MOST

Media **O**riented **S**ystems **T**ransport

Multimedia and Control Networking Technology

MOST DTD Cookbook
Rev. 1.2
12/2009

MOSTCO CONFIDENTIAL

See page 3 for the terms of disclosure

Legal Notice

COPYRIGHT

LICENSE DISCLAIMER

Nothing on any MOST Cooperation Web Site, or in any MOST Cooperation document, shall be construed as conferring any license under any of the MOST Cooperation or its members or any third party's intellectual property rights, whether by estoppel, implication, or otherwise.

CONTENT AND LIABILITY DISCLAIMER

MOST Cooperation or its members shall not be responsible for any errors or omissions contained at any MOST Cooperation Web Site, or in any MOST Cooperation document, and reserves the right to make changes without notice. Accordingly, all MOST Cooperation and third party information is provided "AS IS". In addition, MOST Cooperation or its members are not responsible for the content of any other Web Site linked to any MOST Cooperation Web Site. Links are provided as Internet navigation tools only.

MOST COOPERATION AND ITS MEMBERS DISCLAIM ALL WARRANTIES WITH REGARD TO THE INFORMATION (INCLUDING ANY SOFTWARE) PROVIDED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

In no event shall MOST Cooperation or its members be liable for any damages whatsoever, and in particular MOST Cooperation or its members shall not be liable for special, indirect, consequential, or incidental damages, or damages for lost profits, loss of revenue, or loss of use, arising out of or related to any MOST Cooperation Web Site, any MOST Cooperation document, or the information contained in it, whether such damages arise in contract, negligence, tort, under statute, in equity, at law or otherwise.

FEEDBACK INFORMATION

Any information provided to MOST Cooperation in connection with any MOST Cooperation Web Site, or any MOST Cooperation document, shall be provided by the submitter and received by MOST Cooperation on a non-confidential basis. MOST Cooperation shall be free to use such information on an unrestricted basis.

TRADEMARKS

MOST Cooperation and its members prohibit the unauthorized use of any of their trademarks. MOST Cooperation specifically prohibits the use of the MOST Cooperation LOGO unless the use is approved by the Steering Committee of MOST Cooperation.

SUPPORT AND FURTHER INFORMATION

For more information on the MOST technology, please contact:

MOST Cooperation
Administration
Bannwaldallee 48
D-76185 Karlsruhe
Germany

Tel: (+49) (0) 721 966 50 00
Fax:(+49) (0) 721 966 50 01
E-mail: contact@mostcooperation.com
Web: www.mostcooperation.com

This Specification is Confidential Information of the MOST Cooperation. It may only be disclosed to member companies. Member companies wishing to discuss these Specifications with suppliers or other third parties must ensure that a commercially standard form of non-disclosure agreement has been previously executed by the party receiving such Specifications. Use of these Specifications may only be for purposes for which they are intended by the MOST Cooperation. Unauthorized use or disclosure is a violation of law.

Contents

Document References

All documents which this MOST document have references to are listed here with the actual revision this document is referring to.

Number	Document	Revision
1	MOST Specification	2.5
2	MOST FCat DTD	6.05.3

Document History

Revision 1.2

Change Ref.	Section	Changes
001	General	Updated to match FCat DTD 6.05.3. Correction of clerical errors.
002	2.4	Added description for new attributes Virtual, Wellknown, and Occurrence.
003	2.5	Added description of new attribute Notification.
004	2.5.4	New optional attribute for Enums: SymbolicName.
005	2.5.7	In addition to Classified Stream, function class Container now supports Stream and Short Stream.
006	2.5.9.3.1	New ClassDef ArrayWindow.
007	2.5.11	Sequence Property may now contain Stream parameter.
008	2.5.12	Added note that warns of using Record or Array fragments in Unclassified Properties or Unclassified Methods.
009	2.5.13	New attributes for TStream: MinLength and ByteOrder.
010	2.5.13.2.1	New section on Bit-encoded stream cases. Simple streams contain no StreamCase or exactly one StreamCase.
011	2.5.13.2.2	New section on nested streams.
012	2.5.13.3	New attribute Signedness for StreamSignal element.
013	2.6.2	Removed OPTypes Abort and AbortAck for Trigger Method.
014	2.6.3	Added StartAck OPType to Sequence Method. Added Stream to allowed types for Sequence Method.
015	2.7.1— 2.7.3	Added Channel attribute to all OPTypes.

Revision 1.1

Change Ref.	Section	Changes
		Document creation - based on previous description of DTD in old format. ("MOST Catalog DTD Description DC07"). Correction of various errors and out of date information. Inclusion of elements added in DTD release 6.05.1

1 Introduction

This document describes the MOST Exchange DTD (Rev 6.05.3), alias MOST Catalog DTD or „MOSTCatalog-DTD“. The objective of this document is to aid MOST Working Groups in producing XML-based function catalogues that are consistent with the syntax of the DTD as well as the intended semantics. In addition, the document shall be used when developing tools to edit MOST function catalogues. The guidelines contained within this document that are not directly supported by the syntax of the DTD itself should form the basis of “rule checkers” that can be integrated into the editing tools.

The document is structured as follows. Section 2 describes each element of the DTD in turn and includes guidance on those aspects of the XML-based FCat definitions not necessarily mandated by the syntax defined by the DTD. The emphasis in Section 2 is a description of the various function classes and associated types. Section 3 includes complete examples for all function class types.

2 DTD Elements

2.1 General rules

2.1.1 Element content and white space

In the XML document, descriptions are mapped to a single element, using white space for visual formatting. Therefore, this white space must be conserved. Additionally, when editing the document, no white space should be added that isn't intended as visual markup.

2.1.2 Optional attributes with default values

The DTD assigns a default value to several attributes. If any of them are omitted in an XML instance, validating XML processors will fill in the respective default values. If it is necessary to mark an attribute as truly absent, the special value "#NULL#" should be used.
In some cases, #NULL# is a placeholder for a value that will be defined by the system integrator.

2.1.3 Invalid characters

Certain characters are reserved in XML. Most notably, "<" always starts a tag and "&" an entity. Individual instances of these characters in element content or attributes can be escaped with "<" and "&", respectively. Longer element content sections using several reserved characters can be escaped with CDATA sections, as shown in the example.

```
<PosDescription PosX="#NULL#" PosY="#NULL#">
<![CDATA[FktIDList ::= <FktID>{,<FktID>}]]>
</PosDescription>
```

2.1.4 Naming conventions for XML IDs

To achieve uniform naming throughout XML instances, the following rules apply.

Legal characters for IDs are lowercase letters and the underscore. Every ID is composed of a prefix and an identifier, which may consist of one or several words. Prefix and identifier, as well as the words of the identifier, are separated by underscores. The prefix is uniquely determined by the ID kind: for ClassIDs, it is "class", for TypeIDs "type", for UnitIDs "unit" and for ErrorIDs "error". A couple of examples for various types of IDs are shown below:

ClassID:	class_unclassified_property
TypeID:	type_unsigned_byte
	type_array_of_record
UnitID:	unit_ms
	unit_none
ErrorID:	error_function_specific

OPType IDs are named analogue to the corresponding XML-elements. E.g., the OPType for <PCmdGet> is PcmdGet.

2.1.5 Number formats

When using integer numbers within the XML-file (e.g. for the definition FblockID) either decimal or hexadecimal formats may be used. The following convention shall be followed to distinguish between the two formats:

Decimal: no prefix, e.g. 10 (ten)
Hexadecimal "0x" as prefix, e.g. 0x10 (sixteen)

2.1.6 HTML in PCDATA

Where HTML is used to format descriptions, tagging should start with block-level elements with the exception of list elements like LI, DL, etc. Valid block-level elements are Hx, P, DIV, UL, OL, etc. The use of list elements without a surrounding block-level element will lead to display errors when using ConvertToHTML. The resulting HTML is not valid and the style sheet probably cannot correctly be applied to certain areas.

Example XML code:

```
<ParamDescription>
  &lt;ul&gt;
    &lt;li&gt;0x00..0x1F = general states&lt;/li&gt;
    &lt;li&gt;0x20..0x2F = video specific states&lt;/li&gt;
    &lt;li&gt;0x30..0x3F = tape specific states&lt;/li&gt;
    &lt;li&gt;0x40..0x4F = file handling&lt;/li&gt;
    &lt;li&gt;0x50..0x5F = recording&lt;/li&gt;
  &lt;/ul&gt;
</ParamDescription>
```

2.2 Function catalog

DTD Definition:

```
<!ELEMENT FunctionCatalog (CatalogVersion, (FBlock)+, Definition)>
<!ELEMENT CatalogVersion (Release, Date, Author?, Company?,
          Modification*)>
<!ELEMENT Release (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Company (#PCDATA)>
<!ELEMENT Modification (Change, Reason)>
<!ELEMENT Change (#PCDATA)>
<!ELEMENT Reason (#PCDATA)>
```

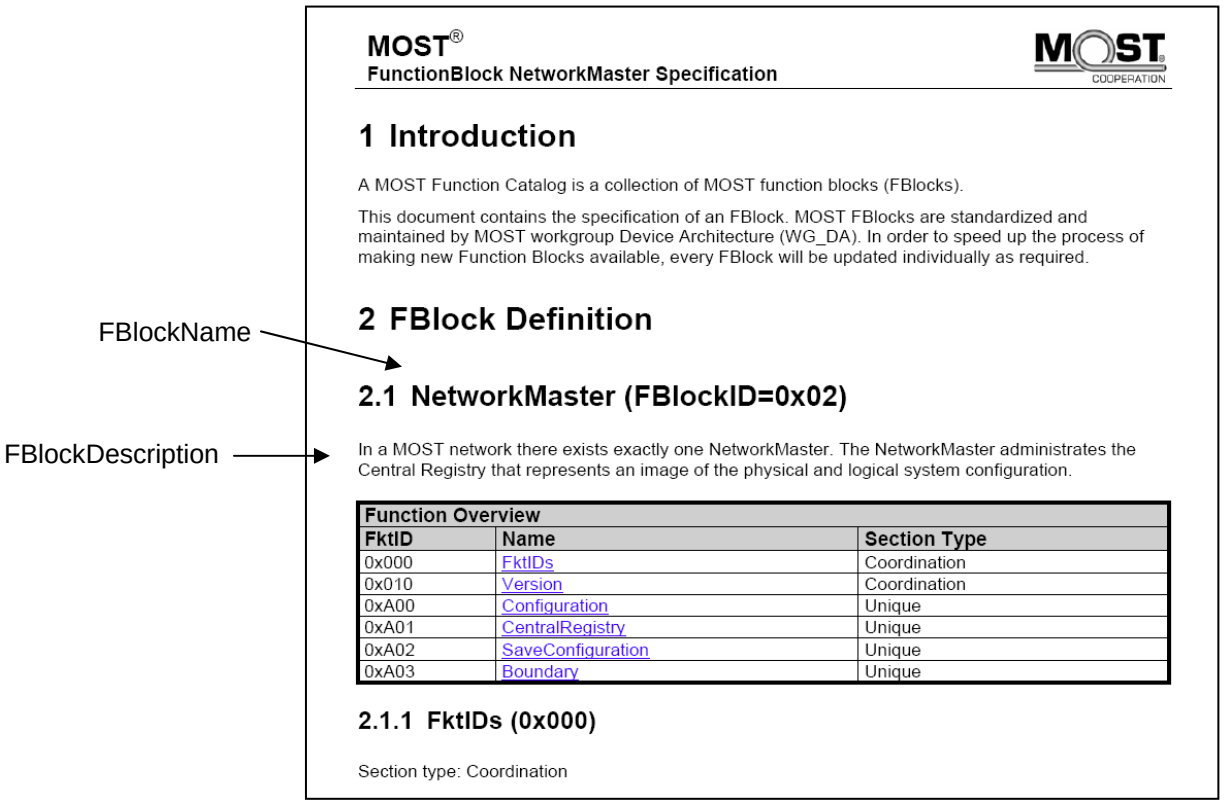
The element `FunctionCatalog` is the root element defined by the MOST Catalog-DTD. This element contains release management information (`CatalogVersion`) as well as the list of Fblocks (`(FBlock)+`) defined within the function catalog and any global definitions required (`Definition`).

The element `CatalogVersion` contains a sequence of `Release`, `Date`, `Author`, `Company` and `Modification` elements. `Author`, `Company` and `Modification` are optional. A list of modification consisting of definitions of the change and reasons for the change may be defined. There are no restrictions on the format of any of the `CatalogVersion` elements.

Example XML code:

```
<FunctionCatalog>
  <CatalogVersion>
    <Release> Version 1.0</Release>
    <Date> 19.02.2006</Date>
    <Author>Simon Burton</Author>
    <Company>DaimlerChrysler</Company>
    <Modification>
      <Change>30.10.2005</Change>
      <Reason>First draft</Reason>
    </Modification>
    <Modification>
      <Change>19.02.2006</Change>
      <Reason>More examples added</Reason>
    </Modification>
  </CatalogVersion>
  <FBlock>
    …
  </FBlock>
  <FBlock>
    …
  </FBlock>
  <Definition>
    …
  </Definition>
</FunctionCatalog>
```

2.3 Function blocks



Example

DTD Definition:

```
<!ELEMENT FBlock (FBlockID, FBlockKind, FBlockName, FBlockDescription,
          FBlockVersion, (Function)*)>
<!ELEMENT FBlockID (#PCDATA)>
<!ELEMENT FBlockKind (#PCDATA)>
<!ELEMENT FBlockName (#PCDATA)>
<!ELEMENT FBlockDescription (#PCDATA)>
<!ELEMENT FBlockVersion (Release, Date, Author?, Company?, Modification*)>
<!ATTLIST FBlockVersion Access(public | private | preliminary) "public">
```

The element `FBlock` contains a function block of the function catalog and all its functions. The element `FBlockID` contains a function block's ID. Within an XML document, no other function block may carry the same ID. It must fall in the range implied by `FBlockKind`. `FBlockKind` may take one of the following values: `Administration`, `Operation`, `Audio`, `Drives`, `Receiver`, `Communication`, `Video` or `Proprietary` (see MOST Specification Rev 2.5, page 29). The element `FBlockName` contains the function block's name. Within an XML document, no other function block may carry the same name. The element `FBlockDescription` contains a description of a function block. No restrictions are currently placed on the format of the `FblockDescription` data. Note that descriptions containing tables and formatted text cannot be easily modelled in the DTD format. The element `FBlockVersion` contains version information for the current function block. The sub-elements `(Release, Date, Author?, Company?, Modification*)` have the same semantics as for the release information for the function catalog (see section 2.2). These elements may be used to record FBlock specific release information. `FBlockVersion` also contains an attribute to describe the function block's access privileges with respect to its status.
Proprietary function blocks must be designated as `private`, candidates for inclusion in the standard as `preliminary` and standardized function blocks as `public`.

Example XML code:

```
<FBlock>
  <FBlockID>0x02</FBlockID>
  <FBlockKind>Administration</FBlockKind>
  <FBlockName>NetworkMaster</FBlockName>
  <FBlockDescription>Contains network management fns</FBlockDescription>
  <FBlockVersion Access="public">
    <Release>2.4.2</Release>
    <Date>10 April 2003</Date>
    <Author>Simon Burton</Author>
    <Company>DaimlerChrysler</Company>
    <Modification>
      <Change>30.10.2005</Change>
      <Reason>First draft</Reason>
    </Modification>
  </FBlockVersion>
  <Function> … </Function>
  <Function> … </Function>
  …
</FBlock>
```

2.4 Functions

A function as represented in the function catalog (without parameter descriptions):

2.2.8 DeckStatus (0x200)

This property controls and shows the state of the drive.

Format of Function

Function classes: Enumeration

FBlock	Function	OPType	Parameter
GeneralPlayer	DeckStatus (0x200)	Set	DeckStatus
		Get	
		SetGet	DeckStatus
		Status	DeckStatus
		Error	ErrorCode, ErrorInfo

DTD Definition:

```
<!ELEMENT Function (FunctionID, FunctionName, FunctionDescription,
          FunctionVersion, FunctionClass)>
<!ATTLIST Function Virtual (true | false) #IMPLIED >
<!ELEMENT FunctionID (#PCDATA)>
<!ATTLIST FunctionID FunctionSection (Coordination | Mandatory | Extension
| Unique |
          Proprietary) #IMPLIED
          Wellknown (true | false) #IMPLIED
          Occurrence (Conditional | Mandatory | Optional) #IMPLIED >
<!ELEMENT FunctionName (#PCDATA)>
<!ELEMENT FunctionDescription (#PCDATA)>
<!ELEMENT FunctionVersion (Release, Date, Author?, Company?,
          Modification*)>
<!ATTLIST FunctionVersion Access(public | private | preliminary) "public">
<!ELEMENT FunctionClass (FunctionClassDesc, FunctionClassError*, (Property
          | Method))>
<!ELEMENT FunctionClassDesc (#PCDATA)>
<!ATTLIST FunctionClass ClassRef IDREF #REQUIRED >
<!ELEMENT FunctionClassError (ErrorCode, ErrorCodeDesc?, ParamName,
          ParamDescription, ErrorParamType)>
<!ATTLIST FunctionClassError ErrorRef IDREF #REQUIRED >
<!ELEMENT ErrorCode (#PCDATA)>
<!ELEMENT ErrorCodeDesc (#PCDATA)>
<!ELEMENT ErrorParamType (TStream | TCStream | TShortStream| TArray |
      TRecord | TBool | TNumber | TEnum | TString | TBitField | TVoid)>
…
<!ELEMENT ParamName (#PCDATA)>
<!ATTLIST ParamName ParamIdx CDATA #IMPLIED>
<!ELEMENT ParamDescription (#PCDATA)>
```

The top level XML element of a function is `Function`. The `Virtual` attribute describes whether the function is located in the MOST ring (`false`) or outside the MOST ring and accessible via gateway (`true`).

Several child elements of `Function` contain the function ID, name, description and class. Versioning information is stored in additional elements. The element `FunctionID` contains a function's ID. Within a function block of an XML document, no other function may carry the same ID. It must lie in the range

implied by its FunctionSection attribute. The optional attribute can take the values: Coordination, Mandatory, Extension, Unique or Proprietary. This attribute classifies a function as part of a section. Every section has an associated range of function IDs (see MOST Specification Rev 2.5, page 33).
The attribute Wellknown is set to true if the function was defined by the MOST Cooperation.
The attribute Occurrence is used to determine whether the use of the function is mandatory, optional, or depends on a condition specific to the application domain.

The element FunctionName contains a function's name. Within a function block of an XML document, no other function may carry the same name. The element FunctionDescription contains a description of a function. The FunctionVersion element contains version information for the current function. Its attribute describes the function's access privileges with respect to its status. Proprietary functions must be designated as private, candidates for inclusion in the standard as preliminary and standardized functions as public. The default value is public. The child elements Release, Date, Author?, Company?, and Modification* are to be used as defined in section 2.2.

The element FunctionClass contains a description of a function and its parameters. Its mandatory attribute establishes a reference to a ClassDef element in the DTD's definition section. The function class text is stored in child element FunctionClassDesc, individual parameters and OPTypes are stored in FunctionClass. The exact location of the parameters depends on the function class, which is categorized as a property or a method. If a function belongs to the „Array" or „Record" classes, the corresponding parameter types are given as well, for example, "Record of { Number Number Number }". To form the element name for a function class, add the prefix "P" to the class name. The only exception is PArray, which represents all three array classes, Array, DynamicArray and LongArray. All function class elements are essentially similar, they differ only in the permissible OPTypes and parameter types.

FunctionClass also contains an optional list of FunctionClassError elements. Each FunctionClassError element contains a description of a possible error of a function. Its attribute establishes a reference to an ErrorDef element in the DTD's definition section. An error applicable to this function shall be referenced here. Each FunctionClassError element contains the child elements ErrorCode, ErrorCodeDesc (optional), ParamName, ParamDescription and ErrorParamType. The element ErrorCode contains the error code of a function specific error. According to MOST Specification Rev 2.5, this is a fixed value of 0x20. The element ErrorCodeDesc contains a description of the error. The element ErrorParamType contains a function specific error parameter - normally, an „ErrorInfo". One of the applicable child elements (TStream, TArray, TRecord, TBool, TNumber, TEnum, TString, TBitField or TVoid) is selected to designate the parameter type.

Function classes are subdivided in two sets: properties and methods. The details of the property or method implemented by the function are described in the child elements Property and Method of the FunctionClass element. The XML format used to describe the different property and method function classes is described in the following sections.

Example XML Code:

```
<Function>
  <FunctionID>0xA02</FunctionID>
  <FunctionName>SaveConfiguration</FunctionName>
  <FunctionDescription>Saves the central registry </FunctionDescription>
  <FunctionVersion Access="public">
    <Release> Version 1.0</Release>
    <Date> 19.02.2006</Date>
    <Author>Simon Burton</Author>
    <Company>DaimlerChrysler</Company>
    <Modification>
      <Change>30.10.2005</Change>
```

```
      <Reason>First draft</Reason>
    </Modification>
  </FunctionVersion>
  <FunctionClass ClassRef="class_trigger">
    <FunctionClassDesc>Trigger</FunctionClassDesc>
    <Method>
      …
    </Method>
  </FunctionClass>
</Function>
```

2.4.1 Multiple parameter occurrences in a function

It is possible for a parameter to occur several times in a function. For example, a parameter may be used in each of the different stream cases of a stream. The printed function catalog describes those parameters exactly once. In the XML document, omitting parameter descriptions isn't always feasible, as information about the parameter and its position may otherwise be lost. Therefore, parameters are always fully described in the XML document. An application processing an XML document may assume equally named parameters of a function to be the same parameter without further comparisons. In addition the attribute `Details` included in each parameter element can be used to suppress the generation of documentation for a particular parameter.

2.4.2 Assignment of OPTypes and parameters

The assignment of OPTypes and parameters in the XML document differs from the printed representation of the function catalog. The printed catalog lists all legal OPTypes in a table, assigning them their respective parameters.

The XML document operates inversely. Parameter and parameter type are specified first. Every parameter directly used by an OPType is assigned a list of OPTypes using this parameter via the `XXParamOPType` element. The respective parameter positions are assigned in the `ParamPos` element. Position numbers for regular parameters start with 1 (for exceptions see next paragraph). This way, a parameter used by several OPTypes has to be described only once.

Because OPTypes are specified in a parameter context, parameterless OPTypes need special treatment. For each function with at least one parameterless OPType a dummy parameter of type `TVoid` must be included. It shall be assigned to the parameterless OPTypes, using a parameter position of 0 and the details="false" flag.

Example XML Code for TVoid:

```
<PNParam details="false">
  <ParamName ParamIdx="1"/>
  <ParamDescription/>
  <PNParamOPType>
    <PNCommand>
      <PCmdGet OPTypeRef="PCmdGet">
        <ParamPos>0</ParamPos>
      </PCmdGet>
    </PNCommand>
    <PNReport/>
  </PNParamOPType>
  <PNParamType>
    <TVoid TypeRef="type_void"/>
  </PNParamType>
</PNParam>
```

2.4.3 “ErrorCode” and “ErrorInfo” parameters

These parameters apply to every function for OPType “Error”. As they are specified in the MOST Specification and identical for every function, the printed catalog does not mention them. The XML document treats them as parameterless OPTypes described above, by assigning the type `TVoid` and applying the attribute `details` the value `false`.

2.5 Properties

Function classes are subdivided in two sets: “properties“ and ”methods“. This section contains further information on defining function classes describing “properties”.

DTD Definition:

```
<!ELEMENT Property (PSwitch | PNumber | PText | PEnum | PContainer |
      PRecord | PArray | PSequence | PUnclassified)>
<!ATTLIST Property Notification (true | false) #IMPLIED >
```

The attribute `Notification` is set to true if the function supports notification.

The individual child elements of `Property` are now described in more detail.

2.5.1 Switch properties

This function class (See MOST Specification Rev 2.5, page 69) is used to model functions consisting of a parameter of boolean type. The property may consist of a number of boolean fields, depending on the use of the `TBool`, `TBoolField`, and `TBitField` types.

Switch example:

TpTa

Basis datatype	Bit #	Code	Description
Boolean	Bit 0	False	Tp Off
		True	Tp On
	Bit 1	False	Ta Off
		True	Ta On
	Bit 2 .. 7	-	Reserved

```
<!ELEMENT PSwitch (PSParam)+>
<!ELEMENT PSParam (ParamName, ParamDescription, PSParamOPType,
          PSParamType)>
<!ATTLIST PSParam details (true | false) #IMPLIED>
<!ELEMENT PSParamOPType ((PSCommand, PSReport?) | (PSReport, PSCommand?))>
<!ELEMENT PSCommand (PCmdSet?, PCmdGet?, PCmdSetGet?, PCmdGetInterface?)>
<!ELEMENT PSReport (PReportStatus?, PReportInterface?, PReportError?)>
<!ELEMENT PSParamType (TBool | TBitField | TVoid)>
```

The `PSParam` element describes a parameter of the Switch function class and the OPTypes that use it. The function class can make use of the following types (as indicated in `PSParamType`): `TBool`, `TBitField`, and `TVoid`. The child element `PSParamOPType` defines the set of OPTypes for the function that make use of the parameter currently being defined. The command OPTypes available to Switch properties are `Set`, `Get`, `SetGet`, and `GetInterface`. The report OPTypes available are `Status`, `Interface`, and `Error`.

2.5.1.1 Use of the TBool type

The TBool type is used to model parameters consisting either of a simple boolean value (true or false) or a record of bitfields without the use of a mask (see TBitField type).

DTD Definition:

```
<!ELEMENT TBool (TBoolField)+>
<!ATTLIST TBool TypeRef IDREF #REQUIRED>
<!ELEMENT TBoolField (BitTrueDesc, BitFalseDesc)>
<!ATTLIST TBoolField BitPos CDATA #REQUIRED>
<!ELEMENT BitTrueDesc (#PCDATA)>
<!ELEMENT BitFalseDesc (#PCDATA)>
```

The TypeRef attribute clarifies the exact usage type. Valid types are:

- type_boolean: This is a single boolean value (exactly 1 bit used)
- type_unsigned_byte: This is a field of eight boolean values (at most 8 bits used)
- type_unsigned_word: This is a field of sixteen boolean values (at most 16 bits used)
- type_unsigned_long: This is a field of thirty-two boolean values (at most 32 bits used).

The type_boolean is usually referred to as a **Boolean** value, while the type_unsigned_byte, type_unsigned_word, and type_unsigned_long are called **Boolfields**. Do not confuse this **Boolfield** with the TBoolField element, they are not related and have completely separate meanings. For TypeRefs type_unsigned_byte, type_unsigned_word, and type_unsigned_long several TBoolField elements are contained in one TBool element.

Each TBoolField element contains a single boolean and descriptions for its “true” and “false” states. For each field, represented in a TBoolField element, the position of this bit or set of bits is given in the BitPos attribute. The grouping of a number of bits into one field is achieved using the “m…n” notation for the BitPos attribute, where m denotes the start bit (lowest) and n the end bit (highest) of the field. In such a case the BitFalseDesc element is left empty. The meaning of the bit when set to true is stored in BitTrueDesc while the meaning of the bit when it is set to false is stored in BitFalseDesc.

Example XML Code:

```
<FunctionClass ClassRef="class_switch">
  <FunctionClassDesc>Switch</FunctionClassDesc>
  <Property>
    <PSwitch>
      <PSParam>
        <ParamName>Tp Ta</ParamName>
        <ParamDescription/>
        <PSParamOPType>
          <PSCommand>
            <PCmdGet OPTypeRef="PCmdSet">
              <ParamPos>1</ParamPos>
            </PCmdGet>
          </PSCommand>
        </PSParamOPType>
        <PSParamType>
          <TBool TypeRef="type_boolean">
            <TBoolField BitPos="0">
              <BitTrueDesc>TP On</BitTrueDesc>
              <BitFalseDesc>TP Off</BitFalseDesc>
            </TBoolField>
            <TBoolField BitPos="1">
```

```
                <BitTrueDesc>TA On</BitTrueDesc>
                <BitFalseDesc>TA Off</BitFalseDesc>
              </TBoolField>
              <TBoolField BitPos="2 .. 7">
                <BitTrueDesc>reserved</BitTrueDesc>
                <BitFalseDesc/>
              </TBoolField>
            </TBool>
          </PSParamType>
        </PSParam>
        <PSParam>
          …
        </PSParam>
      </PSwitch>
  </Property>
</FunctionClass>
```

2.5.1.2 Use of the TBitField type

The difference between a `TBitField` and a `TBool` (see previous section) is the mask which is present in the `TBitField` elements and which is missing in the `TBool` elements. The TypeRef attribute always contains “type_bitfield”.

DTD Definition:

```
<!ELEMENT TBitField (TBoolField)+>
<!ATTLIST TBitField TypeRef IDREF #REQUIRED Length CDATA "1">
<!ELEMENT TBoolField (BitTrueDesc, BitFalseDesc)>
<!ATTLIST TBoolField BitPos CDATA #REQUIRED>
<!ELEMENT BitTrueDesc (#PCDATA)>
<!ELEMENT BitFalseDesc (#PCDATA)>
```

The `Length` attribute of `TBitField` can be 1, 2, or 4, representing the number of bytes to use for the entire Bitfield type. Note that the upper half of the bits is used for the mask, i.e., in a bit field of `Length` 1, bits 4…7 are used as the mask, bits 0…3 are used for the actual content. Similarly, in bit fields of length 2 and length 4, the mask has a size of 1 and 2 bytes, respectively. The content of the embedded `TBoolField` are identical to the `TBoolField` usage in `TBool` (See previous section).

Example XML code:

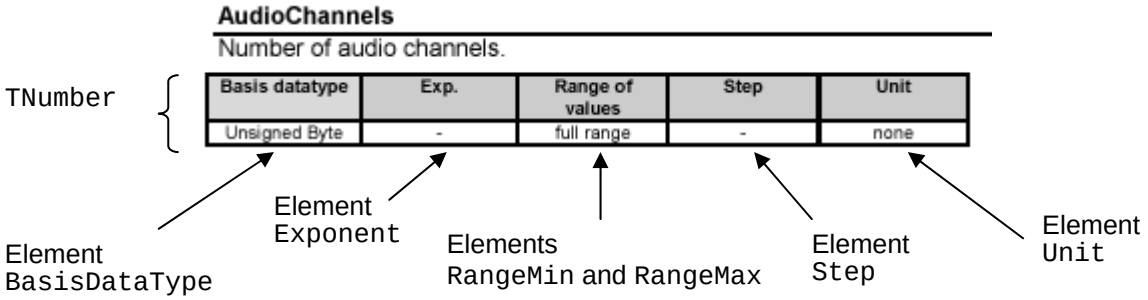
```
<PSParamType>
  <TBitField TypeRef="type_bitfield" Length="2">
    <TBoolField BitPos="0">
      <BitTrueDesc>Normal Audio</BitTrueDesc>
      <BitFalseDesc>-</BitFalseDesc>
    </TBoolField>
    <TBoolField BitPos="1">
      <BitTrueDesc>Directory</BitTrueDesc>
      <BitFalseDesc>-</BitFalseDesc>
    </TBoolField>
    <TBoolField BitPos="2">
      <BitTrueDesc>Category</BitTrueDesc>
      <BitFalseDesc>-</BitFalseDesc>
    </TBoolField>
    <TBoolField BitPos="3">
      <BitTrueDesc>Playlist</BitTrueDesc>
      <BitFalseDesc>-</BitFalseDesc>
    </TBoolField>
```

```
      <TBoolField BitPos="4">
        <BitTrueDesc>Audiobook</BitTrueDesc>
        <BitFalseDesc>-</BitFalseDesc>
      </TBoolField>
      <TBoolField BitPos="5">
        <BitTrueDesc>Podcast</BitTrueDesc>
        <BitFalseDesc>-</BitFalseDesc>
      </TBoolField>
      <TBoolField BitPos="6">
        <BitTrueDesc>FilesInPlaylist</BitTrueDesc>
        <BitFalseDesc>-</BitFalseDesc>
      </TBoolField>
      <TBoolField BitPos="7">
        <BitTrueDesc>SecuredAudio</BitTrueDesc>
        <BitFalseDesc>-</BitFalseDesc>
      </TBoolField>
    </TBitField>
</PSParamType>
```

2.5.2 Number properties

This class is used to model functions consisting of a single parameter with a numeric type (See MOST Specification Rev 2.5, page 70).

A number example:



DTD Definition:

```
<!ELEMENT PNumber (PNParam)+>
<!ELEMENT PNParam (ParamName, ParamDescription, PNParamOPType,
          PNParamType?)>
<!ATTLIST PNParam details (true | false) #IMPLIED>
<!ELEMENT PNParamOPType ((PNCommand, PNReport?) | (PNReport, PNCommand?))>
<!ELEMENT PNCommand (PCmdSet?, PCmdGet?, PCmdSetGet?, PCmdIncrement?,
          PCmdDecrement?, PCmdGetInterface?)>
<!ELEMENT PNReport (PReportStatus?, PReportInterface?, PReportError?)>
<!ELEMENT PNParamType (TNumber | TVoid)>
…
<!ELEMENT TNumber (BasisDataType, Exponent, (RangeMin, RangeMax)?, Step,
          Unit)>
<!ELEMENT BasisDataType (TVoid | TUByte | TSByte | TUWord | TSWord | TULong
          | TSLong)>
<!ELEMENT Exponent (#PCDATA)>
<!ELEMENT RangeMin (#PCDATA)>
<!ELEMENT RangeMax (#PCDATA)>
<!ELEMENT Step (#PCDATA)>
<!ELEMENT Unit EMPTY>
<!ATTLIST Unit UnitRef IDREF #REQUIRED>
```

```
<!ELEMENT TUByte EMPTY>
<!ATTLIST TUByte TypeRef IDREF #REQUIRED>
<!ELEMENT TSByte EMPTY>
<!ATTLIST TSByte TypeRef IDREF #REQUIRED>
<!ELEMENT TUWord EMPTY>
<!ATTLIST TUWord TypeRef IDREF #REQUIRED>
<!ELEMENT TSWord EMPTY>
<!ATTLIST TSWord TypeRef IDREF #REQUIRED>
<!ELEMENT TULong EMPTY>
<!ATTLIST TULong TypeRef IDREF #REQUIRED>
<!ELEMENT TSLong EMPTY>
<!ATTLIST TSLong TypeRef IDREF #REQUIRED>
```

The PNParam element describes a parameter of the Number function class and the OPTypes that use it. The function class can make use of the following types (defined in PNParamType): TNumber and TVoid. The child element PNParamOPType defines the set of OPTypes for the function that make use of the parameter currently being defined. The command OPTypes available to Number properties are Set, Get, SetGet, Increment, Decrement, and GetInterface. The report OPTypes available are Status, Interface, and Error.
The TypeRef attribute selects the actual type, coding, and size of the number element. It is possible to select signed and unsigned values, byte (1 byte, i.e. values 0x00..0xFF or -0xF0..0xEF), word (2 bytes, values 0x0000…0xFFFF or -0xF000… 0xEFFF), and long (4 bytes, values 0x00000000…0xFFFFFFFF or -0xF0000000…0xEFFFFFFF).

The Step element is used whenever the given values should be dividable by a specific number, e.g. if only even numbers should be included in the valid range (i.e., 0, 2, 4, 6, 8, etc.) then step is set to 2. By giving values for RangeMin and RangeMax it is possible to limit the allowed values to a subset of the full range (inclusive, i.e., RangeMin is the smallest value allowed and RangeMax is the largest value allowed). The text replacement “full range” should not be used to signal full range, instead RangeMin and RangeMax instead should be omitted. Exponent is used to set the position of the decimal point.

Example:

Value	Exponent	Result
20	-2	0.2
	0	20
	2	2000

By giving a Unit type it is possible to select a unit that this number will have.

Example XML Code:

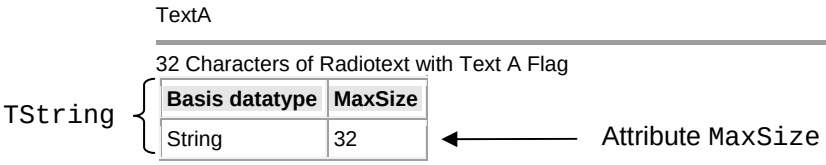
```
<FunctionClass ClassRef="class_number">
  <FunctionClassDesc>Number</FunctionClassDesc>
  <Property>
    <PNumber>
      <PNParam>
        <ParamName>NSteps</ParamName>
        <ParamDescription/>
        <PNParamOPType>
          <PNCommand>
            <PCmdGet OPTypeRef="PCmdSet">
              <ParamPos>1</ParamPos>
            </PCmdGet>
          </PNCommand>
        </PNParamOPType>
        <PNParamType>
```

```
            <TNumber>
              <BasisDataType>
                <TUByte TypeRef="type_unsigned_byte"/>
              </BasisDataType>
              <Exponent>0</Exponent>
              <RangeMin>1</RangeMin>
              <RangeMax>255</RangeMax>
              <Step>1</Step>
              <Unit UnitRef="unit_not_defined"/>
            </TNumber>
          </PNParamType>
        </PNParam>
      </PNumber>
  </Property>
</FunctionClass>
```

2.5.3 Text properties

This function class is used to represent functions consisting of a single text parameter (See MOST Specification Rev 2.5, page 72).

A string example:



DTD Definition:

```
<!ELEMENT PText (PTParam)+>
<!ELEMENT PTParam (ParamName, ParamDescription, PTParamOPType,
          PTParamType)>
<!ATTLIST PTParam details (true | false) #IMPLIED>
<!ELEMENT PTParamOPType ((PTCommand, PTReport?) | (PTReport, PTCommand?))>
<!ELEMENT PTCommand (PCmdSet?, PCmdGet?, PCmdSetGet?, PCmdGetInterface?)>
<!ELEMENT PTReport (PReportStatus?, PReportInterface?, PReportError?)>
<!ELEMENT PTParamType (TString | TVoid)>
…
<!ELEMENT TString EMPTY>
<!ATTLIST TString
   TypeRef IDREF #REQUIRED
   MaxSize CDATA #IMPLIED>
```

The `PTParam` element describes a parameter of the Text function class and the OPTypes that use it. The function class can make use of the following types (defined in `PTParamType`): `TString` and `TVoid`. The child element `PTParamOPType` defines the set of OPTypes for the function that make use of the parameter currently being defined. The command OPTypes available to Text properties are `Set`, `Get`, `SetGet`, and `GetInterface`. The report OPTypes available are `Status`, `Interface`, and `Error`.

`MaxSize` refers to the maximum number of characters used, not to the number of bytes. E.g., for a string encoded in UTF-16 the number of bytes used is at least twice as high as the number of characters.

Example XML Code:

```
<FunctionClass ClassRef="class_text">
  <FunctionClassDesc>Text</FunctionClassDesc>
  <Property>
    <PText>
      <!-- TASendername -->
      <PTParam>
        <ParamName>TASendername</ParamName>
        <ParamDescription/>
        <PTParamOPType>
          <PTCommand>
            <PCmdGet OPTypeRef="PCmdSet">
              <ParamPos>1</ParamPos>
            </PCmdGet>
          </PTCommand>
        </PTParamOPType>
        <PTParamType>
          <TString TypeRef="type_string" MaxSize="13"/>
        </PTParamType>
      </PTParam>
    </PText>
  </Property>
</FunctionClass>
```

2.5.4 Enumeration properties

An enumeration property (See MOST Specification Rev 2.5, page 72) is used if each possible value of the parameter has a specific separate meaning and when at the same time the specific separate meanings cannot occur simultaneously. This is true e.g. for a state engine with several states. Enumerated types assign property descriptions to a continuous range of values. The Enum contains a non-empty list of EnumValues. Each EnumValue has a Code (the actual value), a description corresponding to the value, and an optional symbolic name. The valid range of an EnumValue is 0x00 .. 0xFF.

An enum example:

VideoInteraction

Basis datatype	Range of values	Code	Description
Enum	0x00..0x08	0x00	Title
		0x01	Root
		0x02	Previous Node
		0x03	Next Node
		0x04	Last Position
		0x05	Up
		0x06	Down
		0x07	Right
		0x08	Left

TEnum

Attribute TEnumMax

Attribute Code of element TEnumValue

Text content of element TEnumValue

DTD Definition:

```
<!ELEMENT PEnum (PEParam)+>
<!ELEMENT PEParam (ParamName, ParamDescription, PEParamOPType,
          PEParamType)>
<!ATTLIST PEParam details (true | false) #IMPLIED>
```

```
<!ELEMENT PEParamOPType ((PECommand, PEReport?) | (PEReport, PECommand?))>
<!ELEMENT PECommand (PCmdSet?, PCmdGet?, PCmdSetGet?, PCmdIncrement?,
          PCmdDecrement?, PCmdGetInterface?)>
<!ELEMENT PEReport (PReportStatus?, PReportInterface?, PReportError?)>
<!ELEMENT PEParamType (TEnum | TNumber | TVoid)>
…
<!ELEMENT TEnum (TEnumValue)+>
<!ATTLIST TEnum TypeRef IDREF #REQUIRED TEnumMax CDATA #IMPLIED>
<!ELEMENT TEnumValue (#PCDATA)>
<!ATTLIST TEnumValue Code CDATA #REQUIRED
          SymbolicName CDATA #IMPLIED >
```

The `PEParam` element describes a parameter of the Enumeration function class and the OPTypes that use it. The function class can make use of the following types (defined in `PEParamType`): `TEnum`, `TNumber` and `TVoid`. The child element `PEParamOPType` defines the set of OPTypes for the function that make use of the parameter currently being defined. The command OPTypes available to Enumeration properties are `Set`, `Get`, `SetGet`, `Increment`, `Decrement`, and `GetInterface`. The report OPTypes available are `Status`, `Interface`, and `Error`.

`TEnumMax` stands for the highest value of `TEnumValue`
The `TEnumValue` code can be represented as a hex value or as a decimal value. The hex values must be preceded with “0x”. Hex values are the preferred representation.

Example XML Code:

```
<FunctionClass ClassRef="class_enumeration">
  <FunctionClassDesc>Enumeration</FunctionClassDesc>
  <Property>
    <PEnum>
      <!-- MagazineStatus -->
      <PEParam>
        <ParamName>MagazineStatus</ParamName>
        <ParamDescription>State of Player.</ParamDescription>
        <PEParamOPType>
          <PECommand>
            <PCmdGet OPTypeRef="PCmdSet">
              <ParamPos>1</ParamPos>
            </PCmdGet>
          </PECommand>
        </PEParamOPType>
        <PEParamType>
          <TEnum TypeRef="type_enum" TEnumMax="3">
            <TEnumValue Code="0x00">NoMagazine</TEnumValue>
            <TEnumValue Code="0x01">Magazine loaded</TEnumValue>
            <TEnumValue Code="0x02">DiskCheck</TEnumValue>
            <TEnumValue Code="0x03">DiskChange</TEnumValue>
          </TEnum>
        </PEParamType>
      </PEParam>
    </PEnum>
  </Property>
</FunctionClass>
```

2.5.5 BoolField properties

Properties of this class contain a number of bits that should either be used as flag field, or as controlling bits that are always manipulated together.

The command OPTypes available to BoolField properties are `Set`, `Get`, `SetGet`, and `GetInterface`. The report OPTypes available are `Status`, `Interface`, and `Error`.

Example XML Code:

```
<TBool TypeRef="type_unsigned_byte">
  <TBoolField BitPos="0">
    <BitTrueDesc>left door open</BitTrueDesc>
    <BitFalseDesc>left door closed</BitFalseDesc>
  </TBoolField>
  <TBoolField BitPos="1 ... 7">
    <BitTrueDesc>reserved</BitTrueDesc>
    <BitFalseDesc/>
  </TBoolField>
</TBool>
```

Instead of type_unsigned_byte (8 bits), it is also possible to use type_unsigned_word (16 bits) or type_unsigned_long (32 bits).

2.5.6 BitSet properties

Properties of this class are based on data type BitField. They contain a number of bits, which can be manipulated individually.

The command OPTypes available to BitSet properties are `Set`, `Get`, `SetGet`, and `GetInterface`. The report OPTypes available are `Status`, `Interface`, and `Error`.

Example XML Code:

```
<FunctionClass ClassRef="class_bitset">
  <FunctionClassDesc>BitSet</FunctionClassDesc>
  <Property>
    <PSwitch>
      …
      <PSParam>
        <ParamName>My_Bitfield_Param</ParamName>
        <ParamDescription/>
        <PSParamOPType>
          <PSCommand>
            <PCmdSet OPTypeRef="PCmdSet">
              <ParamPos>1</ParamPos>
            </PCmdSet>
          </PSCommand>
        </PSParamOPType>
        <PSParamType>
          <TBitField TypeRef="type_bitfield" Length="1">
            <TBoolField BitPos="0">
              <BitTrueDesc>reserved</BitTrueDesc>
              <BitFalseDesc/>
            </TBoolField>
            <TBoolField BitPos="">
              <BitTrueDesc/>
```

```
                <BitFalseDesc/>
              </TBoolField>
              …
            </TBitField>
          </PSParamType>
        </PSParam>
      </PSwitch>
   </Property>
</FunctionClass>
```

2.5.7 Container properties

Properties based on the function class “Container” are defined using the **`PContainer`** element (See MOST Specification Rev 2.5, page 76). The container function class uses the type classified stream to define a stream of data of a particular (e.g., MIME) type. The stream of data is prefixed by a media type identifier defined as a null terminated ASCII string.

Classified stream example:

CStreamParam

CStreamParam description

Basis datatype	MaxSize	MediaType
Classified Stream	100	`text/html`

Apart from the classified stream type, the Container function class supports the data types Stream and Short Stream.

```
<!ELEMENT PContainer (PCParam)+>
<!ELEMENT PCParam (ParamName, ParamDescription, PCParamOPType,
          PCParamType)>
<!ATTLIST PCParam details (true | false) #IMPLIED>
<!ELEMENT PCParamOPType ((PCCommand, PCReport?) | (PCReport, PCCommand?))>
<!ELEMENT PCCommand (PCmdSet?, PCmdGet?, PCmdSetGet?, PCmdGetInterface?)>
<!ELEMENT PCReport (PReportStatus?, PReportInterface?, PReportError?)>
<!ELEMENT PCParamType (TVoid | TStream | TCStream | TShortStream)>
…
<!ELEMENT TStream (StreamCase)*>
<!ATTLIST TStream
   TypeRef IDREF #REQUIRED
   Length CDATA #IMPLIED
     MinLength CDATA #IMPLIED
>

<!ELEMENT StreamCase ((PosDescription)?, ((StreamParam)* |
(StreamSignal)*))>
<!ATTLIST StreamCase
    ByteOrder (BigEndian | LittleEndian  )      #IMPLIED

<!ELEMENT StreamParam (ParamName, ParamDescription, ParamType)>
<!ELEMENT ParamName (#PCDATA)>
<!ATTLIST ParamName
   ParamIdx CDATA #IMPLIED
>

<!ELEMENT TCStream EMPTY>
<!ATTLIST TCStream
   Length CDATA "-"
   MediaType CDATA #IMPLIED
   TypeRef IDREF #REQUIRED>
```

```
<!ELEMENT TShortStream (StreamCase)*>
<!ATTLIST TShortStream
   TypeRef IDREF #REQUIRED
   MaxLength CDATA #IMPLIED>
```

The **PCParam** element describes a parameter of the Container function class and the OPTypes that use it. The function class can make use of the following types (defined in PCParamType): TCStream and TVoid. The command OPTypes available to Container properties (defined in PCParamOpType) are: Set, Get, SetGet, GetInterface. The report OPTypes available are Status, Interface, and Error.

If the attribute TypeRef is set to type_cstream, the attribute Length describes the maximum number of bytes that may be contained in the content of this stream, including the MediaType field. The optional attribute MediaType contains the MIME type (according to HTTP 1.1 specification) of the content transferred through this classified stream.

For in-depth descriptions please see the sections corresponding to the data types Stream (2.5.13) and Short Stream (2.5.15).

Example XML Code:

```
<FunctionClass ClassRef="class_container">
  <FunctionClassDesc>Container</FunctionClassDesc>
  <Property>
    <PContainer>
      <PCParam>
        <ParamName>CStreamParam</ParamName>
        <ParamDescription/>
        <PCParamOPType>
          <PCCommand>
            <PCmdSet OPTypeRef="PCmdSet">
              <ParamPos>1</ParamPos>
            </PCmdSet>
          </PCCommand>
          <PCReport>
            <PReportStatus OPTypeRef="PReportStatus">
              <ParamPos>1</ParamPos>
            </PReportStatus>
          </PCReport>
        </PCParamOPType>
        <PCParamType>
          <TCStream Length="100" TypeRef="type_cstream"
            MediaType=”text/html”/>
        </PCParamType>
      </PCParam>
    </PContainer>
  </Property>
</FunctionClass>
```

2.5.8 Record properties

This element contains the parameters and OPTypes of a function of function class „Record“. (See MOST Specification Rev 2.5, page 78).

```
<!ELEMENT PRecord ((PosDescription)*, (PRParam)+)>
<!ATTLIST PRecord NElements CDATA #IMPLIED>
<!ELEMENT PRParam (ParamName, ParamDescription, PRParamOPType,
          PRParamType)>
<!ATTLIST PRParam details (true | false) #IMPLIED>
<!ELEMENT PRParamOPType ((PRCommand, PRReport?) | (PRReport, PRCommand?))>
<!ELEMENT PRCommand (PCmdSet?, PCmdGet?, PCmdSetGet?, PCmdIncrement?,
          PCmdDecrement?, PCmdGetInterface?)>
<!ELEMENT PRReport (PReportStatus?, PReportInterface?, PReportError?)>
<!ELEMENT PRParamType (TRecord | TNumber | TVoid)>
…
<!ELEMENT TRecord (TRecordName?, TRecordDesc?, (TRecordField)+)>
<!ATTLIST TRecord
  TypeRef IDREF #REQUIRED
  NElements CDATA #IMPLIED>
<!ELEMENT TRecordName (#PCDATA)>
<!ELEMENT TRecordDesc (#PCDATA)>
<!ELEMENT TRecordField (TRecordFieldName, TRecordFieldDesc,
          TRecordFieldType)>
<!ATTLIST TRecordField FieldIdx CDATA #REQUIRED>
<!ELEMENT TRecordFieldName (#PCDATA)>
<!ELEMENT TRecordFieldDesc (#PCDATA)>
<!ELEMENT TRecordFieldType (TBool | TBitField | TNumber | TEnum | TString |
          TArray | TStream | TCStream | TShortStream)>
```

The attribute `NElements` of the element `PRecord` is redundant because the element `TRecord` has also an attribute `NElements`. It is highly recommended not to use the `NElements` attribute of `PRecord`. In a future version of the DTD, this attribute might be removed. The `PRecord` element contains an optional list of `PosDescription` elements. Usage of `PosX` and `PosY` for function class “Record” is described in MOST Specification Rev 2.5 on page 78.

The `PRParam` element describes the parameter of the record function class and the OPTypes that use it. The function class can make use of the following types (defined in `PRParamType`): `TRecord`, `TNumber` and `TVoid`. The child element `PRParamOPType` defines the set of OPTypes for the function that make use of the parameter currently being defined. The command OPTypes available to Record properties are `Set`, `Get`, `SetGet`, `Increment`, `Decrement`, and `GetInterface`. The report OPTypes available are `Status`, `Interface`, and `Error`.

2.5.8.1 One-dimensional records

A record as printed in the function catalog:

Data

The content of Data depends on parameter Pos={x,y}.

Basis datatype	Length	Description	
Stream	-	Pos	Data
		{ x=0 }	{TitlePosition, ChapterPosition}
		{ x=1 }	{TitlePosition}
		{ x=2 }	{ChapterPosition}

TitlePosition

Basis datatype	MaxSize
String	

ChapterPosition

Basis datatype	MaxSize
String	

In the above table, a record consists of two record elements, “TitlePosition” and “ChapterPosition”. As for arrays, the entire table of parameter “Data” is mapped to a list of `PosDescription` elements in XML. For one-dimensional records, only the `PosX` attribute of `PosDescription` is required and `PosY` must be assigned `#NULL#`. `PosX` contains the field index of the corresponding parameter. Indexes start at one, zero selects the entire record. Attribute `NElements` of elements `TRecord` and `PRecord` contains the number of fields in the record.

The record field parameters are mapped to `TRecordField` element children of element `TRecord`. Attribute `FieldIdx` of `TRecordField` is the field index in the record. Additional information is specified in subelements `TRecordFieldName`, `TRecordFieldDesc` and `TRecordFieldType`.

2.5.8.2 Two-dimensional records

According to the MOST Specification, a two-dimensional record must be a record of arrays, and one and only one record field may be an array. The main difference to one-dimensional records is the need for both attributes of element `PosDescription` in selecting data from the array. The MOST Specification defines how to employ X and Y values for selection. The attributes `PosX` and `PosY` are mapped in the same way as for arrays (see above) and can either specify a particular value or a range of values. Contrary to two-dimensional arrays, the `RecordFieldType` isn’t restricted to `TArray` for two-dimensional records, because the other fields are of different types. The record’s subparameters are mapped to element `TRecordField`, the array entry type to the `TArray`.

2.5.8.3 Element content of <PosDescription>

The elements of `PosDescription` represent the parameters and their order as applicable for a specific access to the corresponding data structures. The child elements of `PosDescription` are subject to the following rules:

1. Parameters are listed in braces “{}”. Extraneous text outside the braces is prohibited.
2. Parameters are separated by a comma “,”.
3. In complete descriptions of one-dimensional arrays, the first and last entries are given, separated by an ellipsis (“...”). For example, “{SimCard[1], ..., SimCard[NMax]}”.
4. In complete descriptions of arrays of records, the first and last record are given, separated by an ellipsis. An array position is specified for every record field. For example, “{MediaTitle[1], MediaType[1], MediaFilesystem[1], ..., MediaTitle[NMax], MediaType[NMax], MediaFilesystem[NMax]}”.
5. In complete descriptions of records, parameters are given without positions. For example, “{DiskTime, TrackTime, TitleTime}”.
6. In cyclic repetitions of stream parameters, the specific formatting rules of section 2.5.13.1 apply.

Example XML Code:

```
<FunctionClass ClassRef="class_record">
  <FunctionClassDesc>Record</FunctionClassDesc>
  <Property>
    <PRecord>
      <PosDescription PosX="0">{DiskTime, TrackTime,
        TitleTime}</PosDescription>
      <PosDescription PosX="1">{DiskTime}</PosDescription>
      <PosDescription PosX="2">{TrackTime}</PosDescription>
      <PosDescription PosX="3">{TitleTime}</PosDescription>
      <!-- Index -->
      <PRParam>
        <ParamName>Pos</ParamName>
        <ParamDescription>The parameter Pos={x,y} consists of two byte,
and shows which parameters shall be set, inquired, or read. Since this is
an unidimensional construction, the second Byte y is unused (y=0=const) and
the simplified notation Pos={x} is valid. Valid
range:x=0..3</ParamDescription>
        <PRParamOPType>
          …
        </PRParamOPType>
        <PRParamType>
          <TNumber>
            <BasisDataType>
              <TUWord TypeRef="type_unsigned_word"/>
            </BasisDataType>
            <Exponent>0</Exponent>
            <RangeMin>0</RangeMin>
            <RangeMax>3</RangeMax>
            <Step>1</Step>
            <Unit UnitRef="unit_none"/>
          </TNumber>
        </PRParamType>
      </PRParam>
      <!-- Data -->
      <PRParam>
        <ParamName>Data</ParamName>
        <ParamDescription>
          The content of Data depends on parameter Pos={x}.
        </ParamDescription>
        <PRParamOPType>
          …
        </PRParamOPType>
        <PRParamType>
          <TRecord TypeRef="type_record">
            …
          </TRecord>
        </PRParamType>
      </PRParam>
    </PRecord>
  </Property>
</Function>
```

2.5.9 Array properties

This element contains the parameters and OPTypes of a function of function class „Array“ (See MOST Specification Rev 2.5, page 80).

DTD Definition:

```
<!ELEMENT PArray ((PosDescription)*, (PAParam)+)>
<!ATTLIST PArray NMax CDATA #IMPLIED >
<!ELEMENT PosDescription (#PCDATA)>
<!ATTLIST PosDescription
             PosX CDATA #REQUIRED
             PosY CDATA "0">
<!ELEMENT PAParam (ParamName, ParamDescription, PAParamOPType,
          PAParamType)>
<!ATTLIST PAParam details (true | false) #IMPLIED>
<!ELEMENT PAParamOPType ((PACommand, PAReport?) | (PAReport, PACommand?))>
<!ELEMENT PACommand (PCmdSet?, PCmdGet?, PCmdSetGet?, PCmdIncrement?,
          PCmdDecrement?, PCmdGetInterface?)>
<!ELEMENT PAReport (PReportStatus?, PReportInterface?, PReportError?)>
<!ELEMENT PAParamType (TArray | TNumber | TVoid)>
…
<!ELEMENT TArray (TArrayName?, TArrayDesc?, TArrayElementType)>
<!ATTLIST TArray
   TypeRef IDREF #REQUIRED
   NMax CDATA #IMPLIED>
<!ELEMENT TArrayName (#PCDATA)>
<!ELEMENT TArrayDesc (#PCDATA)>
<!ELEMENT TArrayElementType (TBool | TBitField | TNumber | TEnum | TString
          | TArray | TRecord | TStream | TCStream | TShortStream)>
```

The attribute `NMax` of the element `PArray` defines the maximum length of the array. This attribute is redundant because the element `TArray` also has an attribute `NMax`. It is highly recommended not to use the `NMax` attribute of `PArray`. In a future version of the DTD, this attribute might be removed.

The element `PosDescription` contains a description of record elements, array elements or stream cases identified by attributes `PosX` and `PosY`. The description may contain several record or array elements - for example, `PosX` and `PosY` may select the entire array. Usage of `PosX` and `PosY` for function class “Array” is described in MOST Specification Rev 2.5 on page 81. See page 78 for function class “Record”.

The `PAParam` element describes the parameter of the array function class and the OPTypes that use it. The function class can make use of the following types (defined in `PAParamType`): `TArray`, `TNumber`, and `TVoid`. The child element `PAParamOPType` defines the set of OPTypes for the function that make use of the parameter currently being defined. The command OPTypes available to Array properties are `Set`, `Get`, `SetGet`, `Increment`, `Decrement`, and `GetInterface`. The report OPTypes available are `Status`, `Interface`, and `Error`.

The different array types (normal, dynamic and long) are modelled with the same set of elements in the XML document. For “LongArray” functions, only the mother array is described by this element. Further functions for long arrays are modelled as individual functions of the respective function class (e.g., CreateArrayWindow is a function of function class “Unclassified Method”).

The array parameters are specified in the `PAParam` element. As a rule, parameters for normal arrays include at least “Pos” or “Position”, “Data”, “ErrorCode” and “ErrorInfo”. Dynamic and long arrays require at least the parameters “Tag”, “PosY”, “Data”, “ErrorCode” and ErrorInfo”.

2.5.9.1 Normal arrays

The printed function catalog may contain two different kinds of array for a function of class array; one- and two-dimensional arrays. Arrays whose entries are themselves arrays or records are considered two-dimensional. They include individual parameters for every entry.

Every function of function class Array has a parameter to select certain data in the array. It is initially modelled as a normal parameter. Additionally, the `PosDescription` element establishes a relation between parameter value and selected data.

2.5.9.1.1 One-dimensional arrays

A one-dimensional array as printed in the function catalog:

Data

The content of Data depends on parameter Pos={x}.

Basis datatype	Length	Description	
Stream	-	SinkNr	Data
		{ x=0 }	{Offset[1], Offset[2], .., Offset[NMax]}
		{ x>0 }	{Offset[x]}

Offset

Gain/Attenuation in dB

Basis datatype	Exp.	Range of values	Step	Unit
Signed Byte	0	full range	1	dB

The “Data” parameter represents the array as such. Data may be selected from this array via the “Pos” parameter. In the printed catalog, the possible data selections are listed in a table. The XML document accomplishes this in a list of `PosDescription` elements below element `PArray`.

`PosDescription` has two attributes, `PosX` and `PosY`. For one-dimensional arrays, `PosY` is extraneous. It must be assigned the #NULL# value. The `PosX` attribute selects array elements (column “SinkNr” below column “Description”). The content of `PosDescription` describes the respective data (column “Data” below column “Description”).

For the above example, the first `PosDescription` would have a `PosX` of `0` and a content of `{Offset[1], Offset[2] ... Offset[NMax]}`. The second `PosDescription` would have a `PosX` of `>0` and contain `{Offset[x]}.`

The section heading “SinkNr” in the above table is the name of the parameter used in the selection of array entries. Normally, it is named “Pos”. Additionally, the `PArray` and `TArray` elements have an attribute `NMax` each. It contains the maximum array length. For the above example, attribute `NMax` should be assigned value Nmax from the description. The attribute Nmax from `TArray` should be used for this purpose.

The array entries themselves are described within the “Data” parameter, which is always of type Array. The `TArray` element specifies the array entry type with the `TArrayElementType` element. The elements `TArrayName` and `TArrayDescription` are used to describe the array entries. For the above example, `TArrayName` would be assigned the value “Offset” and `TArrayDescription` “Gain/Attenuation in dB”.

Note: *It is planned that in future releases of the DTD, the elements `TArrayName` and `TArrayDescription` are renamed to `TArrayElementName` and `TArrayElementDescription` to reflect their true usage.*

2.5.9.1.2 Two-dimensional arrays

A two-dimensional array as printed in the function catalog.

Data

The content of Data depends on parameter Pos={x,y}.

Basis datatype	Length	Description	
Stream	-	Pos	Data
		{ x=0, y=0 }	{EquGain[1], EquFrequency[1], EquQuality[1], ..., EquGain[NMax], EquFrequency[NMax], EquQuality[NMax]}
		{ x>0, y=0 }	{EquGain[x], EquFrequency[x], EquQuality[x]}
		{ x>0, y=1 }	{EquGain[x]}
		{ x>0, y=2 }	{EquFrequency[x]}
		{ x>0, y=3 }	{EquQuality[x]}

EquGain

Gain or attenuation

Basis datatype	Exp.	Range of values	Step	Unit
Signed Byte	0	full range	1	dB

EquFrequency

Center frequency

Basis datatype	Exp.	Range of values	Step	Unit
Unsigned Word	0	full range	1	Hz

EquQuality

Quality

Basis datatype	Exp.	Range of values	Step	Unit
Unsigned Byte	0	full range	1	none

The main difference between one-dimensional and two-dimensional arrays is the need for both attributes of element `PosDescription` in selecting data from the array. The MOST Specification defines how to employ X and Y values for selection. `PosY` is used in the same way as `PosX` and can be used to specify either a particular value, or a set of values. Another difference to one-dimensional arrays is the additional restriction on array element types. For two-dimensional arrays, only the types Record or Array are legal. The elements of the record or subordinate array must be declared as subparameters of the array. For a record, element `TRecord` accomplishes this. For arrays, use `TArray`.

2.5.9.2 Dynamic arrays

Dynamic arrays are always two-dimensional because they are forcibly arrays of records. They can be recognized in XML documents by their `ClassRef` attribute of element `FunctionClass` pointing to a `ClassDef` whose `ClassDefName` contains the text “Dynamic Array”.

The main difference between a dynamic array and other two-dimensional arrays is the replacing of `ParameterPos` with the parameters `Tag` and `PosY`, which serve to select array data. `Tag` is used to identify the “x” position within the array but instead of being based on the sequential position, `Tag` references a unique identifier for each row that is included as the first parameter of the row element. This has an important bearing on the `PosDescription` element list. The content of that element describes, as usual, the data selected by its attributes `PosX` and `PosY`, but the attribute allocation

differs. Because dynamic arrays use `Tag` instead of `PosX`, the `PosX` attribute is replaced by `Tag`. According to MOST Specification Rev 2.5 (page 83), “0” and “>0” compromise the only legal values.

Another difference to normal two-dimensional arrays concerns the “Data” parameter. As previously mentioned, it is always mapped to an array of `TRecord`, with the first field of the record, “Tag”, prescribed by the MOST Specification. This parameter must be described twice for dynamic arrays, as it occurs both in OPTypes and as a record field. Functions required for the processing of a dynamic array, e.g., “DynArrayIns” or “DynArrayDel”, are modelled separately.

2.5.9.3 Long arrays

A long array consists of a mother array and one or several array windows. Array windows are created dynamically at run time. They are defined in the MOST Specification. The function catalog does not convey array windows, only mother arrays and functions required by them. The mother array models the array as such. Its XML representation corresponds closely to that of a dynamic array. However, the `ClassRef` attribute must reference a different function, whose `ClassDef` has a `ClassDefName` containing the text “LongArray”.

The MOST Specification states that `GetInterface`, `Interface`, and `Error` comprise all legal OPTypes for mother arrays. The access OPTypes belong to the respective array window functions. Because windows aren’t modelled seperately in XML, their OPTypes must be listed as part of the mother array. They must be clearly distinct from mother window OPTypes, to allow selection of the correct function ID. For a list of legal OPTypes, see the MOST Specification.

Example XML Code:

```
<FunctionClass ClassRef="class_array">
  <FunctionClassDesc>Array of { Number } </FunctionClassDesc>
  <Property>
    <PArray>
      <PosDescription PosX="0" PosY="#NULL#">
        {Offset[1], Offset[2], ..., Offset[NMax]}
      </PosDescription>
      <PosDescription PosX="&gt;0" PosY="#NULL#">
        {Offset[x]}
      </PosDescription>
      <!-- Index -->
      <PAParam>
        <ParamName>Pos</ParamName>
        <ParamDescription>The parameter Pos={x,y} consists of two bytes x
and y and shows which parameter shall be set, inquired or read. Since this
is an unidimensional construction, the second Byte y is unused (y=0=const)
and  the simplified notation Pos={x} is valid. The byte x represents in
this case the OutputNo. Valid range: x=0..Nmax, y=0</ParamDescription>
        <PAParamOPType>
          <PACommand>
            <PCmdSet OPTypeRef="PCmdSet">
              <ParamPos>1</ParamPos>
            </PCmdSet>
          </PACommand>
          <PAReport>
            <PReportStatus OPTypeRef="PReportStatus">
              <ParamPos>1</ParamPos>
            </PReportStatus>
          </PAReport>
        </PAParamOPType>
        <PAParamType>
          <TNumber>
```

```
            …
          </TNumber>
        </PAParamType>
      </PAParam>
      <!-- Data -->
      <PAParam>
        <ParamName>Data</ParamName>
        <ParamDescription>
          The content of Data depends on parameter Pos={x}.
        </ParamDescription>
        <PAParamOPType>
          <PACommand>
            <PCmdSet OPTypeRef="PCmdSet">
              <ParamPos>2</ParamPos>
            </PCmdSet>
          </PACommand>
          <PAReport>
            <PReportStatus OPTypeRef="PReportStatus">
              <ParamPos>2</ParamPos>
            </PReportStatus>
          </PAReport>
        </PAParamOPType>
        <PAParamType>
          <TArray TypeRef="type_array">
            <TArrayName>Offset</TArrayName>
            <TArrayDesc>Gain/Attenuation in dB</TArrayDesc>
            <TArrayElementType>
              <TNumber>
                …
              </TNumber>
            </TArrayElementType>
          </TArray>
        </PAParamType>
      </PAParam>
    </PArray>
  </Property>
</FunctionClass>
```

2.5.9.3.1 Array Window

According to the MOST Specification, ArrayWindow is not a function class. In the context of the DTD description, however, it is necessary to distinguish between the mother array and its array windows. Thus, the ArrayWindow class was introduced:

```
<ClassDef ClassID="class_array_window">
      <ClassDefName>ArrayWindow</ClassDefName>
      <ClassDefDesc>#NULL#</ClassDefDesc>
</ClassDef>
```

2.5.10 Map properties

The structure of the function class Map is similar to DynamicArray; however, no ordering of the elements may be assumed. It is optimized for arrays with dynamic changes through both the Controller and the Slave. By not assuming an order of the lines in a Map, the communication overhead for notifications can be minimized.

The XML representation of Map is identical to that of DynamicArray. However, the ClassRef attribute must reference a different function, whose ClassDef has a ClassDefName containing the text “Map”.

The command OPTypes available to Map properties are `Set`, `Get`, `SetGet`, `Increment`, `Decrement`, and `GetInterface`. The report OPTypes available are `Status`, `Interface`, and `Error`.

2.5.11 Sequence properties

Properties based on the function class “Sequence” (See MOST Specification Rev 2.5, page 99) are defined using the `PSequence` element. This class should be used when the property consists of a number of parameters of varying types.

Sequence class example:

Function class: Sequence

FBlock	Function	OPType	Parameter
NavigationStd (0x82)	Pos_Position (0x243)	Set	PositionReliability,Longitude,Latitude,Heading,Speed
		Get	
		GetInterface	
		Status	PositionReliability,Longitude,Latitude,Heading,Speed
		Interface	Flags, Class, OPTypes, Name, Units, DataType, Exponent, Min, Max, Step, Class, OPTypes, Name, Units, DataType, Exponent, Min, Max, Step, Class, OPTypes, Name, Units, DataType, Exponent, Min, Max, Step, Class, OPTypes, Name, Units, DataType, Exponent, Min, Max, Step, Class, OPTypes, Name, Units, DataType, Exponent, Min, Max, Step
		Error	ErrorCode,ErrorInfo

DTD Definition:

```
<!ELEMENT PSequence (PQParam)+>
<!ELEMENT PQParam (ParamName, ParamDescription, PQParamOPType,
          PQParamType)>
<!ATTLIST PQParam details (true | false) #IMPLIED>
<!ELEMENT PQParamOPType ((PQCommand, PQReport?) | (PQReport, PQCommand?))>
<!ELEMENT PQCommand (PCmdSet?, PCmdGet?, PCmdSetGet?, PCmdGetInterface?)>
<!ELEMENT PQReport (PReportStatus?, PReportInterface?, PReportError?)>
<!ELEMENT PQParamType (TBool | TBitField | TNumber | TEnum | TString |
          TStream | TCStream | TShortStream | TVoid)>
```

The `PQParam` element describes a parameter of the Sequence function class and the OPTypes that use it. The function class can make use of the following types (defined in `PQParamType`): `TBool`, `TBitField`, `TNumber`, `TEnum`, `TString`, `TStream`, `TCStream`, `TShortStream`, and `TVoid` (see previous sections). The following OPTypes may make use of a sequence class parameter (defined in PQParamOpType): `Set`, `Get`, `SetGet`, `GetInterface`, `Status`, `Interface`, and `Error`. The `ParamIdx` attribute of the `ParamName` element is used to indicate the position of the parameter in the global sequence. Note the `ParamIdx` attribute is used slightly differently in combination with the `TStream` data type.

Example XML Code:

```
<FunctionClass ClassRef="class_sequence">
  <FunctionClassDesc>Sequence</FunctionClassDesc>
  <Property>
    <PSequence>
      <PQParam>
        <ParamName ParamIdx="1">PositionReliability</ParamName>
        <ParamDescription>Reliability Indicator</ParamDescription>
        <PQParamOPType>
          <PQCommand>
            <PCmdSet OPTypeRef="PCmdSet">
              <ParamPos>1</ParamPos>
            </PCmdSet>
          </PQCommand>
          <PQReport>
            <PReportStatus OPTypeRef="PReportStatus">
              <ParamPos>1</ParamPos>
            </PReportStatus>
          </PQReport>
        </PQParamOPType>
        <PQParamType>
          <TNumber>
            <BasisDataType>
              <TUWord TypeRef="type_unsigned_word"/>
            </BasisDataType>
            <Exponent>0</Exponent>
            <Step>1</Step>
            <Unit UnitRef="unit_none"/>
          </TNumber>
        </PQParamType>
      </PQParam>
      <PQParam>
        <ParamName ParamIdx="2">Longitude</ParamName>
        <ParamDescription>
          Longitude of an absolute position.
        </ParamDescription>
        <PQParamOPType>
          …
        </PQParamOPType>
        <PQParamType>
          <TNumber>
            <BasisDataType>
              <TSLong TypeRef="type_signed_long"/>
            </BasisDataType>
            <Exponent>0</Exponent>
            <Step>1</Step>
            <Unit UnitRef="unit_none"/>
          </TNumber>
        </PQParamType>
      </PQParam>
      …
    </PSequence>
  </Property>
</FunctionClass>
```

2.5.12 Unclassified properties

This element contains the parameters and OPTypes of a function of function class „Unclassified property“. Unclassified properties should only be used when the property in question cannot easily be encapsulated in one of the predefined function classes.

DTD Definition:

```
<!ELEMENT PUnclassified (PUParam)+>
<!ATTLIST Punclassified Length CDATA #IMPLIED>
<!ELEMENT PUParam (ParamName, ParamDescription, PUParamOPType,
      PUParamType)>
<!ATTLIST PUParam details (true | false) #IMPLIED>
<!ELEMENT PUParamOPType((PUCommand, PUReport?)|(PUReport, PUCommand?))>
<!ELEMENT PUCommand (PCmdSet?, PCmdGet?, PCmdSetGet?, PCmdIncrement?,
          PCmdDecrement?, PCmdGetInterface?)>
<!ELEMENT PUReport (PReportStatus?, PReportInterface?, PReportError?)>
<!ELEMENT PUParamType (TBool | TNumber | TString | TEnum | TBitField |
      TStream | TCStream | TShortStream | TRecord | TArray | TVoid)>
<!ELEMENT ParamName (#PCDATA)>
<!ATTLIST ParamName ParamIdx CDATA #IMPLIED>
<!ELEMENT ParamDescription (#PCDATA)>
```

The element `PUnclassified` contains a non-empty list of parameters. The attribute `Length` defines the length of the property. However, this information can be derived from the individual parameters and is therefore left optional. The element `PUParam` defines a parameter of an unclassified property, including the OPTypes used. The attribute `details` is used to block the generation of detailed parameter descriptions, for example when an `ErrorCode` or `ErrorInfo` parameter is being defined.

The child element `ParamName`, used across all function classes, describes contains the parameter name. The attribute `ParamIdx` describes the position of a parameter in a sequence. ParamIdx starts with 1. Because the parameter position may be chosen individually for each OPType, this attribute is suitable for streams (subparameter selection via `ParamName`) and for the function classes “Sequence Property” and “Sequence Method” (overall sequence of the parameters for use when transmitting the function interface). The child element `ParamDescription` contains a description of the parameter.

The child element `PUParamOPType` defines the set of OPTypes for the function that make use of the parameter currently being defined. The command OPTypes available to unclassified properties are `Set`, `Get`, `SetGet`, `Increment`, `Decrement`, and `GetInterface`. The report OPTypes available are `Status`, `Interface`, and `Error`. Each OPType is optional. In other words, the property can be defined to make use of any possible subset of command and report OPTypes. Unclassified functions are not restricted with regard either to OPTypes or parameters. The DTD reflects this.

The child element `PUParamType` defines a parameter type and its subparameter types, if applicable and can reference the following types: `TStream`, `TArray`, `TRecord`, `TBool`, `TNumber`, `TEnum`, `TString`, `TBitField`, `TCStream`, or `TVoid`.

Note: *According to the MOST Specification, the function classes Array and Record require the presence of the Pos and Data parameters. An Array or Record fragment is a parameter Data of type* `TArray` *or* `TRecord` *without the required Pos parameter.*
When using fragments of Arrays and Records in Unclassified Properties or Unclassified Methods, it is impossible for any tool to recognize the data structure.

Example XML Code:

```
<FunctionClass ClassRef="class_unclassified_property">
  <FunctionClassDesc>Unclassified Property </FunctionClassDesc>
  <Property>
^   <PUnclassified>
      <!-- void -->
      <PUParam details="false">
        <ParamName></ParamName>
        <ParamDescription />
        <PUParamOPType>
          <PUCommand>
            <PCmdGet OPTypeRef="PCmdGet">
              <ParamPos>0</ParamPos>
            </PCmdGet>
          </PUCommand>
          <PUReport />
        </PUParamOPType>
        <PUParamType>
          <TVoid TypeRef="type_void" />
        </PUParamType>
      </PUParam>
      <!-- SinkCount -->
      <PUParam>
        <ParamName>SinkCount</ParamName>
        <ParamDescription />
        <PUParamOPType>
          <PUCommand />
          <PUReport>
            <PReportStatus OPTypeRef="PReportStatus">
              <ParamPos>2</ParamPos>
            </PReportStatus>
          </PUReport>
        </PUParamOPType>
        <PUParamType>
          <TNumber>
            <BasisDataType>
              <TUByte TypeRef="type_unsigned_byte" />
            </BasisDataType>
            <Exponent>0</Exponent>
            <Step>1</Step>
            <Unit UnitRef="unit_none" />
          </TNumber>
        </PUParamType>
      </PUParam>
    </PUnclassified>
  </Property>
</FunctionClass>
```

2.5.13 Use of the TStream type

The stream type may be used to describe parameters within unclassified function classes and records. This section describes how to define parameters of type TStream. The printed Function Catalog contains essentially two kinds of streams: those described in a table with one line, and those described in multi-line tables. One-line tables do not distinguish different stream cases, the corresponding stream is always composed the same way and thus called a “simple stream”. Streams with cases are called “complex streams”.

The stream parameters are mapped to the element `TStream`. Its `Length` attribute contains the number of corresponding table contents. `TStream` may contain a list of `StreamCase` elements, every one of which describes a line of the table. Additionally, the `PosDescription` element contains information about the table.

The individual parameters of a `StreamCase` are defined through `StreamParam` elements. Their relative position within a stream case is defined with the attribute `ParamIdx` of `ParamName`.

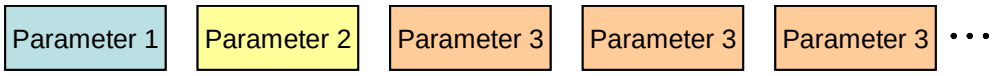
Stream parameters may recur cyclically. This is reflected in the following conventions:

1. Unique parameters are numbered sequentially, starting at one.
2. If one or a block of parameters repeat cyclically, all repeating parameter indices are suffixed with „seq“ and the number of repeating parameters. Repetition is in-order and may be selective. Examples:

   Stream with three parameters, third parameter repeating cyclically.

   Parameter 1: ParamIdx=“1“
   Parameter 2: ParamIdx=“2“
   Parameter 3: ParamIdx=“3seq1“

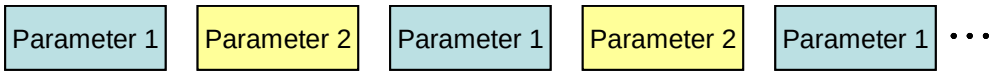
   Transmission sequence:

   Stream with two parameters, both repeating cyclically:

   Parameter 1: ParamIdx=“1seq2“
   Parameter 2: ParamIdx=“2seq2“

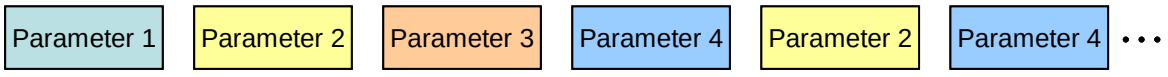
   Transmission sequence:

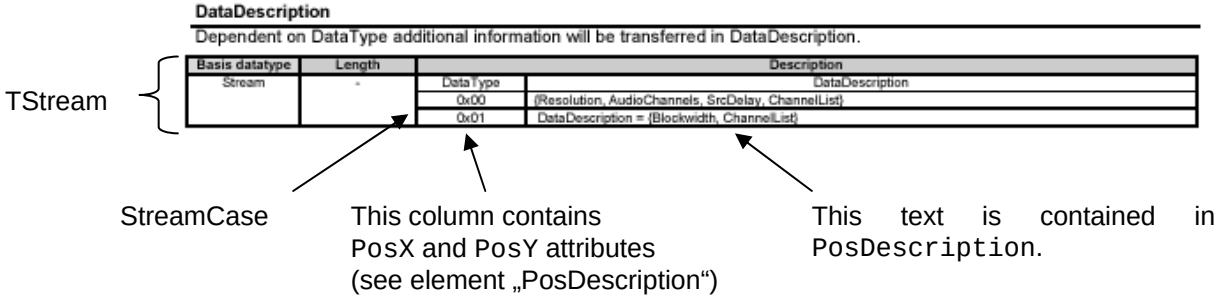
   Stream with four parameters, second and fourth repeating cyclically:

   Parameter 1: ParamIdx=“1“
   Parameter 2: ParamIdx=“2seq2“
   Parameter 3: ParamIdx=“3“
   Parameter 4: ParamIdx=“4seq2“

   Transmission sequence:

A stream example:



DTD Definition:

```
<!ELEMENT TStream (StreamCase)*>
<!ATTLIST TStream
  TypeRef IDREF #REQUIRED
  Length CDATA #IMPLIED
    MinLength CDATA #IMPLIED >

<!ELEMENT StreamCase ((PosDescription)?, ((StreamParam)*|(StreamSignal)*))>
<!ATTLIST StreamCase
   ByteOrder (BigEndian | LittleEndian  )      #IMPLIED
>

<!ELEMENT StreamParam (ParamName, ParamDescription, ParamType)>
<!ELEMENT PosDescription (#PCDATA)>
<!ATTLIST PosDescription PosX CDATA #REQUIRED PosY CDATA "0">
…
<!ELEMENT ParamType (TBool | TBitField | TNumber | TEnum | TString |
      TStream | TCStream | TShortStream | TVoid)>
```

`TStream` contains a number of `StreamCase` elements, each of which is an individual stream case with its own parameters. A stream case is uniquely identified by its `PosDescription`. The content of `PosDescription` must be defined in the context of a specific stream and may, e.g., be dependent on some other parameter in the function. A `PosDescription` of `#NULL#` is used when there is only one stream case in this `TStream`. Each `StreamCase` consists of one or more `StreamParam` elements which themselves can contain a type in the `ParamType` element.
If the `ByteOrder` attribute is missing, the default value is `BigEndian`.

Example XML code:

```
<TStream Length="-" TypeRef="type_stream">
  <StreamCase>
    <PosDescription PosX="#NULL#" PosY="#NULL#">
      DeviceID {,DeviceID}
    </PosDescription>
    <StreamParam>
      <ParamName ParamIdx="1seq1">DeviceID</ParamName>
      <ParamDescription>
        Rx/TxLog of a device or group address
      </ParamDescription>
      <ParamType>
        <TNumber>
          <BasisDataType>
            <TUWord TypeRef="type_unsigned_word"/>
          </BasisDataType>
          <Exponent>0</Exponent>
```

```
            <RangeMin/>
            <RangeMax/>
            <Step>1</Step>
            <Unit UnitRef="unit_none"/>
          </TNumber>
        </ParamType>
      </StreamParam>
    </StreamCase>
</TStream>
```

2.5.13.1 Simple streams

A simple stream example:

FktIDList

List of functions with a maximum of 4.

Basis datatype	Length	Description
Stream	-	FktIDList ::= <FktID> {, <FktID>}

FktID

Function

Basis datatype	Exp.	Range of values	Step	Unit
Unsigned Word	0	Full range	1	none

Mapping a simple stream to XML is straightforward, as there are no cases to distinguish. The simple stream's `TStream` element either contains no `StreamCase` or exactly one `StreamCase` with exactly one `PosDescription`. The latter corresponds exactly to the description column of the above table. Its unnecessary attributes `PosX` and `PosY` must be assigned the `#NULL#` value.

The individual parameters are mapped to `StreamParam` children of the unique `StreamCase` element. In the example mentioned above, there is only one parameter. `StreamParam` has child elements `ParamName`, `ParamDesc` and `ParamType` to specify the parameter in detail.

For cyclically recurring parameters, the `PosDescription` must conform to the following format:

```
      StreamName::=<Param1>, <Param2>, ..., <ParamX>{, <Param1>, <Param2>,
..., <ParamX>}
```

As explained in section 2.1.3, this content must be escaped, e.g. encapsulated in a CDATA section. Cyclically repeated parameters must be specified again within braces, again separated by commas.

Part of the XML-Document for the above example:

```
<TStream Length="-" TypeRef="Stream">
  <StreamCase>
    <PosDescription PosX="#NULL#" PosY="#NULL#">
      FktIDList::=FktID{,FktID}
    </PosDescription>
    <StreamParam>
      <ParamName ParamIdx="1seq1">FktID</ParamName>
      <ParamDescription>Function</ParamDescription>
      <ParamType>
          <TNumber>
          <BasisDataType>
            <TUWord TypeRef="type_unsigned_word"></TUWord>
```

```
            </BasisDataType>
            <Exponent></Exponent>
            <RangeMin></RangeMin>
            <RangeMax></RangeMax>
            <Step></Step>
            <Unit UnitRef="unit_none"></Unit>
          </TNumber>
        </ParamType>
      </StreamParam>
   </StreamCase>
</TStream>
```

2.5.13.2 Complex streams

A complex stream as printed in the Function Catalog:

DataDescription

Depending on DataType, additional information will be transported in DataDescription.

Basis datatype	Length	Description	
Stream	-	DataType	DataDescription
		0x00	{Resolution, AudioChannels, SrcDelay, ChannelList}
		0x01	{Blockwidth, ChannelList}

A complex stream differs from a simple stream in having a multi-line description table. The basis datatype and length column are the same as for simple streams, but the description column contains a sub-table.

In the above example, the sub-table has two columns, “DataType” and “DataDescription”. Data description contains the parameter name of the stream. Data type is the parameter used by the stream to determine the stream case.

This example contains two stream cases for data types 0x00 and 0x01. They are mapped to two `StreamCase` child elements of the corresponding `TStream`. The table row is described via the `PosDescription` element as for simple streams, but now both the `PosX` and `PosY` attributes are required. `PosX` contains the parameter name of the selection parameter and must be equal for all `PosDescription` elements of all stream cases of a `TStream`. `PosY` contains a description of the parameter value for the corresponding stream case. The parameter specification is the same as for simple streams.

This may cause problems. If the same parameter occurs for different stream cases, it must be described several times in the XML document, once for every stream case. This is necessary to obtain a valid DTD instance and allows correct assignment of stream case parameters. An application may assume two stream parameters for different stream cases to be completely equal if they are equal in name.

The child elements of `PosDescription` must conform to one of these choices:

1. A stream case consists of a simple parameter list. In this case, the `PosDescription` contents must conform to the mapping rules for records and arrays (see 2.5.8.3).
2. A stream case has cyclically repeating parameters. In this case, the `PosDescription` contents must comply with the mapping rules for simple streams (see 2.5.13.1).

Example XML code:

```
<TStream TypeRef="type_stream">
  <StreamCase>
    <PosDescription PosX="DataType" PosY="0x00">
      {Resolution, AudioChannels, SrcDelay, ChannelList}
    </PosDescription>
    <StreamParam>
      <ParamName ParamIdx="1">Resolution</ParamName>
      <ParamDescription>
        Resolution of the AudioSamples in byte.
      </ParamDescription>
      <ParamType>
        <TNumber>
          <BasisDataType>
            <TUByte TypeRef="type_unsigned_byte" />
          </BasisDataType>
```

```
            <Exponent>0</Exponent>
            <Step>1</Step>
            <Unit UnitRef="unit_not_defined" />
          </TNumber>
        </ParamType>
      </StreamParam>
      <StreamParam>
        <ParamName ParamIdx="2">AudioChannels</ParamName>
        <ParamDescription>Number of audio channels.</ParamDescription>
        <ParamType>
          <TNumber>
            <BasisDataType>
              <TUByte TypeRef="type_unsigned_byte" />
            </BasisDataType>
            <Exponent>0</Exponent>
            <Step>1</Step>
            <Unit UnitRef="unit_none" />
          </TNumber>
        </ParamType>
      </StreamParam>
      <StreamParam>
        <ParamName ParamIdx="3">SrcDelay</ParamName>
        <ParamDescription>
          Delay of synchronous data related to the Timing Master. Remark:
          The parameter SrcDelay represents the register NDR
        </ParamDescription>
        <ParamType>
          <TNumber>
            <BasisDataType>
              <TUByte TypeRef="type_unsigned_byte" />
            </BasisDataType>
            <Exponent>0</Exponent>
            <Step>1</Step>
            <Unit UnitRef="unit_not_defined" />
          </TNumber>
        </ParamType>
      </StreamParam>
      <StreamParam>
        <ParamName ParamIdx="4">ChannelList</ParamName>
        <ParamDescription>List of particular  channels.</ParamDescription>
        <ParamType>
          <TStream Length="60" TypeRef="type_stream">
            <StreamCase>
              <PosDescription PosX="#NULL#" PosY="#NULL#">
                &lt; Channel&gt;{,&lt;Channel&gt;}
              </PosDescription>
              <StreamParam>
                <ParamName ParamIdx="1seq1">Channel</ParamName>
                <ParamDescription>Number of a channel</ParamDescription>
                <ParamType>
                  <TNumber>
                    <BasisDataType>
                      <TUByte TypeRef="type_unsigned_byte" />
                    </BasisDataType>
                    <Exponent>0</Exponent>
                    <RangeMin>0</RangeMin>
                    <RangeMax>59</RangeMax>
                    <Step>1</Step>
                    <Unit UnitRef="unit_none" />
                  </TNumber>
```

```
                </ParamType>
              </StreamParam>
            </StreamCase>
          </TStream>
        </ParamType>
      </StreamParam>
    </StreamCase>
    <StreamCase>
      <PosDescription PosX="DataType" PosY="0x01">
        DataDescription={BlockWidth, ChannelList}
      </PosDescription>
      <StreamParam>
        <ParamName ParamIdx="1">BlockWidth</ParamName>
        <ParamDescription>
          Number of transferred byte per MOST.
        </ParamDescription>
        <ParamType>
          <TNumber>
            <BasisDataType>
              <TUByte TypeRef="type_unsigned_byte" />
            </BasisDataType>
            <Exponent>0</Exponent>
            <Step>1</Step>
            <Unit UnitRef="unit_not_defined" />
          </TNumber>
        </ParamType>
      </StreamParam>
      <StreamParam>
        <ParamName ParamIdx="2">ChannelList</ParamName>
        <ParamDescription>List of particular  channels.</ParamDescription>
        <ParamType>
          <TStream Length="60" TypeRef="type_stream">
            <StreamCase>
              <PosDescription PosX="#NULL#" PosY="#NULL#">
                &lt;Channel&gt;{,&lt;Channel&gt;}
              </PosDescription>
              <StreamParam>
                <ParamName ParamIdx="1seq1">Channel</ParamName>
                <ParamDescription>Number of a channel</ParamDescription>
                <ParamType>
                  <TNumber>
                    <BasisDataType>
                      <TUByte TypeRef="type_unsigned_byte" />
                    </BasisDataType>
                    <Exponent>0</Exponent>
                    <RangeMin>0</RangeMin>
                    <RangeMax>59</RangeMax>
                    <Step>1</Step>
                    <Unit UnitRef="unit_none" />
                  </TNumber>
                </ParamType>
              </StreamParam>
            </StreamCase>
          </TStream>
        </ParamType>
      </StreamParam>
    </StreamCase>
</TStream>
```

2.5.13.2.1 Bit-encoded Stream Cases

In certain applications, it might be required to determine the content of the stream even more freely than the multi-line description table allows. This usually is the case when arbitrary combinations of stream parameters have to be created dynamically.

Bit-encoded stream cases provide the solution:

```
<PosDescription PosX="ResultColumns" PosY="#BitEncoded#" />
```

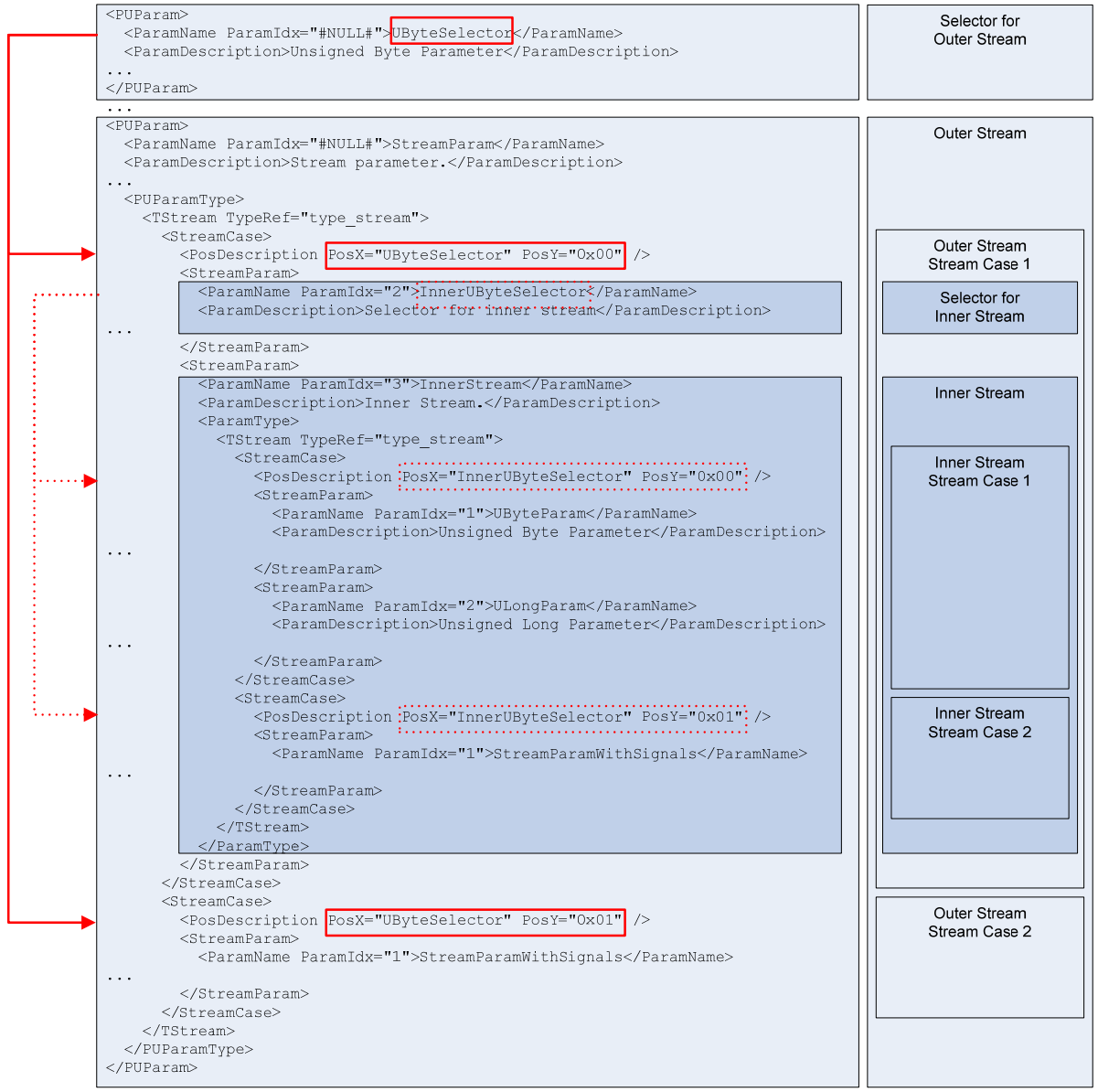
This can be interpreted as:

- Use the Value of predefined parameter “ResultColumns” as selector (quite traditional usage of PosX)

- The value of PosY (“#BitEncoded#”) indicates the different interpretation of the PosX value. It means that “ResultColumns” has to be interpreted bitwise. If bit n is set, the StreamParam with index n will be within the Stream data.

2.5.13.2.2 Nested Streams

In nested streams, the discriminator of an embedded complex stream has to be outside the stream; it may, however, be a parameter of the outer stream.




```
<PUParam>
  <ParamName ParamIdx="#NULL#">UByteSelector</ParamName>
  <ParamDescription>Unsigned Byte Parameter</ParamDescription>
...
</PUParam>
...
<PUParam>
  <ParamName ParamIdx="#NULL#">StreamParam</ParamName>
  <ParamDescription>Stream parameter.</ParamDescription>
...
  <PUParamType>
    <TStream TypeRef="type_stream">
      <StreamCase>
        <PosDescription PosX="UByteSelector" PosY="0x00" />
        <StreamParam>
          <ParamName ParamIdx="2">InnerUByteSelector</ParamName>
          <ParamDescription>Selector for inner stream</ParamDescription>
...
        </StreamParam>
        <StreamParam>
          <ParamName ParamIdx="3">InnerStream</ParamName>
          <ParamDescription>Inner Stream.</ParamDescription>
          <ParamType>
            <TStream TypeRef="type_stream">
              <StreamCase>
                <PosDescription PosX="InnerUByteSelector" PosY="0x00" />
                <StreamParam>
                  <ParamName ParamIdx="1">UByteParam</ParamName>
                  <ParamDescription>Unsigned Byte Parameter</ParamDescription>
...
                </StreamParam>
                <StreamParam>
                  <ParamName ParamIdx="2">ULongParam</ParamName>
                  <ParamDescription>Unsigned Long Parameter</ParamDescription>
...
                </StreamParam>
              </StreamCase>
              <StreamCase>
                <PosDescription PosX="InnerUByteSelector" PosY="0x01" />
                <StreamParam>
                  <ParamName ParamIdx="1">StreamParamWithSignals</ParamName>
...
                </StreamParam>
              </StreamCase>
            </TStream>
          </ParamType>
        </StreamParam>
      </StreamCase>
      <StreamCase>
        <PosDescription PosX="UByteSelector" PosY="0x01" />
        <StreamParam>
          <ParamName ParamIdx="1">StreamParamWithSignals</ParamName>
...
        </StreamParam>
      </StreamCase>
    </TStream>
  </PUParamType>
</PUParam>
```

2.5.13.3 Element Stream Signal <StreamSignal>

The Stream Signal element is used to split the stream into a number of elements (bit groups) with an arbitrary bit length.

Note: *The whole stream has to be byte-aligned according to the MOST Specification.*
SignalIdx starts with 1.

DTD Definition:

```
<!ELEMENT StreamSignal (SignalName, SignalDescription, SignalBitLength)>
<!ATTLIST StreamSignal
  Signedness (Unsigned | Signed)  #IMPLIED
>
<!ELEMENT SignalName (#PCDATA)>
<!ATTLIST SignalName SignalIdx CDATA #IMPLIED>
<!ELEMENT SignalDescription (#PCDATA)>
<!ELEMENT SignalBitLength (#PCDATA)>
```

Example XML Code:

```
<PUParamType>
  <TStream Length="-" TypeRef="type_stream">
    <StreamCase>
      <PosDescription PosX="#NULL#" PosY="#NULL#"/>
      <StreamSignal>
        <SignalName SignalIdx="1">MyFirst3BitVar</SignalName>
        <SignalDescription>Var_3Bit_No1</SignalDescription>
        <SignalBitLength>3</SignalBitLength>
      </StreamSignal>
      <StreamSignal>
        <SignalName SignalIdx="2">MySecond3BitVar</SignalName>
        <SignalDescription>Var_3Bit_No2</SignalDescription>
        <SignalBitLength>3</SignalBitLength>
      </StreamSignal>
      <StreamSignal>
        <SignalName SignalIdx="3seq1">MyFirst4BitVar</SignalName>
        <SignalDescription>Var_4Bit_No1</SignalDescription>
        <SignalBitLength>4</SignalBitLength>
      </StreamSignal>
    </StreamCase>
  </TStream>
</PUParamType>
```

2.5.14 Element Classified Stream <TCStream>

Classified Stream acts as a container for different objects. It consists of a Length field, a MediaType and the actual content.
TCStream is based on HTTP/1.1 and therefore does not include any stream cases. TCStream can, for example, be used to transport images, audio files, or files of unknown type in an *arbitrary byte stream*.

DTD Definition:

```
<!ELEMENT TCStream EMPTY>
<!-- Changed Length to MaxLength, 2006-07-17 -->
<!ATTLIST TCStream
   TypeRef IDREF #REQUIRED
   MaxLength CDATA #IMPLIED
   MediaType CDATA #IMPLIED
>
```

2.5.15 Element Short Stream <TShortStream>

The short stream has a maximum length of 255 bytes. The actual length of the short stream is indicated by a leading single byte value.

DTD Definition:

```
<!ELEMENT TShortStream (StreamCase)*>
<!ATTLIST TShortStream
   TypeRef IDREF #REQUIRED
   MaxLength CDATA #IMPLIED
>
```

A short stream can be structured to have stream cases, just like TStream. It may contain a simple stream or a complex stream, with StreamParams and StreamSignals.

2.6 Methods

This section contains further information on function classes describing “methods”.

DTD Definition:

```
<!ELEMENT Method (MUnclassified | MTrigger | MSequence)>
```

The individual child elements of `Method` are now described in more detail.

2.6.1 Unclassified Methods

This element contains the parameters and OPTypes of a function of function class „Unclassified method“. Unclassified methods should only be used when the method in question cannot easily be encapsulated in one of the predefined function classes.

```
<!ELEMENT MUnclassified (MUParam)+>
<!ELEMENT MUParam (ParamName, ParamDescription, MUParamOPType,
          MUParamType)>
<!ATTLIST MUParam details (true | false) #IMPLIED>
<!ELEMENT MUParamOPType ((MUCommand, MUReport?) | (MUReport, MUCommand?))>
<!ELEMENT MUCommand (MCmdStart?, MCmdStartResult?, MCmdStartResultAck?,
          MCmdGetInterface?, MCmdStartAck?, MCmdAbort?, MCmdAbortAck?)>
<!ELEMENT MUReport (MReportErrorAck?, MReportProcessingAck?,
          MReportProcessing?, MReportResult?, MReportResultAck?,
          MReportInterface?, MReportError?)>
<!ELEMENT MUParamType (TBool | TNumber | TString | TEnum | TBitField |
          TStream | TCStream | TShortStream | TRecord | TArray | TVoid)>
```

The element `MUnclassified` contains a non-empty list of parameters. The element `MUParam` defines a parameter of an unclassified method, including the OPTypes used. The attribute `details` is used to block the generation of detailed parameter descriptions, for example when an `ErrorCode` or `ErrorInfo` parameter is being defined.

The child element `MUParamOPType` defines the set of OPTypes for the function that make use of the parameter currently being defined. The command OPTypes available to unclassified methods are `Start`, `StartResult`, `StartResultAck`, `GetInterface`, `StartAck`, `Abort`, and `AbortAck`. The report OPTypes available are `ErrorAck`, `ProcessingAck`, `Processing`, `Result`, `ResultAck`, `Interface`, and `Error`. Unclassified functions are not restricted with regard either to OPTypes or parameter types. The DTD reflects this.

The child element `MUParamType` defines a parameter type and its subparameter types, if applicable and can reference the following types: `TStream`, `TArray`, `TRecord`, `TBool`, `TNumber`, `TEnum`, `TString`, `TBitField`, `TCStream`, or `TVoid`.

Example XML Code:

```
<FunctionClass ClassRef="class_unclassified_method">
  <FunctionClassDesc>Unclassified Method</FunctionClassDesc>
  <Method>
    <MUnclassified>
      <!-- void -->
      <MUParam details="false">
        <ParamName></ParamName>
        <ParamDescription />
        <MUParamOPType>
          <MUCommand />
          <MUReport>
            <MReportProcessing OPTypeRef="MReportProcessing">
              <ParamPos>0</ParamPos>
            </MReportProcessing>
          </MUReport>
        </MUParamOPType>
        <MUParamType>
          <TVoid TypeRef="type_void" />
        </MUParamType>
      </MUParam>
      <!-- SinkNr -->
      <MUParam>
        <ParamName>SinkNr</ParamName>
        <ParamDescription>
          Number of data sink (within one function block there can
          be more than one), e.g.  0x01 for the
          first sink.
        </ParamDescription>
        <MUParamOPType>
          <MUCommand>
            <MCmdStartResult OPTypeRef="MCmdStartResult">
              <ParamPos>1</ParamPos>
            </MCmdStartResult>
          </MUCommand>
          <MUReport>
            <MReportResult OPTypeRef="MReportResult">
              <ParamPos>1</ParamPos>
            </MReportResult>
          </MUReport>
        </MUParamOPType>
        <MUParamType>
          <TNumber>
            <BasisDataType>
              <TUByte TypeRef="type_unsigned_byte" />
            </BasisDataType>
            <Exponent>0</Exponent>
            <Step>1</Step>
            <Unit UnitRef="unit_not_defined" />
          </TNumber>
        </MUParamType>
      </MUParam>
      <!-- ErrorCode -->
      <MUParam details="false">
        <ParamName>ErrorCode</ParamName>
        <ParamDescription>
          (See Section 2.3.2.5.1 “Error” in [1])
        </ParamDescription>
        <MUParamOPType>
```

```
            <MUCommand />
            <MUReport>
              <MReportError OPTypeRef="MReportError">
                <ParamPos>1</ParamPos>
              </MReportError>
            </MUReport>
          </MUParamOPType>
          <MUParamType>
            <TVoid TypeRef="type_void" />
          </MUParamType>
        </MUParam>
        <!-- ErrorInfo -->
        <MUParam details="false">
          <ParamName>ErrorInfo</ParamName>
          <ParamDescription>
            (See Section 2.3.2.5.1 “Error” in [1])
          </ParamDescription>
          <MUParamOPType>
            <MUCommand />
            <MUReport>
              <MReportError OPTypeRef="MReportError">
                <ParamPos>2</ParamPos>
              </MReportError>
            </MUReport>
          </MUParamOPType>
          <MUParamType>
            <TVoid TypeRef="type_void" />
          </MUParamType>
        </MUParam>
      </MUnclassified>
    </Method>
</FunctionClass>
```

2.6.2 Trigger Methods

This element contains the parameters and OPTypes of a function of function class „Trigger method“ (see MOST Specification Rev 2.5, Page 100). The trigger method class should be used when a sender handle is required for use in Ack messages.

```
<!ELEMENT MTrigger (MTParam)+>
<!ELEMENT MTParam (ParamName, ParamDescription, MTParamOPType,
          MTParamType)>
<!ATTLIST MTParam details (true | false) #IMPLIED>
<!ELEMENT MTParamOPType ((MTCommand, MTReport?) | (MTReport, MTCommand?))>
<!ELEMENT MTCommand (MCmdStart?, MCmdStartResult?, MCmdGetInterface?,
          MCmdStartResultAck?, MCmdStartAck?)>
<!ELEMENT MTReport (MReportErrorAck?, MReportProcessingAck?,
          MReportProcessing?, MReportResult?, MReportResultAck?,
          MReportInterface?, MReportError?)>
<!ELEMENT MTParamType (TNumber | TVoid)>
```

The element `MTrigger` contains a non-empty list of parameters. The element `MTParam` defines a parameter of a trigger method, including the OPTypes used. The attribute `details` is used to block the generation of detailed parameter descriptions, for example when an `ErrorCode` or `ErrorInfo` parameter is being defined.

The child element `MTParamOPType` defines the set of OPTypes for the function that make use of the parameter currently being defined. The command OPTypes available to Trigger methods are `Start`, `StartResult`, `StartResultAck`, `GetInterface`, and `StartAck`. The report OPTypes available are `ErrorAck`, `ProcessingAck`, `Processing`, `Result`, `ResultAck`, `Interface`, and `Error`. Each OPType is optional. In other words the property can be defined to make use of any possible subset of command and report OPTypes.

The child element `MTParamType` defines a parameter type and its subparameter types, if applicable and can reference the following types: `TNumber` (for describing a sender handle), or `TVoid`.

Example XML Code:

```
<FunctionClass>
  <FunctionClassDesc>Trigger</FunctionClassDesc>
  <Method>
    <MTrigger>
      <MTParam details="false">
        <ParamName ParamIdx="1" />
        <ParamDescription />
        <MTParamOPType>
          <MTCommand>
            <MCmdStart OPTypeRef="MCmdStart">
              <ParamPos>1</ParamPos>
            </MCmdStart>
            <MCmdStartResult OPTypeRef="MCmdStartResult">
              <ParamPos>1</ParamPos>
            </MCmdStartResult>
          </MTCommand>
          <MTReport>
            <MReportProcessing OPTypeRef="MReportProcessing">
              <ParamPos>1</ParamPos>
            </MReportProcessing>
            <MReportResult OPTypeRef="MReportResult">
              <ParamPos>1</ParamPos>
            </MReportResult>
          </MTReport>
```

```
          </MTParamOPType>
          <MTParamType>
            <TVoid TypeRef="type_void" />
          </MTParamType>
        </MTParam>
      </MTrigger>
    </Method>
</FunctionClass>
```

2.6.3 Sequence Method

This element contains the parameters and OPTypes of a function of function class „sequence method“ (see MOST Specification Rev 2.5, Page 101). The sequence method class should be used when a number of parameters of different types need to be transmitted.

```
<!ELEMENT MSequence (MQParam)+>
<!ELEMENT MQParam (ParamName, ParamDescription, MQParamOPType,
          MQParamType)>
<!ATTLIST MQParam details (true | false) #IMPLIED>
<!ELEMENT MQParamOPType ((MQCommand, MQReport?) | (MQReport, MQCommand?))>
<!ELEMENT MQCommand (MCmdStart?, MCmdAbort?, MCmdStartResult?,
          MCmdGetInterface?, MCmdStartResultAck?, MCmdAbortAck?,
          MCmdStartAck?)>
<!ELEMENT MQReport (MReportErrorAck?, MReportProcessingAck?,
          MReportProcessing?, MReportResult?, MReportResultAck?,
          MReportInterface?, MReportError?)>
<!ELEMENT MQParamType (TBool | TBitField | TNumber | TEnum | TString |
          TStream | TCStream | TVoid)>
```

The element `MSequence` contains a non-empty list of parameters. The element `MQParam` defines a parameter of an unclassified method, including the OPTypes used. The attribute `details` is used to block the generation of detailed parameter descriptions, for example when an `ErrorCode` or `ErrorInfo` parameter is being defined.

The child element `MQParamOPType` defines the set of OPTypes for the function that make use of the parameter currently being defined. The command OPTypes available to sequence methods are `Start`, `StartAck`, `StartResult`, `StartResultAck`, `GetInterface`, `AbortAck`, and `Abort`. The report OPTypes available are `ErrorAck`, `ProcessingAck`, `Processing`, `Result`, `ResultAck`, `Interface`, and `Error`. The property can be defined to make use of any possible subset of command and report OPTypes.

The child element `MQParamType` defines a parameter type and its subparameter types, if applicable and can reference the following basic types: `TBool`, `TBitField`, `TNumber`, `TEnum`, `TString`, `TStream`, `TCStream`, or `TVoid`.

Example XML Code:

```
<FunctionClass ClassRef="class_sequence">
  <FunctionClassDesc>Sequence</FunctionClassDesc>
  <Method>
    <MSequence>
      <MQParam>
        <ParamName>SenderHandle</ParamName>
        <ParamDescription />
        <MQParamOPType>
          <MQCommand>
            <MCmdStartResultAck OPTypeRef="MCmdStartResultAck">
              <ParamPos>1</ParamPos>
```

```
              </MCmdStartResultAck>
            </MQCommand>
            <MQReport>
              <MReportErrorAck OPTypeRef="MReportErrorAck">
                <ParamPos>1</ParamPos>
              </MReportErrorAck>
              <MReportProcessingAck OPTypeRef="MReportProcessingAck">
                <ParamPos>1</ParamPos>
              </MReportProcessingAck>
              <MReportResultAck OPTypeRef="MReportResultAck">
                <ParamPos>1</ParamPos>
              </MReportResultAck>
            </MQReport>
          </MQParamOPType>
          <MQParamType>
            <TNumber>
              <BasisDataType>
                <TUWord TypeRef="type_unsigned_word"/>
              </BasisDataType>
              <Exponent>0</Exponent>
              <Step>1</Step>
              <Unit UnitRef="unit_none"/>
            </TNumber>
          </MQParamType>
        </MQParam>
        <MQParam>
          <ParamName>DABLearnMode</ParamName>
          <ParamDescription />
          <MQParamOPType>
            <MQCommand>
              <MCmdStartResultAck OPTypeRef="MCmdStartResultAck">
                <ParamPos>2</ParamPos>
              </MCmdStartResultAck>
            </MQCommand>
          </MQParamOPType>
          <MQParamType>
            <TEnum TypeRef="type_enum" TEnumMax="3">
              <TEnumValue Code="0x00">Normal Mode</TEnumValue>
              <TEnumValue Code="0x01">Learn</TEnumValue>
              <TEnumValue Code="0x02">Clear Mode</TEnumValue>
              <TEnumValue Code="0x03">Auto Learn Mode</TEnumValue>
            </TEnum>
          </MQParamType>
        </MQParam>
      </MSequence>
    </Method>
</FunctionClass>
```

2.7 Function OPTypes

The following table lists all possible OPTypes according to MOST Specification Rev 2.5:

OPType	For Properties	For Methods
Commands:		
0	Set	Start
1	Get	Abort
2	SetGet	StartResult
3	Increment	Reserved
4	Decrement	Reserved
5	GetInterface	GetInterface
6	Not allowed	StartResultAck
7	Not allowed	AbortAck
8	Not allowed	StartAck
Reports:		
9	ErrorAck[1]	ErrorAck
A	Not allowed	ProcessingAck
B	Reserved	Processing
C	Status	Result
D	Not allowed	ResultAck
E	Interface	Interface
F	Error	Error

2.7.1 Assignment of OPTypes and parameters

The assignment of OPTypes and parameters in the XML document differs from the printed representation of the function catalog. The printed catalog lists all legal OPTypes in a table, assigning them their respective parameters. The parameter definitions are omitted in this example.

Function class: Unclassified Property

FBlock	Function	OPType	Parameters
NetBlock (0x01)	FBlockIDs (0x000)	Get	
		SetGet	FBlockID, OldInstID, NewInstID
		Status	FBlockIDList
		Error	ErrorCode, ErrorInfo

The XML document operates inversely. Parameter and parameter type are specified first. Every parameter directly used by an OPType is assigned a list of OPTypes using this parameter via the `XXParamOPType` element, PUParamOPType in this case. The respective parameter positions are assigned in the `ParamPos` element. Position numbers start with 1. This way, a parameter used by several OPTypes has to be described only once.
The attribute `Channel` determines the transmission channel that is used by a particular OPType.

[1] ErrorAck is required for properties to report syntax errors in conjunction with illegal OPTypes (ErrorCode 0x01..0x04).

DTD Definitions:

```
<!ELEMENT PUParam (ParamName, ParamDescription, PUParamOPType,
          PUParamType)>
…
<!ELEMENT PUParamOPType  ((PUCommand,PUReport?)|(PUReport,PUCommand?))>
<!ELEMENT PUCommand (PCmdSet?, PCmdGet?, PCmdSetGet?, PCmdIncrement?, …)>
<!ELEMENT PUReport (PReportStatus?, PReportInterface?, PReportError?)>
…
<!ELEMENT PCmdGet (ParamPos?)>
<!ATTLIST PCmdGet OPTypeRef IDREF #FIXED "PCmdGet"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >
<!ELEMENT ParamPos (#PCDATA)>
…
<!ELEMENT PCmdSetGet (ParamPos?)>
<!ATTLIST PCmdSetGet OPTypeRef IDREF #FIXED "PCmdSetGet"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >
…
<!ELEMENT PReportStatus (ParamPos?)>
<!ATTLIST PReportStatus OPTypeRef IDREF #FIXED "PReportStatus"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >
…
<!ELEMENT PReportError (ParamPos?)>
<!ATTLIST PReportError OPTypeRef IDREF #FIXED "PReportError"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >
```

Example XML code:

```
<Function>
  …
  <FunctionName>FBlockIDs</FunctionName>
  …
  <FunctionClass ClassRef="class_unclassified_property">
    <FunctionClassDesc>Unclassified Property</FunctionClassDesc>
    <Property>
      <PUnclassified Length="1">
        <!-- void -->
        <PUParam details="false">
          <ParamName ParamIdx="1"/>
          <ParamDescription/>
          <PUParamOPType>
            <PUCommand>
              <PCmdGet OPTypeRef="PCmdGet">
                <ParamPos>0</ParamPos>
              </PCmdGet>
            </PUCommand>
            <PUReport/>
          </PUParamOPType>
          <PUParamType>
            <TVoid TypeRef="type_void"/>
          </PUParamType>
        </PUParam>
        <!-- FBlockID  -->
        <PUParam>
          <ParamName ParamIdx="6">FBlockID</ParamName>
          <ParamDescription/>
          <PUParamOPType>
            <PUCommand>
              <PCmdSetGet OPTypeRef="PCmdSetGet">
```

```
                <ParamPos>1</ParamPos>
              </PCmdSetGet>
            </PUCommand>
            <PUReport/>
          </PUParamOPType>
          <PUParamType>...</PUParamType>
        </PUParam>
        <!-- NewInstID -->
        <PUParam>
          <ParamName ParamIdx="1">NewInstID</ParamName>
          <ParamDescription/>
          <PUParamOPType>
            <PUCommand>
              <PCmdSetGet OPTypeRef="PCmdSetGet">
                <ParamPos>3</ParamPos>
              </PCmdSetGet>
            </PUCommand>
            <PUReport/>
          </PUParamOPType>
          <PUParamType>...</PUParamType>
        </PUParam>
        <!-- OldInstID -->
        <PUParam>
          <ParamName ParamIdx="2">OldInstID</ParamName>
          <ParamDescription/>
          <PUParamOPType>
            <PUCommand>
              <PCmdSetGet OPTypeRef="PCmdSetGet">
                <ParamPos>2</ParamPos>
              </PCmdSetGet>
            </PUCommand>
            <PUReport/>
          </PUParamOPType>
          <PUParamType>...</PUParamType>
        </PUParam>
        <!-- FBlockIDList -->
        <PUParam>
          <ParamName ParamIdx="3">FBlockIDList</ParamName>
          <ParamDescription/>
          <PUParamOPType>
            <PUCommand/>
            <PUReport>
              <PReportStatus OPTypeRef="PReportStatus">
                <ParamPos>1</ParamPos>
              </PReportStatus>
            </PUReport>
          </PUParamOPType>
          <PUParamType>...</PUParamType>
        </PUParam>
        <!-- ErrorCode -->
        <PUParam details="false">
          <ParamName ParamIdx="4">ErrorCode</ParamName>
          <ParamDescription/>
          <PUParamOPType>
            <PUCommand/>
            <PUReport>
              <PReportError OPTypeRef="PReportError">
                <ParamPos>1</ParamPos>
              </PReportError>
            </PUReport>
```

```
            </PUParamOPType>
            <PUParamType>
              <TVoid TypeRef="type_void"/>
            </PUParamType>
          </PUParam>
          <!-- ErrorInfo -->
          <PUParam details="false">
            <ParamName ParamIdx="5">ErrorInfo</ParamName>
            <ParamDescription/>
            <PUParamOPType>
              <PUCommand/>
              <PUReport>
                <PReportError OPTypeRef="PReportError">
                  <ParamPos>2</ParamPos>
                </PReportError>
              </PUReport>
            </PUParamOPType>
            <PUParamType>
              <TVoid TypeRef="type_void"/>
            </PUParamType>
          </PUParam>
        </PUnclassified>
      </Property>
   <FunctionClass>
</Function>
```

Because OPTypes are specified in a parameter context, parameterless OPTypes need special treatment. A dummy parameter of type “TVoid” must be included in the XML document. It can be assigned the parameterless OPTypes, using a parameter position of 0 and the details=”false” flag.

DTD Definition:

```
<!ELEMENT TVoid EMPTY>
<!ATTLIST TVoid TypeRef IDREF #REQUIRED>
```

2.7.2 OPTypes for Properties

DTD Definition:

```
<!ELEMENT PCmdSet (ParamPos?)>
<!ATTLIST PCmdSet OPTypeRef IDREF #FIXED "PCmdSet"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >
<!ELEMENT ParamPos (#PCDATA)>

<!ELEMENT PCmdGet (ParamPos?)>
<!ATTLIST PCmdGet OPTypeRef IDREF #FIXED "PCmdGet"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT PCmdSetGet (ParamPos?)>
<!ATTLIST PCmdSetGet OPTypeRef IDREF #FIXED "PCmdSetGet"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT PCmdIncrement (ParamPos?)>
<!ATTLIST PCmdIncrement OPTypeRef IDREF  #FIXED "PCmdIncrement"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT PCmdDecrement (ParamPos?)>
<!ATTLIST PCmdDecrement OPTypeRef IDREF #FIXED "PCmdDecrement"
```

```
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT PCmdGetInterface (ParamPos?)>
<!ATTLIST PCmdGetInterface OPTypeRef IDREF #FIXED "PCmdGetInterface"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT PReportStatus (ParamPos?)>
<!ATTLIST PReportStatus OPTypeRef IDREF #FIXED "PReportStatus"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT PReportInterface (ParamPos?)>
<!ATTLIST PReportInterface OPTypeRef IDREF #FIXED "PReportInterface"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT PReportError (ParamPos?)>
<!ATTLIST PReportError OPTypeRef IDREF #FIXED "PReportError"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >
```

2.7.3 OPTypes for Methods

```
<!ELEMENT MCmdStart(ParamPos?)>
<!ATTLIST MCmdStart OPTypeRef IDREF #FIXED "MCmdStart"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MCmdStartResult (ParamPos?)>
<!ATTLIST MCmdStartResult OPTypeRef  IDREF    #FIXED "MCmdStartResult"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MCmdStartResultAck (ParamPos?)>
<!ATTLIST MCmdStartResultAck OPTypeRef IDREF #FIXED
          "MCmdStartResultAck"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MCmdGetInterface (ParamPos?)>
<!ATTLIST MCmdGetInterface OPTypeRef IDREF #FIXED "MCmdGetInterface"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MCmdStartAck (ParamPos?)>
<!ATTLIST MCmdStartAck OPTypeRef IDREF #FIXED "MCmdStartAck"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MCmdAbort (ParamPos?)>
<!ATTLIST MCmdAbort OPTypeRef IDREF #FIXED "MCmdAbort"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MCmdAbortAck (ParamPos?)>
<!ATTLIST MCmdAbortAck OPTypeRef IDREF #FIXED "MCmdAbortAck"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MReportErrorAck (ParamPos?)>
<!ATTLIST MReportErrorAck OPTypeRef IDREF #FIXED "MReportErrorAck"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MReportProcessingAck (ParamPos?)>
<!ATTLIST MReportProcessingAck OPTypeRef IDREF #FIXED
          "MReportProcessingAck"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MReportProcessing (ParamPos?)>
```

```
<!ATTLIST MReportProcessing OPTypeRef IDREF #FIXED "MReportProcessing"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MReportResult (ParamPos?)>
<!ATTLIST MReportResult OPTypeRef IDREF #FIXED "MReportResult"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MReportResultAck (ParamPos?)>
<!ATTLIST MReportResultAck OPTypeRef IDREF #FIXED "MReportResultAck"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MReportInterface (ParamPos?)>
<!ATTLIST MReportInterface OPTypeRef IDREF #FIXED "MReportInterface"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >

<!ELEMENT MReportError (ParamPos?)>
<!ATTLIST MReportError OPTypeRef IDREF #FIXED "MReportError"
          Channel (Control | MOST-High | Ethernet | Other) #IMPLIED >
```

2.8 Definitions

2.8.1 Element <Definition>

The definition elements carry an ID to enable referencing from other parts of the XML document.

Attributes:
None

Child elements:
- List of <ClassDef> (optional)
- List of <PCmdDef> (optional)
- List of <MCmdDef> (optional)
- List of <PReportDef> (optional)
- List of <MReportDef> (optional)
- List of <TypeDef> (optional)
- List of <UnitDef> (optional)
- List of <ErrorDef> (optional)

Data:
None

DTD Definition:

```
<!ELEMENT Definition ((ClassDef)*, (PCmdDef)*, (MCmdDef)*, (PReportDef)*,
(MReportDef)*, (TypeDef)*, (UnitDef)*, (ErrorDef)*)>
```

The following is an excerpt from the definitions usually found trailing a function catalog.

```
<Definition>
  <ClassDef ClassID="class_switch">
    <ClassDefName>Switch</ClassDefName>
    <ClassDefDesc/>
  </ClassDef>
  <ClassDef ClassID="class_number">
    <ClassDefName>Number</ClassDefName>
    <ClassDefDesc/>
  </ClassDef>
  ...
  <PCmdDef PCmdID="PCmdSet">
    <CmdDefName>Set</CmdDefName>
    <CmdDefDesc/>
    <CmdDefOPType>0x0</CmdDefOPType>
  </PCmdDef>
  ...
  <MCmdDef MCmdID="MCmdStart">
    <CmdDefName>Start</CmdDefName>
    <CmdDefDesc/>
    <CmdDefOPType>0x0</CmdDefOPType>
  </MCmdDef>
  ...
  <PReportDef PReportID="PReportStatus">
    <ReportDefName>Status</ReportDefName>
    <ReportDefDesc/>
    <ReportDefOPType>0xC</ReportDefOPType>
  </PReportDef>
  ...
  <MReportDef MReportID="MReportErrorAck">
```

```
      <ReportDefName>ErrorAck</ReportDefName>
      <ReportDefDesc/>
      <ReportDefOPType>0x9</ReportDefOPType>
    </MReportDef>
    ...
    <TypeDef TypeID="type_boolean">
      <TDefName>Boolean</TDefName>
      <TDefDesc/>
      <TDefSize>1</TDefSize>
    </TypeDef>
    ...
    <UnitDef UnitID="unit_none">
      <UnitDefName>none</UnitDefName>
      <UnitDefCode>0x00</UnitDefCode>
      <UnitDefGroup/>
    </UnitDef>
    <UnitDef UnitID="unit_cm">
      <UnitDefName>cm</UnitDefName>
      <UnitDefCode>0x01</UnitDefCode>
      <UnitDefGroup>Distance</UnitDefGroup>
    </UnitDef>
    ...
    <ErrorDef ErrorID="error_general_0x01">
      <ErrorDefCode>0x01</ErrorDefCode>
      <ErrorDefCodeDesc> = FBlockID not available</ErrorDefCodeDesc>
      <ErrorDefInfo>0x00</ErrorDefInfo>
      <ErrorDefInfoDesc/>
    </ErrorDef>
    ...
</Definition>
```

2.8.2 Element <ClassDef>

This element defines a function class.

Attributes:

- ClassID (ID, required)
  Unique ID for references from other elements.

Child elements:

- <ClassDefName>
- <ClassDefDesc>

Data:
None

DTD Definition:

```
<!ELEMENT ClassDef (ClassDefName, ClassDefDesc)>
<!ATTLIST ClassDef ClassID ID #REQUIRED>
```

2.8.2.1 Element <ClassDefName>

This element contains a function class' name, for example, „Array" or „Record".

Attributes:
None

Child elements:
None

Data:
Function class name (PCDATA)

DTD Definition:

```
<!ELEMENT ClassDefName (#PCDATA)>
```

2.8.2.2 Element <ClassDefDesc>

This element contains a description of a function class.

Attributes:
None

Child elements:
None

Data:
Function class description (PCDATA)

DTD Definition:

```
<!ELEMENT ClassDefDesc (#PCDATA)>
```

2.8.3 OPType Definition for „Commands“

2.8.3.1 Element <PCmdDef>

This element defines an OPType of type „Command“. The OPType can be referenced in functions of type „property“.

Attributes:
- PCmdID (ID, required)
  Unique ID for references from other elements.

Child elements:
- <CmdDefName>
- <CmdDefDesc>
- <CmdDefOPType> (optional)

Data:
None

DTD Definition:

```
<!ELEMENT PCmdDef (CmdDefName, CmdDefDesc, CmdDefOPType?)>
<!ATTLIST PCmdDef PCmdID ID #REQUIRED>
```

2.8.3.2 Element <MCmdDef>

This element defines an OPType of type „Command“. The OPType can be referenced in functions of type „method“.

Attributes:
- MCmdID (ID, required)
  Unique ID for references from other elements.

Child elements:
- <CmdDefName>
- <CmdDefDesc>

Data:
None

DTD Definition:

```
<!ELEMENT MCmdDef (CmdDefName, CmdDefDesc, CmdDefOPType?)>
<!ATTLIST MCmdDef MCmdID ID #REQUIRED>
```

2.8.3.3 Element <CmdDefName>

This element contains an OPType’s name. By convention, this must equal the name of the element the OPType will be referenced in, e.g., PCmdSet is referenced from <PCmdSet>.

Attributes:
None

Child elements:
None

Data:
OPType’s name (PCDATA)

DTD Definition:

```
<!ELEMENT CmdDefName (#PCDATA)>
```

2.8.3.4 Element <CmdDefDesc>

This element contains a description of an OPType.

Attributes:
None

Child elements:
None

Data:
OPType description (PCDATA)

DTD Definition:

```
<!ELEMENT CmdDefDesc (#PCDATA)>
```

2.8.3.5 Element <CmdDefOPType>

This element contains the numerical value for a particular OPType.

Attributes:
None

Child elements:
None

Data:
OPType value (PCDATA)

DTD Definition:

```
<!ELEMENT CmdDefOPType (#PCDATA)>
```

2.8.4 OPType Definition for „Reports“

2.8.4.1 Element <PReportDef>

This element defines an OPType of type „Report“. The OPType can be referenced in functions of type „property“.

Attributes:
- PReportID (ID, required)
  Unique ID for references from other elements.

Child elements:
- <ReportDefName>
- <ReportDefDesc>
- <ReportDefOPType> (optional)

Data:
None

DTD Definition:

```
<!ELEMENT PReportDef (ReportDefName, ReportDefDesc, ReportDefOPType?)>
<!ATTLIST PReportDef PReportID ID #REQUIRED>
```

2.8.4.2 Element <MReportDef>

This element defines an OPType of type „Report“. The OPType can be referenced in functions of type „method“.

Attributes:
- MReportID (ID, required)
  Unique ID for references from other elements.

Child elements:
- <ReportDefName>
- <ReportDefDesc>
- <ReportDefOPType> (optional)

Data:
None

DTD Definition:

```
<!ELEMENT MReportDef (ReportDefName, ReportDefDesc, ReportDefOPType?)>
<!ATTLIST MReportDef MReportID ID #REQUIRED>
```

2.8.4.3 Element <ReportDefName>

This element contains an OPType’s name. By convention, this must equal the name of the element the OPType will be referenced in, e.g., MCmdStart is referenced from <MCmdStart>.

Attributes:
None

Child elements:
None

Data:
OPType name (PCDATA)

DTD Definition:

```
<!ELEMENT ReportDefName (#PCDATA)>
```

2.8.4.4 Element <ReportDefDesc>

This element contains a description of an OPType.

Attributes:
None

Child elements:
None

Data:
OPType description (PCDATA)

DTD Definition:

```
<!ELEMENT ReportDefDesc (#PCDATA)>
```

2.8.4.5 Element <ReportDefOPType>

This element contains the numerical value for a particular OPType.

Attributes:
None

Child elements:
None

Data:
OPType description (PCDATA)

DTD Definition:

```
<!ELEMENT ReportDefOPType (#PCDATA)>
```

2.8.5 Element <TypeDef>

This element defines an individual data type.

Attributes:
- TypeID (ID, required)
  Unique ID for references from other elements.

Child elements:
- TDefName
- TDefDesc
- TDefSize

Data:
None

DTD Definition:

```
<!ELEMENT TypeDef (TDefName, TDefDesc, TDefSize)>
<!ATTLIST TypeDef TypeID ID #REQUIRED>
```

2.8.5.1 Element <TDefName>

This element contains a data type name.

Attributes:
None

Child elements:
None

Data:
Data type name (PCDATA)

DTD Definition:

```
<!ELEMENT TDefName (#PCDATA)>
```

2.8.5.2 Element <TDefDesc>

This element contains a data type description.

Attributes:
None

Child elements:
None

Data:
Data type description (PCDATA)

DTD Definition:

```
<!ELEMENT TDefDesc (#PCDATA)>
```

2.8.5.3 Element <TDefSize>

This element contains the size of the parent data type.

Attributes:
None

Child elements:
None

Data:
Size of the parent data type (PCDATA)

DTD Definition:

```
<!ELEMENT TDefSize (#PCDATA)>
```

2.8.6 Element <UnitDef>

This element defines an individual measurement unit.

Attributes:
- UnitID (ID, required)
  Unique ID for references from other elements.

Child elements:
- <UnitDefName>
- <UnitDefCode>
- <UnitDefGroup> (optional)

Data:
None

DTD Definition:

```
<!ELEMENT UnitDef (UnitDefName, UnitDefCode, UnitDefGroup?)>
<!ATTLIST UnitDef UnitID ID #REQUIRED>
```

2.8.6.1 Element <UnitDefName>

This element contains a unit name (“km”, “1/min” …)

Attributes:
None

Child elements:
None

Data:
Unit name (PCDATA)

DTD Definition:

```
<!ELEMENT UnitDefName (#PCDATA)>
```

2.8.6.2 Element <UnitDefCode>

This element contains a code assigned to a unit (see MOST Specification Rev 2.5, page 71).

Attributes:
None

Child elements:
None

Data:
Unit code (PCDATA)

DTD Definition:

```
<!ELEMENT UnitDefCode (#PCDATA)>
```

2.8.6.3 Element <UnitDefGroup>

This element contains a descriptor for the unit's group (e.g., „time" or „frequency")

Attributes:
None

Child elements:
None

Data:
Unit group descriptor (PCDATA)

DTD Definition:

```
<!ELEMENT UnitDefGroup (#PCDATA)>
```

2.8.7 Element <ErrorDef>

This element defines an error, which may be referenced by individual functions.

Attributes:
- ErrorID (ID, required)
  Unique ID for references from other elements.

Child elements:
- <ErrorDefCode>
- <ErrorDefCodeDesc>
- <ErrorDefInfo>
- <ErrorDefInfoDesc>

Data:
None

DTD Definition:

```
<!ELEMENT ErrorDef (ErrorDefCode, ErrorDefCodeDesc, ErrorDefInfo,
          ErrorDefInfoDesc)>
<!ATTLIST ErrorDef ErrorID ID #REQUIRED>
```

2.8.7.1 Element <ErrorDefCode>

This element contains an error code (see MOST Specification Rev 2.5, page 36).

Attributes:
None

Child elements:
None

Data:
The error code (PCDATA)

DTD Definition:

```
<!ELEMENT ErrorDefCode (#PCDATA)>
```

2.8.7.2 Element <ErrorDefCodeDesc>

This element contains a description of an error.

Attributes:
None

Child elements:
None

Data:
Error description (PCDATA)

DTD Definition:

```
<!ELEMENT ErrorDefCodeDesc (#PCDATA)>
```

2.8.7.3 Element <ErrorDefInfo>

This element contains a definition of the information associated with an error (see MOST Specification Rev 2.5, page 36, column „ErrorInfo“).

Attributes:
None

Child elements:
None

Data:
Error information (PCDATA)

DTD Definition:

```
<!ELEMENT ErrorDefInfo (#PCDATA)>
```

2.8.7.4 Element <ErrorDefInfoDesc>

This element contains a description of the information associated with an error.

Attributes:
None

Child elements:
None

Data:
Error information description (PCDATA)

DTD Definition:

```
<!ELEMENT ErrorDefInfoDesc (#PCDATA)>
```