



DRIVECOM

Device Description

Version: 1.1, 28. May 2002

Published by: DRIVECOM Nutzergruppe e.V.
Postfach 1102, D-32817 Blomberg
Telephone : +49 52 35 / 3-4 18 64
Fax : +49 52 35 / 3-4 18 62
Internet: <http://www.drivecom.org/>

All rights, including those related to translation into a foreign language, are reserved. No part of this information is allowed to be reproduced, processed using electronic systems, duplicated, or passed on to third parties in any form (print, photocopy, microfilm or any other process) without the written approval of the DRIVECOM Nutzergruppe e.V.

Subject to change without notice

Authors:

Krumsiek	Phoenix Contact	Blomberg
Hadlich	ifak system GmbH	Barleben
Heines	Phoenix Contact	Blomberg
Lange	Softing GmbH	München
Dr. Leurs	Rexroth Indramat	Lohr / Main
Mirbach	Lenze	Hameln
Pollmeier	ESR Pollmeier	Ober-Ramstadt
Riedl	ifak Magdeburg e.V.	Barleben
Schäfer	Lenze	Hameln
Schleicher	Indramat Refu	Metzingen
Schnurbusch	Lenze	Hameln
Schwarz	Mannesmann Dematic	Wetter
Senneka	Stöber Antriebstechnik	
Teipen	Hanning Elektro-Werke	Oerlinghausen
Vothknecht	Phoenix Contact	Blomberg
Ziegler	SEW-EURODRIVE	Bruchsal

**History:**

Included by	Version, Changes
Riedl	07.07.2000 V0.0.2 draft
Riedl	17.07.2000 V0.0.46 draft
Riedl	V0.2 draft References to XML links rejected, redefine added, domain attribute for language files, interface for script access to DriveServer
Riedl	27.11.2000 V0.8a Harmonisation with FDCML 1.0
Mirbach	V1.0 Inclusion of the results obtained during the implementation of the generic DriveServer
Mirbach	V1.1 Data type ARRAY_OF_VT_UI1 added

Suggestions and comments are to be sent to Stefan Pollmeier gl@esr-pollmeier.de.



Contents

1. Introduction	5
1.1. Overview	5
1.2. References	5
1.3. Abbreviations	5
2. Concept	6
2.1. Overview	6
2.2. Architecture	6
2.3. Sources and profiles	6
3. Basic elements of the language	7
3.1. Overview	7
3.2. AIP	7
3.3. device	8
3.4. DeviceIdentityObject	8
3.5. DeviceManagerObject	9
3.5.1. ProcessDataItemList	10
3.5.2. ParameterItemList	11
3.6. DeviceFunctionObject	12
3.6.1. DictionaryList	12
3.6.2. HelpFileList	12
3.6.3. containerList, componentList	13
3.6.4. varTemplateList	14
3.6.5. Variable	17
3.6.6. Lists	18
3.6.7. Menu	18
3.6.8. Methods	19
3.6.9. Redefinitions	19
4. Property IDs used	19
5. Dictionary (DTD for the language file)	20
6. Help files	20
7. Interface	21
7.1. Overview	21
7.2. Function description	21
7.3. IDL	22
8. Syntax of the language DTD	23



1. Introduction

1.1. Overview

This document describes the method for the electronically readable description of drive parameters and drive functionality. The device description is used for the configuration of the DriveServer. It can also be used for the description of the product in other areas. The DTD presented here encompasses the following aspects:

- Description of the drive parameters
- Support for parameter dependencies
- Logical grouping of the drive parameters supported

For this device description, a Document Type Definition (DTD) in accordance with the XML standard has been defined.

1.2. References

- [1] OLE for Process Control, Data Access Custom Interface Standard, Version 2.0, <http://www.opcfoundation.org>
- [2] DRIVECOM, DriveServer V 1.1
- [3] XML Version 1.0, <http://www.w3c.org/xml>
- [4] Language Specification of Electronic Device Description
- [5] Field Device Configuration Markup Language 1.0

1.3. Abbreviations

DOM	D ocument O bject M odel
DTD	D ocument T ype D efinition
FDCML	F ield D evice C onfiguration M arkup L anguage
OPC	O LE for P rocess C ontrol
W3C	W orld W ide W eb C onsortium
XML	e Xtensible M arkup L anguage

2. Concept

2.1. Overview

The DRIVECOM device description was developed to describe the properties of drives independent of manufacturer. With the aid of the standardized device description, it is possible to configure the DriveServer in a manufacturer-independent way. The way in which a drive is identified on the bus is not part of this specification and is generally performed using manufacturer-specific rules.

The device description is readable text that is written by the drive manufacturer to describe all the information necessary for the configuration and operation of the drive.

2.2. Architecture

The system architecture comprises the language definition in the form of the DTD, the definition of the dictionary in the form of a DTD, numerous device descriptions (XML files) that are loaded onto the DriveServer, and the country-specific language files. The DriveServer can also add layout information in the form of style sheets. The data may be stored internally, e.g., in pre-defined objects (DOM) or in specific structures. The style sheets do not need to be evaluated by the DriveServer itself. They can, however, be passed to clients.

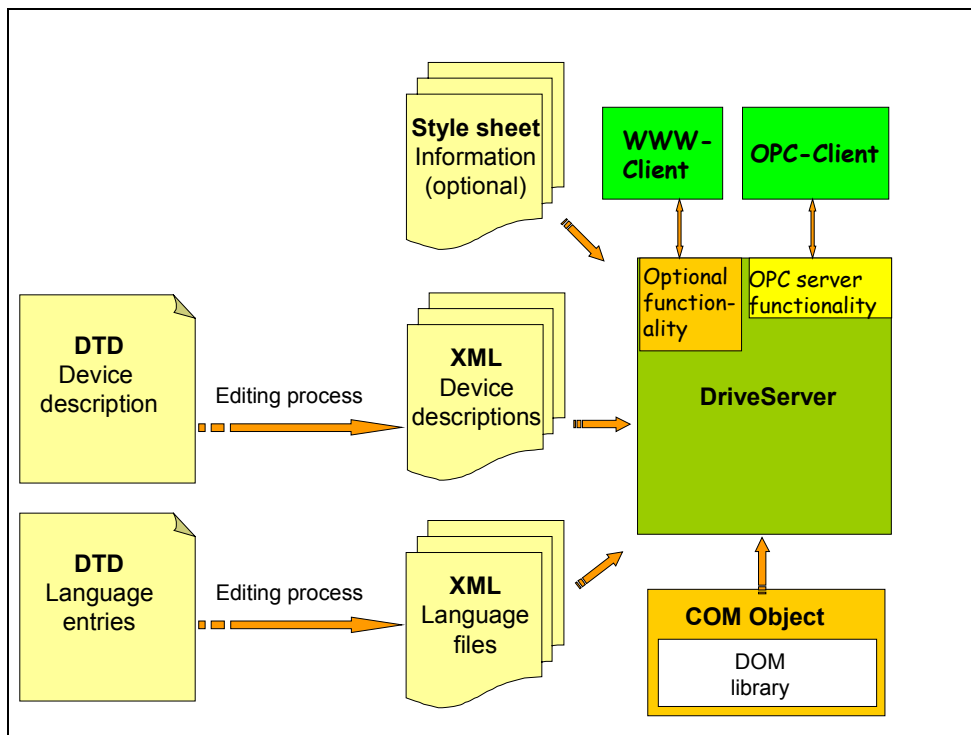


Figure 1: Integration of the device description in the DriveServer concept

2.3. Sources and profiles

The device description contains all the information necessary to describe the drive. For this purpose the descriptions can be divided into standard and device-specific descriptions. Standard descriptions are imported by the actual description and can be specified by groups of manufacturers. Then the manufacturer must only describe the additional parts.

3. Basic elements of the language

3.1. Overview

The DRIVECOM device description is based on XML technology, which was defined by the W3C, and on the FDCML specification.

The semantics of the individual elements are described in this specification and in [5].

The DRIVECOM device description uses the following elements from FDCML:

```
AIP
device
DeviceIdentityObject
DeviceManagerObject
communicationEntity
DeviceFunctionObject
```

3.2. AIP

The 'AIP' element is used in FDCML for the definition of the 'Application Integrated Profiles' (see [5]). In the case of the exclusive usage of the XML device description for the configuration of the DriveServer, this element is used as the entry point to create references to other XML files for import, and to describe the device.

Syntax:

```
<!ELEMENT AIP                               (importList?,
                                           (INTERBUSBasedDevice|
                                           device|
                                           devices))>
```

The 'INTERBUSBasedDevice' and 'devices' elements are not considered further here.

Further DRIVECOM information (device specific parameters and functions) is coded in the 'device' element.

In this specification predominantly the 'DeviceFunctionObject' element is described. This element is empty in the original version of FDCML.

Syntax:

```
<!ELEMENT importList                         (file+)>
<!ELEMENT file                               EMPTY>
<!ATTLIST file                               xml:link (simple) #IMPLIED
                                             href CDATA #REQUIRED>
```

Under 'importList', references to XML files to be imported can be created so that the profile descriptions and manufacturer-specific standard descriptions can be used. The imports must be loaded by the DriveServer first, and also recursively. If the import section is used, at least one file must be given. In the implementation it is possible to resolve the imports in a pre-processor if necessary, and to configure the DriveServer with the file prepared in this manner.

href: String with URL of the file to be incorporated



3.3. device

The 'device' element is used to describe all device-specific functions (see [5]). This specification uses the mandatory attributes and only the 'DeviceIdentityObject', 'DeviceManagerObject' and 'DeviceFunctionObject' child elements. The additional child elements do not need to be completed for DRIVECOM purposes.

Syntax:

```
<!ELEMENT device (DeviceIdentityObject,
                  DeviceManagerObject,
                  DeviceFunctionObject*,
                  ApplicationProcessObject*) >
<!ATTLIST device
  formatName NMTOKEN #FIXED "FDCML"
  formatVersion NMTOKEN #FIXED "1.0"
  fileName CDATA #REQUIRED
  fileCreator CDATA #REQUIRED
  fileCreationDate CDATA #REQUIRED
  fileModificationDate CDATA #REQUIRED
  fileVersion CDATA #REQUIRED
  %id.unique.opt;>
```

fileName:	File name of the XML file
fileCreator:	Specific string for the file creator
fileCreationDate:	Creation date for the file
fileModificationDate:	Date of the last change to the file
fileVersion:	Manufacturer-specific version of the file

3.4. DeviceIdentityObject

All network or bus-dependent properties used for the identification of the device are described in this element (see [5]).

Syntax:

```
<!ELEMENT DeviceIdentityObject (vendor,
                                vendorText?,
                                family,
                                deviceType,
                                designation,
                                deviceText?,
                                orderNumber,
                                version) >
<!ELEMENT vendor (const+|edit+|labelRef) >
<!ATTLIST vendor
  vendorID CDATA #IMPLIED>
```



vendor:	String with name of the drive manufacturer, it is also possible to add references to homepages or other external strings
vendorID:	Optional manufacturer's ID that an organisation can assign <code><!ELEMENT vendorText (const+ edit+ labelRef)></code>
vendorText:	Additional information on the manufacturer, it is also possible to add references to external strings etc. <code><!ELEMENT family (const+ edit+ labelRef)></code>
family:	Manufacturer-specific information on the device family, it is also possible to add references to external strings etc. <code><!ELEMENT deviceType (const+ edit+ labelRef)></code>
deviceType:	String with distinct, manufacturer-specific name for a drive family or device type, it is also possible to add references to external strings etc. <code><!ELEMENT designation (const+ edit+ labelRef)></code> <code><!ATTLIST designation deviceID CDATA #IMPLIED></code>
designation:	String with distinct, manufacturer-specific name for the drive, it is also possible to add references to external strings etc.
deviceID:	Manufacturer-specific device ID (optional) <code><!ELEMENT deviceText (const+ edit+ labelRef)></code>
deviceText:	String with detailed description of a drive family or device type, it is also possible to add references to external strings etc. <code><!ELEMENT orderNumber (const+ edit+ labelRef)></code>
orderNumber:	String with manufacturer-specific order number for a drive family or device type, it is also possible to add references to external strings etc. <code><!ELEMENT version (const+ edit+ labelRef)></code>
version:	String with version/revision number of the unit, it is also possible to add references to external strings etc.

3.5. DeviceManagerObject

In the 'DeviceManagerObject' element all network-specific and communication-specific properties of the device are recorded.

Syntax:

```
<!ELEMENT DeviceManagerObject (dictionaryList?,  
helpFileList?,  
toolList?,  
pictureList?,  
physicalConnectionPointList?,  
localDataItemList?,  
additionalItemList*,  
communicationEntity+)>
```



Only the 'communicationEntity' entry is of significance for DRIVECOM. It describes the relationships between the DriveServer-OPC-Items and the communication objects.

Syntax:

```
<!ELEMENT communicationEntity      (%naming.opt;;
                                     helpFileList?,
                                     toolList?,
                                     pictureList?,
                                     cfgItemList?,
                                     additionalItemList*,
                                     processDataItemList?,
                                     parameterItemList?,
                                     logicalConnectionPointList?,
                                     MAUUsageList?,
                                     internalConnectionPointList?,
                                     slotUsageList?)>
<!ATTLIST communicationEntity      %id.unique.opt;
                                     protocol CDATA #REQUIRED
                                     communicator (NO|YES) 'YES'
                                     communicationEntityType (SLAVE|
                                                             MASTER|
                                                             CLIENT|
                                                             SERVER|
                                                             INTERCONNECTION|
                                                             PEER|
                                                             MASTER_SLAVE) 'SLAVE'
                                     %enabled.def;>
```

Here all entries are optional. (Note: in the original FDCML some entries are mandatory!).

The 'processDataItemList' and 'parameterItemList' entries are of interest for DRIVECOM.

3.5.1. ProcessDataItemList

This element is used to describe the process data objects that exist.

Syntax:

```
<!ELEMENT processDataItemList      (%naming.opt;;
                                     (processDataCategory|
                                     processDataItem)+)>
```

The optional '%naming.opt;' identifier in the list is not required. Only entries for the process data in the form of 'processDataItem' are required.

Syntax:

```
<!ELEMENT processDataItem          (%naming;;
                                     pictureList?,
```



	<pre>(accessPath (bytePos, bitPos?)), datatype, specificProperty*, uses?, processDataItem*, instances?)> <!ATTLIST processDataItem %id.unique.req; direction (I Q) #REQUIRED signalType (D A) #REQUIRED processDataItemType CDATA #IMPLIED %enabled.def;></pre>
naming:	String with name of the process data object for display purposes, is not evaluated for DRIVECOM
pictureList:	Not used for DRIVECOM
accessPath, bytePos, bitPos:	String with syntax of the access path to a data object referring to the DriveServer specification
dataType:	String with data type
specificProperty:	Not used for DRIVECOM
uses:	Not used for DRIVECOM
processDataItem:	Is used for constructing a data object from several process data objects, not used initially for DRIVECOM
%id.unique.req;:	Unique ID for the process data object
direction:	Information of the direction of transmission
signalType:	Analogue or digital
processDataItemType:	Not used for DRIVECOM
%enabled.def;:	Activation of the object, default: YES

3.5.2. ParameterItemList

This element is used to describe existing parameter data objects.

Syntax:

```
<!ELEMENT parameterItemList
(%naming.opt;,
(parameterCategory |
parameterItem |
parameterGroup) +)>
```

The optional '%naming.opt;' identifier in the list is not required. Only entries for the parameter data in the form of 'parameterItem' are required.

```
<!ELEMENT parameterItem
(%naming.opt;,
pictureList?,
(accessPath |
(bytePos, bitPos?)),
```



```
datatype,  
specificProperty*,  
(%values;)?,  
parameterItem*,  
instances?)>  
<!ATTLIST parameterItem      %id.unique.req;  
                               %access.req;  
                               parameterItemType CDATA #IMPLIED  
                               %enabled.def;>
```

For a description of the elements included, see 3.5.1 ProcessDataItem.

3.6. DeviceFunctionObject

The 'DeviceFunctionObject' element is not filled in the FDCML specification and is available for usage within DRIVECOM. The DriveServer-specific objects are described here

Syntax:

```
<!ELEMENT DeviceFunctionObject (dictionaryList?,  
                                helpFileList?,  
                                containerList?,  
                                componentList?,  
                                varTemplateList?,  
                                varList?,  
                                varEnumList?,  
                                menuList?,  
                                methodList?,  
                                redefineList?)>
```

3.6.1. DictionaryList

A list of dictionaries can be added in several places in the XML file, however only this list in the 'DeviceFunctionObject' is of interest for the DriveServer and is evaluated.

Syntax:

```
<!ELEMENT dictionaryList      (dictionary+)>  
<!ELEMENT dictionary          (file)>  
<!ATTLIST dictionary          %lang.req;  
                               dictID CDATA #IMPLIED>
```

Every dictionary comprises a reference to a file and the language supported by the file, as well as a 'dictID' with the aid of which it is possible to reference the dictionary from the other parts of the XML file.

3.6.2. HelpFileList

A list of help files can be added in several places in the XML file, however only this list in 'DeviceFunctionObject' is of interest for the DriveServer and is evaluated.

Syntax:

```

<!ELEMENT helpFileList          (helpFile+)>
<!ELEMENT helpFile              (file)>
<!ATTLIST helpFile              %lang.req;
                                helpFileID CDATA #IMPLIED>

```

Every entry comprises a reference to a file and the language supported by this file, as well as a 'helpFileID' with the aid of which it is possible to reference the help file from other parts of the XML file.

3.6.3. containerList, componentList

The entries in 'containerList' and 'componentList' are used to allocate basic devices to possible modules and vice versa.

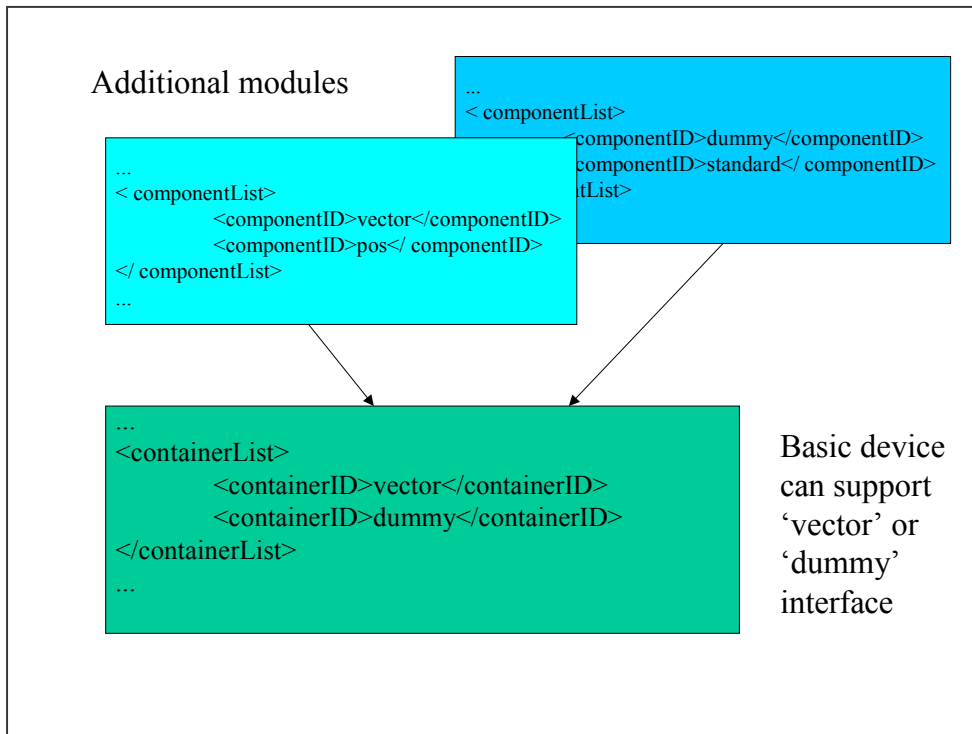


Figure 2: Relationship basic device/module

Figure 2 shows the relationship between the basic device and the matching module types. Thus the basic device, for instance, makes known that the (abstract) 'vector' and 'dummy' interfaces are supported. This signifies that modules that can establish a connection to one of these interfaces (e.g. by means of physical insertion) are thus potentially capable of being 'plugged into' the basic device. This information is necessary for the DriveServer during the identification of the drives so that, if necessary, additional algorithms can be run for the identification of the modules.

A module for its part can also support several interfaces and thus be used for several types of basic device

Syntax:

```

<!ELEMENT containerList        (containerID+)>
<!ELEMENT containerID         (#PCDATA)>
<!ELEMENT componentList       (componentID+)>

```



<code><!ELEMENT componentID</code>	<code>(#PCDATA) ></code>
containerID:	Container feature of the drive components, manufacturer-specific code for the interface property of the container. In this way the container makes known its type. Using the manufacturer-specific algorithm for the drive/description allocation, it is thus possible, if necessary, to search for other components on the drive and identify these.
componentID:	Container feature of the drive components, manufacturer-specific code for the interface feature of the component. Usage see above.

3.6.4. varTemplateList

In this section templates can be defined from which the variables/parameters can inherit all properties except the name. Template names must be unique in the XML file (and in the imported files).

In addition, each template element has the propID attribute (not shown here, see 4 Property IDs); this defines the property value for the later item attribute. The DriveServer can apply these definitions and place them in the list of available properties.

Syntax:

<code><!ELEMENT varTemplateList</code>	<code>(varTemplate+) ></code>
<code><!ELEMENT varTemplate</code>	<code>(classList,</code>
	<code>type,</code>
	<code>DFOAccess,</code>
	<code>limits?,</code>
	<code>readTimeout?,</code>
	<code>writeTimeout?,</code>
	<code>unit?,</code>
	<code>(help* helpRef helpFileRef)?,</code>
	<code>formatstring?,</code>
	<code>defaultvalue?,</code>
	<code>preReadFunc?,</code>
	<code>postReadFunc?,</code>
	<code>preWriteFunc?,</code>
	<code>postWriteFunc?,</code>
	<code>scalingFactor?,</code>
	<code>paramsetList?,</code>
	<code>visible?) ></code>
<code><!ATTLIST varTemplate</code>	<code>%id.unique.req;></code>
varTemplate:	Specific template type with unique name used for referencing.
<code><!ELEMENT classList</code>	<code>(class+) ></code>
<code><!ELEMENT class</code>	<code>EMPTY ></code>
<code><!ATTLIST class</code>	<code>c (DYNAMIC </code>
	<code>LOCAL </code>



	<pre>OPERATE ALARM) #REQUIRED></pre>
classList, class:	<p>Here the way in which the variables can be used by the DriveServer is defined. Several attributes can be set.</p> <p>OPERATE: Normal parameter used to control the drive</p> <p>DYNAMIC: The variable can be changed dynamically from the device. The DriveServer can then, if necessary, request this value cyclically from the BusServer.</p> <p>LOCAL: Variable is only used locally in the DriveServer and is not provided to the drive</p> <p>ALARM: Variable used for signalling alarms</p>
<pre><!ELEMENT type <!ATTLIST type</pre>	<pre>EMPTY> t (VT_BOOL VT_UI1 VT_I2 VT_I4 VT_BSTR VT_DATE VT_CY VT_R4 VT_R8 ARRAY_OF_VT_UI1) #REQUIRED enum (no enumerated bit_enumerated) "no" enum_ref IDREF #IMPLIED></pre>
type:	<p>OPC data type for the data point, if this is an enum, the reference to an enum structure is also expected. The type can still be overwritten in the variable, e.g., to add an enumeration.</p>
<pre><!ELEMENT DFOAccess <!ATTLIST DFOAccess</pre>	<pre>EMPTY> %access.req;></pre>
DFOAccess:	<p>Access right (read ("RO", write ("WO"), read&write("RW"))</p>
<pre><!ELEMENT limits <!ATTLIST limits <!ELEMENT minval <!ATTLIST minval <!ELEMENT maxval <!ATTLIST maxval</pre>	<pre>(minval, maxval)?> funcRef IDREF #IMPLIED> EMPTY> val CDATA #REQUIRED> EMPTY> val CDATA #REQUIRED></pre>
limits:	<p>Limit for the template/variable that is either a constant of the related data type, or has a reference to a method that the DriveServer must open on checking the range. If no range</p>



limits are given, the value range is the range of the related data type.

```
<!ELEMENT readTimeout          (#PCDATA) >
<!ELEMENT writeTimeout         (#PCDATA) >
```

read/writeTimeout:

Time in ms, by which a synchronous Read/Write must be successfully completed. On remote accesses the DriveServer must, if necessary, perform an additional calculation of the delay.

```
<!ELEMENT unit                  (#PCDATA) >
<!ATTLIST unit                  kind (string|
                                   proc|
                                   varRef) "string" >
```

unit:

Unit for the template/variable. The unit can be a string constant, a reference to a method or to a variable (only on the usage of the attribute within a variable).

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

defaultvalue:

Value for the template/variable to which the DriveServer sets all variables of this type after the configuration.

helpRef:

Reference to a language file given in the `dictionaryList` addressed using the `helpID`. The `textID` entry defines the text to be displayed.

helpFileRef:

Reference to a language file given in the `helpFileList` addressed using the `helpFileID`. The `helpID` entry is an entry point within the help.

```
<!ELEMENT preReadFunc          EMPTY>
<!ATTLIST preReadFunc ref       IDREF #REQUIRED>
```

```
<!ELEMENT postReadFunc         EMPTY>
<!ATTLIST postReadFunc         ref IDREF #REQUIRED>
```

```
<!ELEMENT preWriteFunc         EMPTY>
<!ATTLIST preWriteFunc         ref IDREF #REQUIRED>
```

```
<!ELEMENT postWriteFunc        EMPTY>
<!ATTLIST postWriteFunc        ref IDREF #REQUIRED>
```

X_YFunc:

Reference to a method defined in the `methodList` section that is run by the DriveServer at the time to be stipulated (before/after reading/writing the variable).

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

scalingFactor:

Factor by which the value from the BusServer is multiplied on reading, or divided on writing.

```
<!ELEMENT formatstring         EMPTY>
```



<code><!ATTLIST formatstring</code>	<code>str CDATA #REQUIRED</code>
formatstring:	Formatting of a value in accordance with C conventions (e.g. for printf), the formatting can be requested by the OPC client as a property. The formatted string can also be formed in the DriveServer and can then be requested via a property.
<code><!ELEMENT paramsetList</code>	<code>(set*) ></code>
<code><!ELEMENT set</code>	<code>(#PCDATA) ></code>
paramsetList, set:	Entry of an integer for the data record that contains the variable; paramsetList is the list of these parameter sets
<code><!ELEMENT visible</code>	<code>(#PCDATA) ></code>
visible:	This attribute contains an integer. In this way individual variables can be displayed/hidden depending on the context. Default setting for all variables is the value '0', i.e. the variable is always visible. Variables become visible if the context in the DriveServer has a value <code><= visible</code> .

3.6.5. Variable

In this section all variables/parameters for the drive can be listed. The variables always contain a reference to a `varTemplate`. All attributes are inherited from this template. The individual attributes per variable (for a description see 3.6.4 `varTemplateList`) can be explicitly overwritten and then only apply to this variable.

<code><!ELEMENT varList</code>	<code>(var*) ></code>
<code><!ELEMENT var</code>	<code>(%labels;,</code> <code>classList?,</code> <code>type?,</code> <code>uses?,</code> <code>DFOAccess?,</code> <code>limits?,</code> <code>readTimeout?,</code> <code>writeTimeout?,</code> <code>unit?,</code> <code>(help* helpRef helpFileRef)?,</code> <code>defaultvalue?,</code> <code>preReadFunc?,</code> <code>postReadFunc?,</code> <code>preWriteFunc?,</code> <code>postWriteFunc?,</code> <code>scalingFactor?,</code> <code>paramsetList?,</code> <code>visible?) ></code>
<code><!ATTLIST var</code>	<code>name ID #REQUIRED</code> <code>varTemplate IDREF #REQUIRED ></code>



var:	The entry corresponds to a device parameter. The <code>name</code> attribute is the unique identifier within the XML files and in the DriveServer. <code>varTemplate</code> is the reference to the template.
labels:	Variable label to be displayed, this can be either one or more constant strings in the various languages, or a reference to an external dictionary entry.
uses:	Reference to a process or parameter object from the 'communicationEntity' section with the name "DRIVECOM" (see 3.5 DeviceManagerObject).

3.6.6. Lists

Lists are used to allocate numerical values to text. A list can be referenced by several variables. The task of the DriveServer is to resolve the references. The entries on the list are supplied to the items as properties. The task of the client is to display the corresponding text instead of the item value.

```
<!ELEMENT varEnumList          (enum+)>
<!ELEMENT enum                 (enum_entry+)>
<!ATTLIST enum                 name ID #REQUIRED>
```

enum: A list with a unique name used to reference it.

```
<!ELEMENT enumEntry           EMPTY>
<!ATTLIST enumEntry           value CDATA #REQUIRED
                                %labels;>
```

enumEntry: Individual entry in a list. The text can be given either as one or more constant strings in the various languages, or as a reference to a dictionary entry. Depending on the context, value is the numerical value of the variable or the corresponding bit position.

3.6.7. Menu

The menu structure is used to set up the name space for the DriveServer, i.e. the information that is obtained on browsing. The menus can be arranged hierarchically within each other. The variables are entered on the pages if they are visible at run time/browse time.

```
<!ELEMENT menuList           (menu+)>
<!ELEMENT menu               (%labels;,
                                visible?,
                                (help*|helpRef|helpFileRef)?,
                                m_entry+)>
<!ATTLIST menu               %id.unique.req;>
```

menu: A menu has a unique name within the XML file. In addition, the text displayed is also defined using labels. A menu can be subjected to rules on whether it is visible. If this value is not given, it has the value '0' and is always visible. A reference to a help entry is optional (can only be used from OPC V2.03).

```
<!ELEMENT m_entry           EMPTY>
<!ATTLIST m_entry           kind (var|menu) #REQUIRED
```



```
ref IDREF #REQUIRED>
```

m_entry: An individual menu entry can be either a variable reference or a menu reference.

3.6.8. Methods

The 'methodList' section contains the method definitions that are referenced from other areas of the XML file. The methods are to be entered in script language and can be run by the DriveServer at the appropriate time.

```
<!ELEMENT methodList (method+)>
<!ELEMENT method (#PCDATA)>
<!ATTLIST method %id.unique.req;
lang (VBSCRIPT|JSCRIPT) "JSCRIPT">
```

method: A method is uniquely described by its name. In addition, it is possible to specify one of the two script languages in which the method is written. The method itself is stored in the related script language in the element text (however as CDATA-Cast). Each method opened directly by means of references must have a parameter. This parameter is filled with the name of the related variable at run time.

3.6.9. Redefinitions

The new definitions facilitate the declaration of elements (data types, variables, lists, methods, definitions, see above) that have already been defined in the XML document, or in the files incorporated. These elements are erased and re-created.

```
<!ELEMENT redefineList (varTemplateList?,
varList?,
enumList?,
menuList?,
methodList?)>
```

4. Property IDs used

In this section all property IDs additional to the OPC specification are listed. The IDs for the DRIVECOM start at 6000.

ID offset	Data type for the returning VARIANT	Description
0	VT_BSTR	Class of the variable ("DYNAMIC", "LOCAL", "OPERATE", "ALARM")
1	VT_ARRAY VT_BSTR	Help on the parameter or menu entry on browsing, the first array entry is the URL to the file, the second the entry point
2	VT_ARRAY VT_BSTR	Enumeration, this array is available if the parameter has a list definition. It has as many entries as defined in the list. The separation between value/text is made via the position in the string. The text starts from the 10th character. The characters before this form the numerical value and can be converted numerically if necessary.
3	VT_ARRAY VT_BSTR	Bit enumeration, if the parameter has a bit-orientated list



		definition, this array is available. It has as many entries as are defined in the list. The separation between bit position/text is made by the position in the string. The text starts from the 10th character. The characters before this form the numerical value and can be converted numerically if required.
4	VT_I4	Read timeout in ms
5	VT_I4	Write timeout in ms
6	varies	Default value, the data type varies as per the definition, corresponds to the canonical data type that can be requested under property ID '1'
7	VT_R4	Scaling factor
8	VT_BSTR	Formatting string corresponding to the "C-printf" formatting
9	VT_BSTR	Formatted current value of the parameter corresponding to the formatting string
10	VT_I4	Visible value, can only be reached if variable is also actually visible - and thus defined in the name space
11	VT_BSTR	Access paths for the parameter in the drive, not to be confused with the complete access path for an item in the DriveServer (here routing and hierarchy information is required). This string is used in the DriveServer to form the item access path.
12	VT_BSTR	Data type for the parameter in the drive corresponding to FDCML
13	VT_I4	Number of bytes to be transmitted

5. Dictionary (DTD for the language file)

The individual dictionaries are created by language and can be created by the manufacturer as required. The DriveServer replaces the text references with the content of the language files on loading the device descriptions. Whether a dictionary is used at run time, and which dictionary is used, depend on the settings in the DriveServer. The references to the dictionaries are stored in the DriveServer-XML configuration file (see 3.6.1 DictionaryList).

The language files are created using the following syntax:

```
<!ELEMENT DRIVECOM_LANG_V01      (text_entry*)>
<!ATTLIST DRIVECOM_LANG_V01      xml:lang CDATA #REQUIRED>
<!ELEMENT text_entry              EMPTY>
<!ATTLIST text_entry              t_id ID #REQUIRED
                                   text CDATA #REQUIRED>
```

DRIVECOM_LANG_V01 xml:lang: Identifier for the language that the dictionary supports

text_entry t_id: String with ID that is referenced in the device descriptions

text_entry text: String with the actual text content

6. Help files

The help system is not specified in this context. The only precondition is that the referenced files (*.hlp, *.html, *.pdf, etc.) can be accessed. The actual calls must then be performed by the client. This

receives the information on the file and entry point from the DriveServer via the property request for an item.

7. Interface

7.1. Overview

The DriveServer runs scripts in conjunction with the DS_DriveInstance interface. For this purpose two objects are placed in the DriveServer script environment prior to the actual start of the script. These two objects can be addressed in the script using the following names:

- 'OPC_DriveServer': An instance of the OPC-Wrapper-Server to which a 'Connect' has already been performed in the DriveServer.
- 'DS_DriveInstance': A proxy for access to the DS_DriveInstance interface.

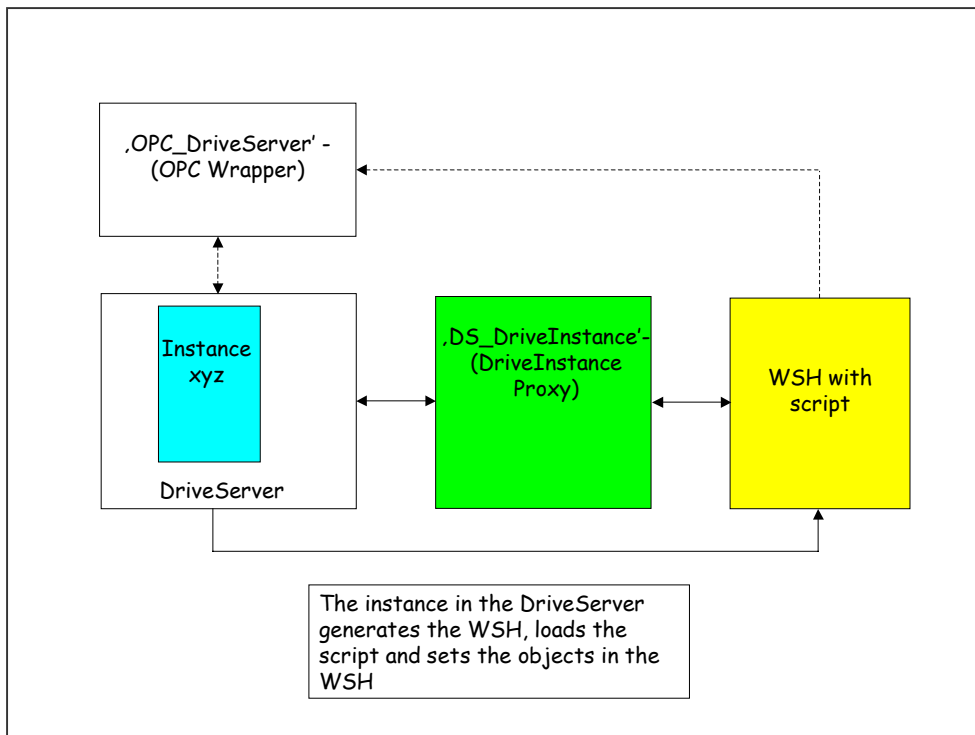


Figure 3: Data flow between instance and script

These two objects are set at script level basis such that they are available the entire time the script is run. Once the script has been processed, these objects are deleted from the DriveServer. In addition, the variable 'strPath2Drive' is set in the script environment by the DriveServer.

The 'DS_DriveInstance' interface is necessary to be able also write to the parameter properties. The OPC interface only allows these values to be read.

7.2. Function description

The property IDs passed to the functions correspond to the OPC specification and the definitions in the configuration DTD 'DRIVECOM_EDD_V01'.

**GetPropertyValue(varVariableName, varPropId) As Variant**

[in] VARIANT varVariableName: Unique name of the variable

[in] VARIANT varPropId: Number of the property

The 'GetPropertyValue' function supplies the property value for the variable given and for the property ID requested. The variable name corresponds to the unique identifier in the XML document.

SetPropertyValue(varVariableName, varPropId, pvarPropValue) As Boolean

[in] VARIANT varVariableName: Unique name of the variable

[in] VARIANT varPropId: Number of the property

[in] VARIANT varPropValue: New value of the property

The 'SetPropertyValue' function sets the property value to the variable given and to the property ID requested. The variable name corresponds to the unique identifier in the XML document. The function returns TRUE if the new value from varPropValue has been applied.

7.3.IDL

```
[
    object,
    uuid(F7A15FA1-9F82-11d4-8DA3-00E07D815C6E),
    dual,
    helpstring("IDS_DriveInterface-Schnittstelle"),
    pointer_default(unique),
    oleautomation
]
interface IDS_DriveInterface : IDispatch
{
    [id(1), helpstring("Methode GetPropertyValue")]
    HRESULT GetPropertyValue ([in] VARIANT varVariableName,
                              [in] VARIANT varPropId,
                              [out, retval] VARIANT* pvarPropValue);

    [id(2), helpstring("Methode SetPropertyValue")]
    HRESULT SetPropertyValue ([in] VARIANT varVariableName,
                              [in] VARIANT varPropId,
                              [in] VARIANT varPropValue,
                              [out, retval] VARIANT_BOOL* pbResult);
}
```

8. Syntax of the language DTD

```
<!-- DRIVECOM Sprachdatei -->
<!ELEMENT DRIVECOM_LANG_V01 (text_entry*)>
<!ATTLIST DRIVECOM_LANG_V01 xml:lang CDATA #REQUIRED>

<!-- Texteintrag -->
<!ELEMENT text_entry EMPTY>
<!-- Text-ID zum Auffinden des Texteintrags, referenziert aus XML-Datei -->
<!ATTLIST text_entry t_id ID #REQUIRED
                    text CDATA #REQUIRED>
```