.7.1.13, OnProgress

Change	Reason
Additional comment concerning the usage	Enhanced description

Description

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

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

It's up to the frame-application how to handle the information.

2.21 Chapter 3.7.5.3, SaveRequest

Change	Reason
Additional comment concerning the usage of IPersistXXX::IsDirty	Clarify the usage of the persistent interface

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.

2.22 Chapter 3.7.7.8, ReleaseDtmForSystemTag

Change	Reason
Added text 'the reference to'	Clarify the behavior

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

2.23 Chapter 3.8.1, Version Interoperability

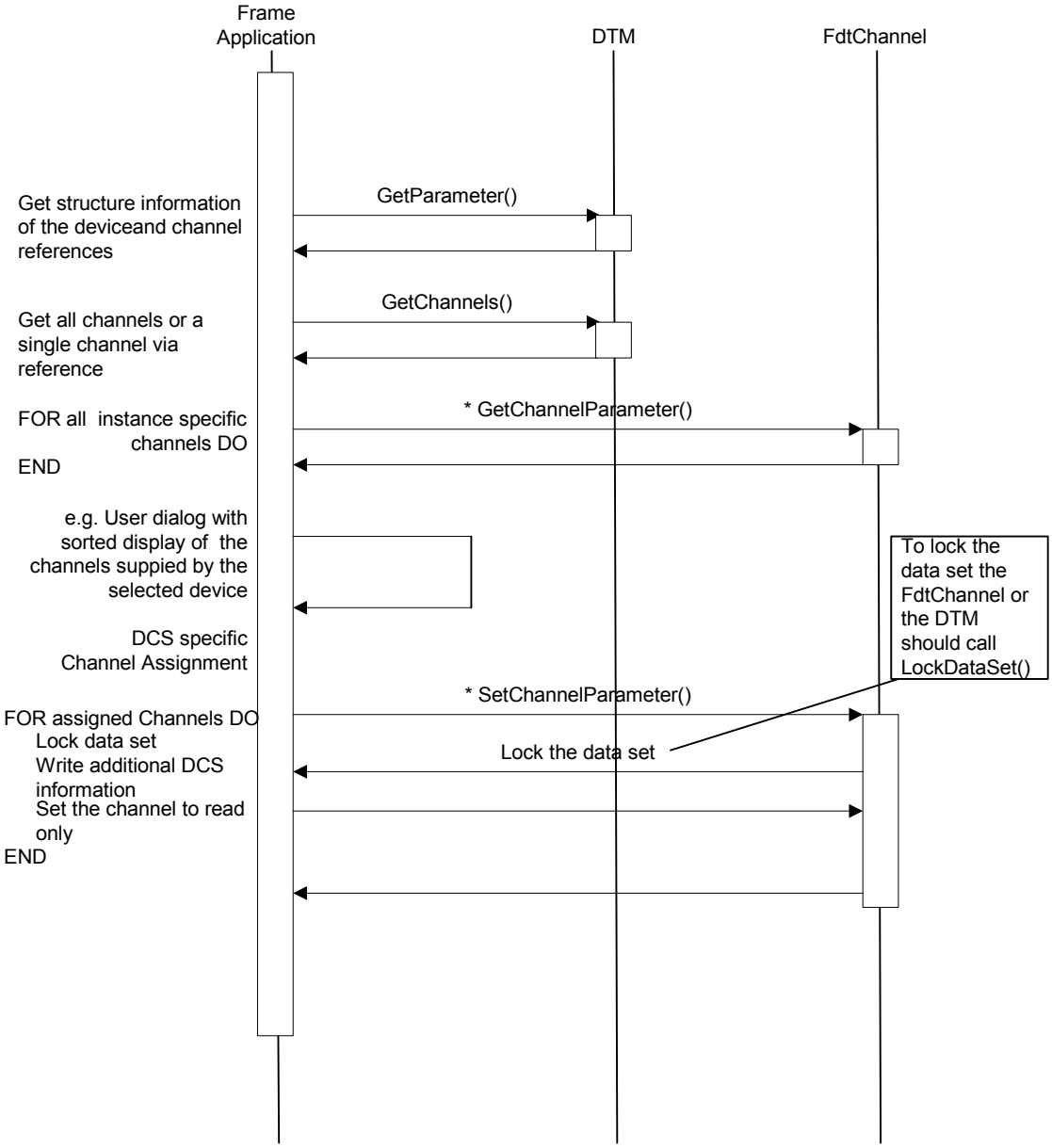
Change	Reason
Entry 'IFdtCommunicationEvents' beneath FDT Channel was removed	According to chapter 3.2.1, FDT Object Model, this entry is not valid
Entry 'IFdtChannel' beneath FDT Channel changed from 'Optional' to 'Mandatory'	IFdtChannel must be a mandatory interface due to the usage within the FDT-Specification
New entry IPersistXXX	Missing

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

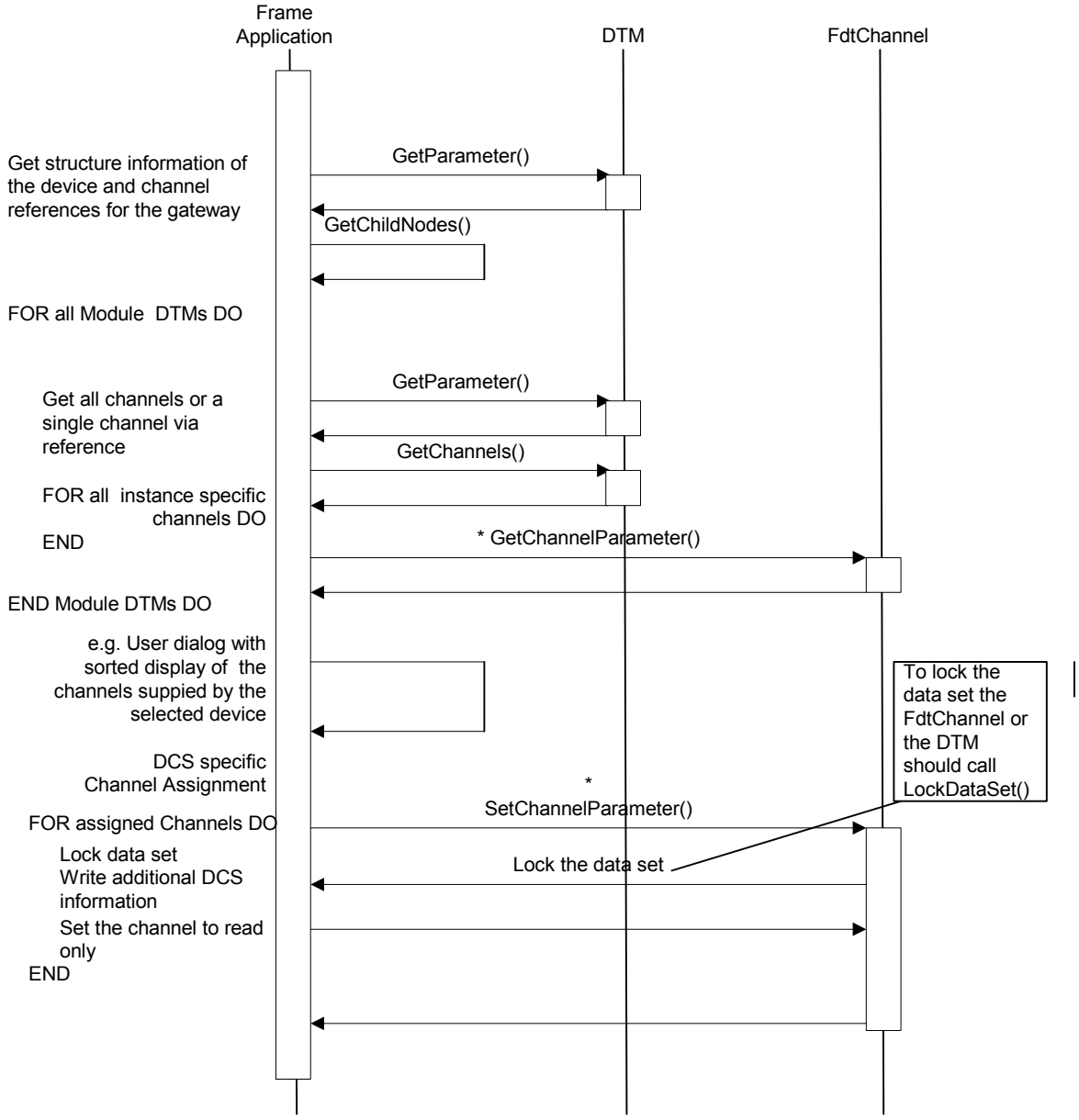
2.24 Chapter 5.7 DCS Channel Assignment

Change	Reason
Sequence chart for ‘Single DTM’	Clarification concerning the usage of LockDataSet()
Sequence chart for ‘Modular DTM Structure’	Clarification concerning the usage of LockDataSet()

Sequence chart for ‘Single DTM’



Sequence chart for 'Modular DTM Structure'



2.25 Chapter 6.1.4, Installation Issues

Change	Reason
Ownership of XML schemas	Additional paragraph concerning installation of XML schemas
Hints concerning the installation procedure	To avoid problems concerning de-installation of FDT related software components

Furthermore a frame application is responsible to install all FDT related XML schemas. A DTM has to use these documents provided by the frame application via IFdtContainer::GetXMLSchemaPath(). DTM specific schemas within FDT related XML documents must be declared as an inline definition.During the de-installation of FDT related components, the procedure has to take care about the availability of the FDT related interface description. To avoid problems it is recommended, to use the following strategy:

- During installation of software components (DTM, frame application), the FDT100.dll, which is packaged with this document, should be installed
- The installation procedure should install the FDT100.dll as a shared component into the system folder of the current windows installation (e.g. for Windows NT4.0 system32) and it is preferred to use the provided merge module (MSM file)
- Due to COM rules, it is only allowed, to install newer version of this component

Furthermore it is highly recommended not to include the FDT specific Interface description (IDL) within own components.

2.26 Chapter 7.2, Data Type Definitions

Change	Reason
Comment concerning data type FdtUUIDString	Clarify the handling concerning case sensitive
Comment concerning data type FdtXPath	Correction concerning the usage within the context of IFdtChannel::GetChannelPath()

Name	Data type	Description
FdtUUIDString	BSTR	String containing a unique identifier according to the Microsoft standard UUID. The format must be e.g. “C2137DD1-7842-11d4-A3C9-005004DC410F” (without bracket). Due to the definition of the UUID format, the value must NOT be handled in a case sensitive way. That means comparing “C2137dd1-7842-11d4-A3C9-005004DC410F” with “C2137DD1-7842-11d4-A3C9-005004DC410f” will return TRUE
FdtXPath	BSTR	String containing a XPath to an element of a XML document **For FDT 1.2 the XPath has to be the root tag “FDT”** • Data exchange via complete documents until the specification of data fragment interchange is released by the W3C • XML documents are accepted as complete document. Otherwise the DTM informs the frame-application via the event interface about the occurred of errors **Exceptions:** • IFdtChannel::GetChannelPath(), refer to method description • Interface IFdtTopology, usage as channel path within the specified methods

2.27 Chapter 10, Appendix – FDT IDL

Change	Reason
Version of IDL-File changed to 1.2	Consistent versions of FDT specification and IDL file

2.28 Chapter 12, Appendix - Implementation Hints

Change	Reason
New Chapter 12.11	Clarification concerning the usage of threading models of ActiveX Controls
New Chapter 12.12	Clarification concerning COM threading issues in general
New Chapter 12.13	Within the method call of IFdtActiveX::OpenActiveXControlRequest(),

Change	Reason
	the container could not get the information concerning resizable user interfaces via the given parameters
New Chapter 12.14	Additional hints concerning creation of XML-Documents using string operations
New Chapter 12.15	Additional hint: Validation concerning online device connection
New Chapter 12.16	Additional hint concerning communication addressing of devices
New Chapter 12.17	Additional hint concerning handling of asynchronous function calls and invokeId

2.29 Chapter 12.11, Threading Model ActiveX Controls

For a proper behavior all ActiveX Controls must be implemented apartment-threaded. The default setting in Visual Basic Projects is ‘Apartment Threaded’. For a detailed description of facts and reasons see MSDN article ‘Q247845’.

2.30 Chapter 12.12, COM Threading issues

FDT applications are software systems consisting of several different components interacting and exchanging data with each other. FDT frame applications, communication/device DTMs and DTM GUIs are the FDT System fundamentals, all based on Microsoft’s COM/ActiveX technology. So the overall system architecture is defined by all involved components.

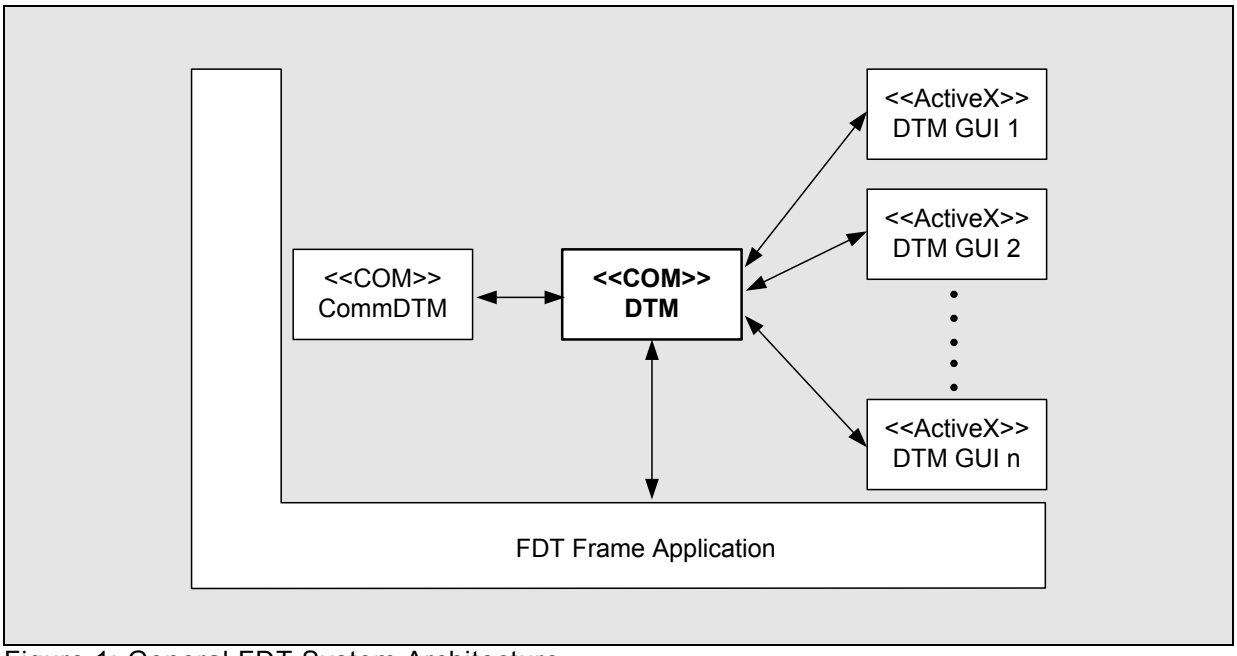


Figure 1: General FDT System Architecture

Depending to that, one or more *threads* are manual started by the frame application, DTMs or automatically by COM runtime.

A *thread* is the basic unit to which the operating system allocates processor time. A thread can execute any part of the process code, including parts currently being executed by another thread. For more details refer to MSDN-Library: Platform SDK: ‘DLLs, Processes, and Threads’ and ‘About Processes and Threads’.

Due to the behavior of the COM runtime environment, some synchronization issues should be taken into account.

2.30.1 Chapter 12.12.1, Threading: A Brief Overview

Microsoft’s COM/ActiveX Technology build-in synchronization is based on so called Apartments. COM Apartments are virtual rooms, which groups COM/ActiveX components of same synchronization type; COM/ActiveX components are always living in exactly one Apartment.

Standard COM knows two different types of Apartments, Single Threaded Apartments (STA) and Multi Threaded Apartments (MTA).
STAs only allow one Thread to enter the Apartment at once and therefore the access to the components, which live in.
MTAs are less restrictive; they allow several Threads to enter Apartment at once that mean all components living in a MTA have to take care about synchronization itself.

Where components are able to live is defined by an entry in Windows Registry. If component is created in invalid Apartment, then COM Runtime automatically creates valid Apartment and puts the new component in.

Components implemented with Visual Basic or ActiveX controls are always STA components. Only in Visual C++ it’s possible to develop MTA components.

For more detailed COM Apartment description please refer to MSDN articles “Q15077” and “Q136885”. For a closer look refer to MSDN-Library, Platform SDK: ‘Processes, Threads, and Apartments’

2.30.2 Chapter 12.12.2, Single Apartment constellation

In *Single Apartment constellation* all components life in the same and only STA, this is normally called ST0 (Single Threaded Apartment 0). In the most common cases components supporting different type of synchronization are running in a frame application. This point directs use to the next chapter *Multi Apartment constellations*.

2.30.3 Chapter 12.12.3, Multi Apartment constellation

In *Multi Apartment constellations* one or more components live in their own Apartment, this leads to more than one running thread. Due to that, synchronization issues have to be considered.

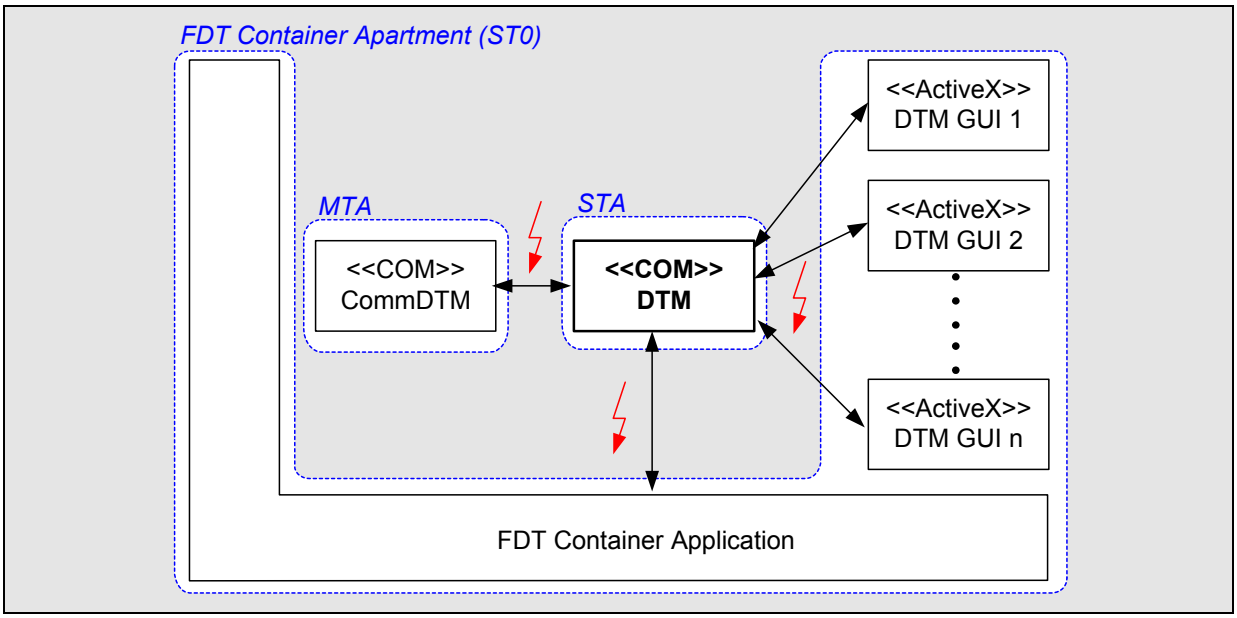


Figure 2: Multi Apartment constellation - DTMs living in own Apartment

Why do we need take care about synchronization within this constellation?

The answer is the exceptions where COM Runtime not fully synchronizes all calls. Within these exceptions, COM runtime allows other threads to enter apartment, while calls to a component across Apartment boundaries is active.

Two exceptions are *modal Dialogs* and Visual Basic *DoEvents()* operation. Both work with an own message loop, which dispatches pending calls to components living in a STA (refer to chapter 12.12.4, Summary)

Similar constellations could also appear by using of other interface methods.

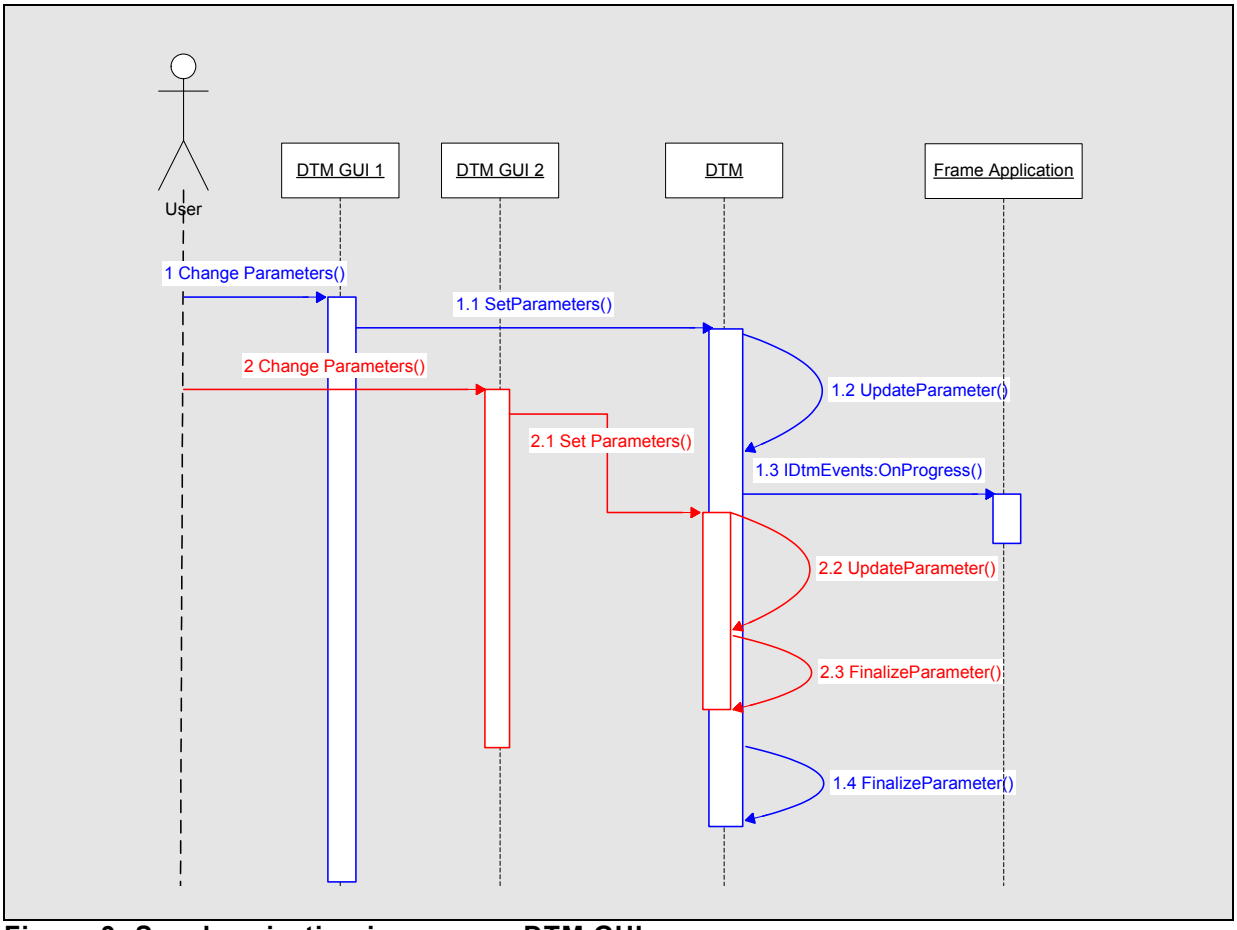


Figure 3: Synchronization issues e.g. DTM GUI

Sequence description:

Step 1: User changes some parameter in DTM GUI 1 and presses *Apply*

Step 1.1: DTM GUI 1 sends the new parameter values to the DTM

Step 1.2-1.3: DTM updates Parameter Model (sets new values and re-calculates dependencies) and fires progress events to inform user about state of processing

Step 2: Now user changes some parameter in DTM GUI 2 and presses also *Apply*

Step 2.1: DTM GUI 2 sends the new parameter values to the DTM; call is blocked until DTM calls IDtmEvents:OnProgress() method of frame application

Step 2.2-2.3: DTM updates again Parameter Model (Parameter model may not be completely valid), during this time first Parameter Model update is suspended.

Step 1.4: After second Parameter Model updated has finished, first one is waked up and finishes work as well.

2.30.4 Chapter 12.12.4, A closer look at message loops

Unlike MS-DOS-based applications, Windows-based applications are event driven. They do not make explicit function calls to obtain input. Instead, they wait for the system to pass input to them.

Each window has a function, called a window procedure that the system calls whenever it has input for the window. The window procedure processes the input and returns control to the

system. For more information about window procedures, refer to MSDN-Library: 'Window Procedure'.

The system uses a queue called *message queue* to route messages to a window procedure; this is done by a *message loop.*

The message loop retrieves messages from the message queue and dispatches them to the window procedure. While dispatching a message, the message loop is blocked. That is one way how COM runtime synchronize messages.

But there are exceptions where COM Runtime not fully synchronizes all calls. Every time, a piece of software executes a message loop on its own, it 'pumps' the message queue and pending messages (common windows messages *and* COM calls) will be dispatched.

An easy way to execute a message loop in Visual Basic is a call to the 'DoEvents' function. All topics concerning message loops discussed below refer also to Visual Basic's DoEvents function.

For more details about DoEvents, refer to MSDN-Library: Visual Basic Reference, 'DoEvents Function'.

Message Loops and FDT

In the context of FDT, pumping the message queue can result in serious problems for the involved components.

Example:

A ButtonClick-Event within an ActiveX-Control causes an IFdtCommunication::ConnectRequest() of a communication DTM. The communication DTM executes a message loop within this ConnectRequest() while it is waiting for communication callback from the device. In this case the user can close the ActiveX-Control or another button can be pressed before the original ConnectRequest() call returns.

> **Therefore it is not allowed to execute a message loop in an asynchronous request call and it's not recommended somewhere else if there are other solutions possible.**

In some cases a DTM needs to process asynchronous calls in synchronous manner. For example a call to IDtmDocumentation::GetDocumentation may require communication with a device to obtain online data before the function call returns.

Because communication in FDT is treated asynchronously, the DTM has to execute a message loop within GetDocumentation in order to wait for responses of the requested communication calls. While waiting for responses it's necessary to dispatch window messages because the called communication channel may live in the same apartment and possibly uses messaging mechanisms that rely on the windows message queue. Furthermore pumping the message queue ensures that all COM-messages are dispatched to the waiting thread.

A similar situation takes place if a DTM needs input from the user before the current function returns. Usually, this is done by means of a modal dialog that executes a message loop likewise.

In those scenarios the involved FDT components must frequently process calls that are dispatched by nested message loops. These calls have possibly top level semantic, e.g. user interactions.

Critical and non-critical message loops

If a nested message loop is executed in a direct sequence of a windows message dispatched by the main message loop (e.g. user input or timers), no further treatment must take place.

The term *direct* signifies that no other FDT component or apartment must appear in the call stack of the function that executes the nested message loop.

If the message loop is triggered within a call from another component (e.g. the host apartment of the frame) extra precautions must be met. These precautions can be divided into tasks for the DTM and tasks for the frame application.

Frame application's point of view

The frame application must be prepared a DTM executes a message loop in a called function that was intended to behave synchronously. In this case the frame may get asynchronous input from user interactions or other components. The frame must handle appearing nested calls in a proper way.

This can be done in various ways; one example is a convenient structure of the call sequence. Consider following situation: The frame wants to display a documentation print that contains online data of a device in a window. Within the IDtmDocumentation::GetDocumentation call the DTM executes a message loop to obtain up-to-date data from the device. If the frame opens the window first and then calls GetDocumentation, it's possible that the user closes the window before the GetDocumentation call returns. This may result in unpredictable behaviour if the frame tries to put the received data in a window that no longer exists. In this case it seems better to retrieve the data first and to open the window second.

If the frame is not able to handle all possible cases properly, it may be helpful to detect the presence of a nested message loop and to lock the user interface or other asynchronous entry points on demand.

A frame application can detect nested message loops by posting a private WM_USER message to one of its windows. If this message is dispatched during processing of an outgoing call to a DTM, a message loop (maybe a modal dialog) is executed in the called DTM function. The frame application may now disable it windows to prevent nested user interactions if it cannot process them properly in any case.

Another solution is to create DTMs and DTM-ActiveX controls in apartments separated from the frame. Although this strategy seems to be a proper solution for the discussed issues, there are several disadvantages:

- Performance may degrade heavily
- Placing ActiveX controls in different apartments results in serious problems handling focus changes and window activation.
- If the nested DTM-message loop is executed in a separate apartment, no processing of window messages will take place in the apartment of the caller. That means that apparent simple tasks like redrawing will not work and the apartment of the caller will not respond to user interactions anyway. If the nested message loop needs some seconds (possibly longer if communication is going on) it may be annoying for the user if the frame is totally not responding.

DTM's point of view

If a DTM executes a message loop (or displays modal dialogs), it is responsible for a proper handling of following situations:

- Possibly nested calls from its own user interface controls
- Possibly nested calls from the frame application (e.g. SetParameters or PrepareToReleaseCommunication)
- Possibly responses from communication channels

- Possibly asynchronous input from private communication components or self-activating events like timers

If a DTM calls IFdtDialog::UserDialog or IDtmEvents::OnErrorMessage, it has to deal with following circumstances:

- Possibly responses from communication channels
- Possibly asynchronous input from communication components or self-activating events like timers

Common solutions for the mentioned problems could be:

- Disable user interface controls before starting the message loop
- Operating on separate copies of the data set
- Queuing of incoming asynchronous calls (e.g. communication responses, see implementation hints below)
- Proper state machine implementation

If a DTM opens private dialogs, it is responsible for disabling its user interfaces. It cannot rely on standard windows behaviour, because the ActiveX controls could run in different top level windows. This is of particularly importance if a private dialog is opened through the DTM business object.

Topics concerning cross-DTM calls:

- DTMs that provide communication capabilities via Communication-Channels must be prepared to get asynchronous calls from within a response call. This situation arises if the called DTM displays a message box in the response function.
- The DTM is not allowed to execute message loops in asynchronous request calls (e.g. IFdtCommunication::TransactionRequest).

2.30.5 Chapter 12.12.5, Summary

The mentioned issues are based on the COM runtime behavior in combination with a multi apartment constellation of different components. Therefore the developer of a frame application or DTM has to take some implementation hints into account:

Performance issues:

Method calls within threads are faster than calls across a thread boundary. E.g. calls from a DTM GUI into the DTM business object may cause every time packaging and sending interface method parameters across thread or process border. For more information refer to MSDN-Library, Platform SDK: ‘Single-Threaded and Multithreaded Communication’.

To solve performance issues, it is helpful to take a closer look to Microsoft’s Distributed interNet Application (DNA) architecture. Even if FDT does not use DNA, there are valuable hints concerning component based software design.

- Some hints concerning component-based applications refer to MSDN-Library:

  ‘Top Windows DNA Performance Mistakes and How to Prevent Them’

Gary Geiger and Jon Pulsipher

and

'Introduction to Designing and Building Windows DNA Applications'

Frank E. Redmond III

See also Microsoft Knowledge Base Article - Q129886.

Using message loops and modal dialogs:

Developers using these functions must take care about synchronization issues described above. A brief summary:

A DTM executing a message loop must be able to handle

- nested calls from its own user interface controls
- nested calls from the frame application
- responses from a communication channel
- lock its own ActiveX controls in case it displays a private modal dialog

Tasks of the frame application:

- A DTM possibly executes message loops in functions called by the frame. Therefore the frame may get asynchronous input from user interactions or other components within those calls. The frame must handle those calls with respect to its integrity and to the DTM state machine.
-
- If the frame application displays modal dialogs within functions called by the DTM it has to ensure a modal behavior (no additional user interactions) for the user

Tasks of the Communication/Gateway DTM:

- The Communication/Gateway DTM must be prepared a DTM executes a message loop in a called response function. The DTM may get asynchronous input from other components within those response calls.

In general, it is not allowed to execute message loops in request calls that are intended to behave asynchronously (e.g. calls to IFdtCommunication)

Calls across thread/apartment boundaries:

Another type of synchronization issues occurs, if calls will be made across thread / apartment boundaries.

E.g. a frame Application may choose architectures within every component lives in an own thread/apartment for safety reason; accidentally shutdowns of DTM GUI controls won't crash the frame application process.
In addition, same constellation exists if DTM GUIs are stand-alone applications instead of ActiveX controls.

For some reasons it could happen now, that two DTM GUIs are accessing the DTM at once, what again could lead to synchronization issues.

Therefore at least 2 different topics should be considered:

- Calls between frame application and DTM (e.g. via SetParameters()) or DTM business Object and DTM user interface

Within this constellation it is recommended for a robust behaviour that, the frame application should take care about the synchronization between the different components. One proposed strategy could be based on Microsoft's Message Filter concept (refer to chapter 12.12.5). However this is not a complete solution for all possible scenarios. That means that DTMs still have to take care about the topics concerning message loops, described above.

- Calls between different DTMs (e.g. concerning communication purposes)

  Here a DTM should be able to handle not synchronized calls. A suitable solution could be a serialization queue implementation (refer to chapter 12.12.6)

  ○

2.30.6 Chapter 12.12.5, Implemetation Hint: Message Filter

For synchronization issues between DTM business object and calls coming from DTM user interface or frame application, Microsoft's MessageFilter implementation in DTM apartment can solve the problem.

A MessageFilter is a piece of code, which locks out calls to components in an Apartment, while another call has already entered it. Different to standard STA behavior this also works while calls across Apartment boundaries are active but not if modal dialog is opened or a message loop is executed.

The MessageFilter blocks calls coming from STA components as long as another call has entered Apartment, whereas calls coming from MTA components are rejected (call returns with error code). Thus MessageFilter is only suitable solution for DTM GUI ActiveX control problem, because those are always STA components.

Additionally, DTMs calling modal dialogs or executing message loops have still to be prepared to receive other calls from DTM user interface or frame application even if MessageFilter is active.

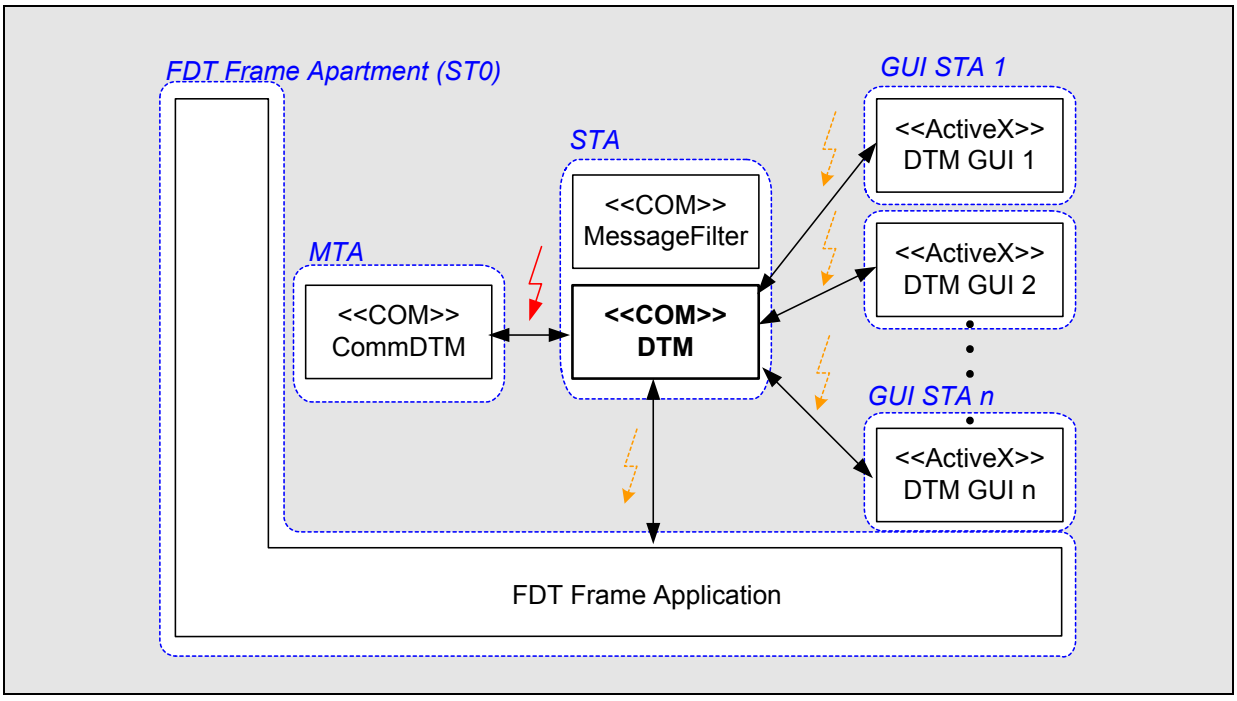


Figure 4: DTM Apartment with MessageFilter

The frame application is responsible for creation of DTM ActiveX controls and knows Apartment Threads IDs. Therefore it is also responsible for registration of the MessageFilter in DTM Apartment. The MessageFilter implementation should only become active for calls coming from a DTM GUI or frame application Thread.

Following VC++ source code examples demonstrate Message Filter registration in DTM Apartment and implementation of IMessageFilter interface. For more detailed explanation please refer to MSDN.

MessageFilter registration in DTM Apartment Thread:

```
BOOL CDTMApartmentThread::InitInstance()
{
	…

	// create the Apartment Message Filter
	CComObject<CMsgFilter> *pMsgFilter = NULL;
	CComObject<CMsgFilter>::CreateInstance(&pMsgFilter);

	// register Message Filter
	CComPtr<IMessageFilter> oPtrMessageFilter((IMessageFilter*)pMsgFilter);
	VERIFY(SUCCEEDED( CoRegisterMessageFilter(oPtrMessageFilter, NULL) ));

	…
}
```

IMessageFilter interface implementation:

```
DWORD CMsgFilter::HandleInComingCall(DWORD dwCallType, HTASK threadIDCaller,
DWORD dwTickCount, LPINTERFACEINFO lpInterfaceInfo)
{
	HTASK hKnownThreadID;

	// loop through ID list of all known Threads
	for (long i=0; i < m_oKnownThreadIDs.size(); i++)
	{
		hKnownThreadID = (HTASK)m_oKnownThreadIDs[i];

		// block call if coming from known Thread and
		// already another call has already entered the DTM Apartment
		if ( hKnownThreadID==threadIDCaller &&
		     dwCallType==CALLTYPE_TOPLEVEL_CALLPENDING )
		{
		      return SERVERCALL_RETRYLATER;
		}
	}

	return SERVERCALL_ISHANDLED;
}

DWORD CMsgFilter::MessagePending(HTASK threadIDCaller, DWORD dwTickCount, DWORD
dwPendingType)
{
	// implement default behavior
	return PENDINGMSG_WAITDEFPROCESS;
}

DWORD CMsgFilter::RetryRejectedCall(HTASK threadIDCaller, DWORD dwTickCount,
DWORD dwRejectType)
{
	// let server call wait for another 3 seconds
	return 3000;
}
```


2.30.7 Chapter 12.12.6, Implementation Hint: Serialisation Queue

Following Visual Basic source code example demonstrates queuing of communication responses in a DTM.

The queue is implemented in class “clsCommQueue”. The queued responses are stored in objects of class “clsQueuedCommCall”. The queue can be locked and unlocked. If the queue is locked, incoming calls are stored and executed when the last lock is released. The helper class “clsQueueLock” provides exception-safe handling of lock/unlock calls.

The implementation of IDtmParameter and IFdtCommunicationEvents in the DTM main object class “clsDTM” demonstrates the usage of these classes. clsDTM holds the communication object “m_CommunicationObj” which is supposed to process communication responses. The queue object “m_CommQueue” is used to queue the calls. Note that the queue is locked during processing of calls to IDtmParameter::GetParameters and IDtmParameter::SetParameters. This prevent communication events from interfering the access to the data set.

```
'------------------------------------------------------------------------------
' Class clsCommQueue
'------------------------------------------------------------------------------
Option Explicit

Private m_nLocks As Long
Private m_QueuedCalls As Collection
Private m_DtmEvents As Fdt100.IDtmEvents
Private m_SystemTag As String

Public Sub Init(ByVal DtmEvents As Fdt100.IDtmEvents, _
                ByVal SystemTag As String)
    ' initialize queue
    Set m_QueuedCalls = New Collection
    Set m_DtmEvents = DtmEvents
    m_SystemTag = SystemTag
End Sub

' code for queuing of response calls ----------------------------------------------
Public Sub LockQueue()
    ' add lock
    m_nLocks = m_nLocks + 1
End Sub

Public Sub UnlockQueue()
    ' release lock
    Dim aCall As clsQueuedCommCall
    If (m_nLocks > 0) Then
        If (m_nLocks = 1) Then

            ' last lock, process queued calls before releasing the last lock
            ' if calls arrive during processing, these calls are still queued
            Do While (m_QueuedCalls.Count > 0)
                ' retrieve earliest call
                Set aCall = m_QueuedCalls.Item(1)
                Call m_QueuedCalls.Remove(1)
                ' execute call
                Call aCall.ExecuteCall(m_DtmEvents, m_SystemTag)
            Loop

        End If
        m_nLocks = m_nLocks - 1
    End If
End Sub

Public Function CreateLock() As clsQueueLock
    ' Creates a lock object and locks the queue. The lock is
    ' automatically released when the lock object is destroyed
    Dim ql As New clsQueueLock
    Call ql.LockQueue(Me)
    Set CreateLock = ql
End Function

Public Sub QueueCall(ByVal CommObj As IFdtCommunicationEvents, _
```



```
                    ByVal CommCall As QueuedCommCall, _
                    UUID As Fdt100.FdtUUIDString, _
                    Optional Response As Fdt100.FdtUUIDString = "")
    ' Queues a call of type 'QueuedCommCall'. The call
    ' is executed at once if the queue is not locked.
    Dim aCall As New clsQueuedCommCall

    ' put call into queue
    Set aCall.CommObj = CommObj
    aCall.CommCall = CommCall
    aCall.UUID = UUID
    aCall.Response = Response
    Call m_QueuedCalls.Add(aCall)

    If (m_nLocks = 0) Then
        ' queue currently not locked: lock queue to
        ' ensure that nested calls are properly queued
        Call LockQueue
        ' process queued calls
        Call UnlockQueue
    End If
End Sub


'-------------------------------------------------------------------------------
' Class clsQueuedCommCall
'-------------------------------------------------------------------------------
Option Explicit

Public Enum QueuedCommCall
    qccOnAbort
    qccOnConnectResponse
    qccOnDisconnectResponse
    qccOnTransactionResponse
End Enum

Public CommObj As Fdt100.IFdtCommunicationEvents
Public CommCall As QueuedCommCall
Public UUID As Fdt100.FdtUUIDString
Public Response As Fdt100.FdtUUIDString

Public Sub ExecuteCall(ByVal DtmEvents As Fdt100.IDtmEvents, _
                       ByVal SystemTag As String)
    On Error GoTo CallErr

    Select Case CommCall
        Case qccOnAbort
            Call CommObj.OnAbort(UUID)
        Case qccOnConnectResponse
            Call CommObj.OnConnectResponse(UUID, Response)
        Case qccOnDisconnectResponse
            Call CommObj.OnDisconnectResponse(UUID, Response)
        Case qccOnTransactionResponse
            Call CommObj.OnTransactionResponse(UUID, Response)
    End Select
CallExit:
    Exit Sub

CallErr:
    Call DtmEvents.OnErrorMessage(SystemTag, "Error #" & Err.Number & " at " & _
                                  Err.Source & vbNewLine & Err.Description)
    Resume CallExit
End Sub


'-------------------------------------------------------------------------------
' Class clsQueueLock
'-------------------------------------------------------------------------------
Option Explicit

Private m_CommQueue As clsCommQueue

Private Sub Class_Terminate()
    Call UnlockQueue
End Sub

Public Sub LockQueue(CommQueue As clsCommQueue)
    If m_CommQueue Is Nothing Then
```



```
        Set m_CommQueue = CommQueue
        Call m_CommQueue.LockQueue
    End If
End Sub

Public Sub UnlockQueue()
    If Not m_CommQueue Is Nothing Then
        Call m_CommQueue.UnlockQueue
        Set m_CommQueue = Nothing
    End If
End Sub


'-------------------------------------------------------------------------------
' Class clsDTM: portion of a DTM main object
'-------------------------------------------------------------------------------
Option Explicit

Implements Fdt100.IFdtCommunicationEvents
Implements Fdt100.IDtmParameter

Private m_CommQueue As clsCommQueue
Private m_CommunicationObj As Fdt100.IFdtCommunicationEvents

' IDtm and other code
'.
'.
'.

' IFdtCommunicationEvents  ----------------------------------------------------
Private Sub IFdtCommunicationEvents_OnAbort( _
            ByVal communicationReference As Fdt100.FdtUUIDString)
    Call m_CommQueue.QueueCall(m_CommunicationObj, qccOnAbort, _
                               communicationReference)
End Sub

Private Sub IFdtCommunicationEvents_OnConnectResponse( _
            ByVal invokeId As Fdt100.FdtUUIDString, _
            ByVal Response As Fdt100.FdtXmlDocument)
    Call m_CommQueue.QueueCall(m_CommunicationObj, qccOnConnectResponse, _
                               invokeId, Response)
End Sub

Private Sub IFdtCommunicationEvents_OnDisconnectResponse( _
            ByVal invokeId As Fdt100.FdtUUIDString, _
            ByVal Response As Fdt100.FdtXmlDocument)
    Call m_CommQueue.QueueCall(m_CommunicationObj, qccOnDisconnectResponse, _
                               invokeId, Response)
End Sub

Private Sub IFdtCommunicationEvents_OnTransactionResponse( _
            ByVal invokeId As Fdt100.FdtUUIDString, _
            ByVal Response As Fdt100.FdtXmlDocument)
    Call m_CommQueue.QueueCall(m_CommunicationObj, qccOnTransactionResponse, _
                               invokeId, Response)
End Sub


' IDtmParameter  --------------------------------------------------------------
Private Function IDtmParameter_GetParameters( _
            ByVal parameterPath As Fdt100.FdtXPath) As Fdt100.FdtXmlDocument
    Dim CommLock As clsQueueLock
    ' queue asynchronous responses during processing of parameter document
    ' the lock object is automatically destroyed when leaving this function
    Set CommLock = m_CommQueue.CreateLock
    ' get parameters
    ' ...
End Function

Private Function IDtmParameter_SetParameters( _
            ByVal parameterPath As Fdt100.FdtXPath, _
            ByVal FdtXmlDocument As Fdt100.FdtXmlDocument) As Boolean
    Dim CommLock As clsQueueLock
    ' queue asynchronous responses during processing of parameter document
    ' the lock object is automatically destroyed when leaving this function
    Set CommLock = m_CommQueue.CreateLock
    ' set parameters
    ' ...
```



```
End Function
```

2.31 Chapter 12.13, Resizable User Interfaces

Within the context of the method call IFdtActiveX::OpenActiveXControlRequest(), a container could determine if a control supports a resizable user interface by following strategy:

1) Try to resize the control by setting the properties for height and width to a slightly higher value
2) Request the controls height and width
3) Compare the values with the values, which where set by the container
4) If the values are equal, the control supports a resizable user interface

2.32 Chapter 12.14, Creating XML-Documents using String Operations

Typically, XML documents should be created based on the W3C's 'Document Object Model' (DOM, refer also to FDT-Specification, Chapter '2.6 Parameter interchange via XML').

Developments, which are based on string operations like concatenation to create XML-Documents, must at least a special handling concerning the following items into account:

- White Space
  - Horizontal tab
  - Line-feed
  - Carriage-return
  - ASCII space character
- End-of-line Handling
  - Carriage-return Line-feed
  - Line-feed only
  - Carriage-return only
- Character References
  - Typically used as a substitute for a literal form (e.g.'&#169' to display '©')
- Entity References
  - Allow the insertion of any string literal into e.g. attribute values. If a string should look like 'ABC&D', the string created with string operations must look like 'ABC&D'

For complete definition and additional information refer to (check also for latest version):

Extensible Markup Language
W3C Recommendation 6 October 2000
http://www.w3.org/TR/2000/REC-xml-20001006

2.33 Chapter 12.15, Validation concerning online device connection

After an online connection to a device is established, a DTM should check, weather it could handle this device in a proper way (e.g. by checking the device type via communication protocol specific data).

2.34 Chapter 12.16, Communication addressing of devices

As described in chapter ‘5.2 Nested Communication’ the parent component is responsible to manage the communication addressing. Therefore, the following topics should be taken into account:

- To hand over the address information, IDtmParameter::SetParameters() must be used (refer to 3.3.11.2 SetParameters)
- Concerning the HART-Protocol, the attributes ‘tag’ and ‘slaveAddress’, concerning PROFIBUS, the attribute ‘busAddress’ is used to hand over the address information.
- Device DTMs should not manage the address information

2.35 Chapter 12.17, Handling of asynchronous function calls and invokeId

The invokeId used by several functions is generated by components and returned to the caller via a callback function as a way to identify the returned data. Depending on the mix of client and server threading models used, it has been found in practice that the callback can occur before the called method returns to the caller.

Thus, if the caller wants to manage e.g. the request concerning a communication transaction in some list of 'outstanding transactions' in order to verify completion of a transaction it will need to manage the invokeId BEFORE making the method call.

This hint is similar to a note found in "OPC Data Access Custom Interface Standard Version 2.05" chapter 4.5.6

2.36 Chapter 13.1, FDT Data Types

Change	Reason
Enhanced description concerning attribute ‘displayFormat’	The format of this attribute was not proper defined
Element ‘CommunicationTypeEntry’ within the element ‘BusCategory’ Optional attribute ‘communicationType’ within element ‘CommunicationTypeEntry’	By means of the information of the Windows-Registry and IDtm:GetInformation() it can not differentiated between the bus protocols a DTM requires or supported
Optional attribute ‘busCategoryName’ within element ‘BusCategory’	Vendor specific protocols, defined by a DTM can not human readable displayed by a frame-application
New enum ‘notSupportedFeature’ within attribute type communicationError	Returned as communication error, if the extended address format according to the PROFIBUS specification is not supported
Optional attribute ‘label’ within element ‘DtmVariable’ and ‘DtmVariables’	Separation of language independent and language neutral identification of the Variables

Attribute	Description

Attribute	Description
busCategoryName	Human readable string for the bus type. Depending on the bus type the following strings are mandatory: DPV0 -> 'Profibus DP/V0' DPV1 -> 'Profibus DP/V1' HART -> 'HART' Others (e.g. vendor specific protocols) -> DTM specific
displayFormat	Describes the display format for a display attribute (e.g. "%3.2f " for a float value). The format string follows the rules of C, specified in IEEE Std1003.2-1992
communicationType	Indicates the type of the bus category: 'supported': Bus protocol which can be provided by the DTM at a channel. Notice: The actual supported bus protocols depend on the configuration of the DTM. This attribute shows the possible supported, not the actual available protocols. 'required': Bus protocol which will be expected by the DTM from the parent DTM
Label	Human readable name. If the optional attribute 'label' exists the attribute 'name' contains a language independent identifier

Tag	Description
CommunicationTypeEntry	Enumeration element for the communication type

The today's optional attributes 'communicationType' and 'busCategoryName' will be mandatory in FDT 2.0. Further on, the information within the Windows-Registry regarding to communication will be obsolete in FDT 2.0, too. Therefore it is strongly recommended that frame-applications evaluate the new information from IDtm:GetInformation() from now on (e.g. for generation of a product catalog).

Also the today's optional attribute 'label' will be mandatory in FDT 2.0. It is strongly recommended that DTMs and frame-applications support the attribute 'label' from now on.

Please notice anymore if a DTM provide a XML document with the optional tag 'nodeId' (e.g. on IDtmParameter::GetParameters()) this attribute must be properly used within the document for setting changes (e.g. on IDtmParameter::SetParameters())

2.37 Chapter 13.12, Functions of a DTM

Change	Reason
Usage of attribute 'separator'	Enhanced description
Usage of attribute 'toggle'	Clarify usage
Usage of element 'StandardFunction'	Enhanced description
Behaviour if element 'Status' not available	Enhanced description
Optional attribute 'printableStandardFunction' within element 'StandardFunction'	Clarify usage
Optional attribute 'resizableStandardFunction' within	Clarify usage

Change	Reason
element ‘StandardFunction’	

Attribute	Description
separator	Special menu entry. If this attribute is set to TRUE, the entry acts as a separator. So there is a need to define an extra ‘Function’ element to create a separator entry.
toggle	Status of a menu entry whether it is active or not. This attribute is obsolete

Tag	Description
StandardFunction	Definition of a single function entry, which is a standard function (concerning the applicationIds). This entry has not an own label. The label must be supplied by the frame application. Standard functions are intended to be functions with user interface.
Status	Description of a menu. This element is used in an optional manner. If the status element is not available, the default behavior of the related function entry is defined as ‘checked’ = FALSE ‘enabled’ = TRUE ‘hidden’ = FALSE ‘separator’ = FALSE

2.38 Chapter 13.13, Functions of a DTM Channel Object

Refer to: 2.37 Chapter 13.12, Functions of a DTM

2.39 Chapter 13.21 Basic Channel Schema

Change	Reason
Additional schema which is used to access a channel without process data	Clarify usage of channels without process data

Used at: IFdtChannel::GetChannelParameters()

IFdtChannel::SetChannelParameters()

The XML document describes how to access a channel without process data.

Attribute	Description
gatewayBusCategory	Unique identifier for a supported bus type like Profibus

Attribute	Description
	or HART according to the FDT specific CATID

Tag	Description
FDT	Root tag
FDTBasicChannel	Description of the channel
FDTChannelType	Description of the channel component in case of channels with gateway functionality

2.40 Chapter 15.1.1, DPV0 Communication

Change	Reason
Optional attribute 'communicationReference' within element 'Abort'	Without the proper communication reference, the communication component could not determine the connection to abort. **Even if this attribute is optional, it is highly recommended, to fill in the proper communication reference.**

2.41 Chapter 15.1.2, DPV1 Communication

Change	Reason
Optional attribute 'communicationReference' within element 'Abort'	Without the proper communication reference, the communication component could not determine the connection to abort. **Even if this attribute is an optional value, it is highly recommended, to fill in the proper communication reference**
Optional attribute 'maxLenDataUnit' within the element 'ConnectResponse'	Due to the fact, that not all communication components within the chain of nested communication may not provide the full length of 244 byte for data transport, a slave should be able to get this information in advance.

Attribute	Description
MaxLenDataUnit	Optional attribute, to describe the amount of data, which can be transferred via the established connection • If this attribute is not available, no special length restriction is announced • Each communication component within the chain of interfaces concerning nested communication could introduce this attribute • Each communication component should change the contents of the attribute based on the following rule: The new value is the minimum of the current value and the restriction of its own implementation • If a communication component has no restriction, it should hand over the given value • If a communication component is able to reuse an established connection concerning a new connect request, it should take into account the data length determined for the existing connection If the data length is not applicable, the DTM should send an error message via OnErrorMessage()
scl	Access level as described in the PROFIBUS specification. *1)
networkMACAddress	Network or MAC address as described in the PROFIBUS specification. *1) If there is a networkMACAddress, (i.e. networkMACAddress <> ""), the Profibus-DPV1-Master-DTM must set the corresponding “address type” in the Initiate request telegram “1”, else “0”. Example: Device with address 20 (14hex) addressed via a linking device: networkMACAddress=“00000000000014”

Additional or modified Elements:

Tag	Description
NetworkAddress	Describes the extended address format
srcNetworkAddress	Describes the extended address of the source *1)
destNetworkAddress	Describes the extended address of the destination *1)
ConnectRequest	Optional elements "srcNetworkAddress" and "destNetworkAddress"
ConnectResponse	Optional elements "srcNetworkAddress" and "destNetworkAddress" Optional attribute "maxLenDataUnit"

*1) PROFIBUS Guideline (Order No. 2.082): Chapter “Services on communication relationships MSAC_C2”: MSAC2_Initiate

2.42 Chapter 16.1, Communication Schema

Change	Reason
Optional attribute 'communicationReference' within element 'Abort'	Without the proper communication reference, the communication component could not determine the connection to abort. **Even if this attribute is an optional value, it is highly recommended, to fill in the proper communication reference.**

2.43 Chapter 16.3, Topology Scan Schema

The today's optional attributes 'fdthart:manufacturerId', 'fdthart:deviceTypId' and 'fdthart:shortAddress' will be mandatory in FDT 2.0. With the information supported by these attributes an assignment from the scanned field-devices to the according DTMs are possible. Therefore it is strongly recommended to support these attribute from now on.

2.44 Appendix - Adapted FDT XML Schema

13.1 FDT Data Types

```
<Schema name="FDTDataTypesSchema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <!--Definition of Attributes-->
    <AttributeType name="alarmType" dt:type="enumeration" dt:values="highHighAlarm highAlarm
lowLowAlarm lowAlarm"/>
    <AttributeType name="binData" dt:type="bin.hex"/>
    <AttributeType name="bitLength" dt:type="ui4"/>
    <AttributeType name="byteArray" dt:type="bin.hex"/>
    <AttributeType name="byteLength" dt:type="ui4"/>
    <AttributeType name="classificationId" dt:type="enumeration" dt:values="flow level pressure temperature
valve positioner actuator nc_rc encoder speedDrive hmi analogInput analogOutput digitalInput digitalOutput
electrochemicalAnalyser dtmSpecific"/>
    <AttributeType name="communicationError" dt:type="enumeration" dt:values="abort busy
invalidCommunicationReference noConnection noParallelServices noPendingRequest unknownError timeout
dtmSpecific notSupportedFeature"/>
    <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="label" dt:type="string"/>

    <!--Definition of Elements-->

    <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="path" required="yes"/>
</ElementType>
<ElementType name="SubstituteType" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="substituteType" required="yes"/>
</ElementType>
<ElementType name="LanguageId" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="languageId" required="yes"/>
</ElementType>
<ElementType name="SupportedLanguages" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="LanguageId" minOccurs="1" maxOccurs="*"/>
</ElementType>
<!-- Definition of Buscategories -->
<ElementType name="BusCategory" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="busCategory" required="yes"/>
    <attribute type="busCategoryName" required="no"/>
    <element type="CommunicationTypeEntry" minOccurs="0" maxOccurs="*"/>
</ElementType>
<ElementType name="BusCategories" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="BusCategory" minOccurs="1" maxOccurs="*"/>
</ElementType>
<ElementType name="DtmDeviceType" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="readAccess" required="no"/>
    <attribute type="writeAccess" required="no"/>
    <attribute type="manufacturerId" required="no"/>
    <attribute type="deviceTypeId" required="no"/>
    <attribute type="subDeviceType" required="no"/>
    <attribute type="deviceTypeInformation" required="no"/>
    <attribute type="classificationId" required="no"/>
    <element type="VersionInformation" minOccurs="1" maxOccurs="1"/>
    <element type="SupportedLanguages" minOccurs="1" maxOccurs="1"/>
    <element type="BusCategories" minOccurs="0" maxOccurs="1"/>
    <element type="DeviceIcon" minOccurs="0" maxOccurs="1"/>
</ElementType>
<!-- Definition of Version Information -->
<ElementType name="VersionInformation" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="readAccess" required="no"/>
    <attribute type="writeAccess" required="no"/>
    <attribute type="name" required="yes"/>
    <attribute type="vendor" required="no"/>
    <attribute type="version" required="no"/>
    <attribute type="date" required="no"/>
    <attribute type="descriptor" required="no"/>
</ElementType>
<ElementType name="NumberData" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="number" required="yes"/>
</ElementType>
<ElementType name="StringData" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="string" required="yes"/>
</ElementType>
<ElementType name="TimeData" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="time" required="yes"/>
</ElementType>
<!-- Definition of Binary Variable -->
<ElementType name="BinaryVariable" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="binData" required="yes"/>
</ElementType>
<!-- Definition of Enumerator Variable -->
<ElementType name="EnumeratorEntry" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="index" required="yes"/>
    <attribute type="name" required="yes"/>
```



```
    <attribute type="descriptor" required="no"/>
</ElementType>
<ElementType name="Variable" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="EnumeratorEntry" minOccurs="1" maxOccurs="1"/>
</ElementType>
<ElementType name="EnumeratorEntries" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="EnumeratorEntry" minOccurs="1" maxOccurs="*"/>
</ElementType>
<ElementType name="BitEnumeratorEntries" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="EnumeratorEntry" minOccurs="1" maxOccurs="*"/>
</ElementType>
<ElementType name="EnumeratorVariable" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="Variable" minOccurs="1" maxOccurs="1"/>
    <element type="EnumeratorEntries" minOccurs="0" maxOccurs="1"/>
</ElementType>
<!-- Definition of Bit Enumerator Variable -->
<ElementType name="BitVariable" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="EnumeratorEntry" minOccurs="0" maxOccurs="*"/>
</ElementType>
<ElementType name="BitEnumeratorVariable" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="BitVariable" minOccurs="1" maxOccurs="1"/>
    <element type="BitEnumeratorEntries" minOccurs="0" maxOccurs="1"/>
</ElementType>
<!-- Definition of Unit -->
<ElementType name="Unit" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="readAccess" required="no"/>
    <attribute type="writeAccess" required="no"/>
    <group order="one" minOccurs="0" maxOccurs="1">
        <element type="EnumeratorVariable"/>
        <element type="ChannelReference"/>
    </group>
</ElementType>
<!-- Definition of LowerRange -->
<ElementType name="LowerRange" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <group order="one" minOccurs="0" maxOccurs="1">
        <element type="NumberData"/>
        <element type="ChannelReference"/>
    </group>
</ElementType>
<!-- Definition of UpperRange -->
<ElementType name="UpperRange" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <group order="one" minOccurs="0" maxOccurs="1">
        <element type="NumberData"/>
        <element type="ChannelReference"/>
    </group>
</ElementType>
<!-- Definition of Ranges -->
<ElementType name="Range" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="LowerRange" minOccurs="1" maxOccurs="1"/>
    <element type="UpperRange" minOccurs="1" maxOccurs="1"/>
    <element type="Unit" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="Ranges" content="eltOnly" model="closed">
    <attribute type="readAccess" required="no"/>
    <attribute type="writeAccess" required="no"/>
    <attribute type="nodeId" required="no"/>
    <element type="Range" minOccurs="1" maxOccurs="*"/>
</ElementType>
<!-- Definition of Channel References -->
<ElementType name="ChannelReference" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="idref" required="yes"/>
```



```
</ElementType>
<ElementType name="ChannelReferences" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="ChannelReference" minOccurs="1" maxOccurs="*"/>
</ElementType>
<!-- Definition of Alarms -->
<ElementType name="StaticValue" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="staticValue" required="yes"/>
</ElementType>
<ElementType name="Alarm" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="alarmType" required="yes"/>
    <element type="Unit" minOccurs="0" maxOccurs="1"/>
    <group order="one" minOccurs="0" maxOccurs="1">
        <element type="StaticValue"/>
        <element type="ChannelReferences"/>
    </group>
</ElementType>
<ElementType name="Alarms" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="Alarm" minOccurs="1" maxOccurs="*"/>
</ElementType>
<!-- DtmVariable -->
<ElementType name="Display" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="string" required="yes"/>
</ElementType>
<ElementType name="DtmVariableReference" content="empty" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="reference" required="yes"/>
</ElementType>
<!-- StructuredVariable -->
<ElementType name="StructuredElement" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="bitLength" required="yes"/>
    <group order="one" minOccurs="1" maxOccurs="1">
        <element type="Display"/>
        <element type="DtmVariableReference"/>
    </group>
</ElementType>
<ElementType name="StructuredElements" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="StructuredElement" minOccurs="1" maxOccurs="*"/>
</ElementType>
<ElementType name="StructuredVariable" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <element type="BinaryVariable" minOccurs="1" maxOccurs="1"/>
    <element type="StructuredElements" minOccurs="1" maxOccurs="1"/>
    <attribute type="dataTypeDescriptor" required="no"/>
</ElementType>
<ElementType name="Variant" content="eltOnly" model="closed">
    <attribute type="nodeId" required="no"/>
    <attribute type="dataType" required="yes"/>
    <attribute type="byteLength" required="no"/>
    <attribute type="displayFormat" required="no"/>
    <group order="one" minOccurs="1" maxOccurs="1">
        <element type="StringData"/>
        <!-- for types: ascii, packedAscii, password, bitString, hexString -->
        <element type="NumberData"/>
        <!-- for types: float, double, int, unsigned, index -->
        <element type="TimeData"/>
        <!-- for types: date, time, dateAndTime, duration -->
        <element type="EnumeratorVariable"/>
        <!-- for type: enumerator -->
        <element type="BitEnumeratorVariable"/>
        <!-- for type: bitEnumerator -->
        <element type="BinaryVariable"/>
        <!-- for types: binary, dtmSpecific -->
        <element type="StructuredVariable"/>
        <!-- for type: structured -->
    </group>
```



```
    </ElementType>
    <!-- SubstituteValue -->
    <ElementType name="SubstituteValue" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="SubstituteType"/>
            <element type="Variant"/>
        </group>
    </ElementType>
    <!-- Value -->
    <ElementType name="Value" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="readAccess" required="no"/>
        <attribute type="writeAccess" required="no"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="Display"/>
            <element type="Variant"/>
        </group>
    </ElementType>
    <!-- DTMVariableStatus -->
    <ElementType name="StatusInformation" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="readAccess" required="no"/>
        <attribute type="writeAccess" required="no"/>
        <element type="EnumeratorEntry" minOccurs="1" maxOccurs="*"/>
    </ElementType>
    <!-- DtmVariable -->
    <ElementType name="DtmVariable" content="eltOnly" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="name" required="yes"/>
        <attribute type="label" required="no"/>
        <attribute type="descriptor" required="no"/>
        <element type="Value" minOccurs="1" maxOccurs="1"/>
        <element type="Unit" minOccurs="0" maxOccurs="1"/>
        <element type="Ranges" minOccurs="0" maxOccurs="1"/>
        <attribute type="statusFlag" required="no"/>
        <element type="StatusInformation" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <ElementType name="DtmVariables" content="mixed" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="name" required="yes"/>
        <attribute type="label" required="no"/>
        <attribute type="descriptor" required="no"/>
        <group order="many">
            <element type="DtmVariables" minOccurs="0" maxOccurs="*"/>
            <element type="DtmVariable" minOccurs="0" maxOccurs="*"/>
        </group>
    </ElementType>
    <!-- Communication Data -->
    <ElementType name="CommunicationData" content="empty" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="byteArray" required="yes"/>
    </ElementType>
    <!-- Definition of FDT specic communication errors for nested communication-->
    <ElementType name="CommunicationError" content="mixed" model="closed">
        <attribute type="nodeId" required="no"/>
        <attribute type="communicationError" required="yes"/>
        <attribute type="tag" required="yes"/>
        <attribute type="errorCode" required="no"/>
        <attribute type="descriptor" required="no"/>
    </ElementType>
</Schema>
```


13.4 DTM Information

Example :

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMInformationSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml"
xmlns:fdtappid="x-schema:FDTApplicationIdSchema.xml">
    <DtmInfo>
        <FDTVersion major="1" minor="2"/>
        <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
        <DtmDeviceTypes>
            <fdt:DtmDeviceType>
                <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-
05"/>
                <fdt:SupportedLanguages>
                    <fdt:LanguageId languageId="1131"/>
                </fdt:SupportedLanguages>
                <fdt:BusCategories>
                    <fdt:BusCategory busCategory="036D1499-387B-11D4-86E1-00E0987270B9"
busCategoryName="Profibus DP/V1">
                        <fdt:CommunicationTypeEntry communicationType="required"/>
                    </fdt:BusCategory>
                    <fdt:BusCategory busCategory="036D1498-387B-11D4-86E1-00E0987270B9"
busCategoryName="HART">
                        <fdt:CommunicationTypeEntry communicationType="supported"/>
                    </fdt:BusCategory>
                </fdt:BusCategories>
            </fdt:DtmDeviceType>
        </DtmDeviceTypes>
    </DtmInfo>
</FDT>
```

13.8 Supported Fieldbus Protocols

Example :

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:DTMProtocolsSchema.xml" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
    <fdt:BusCategories>
        <fdt:BusCategory busCategory="036D1498-387B-11D4-86E1-00E0987270B9"
busCategoryName="HART">
            <fdt:CommunicationTypeEntry communicationType="supported"/>
        </fdt:BusCategory>
    </fdt:BusCategories>
</FDT>
```

13.12 Functions of a DTM

```
<Schema name="DTMFunctionsSchema" 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">
    <!--Definition of Attributes-->
    <AttributeType name="functionId" dt:type="i4"/>
    <AttributeType name="checked" dt:type="boolean"/>
    <AttributeType name="enabled" dt:type="boolean"/>
    <AttributeType name="hasGUI" dt:type="boolean" default="0"/>
    <AttributeType name="printable" dt:type="boolean" default="0"/>
    <AttributeType name="help" dt:type="string"/>
    <AttributeType name="hidden" dt:type="boolean"/>
    <AttributeType name="label" dt:type="string"/>
    <AttributeType name="path" dt:type="uri"/>
    <AttributeType name="mime-type" dt:type="string"/>
    <AttributeType name="programName" dt:type="string"/>
    <AttributeType name="resizable" dt:type="boolean" default="0"/>
    <AttributeType name="separator" dt:type="boolean"/>
    <AttributeType name="toggle" dt:type="boolean"/>
    <AttributeType name="printableStandardFunction" dt:type="boolean" default="1"/>
```



```
    <AttributeType name="resizableStandardFunction" dt:type="boolean" default="1"/>

    <!--Definition of Elements-->
    <ElementType name="MimeType" content="empty" model="closed">
        <attribute type="mime-type" required="no"/>
    </ElementType>
    <ElementType name="OpenWith" content="empty" model="closed">
        <attribute type="programName" required="no"/>
    </ElementType>
    <ElementType name="Status" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="toggle" required="yes"/>
        <attribute type="checked" required="yes"/>
        <attribute type="enabled" required="yes"/>
        <attribute type="hidden" required="yes"/>
        <attribute type="separator" required="yes"/>
    </ElementType>
    <ElementType name="Icon" content="empty" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="path" required="yes"/>
    </ElementType>
    <!--Definition of a single function entry, which is not a standard function (concerning the applicationIds)-->
    <!--This entry has its own label-->
    <ElementType name="Function" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="label" required="yes"/>
        <attribute type="fdt:name" required="yes"/>
        <attribute type="help" required="yes"/>
        <attribute type="hasGUI" required="no"/>
        <attribute type="printable" required="no"/>
        <attribute type="resizable" required="no"/>
        <attribute type="functionId" required="yes"/>
        <element type="Icon" minOccurs="0" maxOccurs="1"/>
        <element type="Status" minOccurs="0" maxOccurs="1"/>
        <element type="appId:FDTApplicationIds" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <!--Definition of a document entry, which specifies a path to a DTM provided document-->
    <!--This entry has its own label-->
    <ElementType name="Document" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="label" required="yes"/>
        <attribute type="help" required="yes"/>
        <attribute type="path" required="yes"/>
        <group order="one" minOccurs="1" maxOccurs="1">
            <element type="MimeType" minOccurs="1" maxOccurs="1"/>
            <element type="OpenWith" minOccurs="1" maxOccurs="*"/>
        </group>
        <element type="Icon" minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <!--Definition of a single function entry, which is a standard function (concerning the applicationIds)-->
    <!--This entry has not an own label. The label must be supplied by the frame application-->
    <ElementType name="StandardFunction" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:name" required="yes"/>
        <attribute type="help" required="yes"/>
        <attribute type="printableStandardFunction" required="no"/>
        <attribute type="resizableStandardFunction" required="no"/>
        <attribute type="functionId" required="yes"/>
        <element type="Icon" minOccurs="0" maxOccurs="1"/>
        <element type="Status" minOccurs="0" maxOccurs="1"/>
        <element type="appId:ApplicationId" minOccurs="1" maxOccurs="1"/>
    </ElementType>
    <!--Definition of a group of standard function entries-->
    <!--This entry has not an own label. The label must be supplied by the frame application. It can not contain standard
functions-->
    <ElementType name="StandardFunctions" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="appId:ApplicationId" minOccurs="1" maxOccurs="1"/>
        <attribute type="fdt:name" required="yes"/>
        <attribute type="help" required="yes"/>
        <element type="Status" minOccurs="0" maxOccurs="1"/>
        <group order="many" minOccurs="0" maxOccurs="*">
```



```
            <element type="Function" minOccurs="0" maxOccurs="*"/>
            <element type="Functions" minOccurs="0" maxOccurs="*"/>
        </group>
    </ElementType>
    <!--Definition of a group of function entries-->
    <!--This entry has its own label and could contain standard, non standard functions and documents-->
    <ElementType name="Functions" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="label" required="yes"/>
        <attribute type="fdt:name" required="yes"/>
        <attribute type="help" required="yes"/>
        <element type="Status" minOccurs="0" maxOccurs="1"/>
        <group order="many" minOccurs="0" maxOccurs="*">
            <group order="one" minOccurs="0" maxOccurs="*">
                <element type="Function" minOccurs="0" maxOccurs="*"/>
                <element type="Document" minOccurs="0" maxOccurs="*"/>
                <element type="StandardFunction" minOccurs="0" maxOccurs="*"/>
            </group>
            <group order="one" minOccurs="0" maxOccurs="*">
                <element type="Functions" minOccurs="0" maxOccurs="*"/>
                <element type="StandardFunctions" minOccurs="0" maxOccurs="*"/>
            </group>
        </group>
    </ElementType>
    <ElementType name="FDT" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="Functions" minOccurs="1" maxOccurs="1"/>
    </ElementType>
</Schema>
```

13.21 Basic Channel Schema

```
<Schema name="FDTBasicChannelParameterSchema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
    <!--Definition of Attributes-->
    <AttributeType name="gatewayBusCategory" dt:type="uuid"/>
    <!--Definition of Elements-->
    <ElementType name="FDTBasicChannel" content="empty" model="closed" order="seq">
        <attribute type="fdt:nodeId" required="no"/>
        <attribute type="fdt:tag" required="yes"/>
        <attribute type="fdt:id" required="yes"/>
    </ElementType>
    <ElementType name="FDTChannelType" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="fdt:VersionInformation" minOccurs="1" maxOccurs="1"/>
    <attribute type="gatewayBusCategory" required="no"/>
    </ElementType>
    <ElementType name="FDT" content="eltOnly" model="closed">
        <attribute type="fdt:nodeId" required="no"/>
        <element type="FDTChannelType" minOccurs="1" maxOccurs="1"/>
        <element type="FDTBasicChannel" minOccurs="1" maxOccurs="1"/>
    </ElementType>
</Schema>
```

Example:

```
<?xml version="1.0"?>
<FDT xmlns="x-schema:FDTBasicChannelParameterSchema.xml" xmlns:fdt="x-
schema:FDTDataTypesSchema.xml">
    <FDTChannelType>
        <fdt:VersionInformation name="myname" vendor="myVendor" version="1.0" date="2000-08-05"/>
```



```
    </FDTChannelType>
    <FDTBasicChannel fdt:tag="myTag" fdt:id="myId" />
</FDT>
```

15.1.1 DPV0 Communication

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



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


15.1.2 DPV1 Communication

```
<Schema name="FDTProfibusDPV1CommunicationSchema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
        <!--Definition of Attributes-->
        <AttributeType name="api" dt:type="ui1"/>
        <AttributeType name="busAddress" dt:type="ui1"/>
        <AttributeType name="errorCode" dt:type="bin.hex"/>
        <AttributeType name="index" dt:type="ui1"/>
        <AttributeType name="communicationReference" dt:type="uuid"/>
        <AttributeType name="slot" dt:type="ui1"/>
        <AttributeType name="sequenceTime" dt:type="ui4"/>
        <AttributeType name="delayTime" dt:type="ui4"/>
        <AttributeType name="maxLenDataUnit" dt:type="ui1"/>
        <AttributeType name="scl" dt:type="ui1"/>
        <AttributeType name="networkMACAddress" dt:type="bin.hex"/>
        <!--Definition of Elements-->
        <ElementType name="NetworkAddress" content="empty" model="closed">
                <attribute type ="api" required="yes"/>
                <attribute type ="scl" required="yes"/>
                <attribute type ="networkMACAddress" required="yes"/>
        </ElementType>
        <ElementType name="srcNetworkAddress" content="eltOnly" model="closed">
                <element type="NetworkAddress" minOccurs="1" maxOccurs="1"/>
        </ElementType>
        <ElementType name="destNetworkAddress" content="eltOnly" model="closed">
                <element type="NetworkAddress" minOccurs="1" maxOccurs="1"/>
        </ElementType>
        <ElementType name="ConnectRequest" content="eltOnly" model="closed">
                <attribute type="fdt:nodeId" required="no"/>
                <attribute type="api" required="yes"/>
                <attribute type="busAddress" required="yes"/>
                <element type="srcNetworkAddress" minOccurs="0" maxOccurs="1"/>
                <element type="destNetworkAddress" minOccurs="0" maxOccurs="1"/>
        </ElementType>
        <ElementType name="ConnectResponse" content=" eltOnly" model="closed">
                <attribute type="fdt:nodeId" required="no"/>
                <attribute type="api" required="yes"/>
                <attribute type="busAddress" required="yes"/>
                <attribute type="communicationReference" required="yes"/>
                <attribute type="errorCode" required="yes"/>
                <attribute type="maxLenDataUnit" required="no"/>
                <element type="srcNetworkAddress" minOccurs="0" maxOccurs="1"/>
                <element type="destNetworkAddress" minOccurs="0" maxOccurs="1"/>
        </ElementType>
        <ElementType name="DisconnectRequest" content="empty" model="closed">
                <attribute type="fdt:nodeId" required="no"/>
                <attribute type="api" required="yes"/>
                <attribute type="busAddress" required="yes"/>
                <attribute type="communicationReference" required="yes"/>
        </ElementType>
        <ElementType name="DisconnectResponse" content="empty" model="closed">
                <attribute type="fdt:nodeId" required="no"/>
                <attribute type="api" required="yes"/>
                <attribute type="busAddress" required="yes"/>
                <attribute type="communicationReference" required="yes"/>
                <attribute type="errorCode" required="yes"/>
        </ElementType>
        <ElementType name="ReadRequest" content="empty" model="closed">
                <attribute type="fdt:nodeId" required="no"/>
                <attribute type="slot" required="yes"/>
                <attribute type="index" required="yes"/>
                <attribute type="communicationReference" required="yes"/>
        </ElementType>
        <ElementType name="ReadResponse" content="eltOnly" model="closed">
                <attribute type="fdt:nodeId" required="no"/>
                <attribute type="slot" required="yes"/>
                <attribute type="index" required="yes"/>
                <attribute type="communicationReference" required="yes"/>
                <attribute type="errorCode" required="yes"/>
                <element type="fdt:CommunicationData" minOccurs="1" maxOccurs="1"/>
        </ElementType>
```



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

16.1 Communication Schema

```
<Schema name="FDTHARTCommunicationSchema" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes" xmlns:fdt="x-schema:FDTDataTypesSchema.xml">
        <!--Definition of Attributes-->
        <AttributeType name="address1" dt:type="ui1"/>
        <AttributeType name="address2" dt:type="ui1"/>
        <AttributeType name="address3" dt:type="ui1"/>
        <AttributeType name="commandNumber" dt:type="ui1"/>
        <AttributeType name="communicationReference" dt:type="uuid"/>
        <AttributeType name="deviceStatus" dt:type="ui1"/>
        <AttributeType name="deviceTypeId" dt:type="ui1"/>
        <AttributeType name="longFrameRequired" dt:type="boolean"/>
        <AttributeType name="manufacturerId" dt:type="ui1"/>
        <AttributeType name="preambleCount" dt:type="ui1"/>
        <AttributeType name="primaryMaster" dt:type="boolean"/>
        <AttributeType name="shortAddress" dt:type="ui1"/>
        <AttributeType name="value" dt:type="ui1"/>
        <AttributeType name="sequenceTime" dt:type="ui4"/>
        <AttributeType name="delayTime" dt:type="ui4"/>
        <!--Definition of Elements-->
        <ElementType name="CommunicationStatus" content="empty" model="closed">
                <attribute type="fdt:nodeId" required="no"/>
```



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



```
        </ElementType>
        <ElementType name="SequenceStart" content="empty" model="closed">
                <attribute type="communicationReference" required="yes"/>
        </ElementType>
        <ElementType name="Abort" content="empty" model="closed">
                <attribute type="communicationReference" required="no"/>
        </ElementType>
        <ElementType name="FDT" content="eltOnly" model="closed">
                <attribute type="fdt:nodeId" required="no"/>
                <group order="one" minOccurs="1" maxOccurs="1">
                        <element type="ConnectRequest"/>
                        <element type="ConnectResponse"/>
                        <element type="DisconnectRequest"/>
                        <element type="DisconnectResponse"/>
                        <element type="DataExchangeRequest"/>
                        <element type="DataExchangeResponse"/>
                        <element type="SequenceBegin"/>
                        <element type="SequenceEnd"/>
                        <element type="SequenceStart"/>
                        <element type="Abort"/>
                        <element type="fdt:CommunicationError"/>
                </group>
        </ElementType>
</Schema>
```

2.45 Appendix – FDT IDL

```
// FDT 1.20 Interfaces


[
  uuid(036D1471-387B-11D4-86E1-00E0987270B9),
  version(1.2)
]
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 windowTiltle,
                [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);
  };

      //
      // FDT Container interfaces
      // ========================
      //

      // IDtmEvents
      [
   odl,
   uuid(F15BA42E-BBF1-42ed-8009-7F664A002CFB),
   dual,
   oleautomation
  ]
  interface IDtmEvents : IDispatch {
    [id(0x1)]
    HRESULT OnParameterChanged(
                [in] BSTR systemTag,
                [in] FdtXmlDocument parameter);
    [id(0x2)]
    HRESULT OnErrorMessage(
                [in] BSTR systemTag,
                [in] BSTR errorMessage);
    [id(0x3)]
    HRESULT OnProgress(
                [in] BSTR systemTag,
                [in] BSTR title,
                [in] short percent,
                                                [in] VARIANT_BOOL show);
    [id(0x4)]
    HRESULT OnUploadFinished(
                                [in] FdtUUIDString invokeId,
                                [in] VARIANT_BOOL success);
```



```
    [id(0x5)]
    HRESULT OnDownloadFinished(
                        [in] FdtUUIDString invokeId,
                        [in] VARIANT_BOOL success);
    [id(0x6)]
    HRESULT OnApplicationClosed(
                        [in] FdtUUIDString invokeId);
    [id(0x8)]
    HRESULT OnFunctionChanged(
                        [in] BSTR systemTag);

          [id(0x9)]
    HRESULT OnChannelFunctionChanged(
                        [in] BSTR systemTag,
          [in] FdtXPath channelPath);
    [id(0xa)]
    HRESULT OnPrint(
          [in] BSTR systemTag,
          [in] FdtXmlDocument functionCall);
    [id(0xb)]
    HRESULT OnNavigation(
                        [in] BSTR systemTag);
    [id(0xc)]
          HRESULT OnOnlineStateChanged(
                                    [in] BSTR systemTag,
                                    [in] VARIANT_BOOL onlineState);
          [id(0xd)]
          HRESULT OnPreparedToRelease(
                                    [in] BSTR systemTag);
          [id(0xe)]
          HRESULT OnPreparedToReleaseCommunication(
                                    [in] BSTR systemTag);
          [id(0xf)]
          HRESULT OnInvokedFunctionFinished(
                                    [in] FdtUUIDString invokeId,
                                    [in] VARIANT_BOOL success);
          [id(0x10)]
          HRESULT OnScanResponse(
                                    [in] FdtUUIDString invokeId,
                                    [in] FdtXmlDocument response);

};
```



```
        // IDtmAuditTrailEvents
        [
  odl,
  uuid(036D1479-387B-11D4-86E1-00E0987270B9),
  dual,
  oleautomation
]
interface IDtmAuditTrailEvents : IDispatch {
   [id(0x1)]
   HRESULT OnStartTransaction(
               [in] BSTR systemTag,
               [out, retval] VARIANT_BOOL* result);
   [id(0x2)]
   HRESULT OnAuditTrailEvent(
               [in] BSTR systemTag,
               [in] FdtXmlDocument logEntry);
   [id(0x3)]
   HRESULT OnEndTransaction(
               [in] BSTR systemTag,
               [out, retval] VARIANT_BOOL* result);
};

        // IFdtActiveX
        [
  odl,
  uuid(5959f485-2c51-4a55-80a7-dd3c45d8baf2),
  dual,
  oleautomation
]
interface IFdtActiveX : IDispatch {
   [id(0x1)]
   HRESULT OpenActiveXControlRequest(
               [in] BSTR systemTag,
               [in] FdtXmlDocument functionCall,
               [out, retval] VARIANT_BOOL* result);
   [id(0x2)]
   HRESULT CloseActiveXControlRequest(
               [in] FdtUUIDString invokeId,
               [out, retval] VARIANT_BOOL* result);
};

        // IFdtBulkData
        [
```



```
  odl,
  uuid(036D148D-387B-11D4-86E1-00E0987270B9),
  dual,
  oleautomation
]
interface IFdtBulkData : IDispatch {
   [id(0x60030000)]
   HRESULT GetProjectRelatedPath(
               [in] BSTR systemTag,
               [out, retval] BSTR* result);
   [id(0x60030001)]
   HRESULT GetInstanceRelatedPath(
               [in] BSTR systemTag,
               [out, retval] BSTR* result);
};

      // IFdtContainer
      [
  odl,
  uuid(036D1487-387B-11D4-86E1-00E0987270B9),
  dual,
  oleautomation
]
interface IFdtContainer : IDispatch {
   [id(0x1)]
   HRESULT SaveRequest(
               [in] BSTR systemTag,
               [out, retval] VARIANT_BOOL* result);
   [id(0x2)]
   HRESULT LockDataSet(
               [in] BSTR systemTag,
               [out, retval] VARIANT_BOOL* result);
   [id(0x3)]
   HRESULT UnlockDataSet(
               [in] BSTR systemTag,
               [out, retval] VARIANT_BOOL* result);
   [id(0x4)]
   HRESULT GetXmlSchemaPath(
               [out, retval] BSTR* result);
};

      // IFdtDialog
      [
```



```
  odl,
  uuid(15C19931-6161-11d4-A0A9-005004011A04),
  dual,
  oleautomation
]
interface IFdtDialog : IDispatch {
   [id(0x1)]
   HRESULT UserDialog(
                [in] BSTR systemTag,
                                                [in] FdtXmlDocument userMessage,
                [out, retval] FdtXmlDocument* result);
};

      // IFdtTopology
      [
  odl,
  uuid(036D147A-387B-11D4-86E1-00E0987270B9),
  dual,
  oleautomation
]
interface IFdtTopology : IDispatch {
   [id(0x1)]
   HRESULT GetParentNodes(
                [in] BSTR systemTag,
                [out, retval] FdtXmlDocument* result);
   [id(0x2)]
   HRESULT GetChildNodes(
                [in] BSTR systemTag,
                [in] FdtXPath channelPath,
                [out, retval] FdtXmlDocument* result);
   [id(0x3)]
   HRESULT CreateChild(
                [in] FdtXmlDocument deviceType,
                [in] FdtXPath channelPath,
                [out, retval] BSTR* result);
   [id(0x4)]
   HRESULT DeleteChild(
                [in] BSTR systemTag,
                [in] FdtXPath channelPath,
                [out, retval] VARIANT_BOOL* result);
                 [id(0x5)]
                 HRESULT GetDtmForSystemTag(
                                                [in] BSTR systemTag,
```



```
                                        [out,retval] IDtm **result);
            [id(0x6)]
            HRESULT GetDtmInfoList(
                                        [out, retval] FdtXmlDocument* result);
            [id(0x7)]
            HRESULT MoveChild(
                                        [in] BSTR systemTag,
            [in] FdtXPath destinationchannelPath,
            [out, retval] VARIANT_BOOL* result);
    [id(0x8)]
    HRESULT ReleaseDtmForSystemTag(
                                        [in] BSTR systemTag,
                                        [out, retval] VARIANT_BOOL* result);
};

    //
    // Collections
    // ===========

    // IFdtChannelCollection
[
  odl,
  uuid(E4F31A10-45BF-11d4-BBB3-0060080993FF),
  dual,
  oleautomation
]
    interface IFdtChannelCollection : IDispatch {
            [propget, id(0x1)]
            HRESULT Item(
                                        [in] VARIANT *pvarIndex,
                                        [out,retval] IFdtChannel **ppRes);
            [propget, id(0x2)]
            HRESULT Count(
                                        [out,retval] long* plCount);
            //support VB FOR EACH syntax via an IEnumVariant
            [propget, id(DISPID_NEWENUM)]
            HRESULT NewEnum(
                                        [out,retval] IUnknown** ppEnumVariant);
    };

};
```


PROFIBUS Nutzerorganisation e.V.
Haid-und-Neu-Str. 7
D-76131 Karlsruhe
Phone: ++ 721 / 96 58 590
Fax: ++ 721 / 96 58 589
e-mail: pi@profibus.com
http://www.profibus.com