SIMATIC NET

GSDML Getting Started Made Easy

Date August 23, 2012

Getting Started

<<…>> translator's

<<this page is blank>>

SIEMENS

SIMATIC NET

GSDML

Getting Started

Simple Instructions for Creating a
Device Description Data (GSD) File
for PROFINET IO)

Version: 1.3
Date: 08/12

Liability Exclusion

We have checked the contents of this document regarding agreement with the hardware and software described. Nevertheless, deviations can not be ruled out so that we do not guarantee complete agreement. However, the data in this document is checked periodically. Required corrections are included in subsequent editions. We appreciate suggestions for improvement.

Copyright

Subject to technical changes.

Version IDs

Version	Date	Change
V 1.0	September 4, 2008	Initial version
V 1.1	March 2, 2009	Corrections incorporated
V 1.2	April 19, 2012	Updated to the current version
V 1.3	August 23, 2012	Updated to the current version

Content

1 About this Document

This objective of this document is to facilitate getting acquainted with the application and the structure of the GSDML. It does not replace the standard document, but builds on it; and based on a few examples, wants to show you how the elements and attributes of a GSDML document work.

The document "GSDML Specification for PROFINET IO" Is made available by means of the Webserver of the PNO (www.profinet.com → Downloads).

If there should be contradictions with respect to the GSDML specification, the GSDML specification is the standard.

In the future, the document is adapted to the requirements of the market. Feedback is welcome for that reason.

Please report missing passages or incompatibilities to ComDeC@siemens.com.

The syntax of XML itself and the structure of schematic files will not be explained here. There are many good books serving this purpose.

Note: Some Links may currently grasp to nothing. We are working on it and will update it when they are correctly available.

2 Introduction

By introducing the GSDML (**G**eneral **S**tation **D**escription **M**arkup **L**anguage), the feature of XML based documents to generate any hierarchical levels was used to map the hierarchical device model as unchanged as possible.

The result is a description language that is able to describe over several levels the attributes of a device family. When the XML schematics representation was generated, the attempt was made to include as much as possible of the GSD semantics, but to also, where it was advisable and necessary, to redesign the mapping.

Note on Terminology

GSDML is one language to describe PROFINET IO field devices. By using this language, a GSD (General Station Description) is generated in turn. For that reason, it is correct to refer to a "GSD file" although it is structured in XML notation.

When the term "GSD" or "GSD file" is used in this document, it always refers to the form based on the XML If the keyword-based GSD is referred to, it will be mentioned explicitly.

2.1 What is New

Compared with the keyword-based GSD (PROFIBUS), the following are essential changes regarding the XML-based GSD, in addition to the syntactical change because of XML:

- The "Device Access Point" (DAP) is described explicitly for the GSDML. For the GSD on the other hand, only the DP slave is described as a whole. The result of adding the DAPs is that almost all parameters of the device are described at the DAP element.

- A GSDML document can contain any number of DAP definitions. This allows for generating a file for a device family (and not only for individual field devices). The advantage is obvious, particularly to field devices that have a modular configuration. The description of the modules can be maintained centrally in **one** GSD file – the generation and update effort is clearly reduced in comparison to a keyword-based GSD since there, the module descriptions are present redundantly and have to be kept identical in the different files 'manually'.

- A GSDML document can maintain any number of foreign language texts in **one** file. This leads to a further reduction of the number of required files since in the keyword-based GSD, a separate file is required for each language.

Add to this the capabilities that result in using the XML standard: validation of XML files by means of schematic description and transformation into other formats by means of XLS.

3 Using Schematic Files for Validation

The structure of an XML document can be checked by using a schematic file. In this connection, we refer to the validation of a document. During this process, a check is performed, for example, whether the element structure and the attributes used in the XML document agree with the schematic definition. For example, the schematic file contains information whether an attribute has to be present or whether this attribute is optional.

The document is validated with an XML parser. An overview is provided in the section "Tools".

It is also possible to check the correct structure of a GSD file with the tool "PROFINET XML Viewer“. It is available for downloading from the PNO web pages. If you should be using this tool, you don’t need any expertise regarding the functions and application of schematic files, since the PROFINET XML Viewer installs it automatically and uses it for validation. In this case, you can skip this chapter. More information about the tool "PROFINET XML Viewer“ is also provided in the section "Tools“.

3.1 Using the Schematic Files in the GSDML Context

The PNO working committee generates, updates and makes available the GSDML schematic files for downloading by means of the PNO Webserver.

Using a validated XML parser, the author of a GSD file is now able to check an existing GSD file against the schematic and correct it if necessary.

When importing a GSD file to a tool, the tool also can -as the first step- automatically check the file against the schematic files and if there is an error, reject the import of the GSD file.

The figure below shows in principle the sequence for schematic handling:

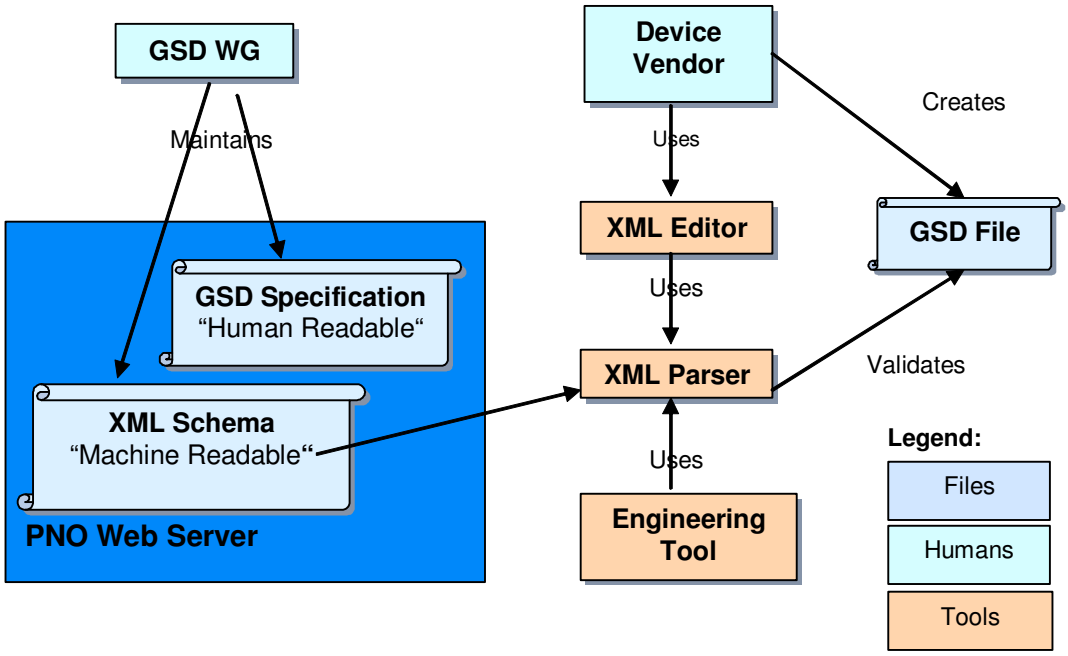


Figure 1: GSDML Work Flow

3.2 Schematic GSDML Files

Four schematic files are necessary to validate a GSD file. These are arranged as follows:

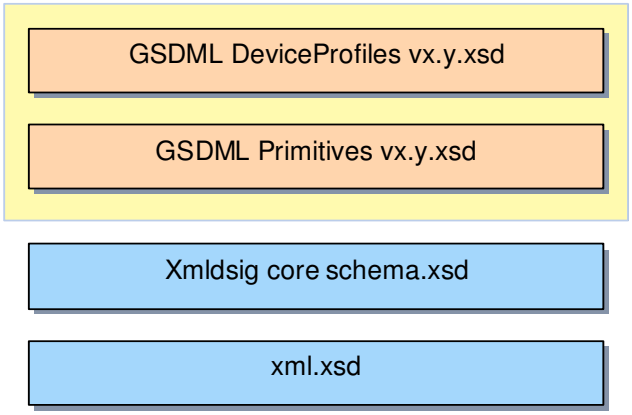


Figure 2: Schema Layer

The schematic files are updated and developed further by different groups. The GSDML schematics are based on the fundamentals of the XML@PROFIBUS Guideline that also can be loaded by means of the PNO Webserver. To this end, the URL for name space identification also is used to download the schematic files.

The meaning of the schematic files in detail:

xml.xsd (Namespace http://www.w3.org/XML/1998/namespace)

This schematic defines attributes and an attribute group suitable for use by schematics wishing to allow xml:lang or xml:space attributes on elements they define. The World Wide Web consortium makes this schematic file available for downloading under the URL http://www.w3.org/2001/xml.xsd.

xmldsig-core-schema.xsd (name space http://www.w3.org/2000/09/xmldsig#)

This document specifies XML digital signature processing rules and syntax. XML signatures provide integrity, message authentication, and/or signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere. The World Wide Web consortium makes this schematic file available for downloading under the URL http://www.w3.org/2000/09/xmldsig#.

GSDML Primitives-vx.y.xsd (name space "http://www.profibus.com/GSDML/2003/11/Primitives")

This schematic file contains fundamental structure definitions.

x.y in the file name is a substitution in this case for the version of the schematic. The first version is designated as 1.0.

GSDML DeviceDescription-vx.y.xsd (Name space "http://www.profibus.com/GSDML/2003/11/DeviceDescription")

This is the actual schematic document for defining the GSDML. It uses the GSDML primitives schematic, and can only be used in conjunction with it.

Bu using the name space URL, the schematic files can be loaded from the corresponding Webserver. When entering the URL, you will get to an HTML page where you can load the corresponding schematic file to your hard disk. To this end, set up a directory ".xsd“ for storing these four schematic files.

4 Rules for Generating the GSD File Name

In contrast to the keyword-based GSD, the length of the file name is not limited to eight characters. The file name has to correspond to the following rules:

GSDML-[GSDML schematic version]-[manufacturer name]-[device family name]-[date].xml

The following rules apply to the keywords in brackets:

GSDML schematic version: Version ID of the referenced schematic. This version ID has to correspond to the version ID in the file name of the GSDML DeviceProfile-[GSDML schematic version].xsd. Optional the release time of the GSDML based file in format hhmmss (Optional). “hh” is “00” up to “24”.

Manufacturer name: Name of the device manufacturer. Hyphens and blanks are permitted in the name. To prevent generating the same file name for different manufacturers it is recommended -in addition to the name- to also integrate the PNO ID (VendorID) as part of the name.

Name of the device family: defines which device family is described in the GSDML. Hyphens and blanks are permitted in the name. The name should correspond to the attribute content "Productfamily" of the element "GSDML_DeviceIdentity".

Date: Date when the GSD file was released, in the format yyyymmdd. However, the manufacturer has to take into account that different GSD files with the same date ID are not released for the same device family. <<?>>

Time: Time when the GSD file was released, in the format hhmmss (optional).

This makes counting the version of the GSD file itself superfluous and ensures that different versions don’t overwrite an existing file.

If a GSD file with a more recent date/time is installed for the same product family, as a rule only the field devices of the latest GSD file are displayed in an engineering tool in the selection catalog.

Example of a valid file name:

GSDML-V1.0-Siemens-002A-ET200X-20030818.xml

5 Rough Structure of a GSD File

5.1 Processing Instructions for the XML Parser

As for any XML document, the processing instructions have to be defined at the start of a GSD file. As a rule, they look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Here, only the definition of the encoding can be varied. There are no restrictions for this in the GSDML – all encodings corresponding to the standard can be specified. However, there are editors that have problems with Unicode data and who in particular don't interpret correctly the byte order mark. Otherwise, using UTF-8 is recommended as a rule since all country-specific special characters can be represented with it, without this causing the size of the file to increase significantly (as in the case of UTF-16 or UTF-32).

5.2 Root Element ISO15745Profile

The root element of a GSD file is "ISO15745Profile". The name space used and the import instruction for the schematic file **must** be specified.

```
<ISO15745Profile xmlns="http://www.profibus.com/GSDML/2003/11/DeviceProfile"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.profibus.com/GSDML/2003/11/DeviceProfile
..\XSD\GSDML-DeviceProfile-v1.0.xsd">
```

The schematic file should be located in a directory "XSD" parallel to the GSD file! This rule is necessary so that an engineering tool or a certification center (where the GSD files of the most varied manufacturers are stored together) does not have to store the identical schematic files multiple times in order to perform a validation.

As specified in ISO 15745, an ISO15745Profile consists of a ProfileHeader and a ProfileBody.

The figure below shows the structure of a GSD file in principle:

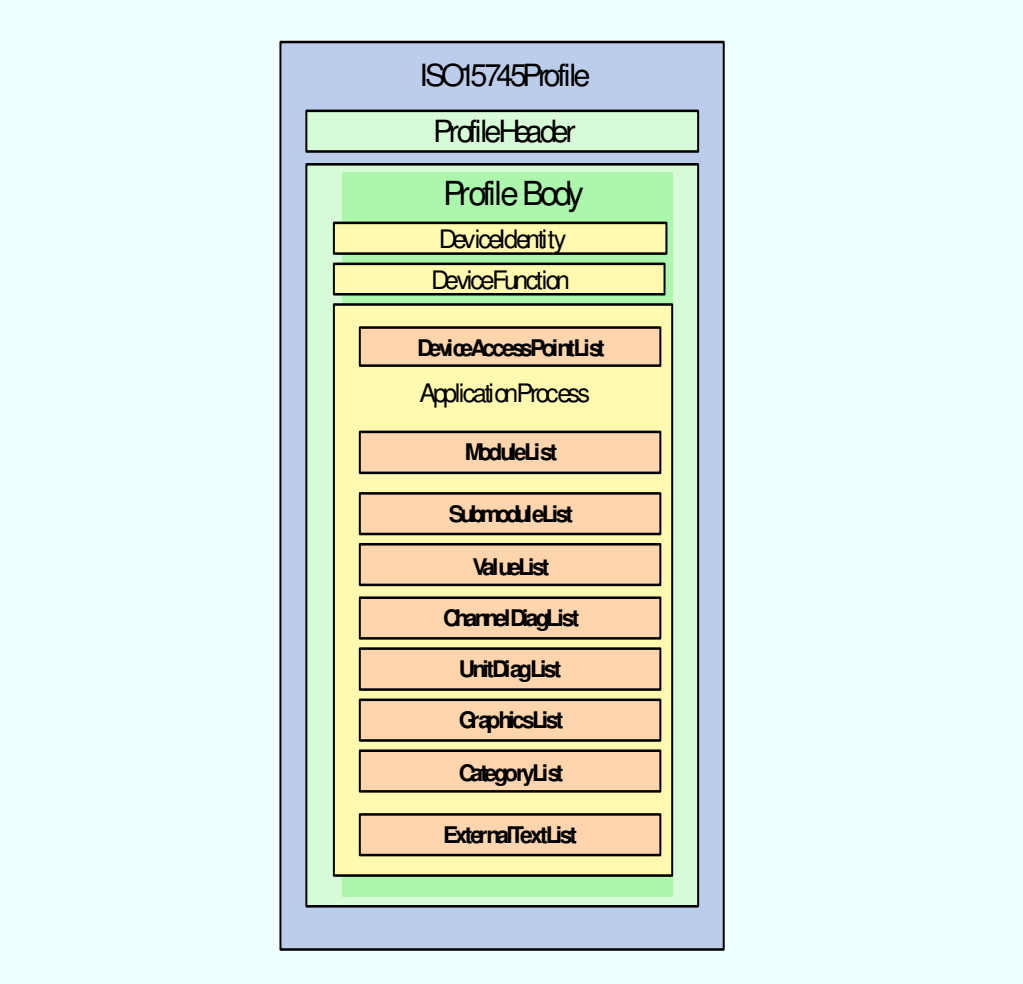


Figure 3: Structure of a GSD File in Principle

5.2.1 ProfileHeader

The ProfileHeader is to be assigned as follows:

```
<ProfileHeader>
    <ProfileIdentification>PROFINET Device Profile</ProfileIdentification>
    <ProfileRevision>1.00</ProfileRevision>
    <ProfileName>Device Profile for PROFINET Devices</ProfileName>
    <ProfileSource>PROFIBUS Nutzerorganisation e. V. (PNO)</ProfileSource> <<user organization>>
    <ProfileClassID>Device</ProfileClassID>
    <ISO15745Reference>
        <ISO15745Part>4</ISO15745Part>
        <ISO15745Edition>1</ISO15745Edition>
        <ProfileTechnology>GSDML</ProfileTechnology>
    </ISO15745Reference>
</ProfileHeader>
```

5.2.2 ProfileBody

The ProfileBody contains the actual data of the field device and consists of three parts:

- "DeviceIdentity" includes information for identifying the field device,
- "DeviceFunction" includes data that describe the function.
- "ApplicationProcess". This is the main part of the description data.The most important sections of the ApplicationProcess block are listed in the following sub-chapters.

5.2.2.1 DeviceAccessPointList

This section contains the description of the individual Device Access Points (network access point). As mentioned above, a GSD file can contain the description for any number of interface modules. The sum of all Device Access Points forms a Device Family. The degree of expansion of a DAP can be configured for modular field devices. Different modules (base modules) can be assigned to each DAP.

5.2.2.2 ModuleList

This section includes the description of the individual modules of a field device. These can be insertable (in the case of field devices set up in the modular mode) or permanently integrated in a field device.

5.2.2.3 SubModuleList

This section includes the description of the individual sub-modules of a field device. These can be insertable (in the case of field devices set up in the modular mode) or permanently integrated in a field device

5.2.2.4 ValueList

This section includes the ValueList. It contains -the individual parameters of a field device- the assignment between a concrete value and the associated text.

5.2.2.5 ChannelDiagList

This section includes the ChannelDiagList. It contains the assignment between a channel error of a field device and the corresponding texts.

5.2.2.6 UnitDiagList

This section includes the UnitDiagList. It describes the structural configuration of the generic diagnostic indications of a field device.

5.2.2.7 GraphicsList

This section includes the references to graphic representations of a field device.

5.2.2.8 CategoryList

This section includes a certain category (for example, Digital Input, Analog Output, etc.). It is used for the arrangement and better locatability within the module catalog of an engineering system.

5.2.2.9 ExternalTextList

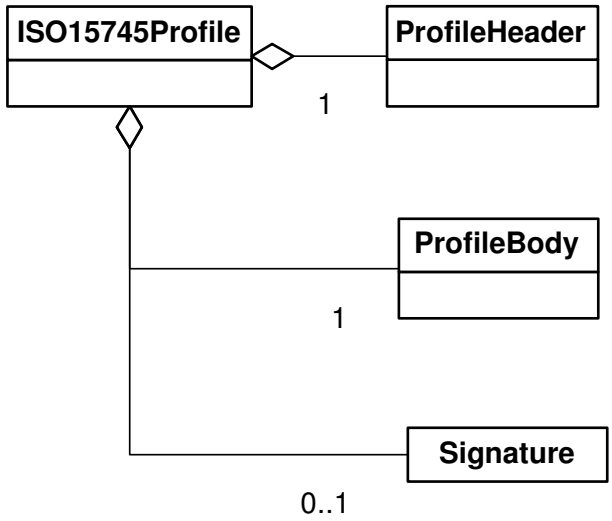
This section includes the "ExternalTextList“ of the GSDML. All texts are stored that can be referenced by the other sections. More detail is provided for this in the chapter "Integration of Foreign Languages".

6 Generating a GSD file

The GSDML schematic can be useful, of course, to generate a GSD file "from scratch". The rules it contains regarding the syntax of a GSDML-based file suffice to generate a correct and validating GSD file.

However, it is simpler to start with the existing GSD file and adapt it to its own field devices. A basis to start is provided, for example, in the sample files of the PNO that can be loaded from <<?>> the PNO Webserver under the address http://www.profibus.com/pall/applications/products/article/00085/.

The graphics in the GSD file are shown in UML notation and facilitate readability as well as understanding the hierarchy of the definitions to be entered. Here a brief example:



For the sake of simplicity, please read the UML schematic from the left to the right. This also opens up for you the hierarchy of the data structure. The numerical values specified have the following meaning:

1	This definition has to be specified always
0 … 1	This definition is optional
0 … *	The value range means that nothing has to be specified, but also that any number of definitions can be specified
1 … *	This value range means that at least one definition has to be specified, but any number of definitions can also be specified

If you are assuming an existing GSD file, you have to perform -in addition to the adaptations to your field devices- the following adaptations in any case:

- The file name has to be structured according to Chapter 4.
- As "VendorID“ (manufacturer ID), enter the number that is defined for your company at the PNO. An overview is provided under http://www.profibus.com/IM/Man_ID_Table.xml. Please note that you enter the number in the GSDML as hexa-decimal.

- Change the DeviceID to the ID of your device family
- If needed, adapt the "MainFamily" that represents a rough technological classification of your field device. This attribute is used to structure the HW catalog in the engineering system. Refer also to Chapter 15.2.
- In the attribute "ProductFamily", enter the name of your product family.

The figure below shows the locations that were changed in the GSD file:

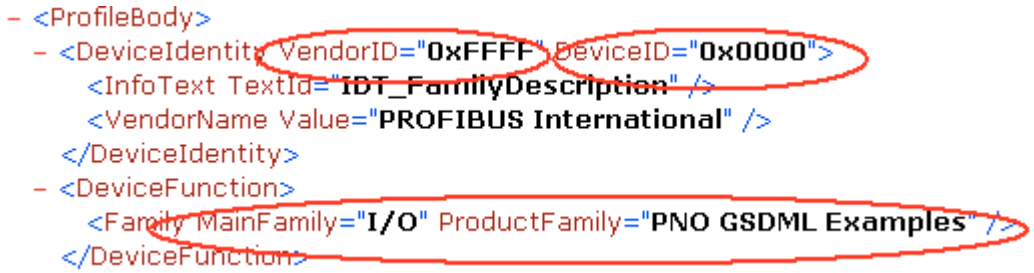


Figure 4: VendorID, DeviceID and Product Information in the GSD

Example for Configuring a Device Identification:

The structure of a DeviceID can be specified within the company but should be uniform within the same company (please coordinate with the Marketing Department and possibly Sales).

Below an example to illustrate the idea:

Vendor_ID		Device_ID	
		Device Class	Device Family
00	**2A**	Byte	Byte
Example for	Siemens	00= Development Kit	00= reserved 01= ERTEC 200 DevKit 02= ERTEC 400 DevKit 03= …
		01= Controllers	01= S7 300 02= S7 400 03= …

7 Device Access Point List (DAP)

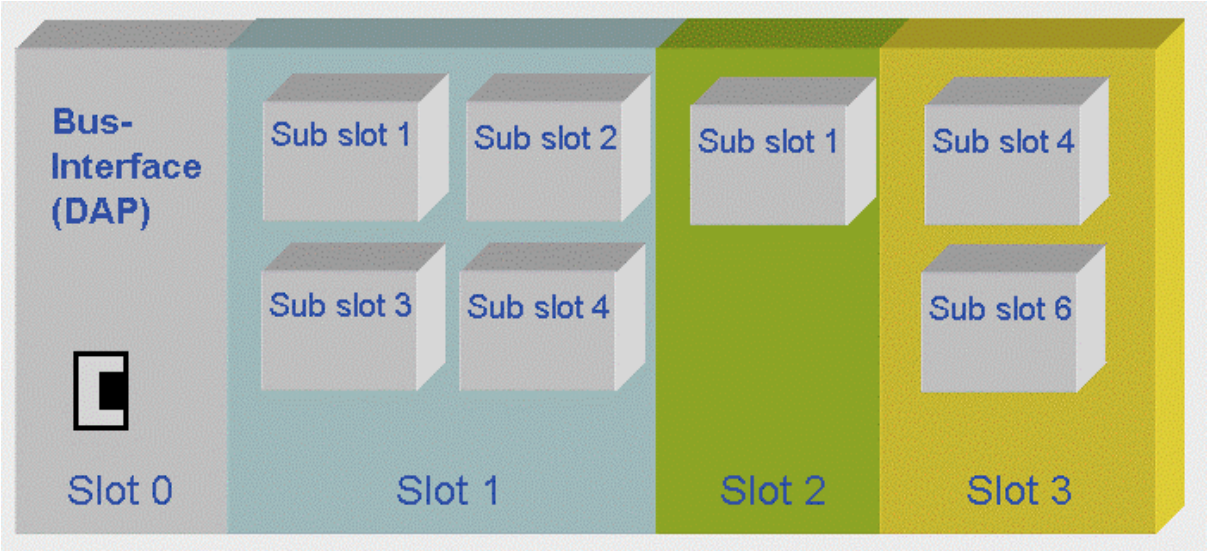


Figure x: Structure of a Modular Field Device with the Associated Modules.

The definitions with the DAP start within the ApplicationProcess. A DAP represents the bus interface and thus determines the essential features of the IO device. Therefore, the technical capabilities of the IO device and the possible modules with their attributes have to be described in the GSD file. Several DAPs can be defined in one GSD file.

The following is always associated with a DAP:

- Physical Device Management (PDEV); it contains all HW oriented definitions (refer also to the chapter "Physical Device Management“).
- The supported Real Time classes
- Redundancy definitions
- The port definitions for each switch port

Examples of these definitions are provided in the respective chapters.

Example: Here, for example, a DAP based on the ERTEC 200 is defined. Since the ERTEC 200 has two switch ports, they also have to be defined within the DAP with the associated sub-slot numbers

```
<ApplicationProcess>
  <DeviceAccessPointList>
    <DeviceAccessPointItem ID="DAP 2" PhysicalSlots="0..16"
     ModuleIdentNumber="0x00000002" MinDeviceInterval="16"
     ImplementationType="ERTEC200" DNS_CompatibleName="ERTEC-DEVKit"
     ExtendedAddressAssignmentSupported="true" FixedInSlots="0"
     ObjectUUID_LocalIndex="1" RequiredSchemaVersion="V2.2"
     MaxSupportedRecordSize="4068"
     <ModuleInfo CategoryRef="ID_ERTEC200DEVKit">
       <Name TextId="TOK_Standard" />
       <InfoText TextId="TOK_ModInfo_InfoTextId_DAP2" />
       <VendorName Value="SIEMENS" />
       <OrderNumber Value="6GK1 953-0BA00" />
       <HardwareRelease Value="A1.0" />
       <SoftwareRelease Value="Z1.0" />
     </ModuleInfo>
```



```
<SubslotList>  <!—here, the definitions start for addressing the PDEV -- >
                <IOConfigData MaxInputLength="1440" MaxOutputLength="1440" />
<UseableModules>
  <ModuleItemRef ModuleItemTarget="ID_Mod_01" AllowedInSlots="1..16" />
  <ModuleItemRef ModuleItemTarget="ID_Mod_02" AllowedInSlots="1..16" />
  <ModuleItemRef ModuleItemTarget="ID_Mod_03" AllowedInSlots="1..16" />
</UseableModules>
<VirtualSubmoduleList>
   <VirtualSubmoduleItem ID="DAP 2" SubmoduleIdentNumber="0x0001">
     <IOData IOPS_Length="1" IOCS_Length="1" />
     <ModuleInfo>
     <Name TextId="TOK_Standard" />
     <InfoText TextId="TOK_ModInfo_InfoTextId_DAP2" />   </ModuleInfo>
  </VirtualSubmoduleItem>
</VirtualSubmoduleList>

 <SystemDefinedSubmoduleList>

Here, the Pdev definitions are located; refer to Chapter 8

 </SystemDefinedSubmoduleList>
```


8 Definition of the Physical Device Managements (PDev)

In the case of PROFINET, the physical interfaces of a field device and their switch ports as well as the performance is called PDev. An Ethernet interface is uniquely identified by its name, IP and MAC address, and can consist of several switch ports.

In a field device, all HW oriented port information and transition parameters are stored retentively in the PDev. These are organized as follows:

- **Interface Submodule** corresponds to an instance. Here, for example, the planning data for IRT communication, clock synchronization, media redundancy etc, are stored.
- **Port Submodules** Data for the port settings, neighborhood information, fiber optic parameters, etc. are stored here

To this end, the subslot numbers are defined for the respective DAP according to the entry in the GSD file. These numbers start with 0x8000 for the interface subslot. The distribution is as follows:

Subslot number 0x8i00 → i = number of interface module (0 ..0x0F) and

Subslot number 0x8ipp → pp = number of the respective port (1 ... 0x0FF)

Addressing for the PDev is defined in the GSD file of the IO device. Below, an example is provided of what such a definition in the GSD file can look like.

Example

The PDev definition is located within the SystemDefinedSubmoduleList and belongs to the respective DAP. To address the PDev, Subslot 0x8000 (32768 dec) was selected for the example below. Here, the following is also located: the supported Real Time classes, the IRT definition and definitions for Fast Startup. These definitions will be discussed in greater detail in the corresponding chapters below. For the sake of clarity, they are included in the example, however.

```
<SystemDefinedSubmoduleList>
    <InterfaceSubmoduleItem SubslotNumber="32768" SubmoduleIdentNumber="0x0002"
      SupportedRT_Classes="RT_CLASS_1;RT_CLASS_2;RT_CLASS_3"
      TextId="TOK_DAP_InterfaceModule" IsochroneModeSupported="true"
      SupportedProtocols="SNMP;LLDP" SupportedMibs="MIB2"
      NetworkComponentDiagnosisSupported="true" DCP_HelloSupported="true">
      <RT_Class3Properties MaxBridgeDelay="2920" MaxNumberIR_FrameData="400" />
      <SynchronisationMode SupportedRole="SyncSlave" MaxLocalJitter="50"
       T_PLL_MAX="1000" SupportedSyncProtocols="PTCP" />
      <ApplicationRelations NumberOfAdditionalInputCR="0"
        NumberOfAdditionalMulticastProviderCR="0" NumberOfAdditionalOutputCR="0"
        NumberOfMulticastConsumerCR="0" PullModuleAlarmSupported="true">
        <TimingProperties SendClock="16 32 64 128" ReductionRatio="1 2 4 8 16 32 64 128
                                                                   256 512" />
      </ApplicationRelations>
    </InterfaceSubmoduleItem>
   <PortSubmoduleItem SubslotNumber="32769" SubmoduleIdentNumber="0x0003"
    MAUType="100BASETXFD" TextId="TOK_Port1" MaxPortRxDelay="302"
    MaxPortTxDelay="108" PortDeactivationSupported="true"
    LinkStateDiagnosisCapability="Up+Down" />
       <PortSubmoduleItem SubslotNumber="32770" SubmoduleIdentNumber="0x0003"
    MAUType="100BASETXFD" TextId="TOK_Port2" MaxPortRxDelay="302"
    MaxPortTxDelay="108" PortDeactivationSupported="true"
    LinkStateDiagnosisCapability="Up+Down" />
</SystemDefinedSubmoduleList>
```


9 Definition of Real Time Classes

PROFINET provides a cascadable Real Time concept. In order to adjust communication optimally to the needs of the automation plant, it is necessary to specify in the GSD file which real time classes the respective field device supports. These definitions are DAP-specific and therefore have to be specified in the PDev (InterfaceSubmodule).

In the GSD file, the definition of the real time classes could look like this:

```
<InterfaceSubmoduleItem SubslotNumber="32768" SubmoduleIdentNumber="0x0000002"
      SupportedRT_Classes="RT_CLASS_1;RT_CLASS_2;RT_CLASS_3"
      TextId="TOK_DAP_InterfaceModule" IsochroneModeSupported="true"
      SupportedProtocols="SNMP;LLDP" SupportedMibs="MIB2"
</InterfaceSubmoduleItem>
```

9.1 Isochronous Real Time

Isochronous or synchronous Real Time is part of synchronized communication (RT_Class_2 and RT_Class_3, synchronized). To this end, the corresponding definitions for the respective DAP have to be provided within the PDev. PROFINET uses the PTCP protocol for synchronization. Since a synchronization frame is sent over the entire bus to all stations, the time lag also has to be specified when a frame passes through a switchport in send and receive direction.

Example

```
<SystemDefinedSubmoduleList>
   <InterfaceSubmoduleItem SubslotNumber="32768" SubmoduleIdentNumber="0x0002"
    SupportedRT_Classes="RT_CLASS_1;RT_CLASS_2;RT_CLASS_3"
    TextId="TOK_DAP_InterfaceModule" IsochroneModeSupported="true"
    SupportedProtocols="SNMP;LLDP" SupportedMibs="MIB2"
    NetworkComponentDiagnosisSupported="true" DCP_HelloSupported="true">
    <RT_Class3Properties MaxBridgeDelay="2920" MaxNumberIR_FrameData="400" />
    <SynchronisationMode SupportedRole="SyncSlave" MaxLocalJitter="50"
     T_PLL_MAX="1000" SupportedSyncProtocols="PTCP" />

    <ApplicationRelations NumberOfAdditionalInputCR="0"
      NumberOfAdditionalMulticastProviderCR="0" NumberOfAdditionalOutputCR="0"
      NumberOfMulticastConsumerCR="0" PullModuleAlarmSupported="true">
      <TimingProperties SendClock="16 32 64 128" ReductionRatio="1 2 4 8 16 32 64 128
                                                                 256 512" />
    </ApplicationRelations>
  </InterfaceSubmoduleItem>
  <PortSubmoduleItem SubslotNumber="32769" SubmoduleIdentNumber="0x0003"
   MAUType="100BASETXFD" TextId="TOK_Port1" MaxPortRxDelay="302"
   MaxPortTxDelay="108" <! – delay in switch in send and receive direction -->
   PortDeactivationSupported="true"
   LinkStateDiagnosisCapability="Up+Down" />

  <PortSubmoduleItem SubslotNumber="32770" SubmoduleIdentNumber="0x0003"
   MAUType="100BASETXFD" TextId="TOK_Port2" MaxPortRxDelay="302"
   MaxPortTxDelay="108" PortDeactivationSupported="true"
   LinkStateDiagnosisCapability="Up+Down" />
</SystemDefinedSubmoduleList
```


10 Modules, Submodules, Slots and Subslots

It is possible to define different modules and submodules in a GSD file. They are defined only once within a GSD file and if needed, can be assigned to different DAPs. The modules/submodules define the physical configuration of an electronic base module. Each is clearly referenced with an eight digit ident number. For that reason, the module and submodule numbers have to be unique (assigned only once). For actual operation, the modules/submodules are assigned to the physical slots of a module system. It is also possible to assign a module/submodule to several slots. The I/O data is always addressed by means of a slot-subslot combination.

The combination consisting of DeviceID, ModuleIdentnumber and SubmoduleIdentnumber has to be unique within the manufacturer.

Since it has to be possible to identify a hardware clearly and unmistakably from the view of the engineering tool as well as the process-oriented field device, the following rules are applied to using Ident numbers:

- Modules are identified through unique non-interchangeable ident numbers within a field device.
- Submodules are identified with unique submodule ident numbers within a module.

In addition, the following has to be noted:

If a module from the same or another device family can be used in different field devices, the ModuleIdentNumber has to be uniform over these field devices.

If a submodule can be used in different modules, the SubmoduleIdentNumber has to be uniform over these modules.

The following applies to such a SubmoduleIdentnumber:

These submodules don't only identify the physical properties, but also the representation of the cyclical I/O data. If such submodules permit several representations of the data length, different SubmoduleIdentnumbers have to be used for different representations (length and data format). Only in this way is it possible to clearly identify physically identical submodules within a module, through different representation of the data.

The VendorID, DeviceID and the ModuleIdentnumber represent a 1:1 relationship (that means, there is never more than one IdentNumber for the same hardware). A submodule on the other hand can also have several different SubmoduleIdentnumbers for different representations of the I/O data.

To deviate from these rules is allowed only if

- Existing modular field devices become PROFINET IO devices, since for this transition only the head module has to e exchanged.
- Gateways (links) to other networks are used since normally, the field devices of the secondary network are mapped to the PROFINET IO address space.

Submodule definitions can occur at different locations within a GSD file. Some of these locations are listed below:

- Interface and PortSubmodules within a DAPs in the *SystemDefinedSubmoduleList*
- PortSubmodules as so-called "normal" modules in the *SystemDefinedSubmoduleList or SubmoduleList (plugable PortSubmodulse)*
- "Normal" modules within the *VirtualSubmoduleList*
- "Normal" submodules within the global *SubmoduleList*

All these *IdentificationNumbers* can be read out as follows:

- RPCObject UUID (Vendor and DeviceID)
- I&M functions (Vendor and DeviceID)
- Acyclical reading with RealIdentificationData (ModuleIdentNumber and SubmoduleIdentNumber)

Example:

In the example below, three modules are defined that can be read out <<? part of sentence missing in the original>> based on their ID (here ID_Mod_01 for instance). They are defined in two steps. Within the "UseableModules", all modules are identified with their unique IDs.

1. Definition of the Modules:

```
<UseableModules>
   <ModuleItemRef ModuleItemTarget="ID_Mod_01" AllowedInSlots="1..16" />
   <ModuleItemRef ModuleItemTarget="ID_Mod_02" AllowedInSlots="1..16" />
   <ModuleItemRef ModuleItemTarget="ID_Mod_03" AllowedInSlots="1..16" />
</UseableModules>
```

2. Definition of the properties of the modules within the ModuleList with reference to the Module_ID. The actual data length is specified within the VirtualSubmodule definition.

```
<ModuleList>
   <ModuleItem ID="ID_Mod_01" ModuleIdentNumber="0x00000020">
   <ModuleInfo>
      <Name TextId="TOK_TextId_Module_1IO" />
      <InfoText TextId="TOK_InfoTextId_Module_1IO" />
      <HardwareRelease Value="1.0" />
      <SoftwareRelease Value="1.0" />
   </ModuleInfo>
   <VirtualSubmoduleList>
      <VirtualSubmoduleItem ID="1" SubmoduleIdentNumber="0x00000001" API="0">
         <IOData IOPS_Length="1" IOCS_Length="1">
            <Input Consistency="All items consistency">
               <DataItem DataType="OctetString" TextId="TOK_Input_DataItem_1"
                Length="1" UseAsBits="true" />
            </Input>
            <Output Consistency="All items consistency">
                <DataItem DataType="OctetString" TextId="TOK_Output_DataItem_1"
                 Length="1" UseAsBits="true" />
            </Output>
         </IOData>
         <ModuleInfo>
            <Name TextId="TOK_TextId_Module_1IO" />
            <InfoText TextId="TOK_InfoTextId_Module_1IO" />
         </ModuleInfo>
      </VirtualSubmoduleItem>
    </VirtualSubmoduleList>
   </ModuleItem>
```

Here, additional modules follow.

10.1 Definition of Modules with IRT Capability

According to the previous example it is assumed that the module with the ID_03 is to be IRT capable. The definition of the IRT capable module has the following structure.

Example:

```
<ModuleItem ID="ID_Mod_03" ModuleIdentNumber="0x00000003">
   <ModuleInfo>
      <Name TextId="TOK_TextId_Module_1I_Isochron" />
      <InfoText  TextId="TOK_InfoTextId_Module_1I_Isochron" />
      <HardwareRelease Value="1.0" />
      <SoftwareRelease Value="1.0" />
   </ModuleInfo>
   <VirtualSubmoduleList>
      <VirtualSubmoduleItem ID="03" SubmoduleIdentNumber="0x00000001" API="0">
         <IOData IOPS_Length="1" IOCS_Length="1">'
            <Input Consistency="All items consistency">
               <DataItem DataType="OctetString"
                TextId="TOK_Inp_DataItem_1_Isochron" Length="1" UseAsBits="true" />
            </Input>
         </IOData>
         <ModuleInfo>
            <Name TextId="TOK_TextId_Module_1I_Isochron" />
            <InfoText TextId="TOK_InfoTextId_Module_1I_Isochron" />
         </ModuleInfo>
         <IsochroneMode IsochroneModeRequired="true" T_DC_Base="8" T_DC_Min="8"
          T_DC_Max="128" T_IO_Base="125000" T_IO_InputMin="1" T_IO_OutputMin="1" />
      </VirtualSubmoduleItem>
   </VirtualSubmoduleList>
</ModuleItem>
```


11 Diagnostic Definitions

In PROFINET IO, alarms are used to signal, in the form of diagnoses, critical states from the process or the field device to the control system. These can be defined

- Manufacturer specific
- Profile specific or
- System specific

In order to be displayed in the engineering tool in a readable form, they have to be defined in a certain structure in the GSD file. The manufacturer-specific and profile specific alarms have to be described in the GSD file regarding the value range as well as the texts <<?>> . The system-specific alarms can be assumed as known in the ES tool regarding the structure as well as the texts.

The table below shows which alarm types of PROFINET can be described in the GSD file for PROFINET:

UserStructureIdentifier	ChannelErrorType	ExtChannelErrorType	GSDML Element
0x0000-0x7FFF ManufacturerSpecific	–	–	UnitDiagTypeList/UnitDiagTypeItem, enumerated by UserStructureIdentifier
0x8000 ChannelDiagnosis	0x0000-0x000E Reserved or system defined	–	–
	0x000F-0x001F Manufacturer specific, but with recommendation	–	ChannelDiagList/ChannelDiagItem (no ExtChannelDiagList), enumerated by (Channel)ErrorType
	0x0020-0x00FF Reserved for common profiles	–	ChannelDiagList/ChannelDiagItem (no ExtChannelDiagList), enumerated by (Channel)ErrorType
	0x0100-0x7FFF Manufacturer specific	–	ChannelDiagList/ChannelDiagItem (no ExtChannelDiagList), enumerated by (Channel)ErrorType
	0x8000-0x8FFF Reserved or system defined	–	–
	0x9000-0x9FFF Reserved for profiles	–	ChannelDiagList/ProfileChannelDiagItem (no ExtChannelDiagList), enumerated by API and (Channel)ErrorType
0x8001 Multiple	Will be split into single channel diagnosis alarms and decoded accordingly.		
0x8002 ExtChannelDiagnosis	0x0000-0x000E Reserved or system defined	–	–
	0x000F-0x001F Manufacturer specific, but with recommendation	0x0001-0x7FFF Manufacturer specific	ChannelDiagList/ChannelDiagItem, enumerated by (Channel)ErrorType, with ExtChannelDiagList/ExtChannelDiagItem, enumerated by (ExtChannel)ErrorType
	0x0020-0x00FF Reserved for common profiles	0x0001-0x7FFF Manufacturer specific	ChannelDiagList/ChannelDiagItem, enumerated by (Channel)ErrorType, with ExtChannelDiagList/ExtChannelDiagItem, enumerated by (ExtChannel)ErrorType
	0x0100-0x7FFF Manufacturer specific	0x0001-0x7FFF Manufacturer specific	ChannelDiagList/ChannelDiagItem, enumerated by (Channel)ErrorType, with ExtChannelDiagList/ExtChannelDiagItem, enumerated by (ExtChannel)ErrorType
	0x8000-0x8FFF Reserved or system defined	–	–
	0x9000-0x9FFF Reserved for profiles	0x9000-0x9FFF Reserved for profiles	ChannelDiagList/ProfileChannelDiagItem, enumerated by (Channel)ErrorType, with ExtChannelDiagList/ProfileExtChannelDiagItem, enumerated by API and (ExtChannel)ErrorType
0x8003	The structure of the QualifiedChannelDiagnosis is that of the ExtChannelDiagnosis except for an additional		

UserStructureIdentifier	ChannelErrorType	ExtChannelErrorType	GSDML Element
QualifiedChannelDiagnosis	system defined QualifiedChannelQualifier. So the QualifiedChannelDiagnosis is described with the same GSDML elements as the ExtChannelDiagnosis.		
0x8004-0x8FFF Reserved or system defined	–	–	–
0x9000-0x9FFF Reserved for profiles	–	–	UnitDiagTypeList/ProfileUnitDiagTypeItem, enumerated by API and UserStructureIdentifier
0xA000-0xFFFF Reserved	–	–	–

Table 1: User-Specific Diagnoses

Example:
Within the ***ChannelDiagList***, the manufacturer specifies -within the keyword ***ChannelDiagItem*** under the definition ***ErrorType***- the error number that is described in more detail with the keyword ***TextId***. The definition starts on the same level as the DeviceAccessPointList. It is advisable to first define -either within the company or at least for a module system- all possible error numbers uniformly, since the user software in the PROFINET IO field device can then use uniform error numbers.

```
<ChannelDiagList>
   <ChannelDiagItem ErrorType="16">
      <Name TextId="TOK_Name_ErrorType16" />
      <Help TextId="TOK_HelpName_ErrorType16" />
   </ChannelDiagItem>
   <ChannelDiagItem ErrorType="17">
      <Name TextId="TOK_Name_ErrorType17" />
      <Help TextId="TOK_HelpName_ErrorType17" />
   </ChannelDiagItem>
   <ChannelDiagItem ErrorType="18">
      <Name TextId="TOK_Name_ErrorType18" />
      <Help TextId="TOK_HelpName_ErrorType18" />
   </ChannelDiagItem>
</ChannelDiagList>
```

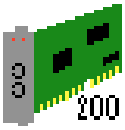
This parameter assignment error then has to be defined also in the language list for all other languages used.

```
<ExternalTextList>
   <PrimaryLanguage>
      <Text TextId="TOK_Name_ErrorType16" Value="parameter assignment error" />
      <Text TextId="TOK_Name_ErrorType17" Value="Power supply fault" />
      <Text TextId="TOK_Name_ErrorType18" Value="fuse blown" />
   </PrimaryLanguage>
</ExternalTextList>
```


12 Graphic Symbols

When configuring PROFINET systems, the field devices are represented using graphic symbols. The definition of such graphic symbols could look like this (from the development kit for ERTEC 200).

```
<Graphics>
   <GraphicItemRef Type="DeviceSymbol" GraphicItemTarget="ID_Graph_2" />
</Graphics>
```



13 Texts in GSDML

13.1 General

Most texts in the GSDML can be defined language-dependent. To this end, only “Ids“ are used in the description part (designated as attribute with the name "TextId") that refer to the actual texts in the "ExternalTextList". There are no length limitations for the texts themselves since as a rule it is possible to respond in the engineering tools to texts of any length with scroll bars, for example. Nevertheless, it is pointed out that if texts are “overly long“, it can happen that the entire text is not displayed.

For that reason, it is recommended to stay within the following length definitions:

Use	Element Name	Recommended Maximum Length in Octets
Category names	CategoryItem	30
Parameter names	ValueItem	30
Parameter texts	Âssign	
Designating cyclical IO data	DataItem	30
Help information	Help	1024
Manufacturer specific diagnosis: Designation	ChannelDiagItem/Name	30
Manufacturer specific diagnosis: Help text	ChannelDiagItem/Text	1024
Name of a parameter assignment data set	ParameterRecordDataItem	30
Name of a module	ModuleInfo/Name	30
General information about a module	ModuleInfo/InfoText	250
General information about an IO device	DeviceIdentity/InfoText	250

Table 2: Recommended Text Lengths

13.2 Incorporating Foreign Languages

All language-dependent texts are stored in the section "ExternalTextList" of a GSD file. The ExternalTextList includes at least the section "PrimaryLanguage" that has to contain all texts in **English**(en)

In addition, any number of sections "Language" is possible where the texts of another language are stored respectively. The language itself is defined with the attribute "xml:lang" and follows the rules of "2 Letter Codings" of the ISO639-1 Codes for the representation of names of languages – Part 1: Alpha-2 code..

The encodings for the most common languages are as follows:

Abbreviation	Language
de	Deutsch
es	Spanisch
fr	Französisch
it	Italienisch

Table 3: Language Encodings in the GSDML

The sections "PrimaryLanguage" and "Language" itself have identical structures. They include any number of "Text" entries. The attribute "TextId" is used for identification and the attribute "Value" contains the actual text.

An "ExternalTextList" with two languages can look like this:

```
<ExternalTextList>
    <PrimaryLanguage>
        <Text TextId="IDT_REF_WIREBREAK" Value="Wire break"/>
    </PrimaryLanguage>
    <Language xml:lang="de">
        <Text TextId="IDT_REF_WIREBREAK" Value="Drahtbruch"/>
    </Language>
</ExternalTextList>
```

In addition to this method, it is also possible to move texts to a so-called "satellite files". This allows for adding a language without the (possibly already certified) GSD file having to be changed.

A satellite file is structured as follows (since GSDML Version V 2.2):

```
<?xml version="1.0" encoding="UTF-8"?>
<ExternalTextDocument xml:lang="fr" xmlns="http://www.profibus.com/Common/2003/11/Primitives"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.profibus.com/Common/2003/11/Primitives ..\..\XSD\Common-Primitives-v2.2.
xsd">
    <Text TextId=" IDT_REF_WIREBREAK" Value="rupture de fil"/>

</ExternalTextDocument>
```

For GSDML older than V2.2 the namespace http://www.profibus.com/Common/2003/11/Primitives and the Scheme Common-Primitives-v1.0.xsd shall be used.

So that the satellite file can be located in the engineering tool it has to correspond to certain name conventions. The rule is that the same name has to be used as in the corresponding GSD file, with the addition "-Text-<2Letter Code>".

Example:

GSD file name is: GSDML-V1.0-Siemens-ET200X-20030818.xml

Satellite file in Italian has to be:
GSDML-V1.0-Siemens-ET200X-20030818-Text-it.xml

14 Using References

In a GSD file, numerous references are used in order to avoid describing the same data multiple times. Example: Two modules support the same parameter "Measuring Type". In addition to the textual explanations, numerous parameters are stored in the GSD for this parameter (such as encoding). By using references, it is now possible to describe the parameter "Measuring Type" at one location in the GSD and to reference it from all modules that are using this parameter. Otherwise, all texts would have to be specified separately in all languages for each module. This would, on the one hand, increase the effort to update the file and on the other hand increase the size of the GSD file considerably.

14.1 Checks Using the Schematic

IDs are used to reference elements in the GSD: in the example below, the ID of a module is shown:

```
<ModuleItem ID="IDM_1" ModuleIdentNumber="0x12345678">
```

To reference this module, it is sufficient to specify the ID of this module:

```
<ModuleItemRef ModuleItemTarget="IDM_1" AllowedInSlots="1.. 63"/>
```

Referencing only works, of course, if the IDs are kept unique. This is checked and ensured by means of schematic validation. The example below shows the error indication of XML Spy:

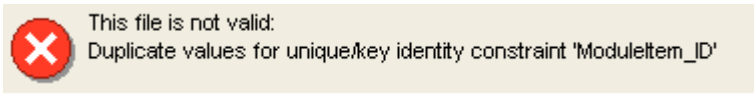


Figure 5: Error Message for Double IDs

In addition, a check is performed whether a valid ID was specified for references:

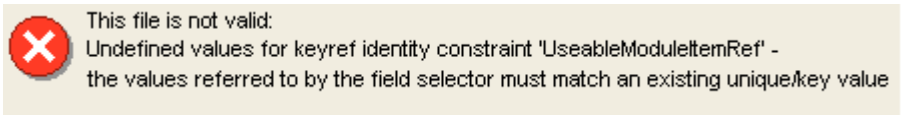


Figure 6: Error Message if Reference is Invalid

14.2 Tips to Name IDs Meaningfully

A GSD file can become quite large. Correspondingly many IDs have to be assigned and kept consistent. When the IDs are assigned, the GSDML schematic itself only checks whether the string has leading or ending blanks. These are detected through schematic validation and rejected. Likewise, line breaks are not permitted in a string.

Otherwise, any string can be used for IDs.

To avoid confusion, it is recommended starting the IDs with a type-specific prefix:

- The string indicates which object the ID references
- Unique ID assignment is facilitated
- In an editor, the IDs can be displayed sorted in a list

The table below provides recommendations for assigning ID prefixes:

Object	Prefix
DeviceAccessPointItem	IDD_
Module	IDM_
Submodule	IDS_
ValueItem	IDV_
GraphicItem	IDG_
Category	IDC_
InfoText	IDT_INFO_
ChannelDiagItem/Name	IDT_DIAG_NAME_
RecordDataItem/Name	IDT_RECORD_NAME_
ModuleInfo/Name	IDT_MODULE_NAME_
Category	IDT_CATEGORY_
Help	IDT_HELP_
Assign	IDT_ASSIGN_
Ref	IDT_REF_
DataItem	IDT_DATAITEM_

Table 4: Prefix Use of IDs

15 Catalog Information in the GSD

The previous (keyword based) GSD did not include information that allowed representing the module selection of modular field devices in a structured mode. This made selecting the desired modules very difficult for field devices with a large number of module variants, since the only criterion for distinguishing the module was the module's name.

In the GSDML specification, this problem was taken into account by adding categories. Each module can be assigned to a category and a subcategory. This assignment has **no** effect on the runtime properties of the module, but is used only to improve the representation for module selection; for example, in the form of a catalog in the engineering system.

Bu using a category, technically related modules can be grouped. For example, all analog input modules can be assigned to the category "Analog Input".

15.1 Category Assignment

Each module (including the DAP) has in the GSD a mandatory element "ModuleInfo". In addition to other sub-elements, it includes two optional attributes "CategoryRef" and "Subcategory1Ref". With these attributes, the assignment to a main category or a sub-category is made. The contents of the attributes have to include an ID of the element "CategoryItem" of the global "CategoryList".

The structure of a CategoryItem itself is very simple – in addition to the ID for referencing, it only contains a TextId for a (language dependent) text.

The example below shows the relationships:

```
<ModuleList>
    <ModuleItem ID="IDT_M1" ModuleIdentNumber="0x12345678">
          <ModuleInfo CategoryRef=" IDC_ 1" SubCategory1Ref=" IDC_2">
          ……

<CategoryList>
    <CategoryItem ID=" IDC_ 1" TextId=" IDT_CATEGORY_1"/>
    <CategoryItem ID=" IDC_ 2" TextId=" IDT_CATEGORY_2"/>
</CategoryList>

<ExternalTextList>
    <PrimaryLanguage>
        <Text TextId=" IDT_CATEGORY_1" Value="Motorstarter"/>
        <Text TextId=" IDT_CATEGORY_2" Value="Direktstarter"/>
    </PrimaryLanguage>
</ExternalTextList>
```

15.2 Representing the Categories

It is left to the engineering tool manufacturer how the categories are represented in the engineering tool. But the use of the categories can be illustrated quite simply with the module catalog of the Siemens tool "HWConfig":

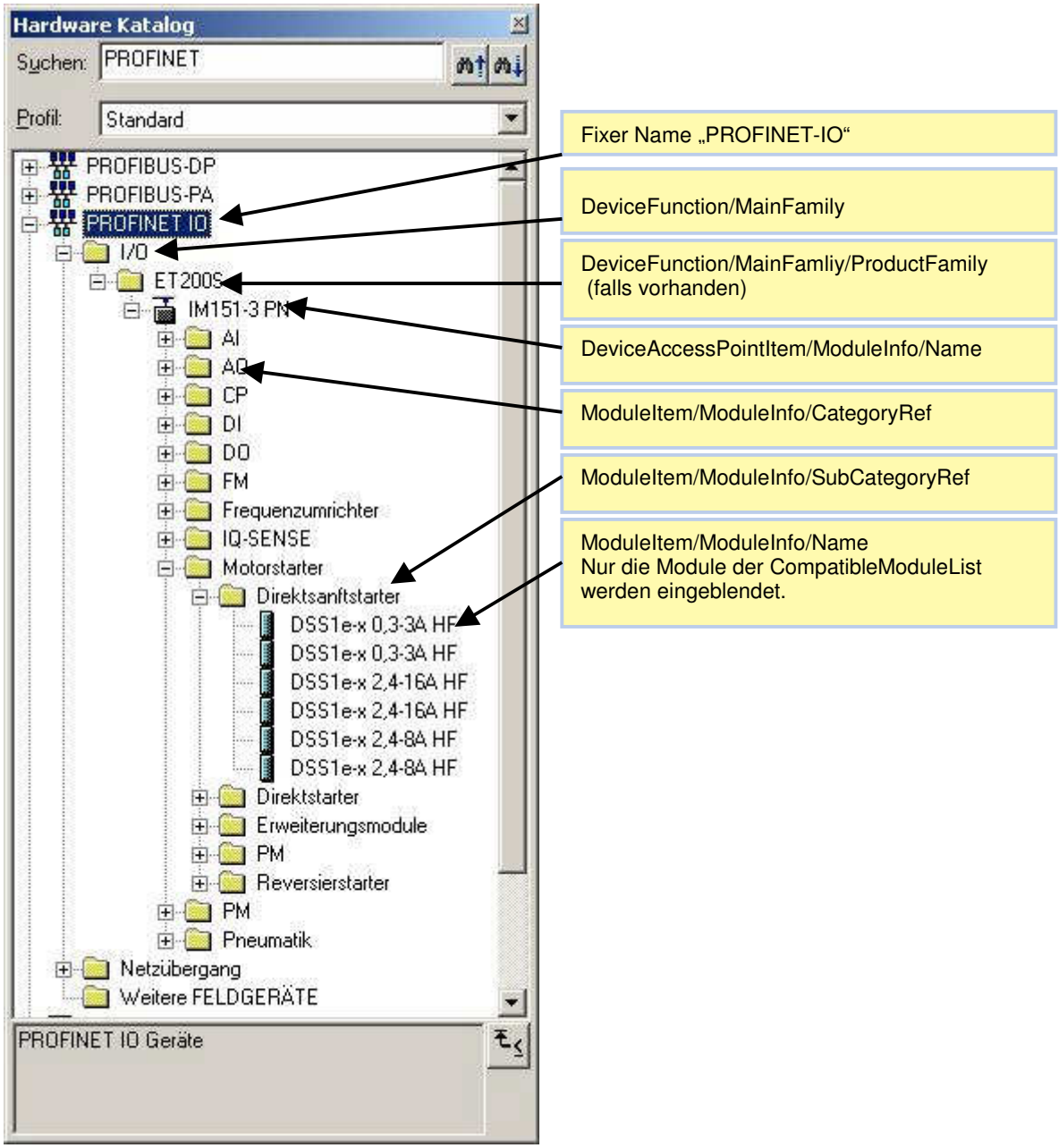


Figure 7: Catalog Representation Using STEP7 as an Example

<<**Suchen** = search; **Frequenzumrichter** = frequency converter; **Direktsanftstarter** = direct soft starter; **Erweiterungsmodule** = expansion modules; **Netzübergang** = network transition; **Weitere** .. = other field devices>>

16 Description of ParameterRecordDataObjects

The GSDML allows for describing parameters that the IO controller transmits to a submodule of the IO device at startup. For PROFINET IO, these parameters are stored in the form of RecordDataObjects (comparable to data records).

To describe this data, the element "ParameterRecordDataItem" is used in the GSDML. It can be located optionally below a submodule.

The individual parameters of a submodule can be distributed over several RecordDataObjects. Each RecordDataObject can (within a submodule) be clearly addressed by means of the attribute "Index".

With the attribute "TransferSequence" you can specify the time sequence in which the individual ParameterRecordDataObjects are transferred.

Other attributes of a ParameterRecordDataItem in the GSDML are

- Length of the RecordDataObject in octets (attribute "Length)
- Name of the RecordDataObject (child element "Name). It is used only for display purposes in the engineering tool.
- Initialization of the data structure (child element "Const")
- Definition of the individual parameters (child element "Ref)

16.1 "Const" Definition of a ParameterRecordDataItem

The data structure of a ParameterRecordDataItem can be initialized by using the "Const" element – that is, filled with default values.

In this case, a ParameterRecordDataItem can contain any number of Const definitions.

A Const element has two attributes:

<Const ByteOffset="1" Data="0x01,0x00"/>
The attribute "Data" contains the byte values -separated by a comma- of the RecordDataObjects in hex representation (starting with "0x").

The optional attribute "ByteOffset" describes, starting with which byte, the following bytes from the "Data" attribute are to be inserted. If this attribute is missing, ByteOffset "0" is assumed.

In general, the rule applies that the ByteOffset + the number of described bytes of the data element must not exceed the length of the RecordDataObject.

The example below shows the initialization of a 4 byte long RecordDataObject by means of the Const definition:

```
<ParameterRecordDataItem Index="1" Length="4">
    <Name TextId=" IDT_REF_GeneralParameter"/>
    <Const  Data="0xFF,0x00"/>
    <Const  ByteOffset="3" Data="0xAA"/>
</ParameterRecordDataItem>
```


Bit → Byte	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	1	0	1	0	1	0	1	0

Fields described by <<?>> Const definition

Figure 8: Effect of the Const Definition in a Record Data Object

In this example, the value of the byte with Offset "2" is not specified by the Const definition. In this case, the GSDML specification specifies that these values are filled with "0".

16.2 "Ref" Definition of a ParameterRecordDataItem

The individual parameters of a ParameterRecordDataItem are described by using the "Ref Element". This element has many attributes that specify the properties of the parameter.

The example below illustrates the use of the Ref Element:

The RecordDataObject described above is to include a parameter "Measuring Type". This parameter, encoded by using two bits, is to be able to accept the following values:

- 0..20 mA (encoding 00)
- 4..20 mA(encoding 01)
- 0..10 V (encoding 10)
- +-10 V (encoding 11)

The object could look like this:

Bit → Byte	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	1
1	0	0	0	MT Bit 1	MT Bit 0	0	0	0
2	0	0	0	0	0	0	0	0
3	1	0	1	0	1	0	1	0

Figure 9: Effect of the Ref Definition in a Record Data Object

<<MT = measuring type>>

In the GSD, this parameter would be described as follows:

```
<ParameterRecordDataItem Index="1" Length="4">
    <Name TextId=" IDT_REF_GeneralParameter"/>
    <Const  Data="0xFF,0x00"/>
    <Const  ByteOffset="3" Data="0xAA"/>
    <Ref ValueItemTarget=" IDV_ Messbereich" ByteOffset="1" BitOffset="3" DataType="BitArea"
BitLength="2" TextId="IDT_REF_ Messbereich"/>    <<measuring range>>

</ParameterRecordDataItem>


<ValueList>
    <ValueItem ID=" IDV_Messbereich">                                    <<measuring range>>>
        <Assignments>
            <Assign TextId="IDT_ASSIGN_0-20mA" Content="0"/>
            <Assign TextId="IDT_ASSIGN_4-20mA" Content="1"/>
            <Assign TextId="IDT_ASSIGN_0-10V" Content="2"/>
            <Assign TextId="IDT_ASSIGN_+-10V" Content="3"/>
        </Assignments>
    </ValueItem>
</ValueList>

<ExternalTextList>
    <PrimaryLanguage>
        <Text TextId="IDT_REF_GeneralParameter" Value="Allgemeine Parameter"/> <<general parameters>>
        <Text TextId="IDT_REF_Messbereich" Value="Messbereich"/>                <<meas. range>>
        <Text TextId="IDT_ASSIGN_0-20mA " Value="Strom 0..20 mA"/>              <<current>>
        <Text TextId="IDT_ASSIGN_4-20mA " Value="Strom 4..20 mA"/>
        <Text TextId="IDT_ASSIGN_0-10V " Value="Spannung 0..10 Volt"/>          <<voltage>>
        <Text TextId="IDT_ASSIGN_+-10V " Value="Spannung + - 10 Volt"/>
    </PrimaryLanguage>
</ExternalTextList>
```

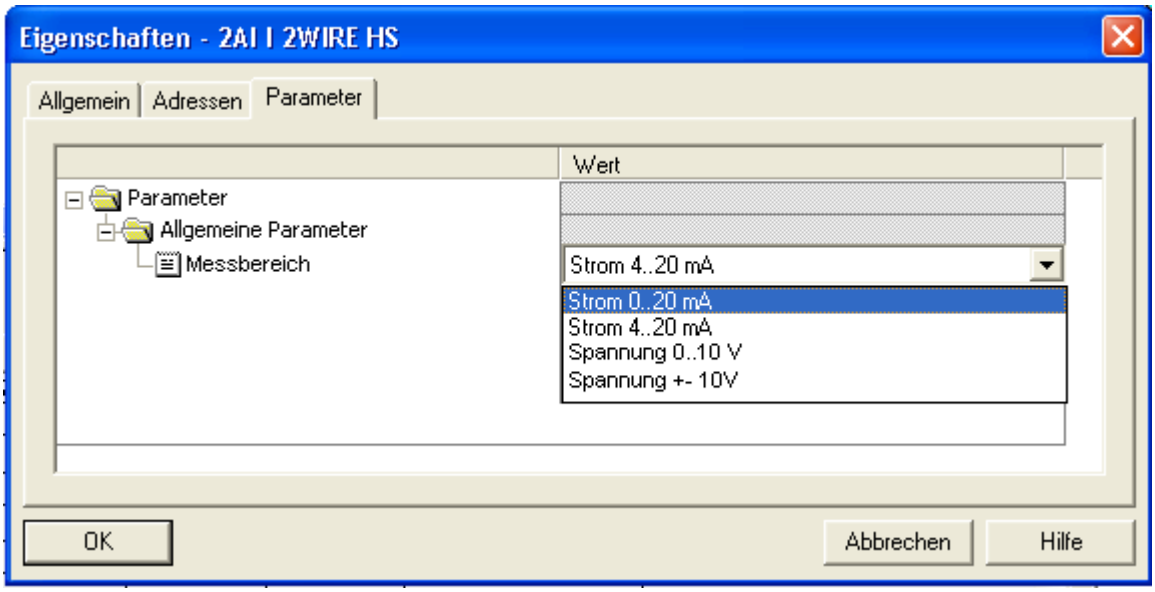
The following special considerations are to be noted:

- By means of "ValueItemTarget", each concrete value can be assigned a text. In the engineering tool, this text is offered to the user as selection criterion. If there is no "ValueItemTarget", the tool will provide this parameter as a pure editing field; that is, in the above case, only the values "0" to "3“ could be entered.

- By using the attribute "AllowedValues" of the Ref Element, the value range can be limited. A similar module that is only able to measure current could limit this with the attribute "AllowedValues="0 1".

- If neither "ValueItemTarget" nor "AllowedValues" is available, the value range that can be entered is limited only by the data type.

The dialog below shows a possible representation of the parameter in the engineering tool:



<<**Eigenschaften** = attributes; **Allgemein** = general; **Messbereich** = measuring range; **Strom** = current; **Spannung** = voltage; **Abbreche**n = Cancel; **Hilfe** = Help>>

Figure 10: Representation of a Parameter in the Engineering Tool

17 SNMP and MIB2

SNMP means Simple Network Management Protocol. If a field device (the DAP) supports the SNMP protocol, an engineering tool can, for example, read out the MIB2 and LLDP library by means of the SNMP protocol. This information is very useful when replacing field devices or to display the plant topology. Below, an example shows what a definition can look like:

```
<InterfaceSubmoduleItem SubslotNumber="32768" SubmoduleIdentNumber="0x0000002"
   SupportedRT_Classes="RT_CLASS_1;RT_CLASS_2;RT_CLASS_3"
   TextId="TOK_DAP_InterfaceModule" IsochroneModeSupported="true"
    SupportedProtocols="SNMP;LLDP" SupportedMibs="MIB2"
    NetworkComponentDiagnosisSupported="true"      . . .
```


18 Fast Startup

Usually, Ethernet stations need, after Power On, several seconds to access data exchange. However, in some areas of automation technology, a much faster power-up of a field device is desired. This variant of power-up is called Fast Start Up (FSU). In this case, the field device becomes active itself after voltage return and signals with the DCP service DCP.Hello. This definition is DAP specific and is therefore located within the interface submodule. To define a FSU requires three steps:

1. PowerOnToCommReady within the DAP

2. The PDev definition (usually SubslotItem SubslotNumber="32768") and

3. The definition that DCP.Hello is supported: DCP_HelloSupported="true"

The definition of the optimum FSU service could look like this, for example:

1. PowerOnToCommReady within the DAP

```
<DeviceAccessPointItem ID="DAP 2" PhysicalSlots="0..16"
    ModuleIdentNumber="0x00000002" MinDeviceInterval="16"
    ImplementationType="ERTEC400" DNS_CompatibleName="ERTEC-DEVKit"
    ExtendedAddressAssignmentSupported="true" FixedInSlots="0"
    ObjectUUID_LocalIndex="1"
    RequiredSchemaVersion="V2.2" MaxSupportedRecordSize="4068"
    ParameterizationSpeedupSupported="true"
    PowerOnToCommReady="700">   <!—time in ms until the first data exchange  ->
```

2. The PDev definition (usually SubslotItem SubslotNumber="32768") and

The associated PDev

```
<SubslotList>
   <SubslotItem SubslotNumber="32768" TextId="TOK_Subslot_8000" />
    <SubslotItem SubslotNumber="32769" TextId="TOK_Subslot_8001" />
    <SubslotItem SubslotNumber="32770" TextId="TOK_Subslot_8002" />
    <SubslotItem SubslotNumber="32771" TextId="TOK_Subslot_8003" />
    <SubslotItem SubslotNumber="32772" TextId="TOK_Subslot_8004" /
  </SubslotList>
```

3. The definition that DCP.Hello is supported DCP_HelloSupported="true"

The following has to be specified in the corresponding subslot of the PDev:

```
<InterfaceSubmoduleItem SubslotNumber="32768" SubmoduleIdentNumber="0x0000002"
    SupportedRT_Class="Class2" SupportedRT_Classes="RT_CLASS_1;RT_CLASS_2"
    TextId="TOK_DAP_InterfaceModule" IsochroneModeSupported="true"
     SupportedProtocols="SNMP;LLDP" SupportedMibs="MIB2"
     NetworkComponentDiagnosisSupported="true" DCP_HelloSupported="true">

. . .
```


19 Media Redundancy

The Media-Redundancy protocol consists of a **M**edia **R**edundancy **M**anager (MRM) and a **M**edia **R**edundancy **C**lient (MRC)
Usually the IO-Controller takes over the functionality of a “MRM“. This is why the following example just takes care of a MRC definition within the InterfaceSubmodule.

```
<InterfaceSubmoduleItem ID="IDS_3I" SubslotNumber="32768"
  SubmoduleIdentNumber="0x0002" SupportedRT_Class="Class3"
  SupportedRT_Classes="RT_CLASS_1;RT_CLASS_2;RT_CLASS_3"
  TextId="TOK_DAP_InterfaceModule" IsochroneModeSupported="true"
  SupportedProtocols="SNMP;LLDP" SupportedMibs="MIB2"
  NetworkComponentDiagnosisSupported="true" DCP_HelloSupported="true">
  <RT_Class3Properties MaxBridgeDelay="2920" MaxNumberIR_FrameData="128" />
  <SynchronisationMode SupportedRole="SyncSlave" MaxLocalJitter="50" T_PLL_MAX="1000"
  SupportedSyncProtocols="PTCP" />
  <ApplicationRelations NumberOfAR="2" NumberOfAdditionalInputCR="0"
  NumberOfAdditionalMulticastProviderCR="0" NumberOfAdditionalOutputCR="0"
  NumberOfMulticastConsumerCR="0" PullModuleAlarmSupported="true">
    <TimingProperties SendClock="8 16 32 64 128" ReductionRatio="1 2 4 8 16 32 64 128
    256 512" />
   <RT_Class3TimingProperties SendClock="8 16 32 64 128" ReductionRatio="1 2 4 8 16" />
  </ApplicationRelations>
  <MediaRedundancy SupportedRole="Client" /> <! Definition of MRP-Client>

</InterfaceSubmoduleItem>
```


20 PROFIsafe Definitions in the GSD File

I/O modules that support PROFIsafe functionality have to be defined correspondingly in the GSD file within the ´*VirtualSubmoduleList*´. Below, examples are provided that show what such definitions can look like.

```
<VirtualSubmoduleList>
  <VirtualSubmoduleItem ID="IDS_2"
   SubmoduleIdentNumber="0x00000010"
   PROFIsafeSupported="true">
   <IOData F_IO_StructureDescCRC="42144">
    <Input Consistency="All items consistency">
     <DataItem DataType="Integer16" TextId="IDT_DATAITEM_Int" />
     <DataItem DataType="Float32" TextId="IDT_DATAITEM_Real" />
     <DataItem DataType="F_MessageTrailer4Byte"
      TextId="IDT_DATAITEM_PS_Safety" />
    </Input>
    <Output Consistency="All items consistency">
     <DataItem DataType="F_MessageTrailer4Byte"
      TextId="IDT_DATAITEM_PS_Safety" />
    </Output>
   </IOData>
   <RecordDataList>
    <F_ParameterRecordDataItem Index="1" F_ParamDescCRC="46722">
     <F_Check_iPar DefaultValue="Check" Changeable="false" Visible="false" />
     <F_SIL DefaultValue="SIL2" AllowedValues="SIL1 SIL2" Changeable="true"
      Visible="true" />
     <F_CRC_Length DefaultValue="3-Byte-CRC" />
     <F_Block_ID DefaultValue="2" />
     <F_Par_Version />
     <F_Source_Add />
     <F_Dest_Add AllowedValues="1..65000" />
     <F_WD_Time DefaultValue="100" AllowedValues="1..2000" />
     <F_Par_CRC DefaultValue="3969" />
    </F_ParameterRecordDataItem>
   </RecordDataList>
   <ModuleInfo>
    <Name TextId="IDT_NAME_VS2" />
    <InfoText TextId="IDT_INFO_VS2" />
   </ModuleInfo>
  </VirtualSubmoduleItem>
 </VirtualSubmoduleList>
```


21 Compatibility between Different GSDML Versions

As in the case of PROFIBUS, further developments of the PROFINET technology usually also have an effect on the GSD. These are expressed with a more recent schematic version of the GSDML. When developing the GSDML, it is kept in mind that existing GSD files are also compatible with the more recent schematic version (upward compatibility).

The other direction, however ("more recent" GSD encounters "older" GSDML schematic) requires a more detailed examination:

As soon as structures are used in the GSD that have not yet been defined in the present schematic, a schematic validation will indicate errors. The consequence would be that either the latest engineering tools would always have to be used (that "know" a new schematic version) or that the field device manufacturer develops and updates several GSD versions. However, neither variant is desired.

An alternative would be to dispense with the schematic validation in the tools, and to ignore all unknown GSD structures in the GSD Interpreter. But this would have the disadvantage that the correct function of the field device can not be ensured since it may be necessary for the field device to also receive the data that is newly described in the GSD.

For these reasons, the attribute "RequiredSchemaVersion" was added to the GSDML Version 2.0. With it, the field device manufacturer can control which GSD structures are allowed to be overread by a tool without it causing errors.

The way it works is as follows: With this attribute it can be specified on the level of a single module or DAP which schematic version an engineering tool has to be able to handle at least in order for the module to be operated usefully. The example below shows the entry in a GSD of Schematic Version 2:

...

```
<ModuleItem ID="IDM_1" ModuleIdentNumber="0x00000000"
RequiredSchemaVersion="V1.0">
[..]

<VirtualSubmoduleList>
<VirtualSubmoduleItem ID="IDS_2" SubmoduleIdentNumber="0x00000010">
<IOData>
<Output>
<DataItem DataType="Unsigned8" UseAsBits="true" TextId="IDT_DATAITEM_Digital
Outputs">
  <BitDataItem BitOffset="0" TextId="IDT_DATAITEM_CHANNEL0" />
  <BitDataItem BitOffset="1" TextId="IDT_DATAITEM_CHANNEL1" />
  <BitDataItem BitOffset="2" TextId="IDT_DATAITEM_CHANNEL2" />
  <BitDataItem BitOffset="3" TextId="IDT_DATAITEM_CHANNEL3" />
</DataItem>
  </Output>
  </IOData>
</ModuleItem>
```

Here, it is specified for the module that it can be operated also with Version 1.0 of the GSDML. In the example, the expansion in comparison to Version 1.0 consists of the description of the bit structure of the output data (element BitDataItem). Since this description as no effect on the function of the module itself, these elements can be overread. An engineering tool that can only handle a GSDML Version 1.0 will read the following data, for example:

.....

```
<ModuleItem ID="IDM_1" ModuleIdentNumber="0x00000000">
  <VirtualSubmoduleList>
```



```
    <VirtualSubmoduleItem ID="IDS_2" SubmoduleIdentNumber="0x00000010">
  <IOData>
  <Output>
<DataItem DataType="Unsigned8" UseAsBits="true" TextId="IDT_DATAITEM_Digital
Outputs">
  </DataItem>
   </Output>
   </IOData>
</ModuleItem>
```

However, in other cases, it is necessary from the view of the field device to receive the parameters described in the GSD. An example of this could be a field device that can be used reasonably only in a synchronous environment. Since in the GSDML V1.0 no description resources are available for this, it doesn't make sense to configure this field device with a tool that is only able to interpret a Version 1.0 of the GSDML.

From the view of the engineering tool, modules or DAPs are not offered for configuration that, as "RequiredSchemaVersion", have a version that is not supported.

22 Tools

22.1 PROFINET XML Viewer

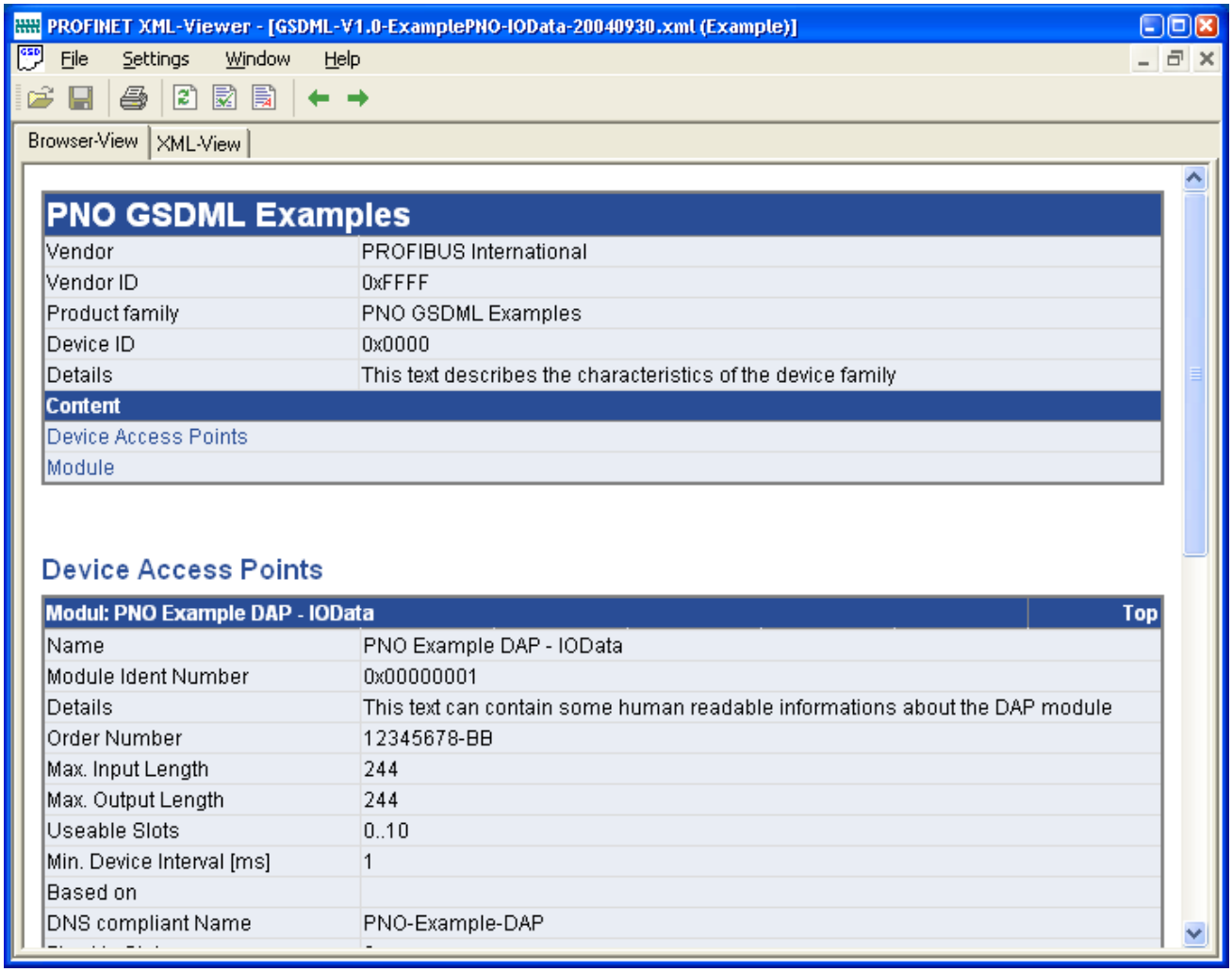
This tool is made available free of charge by the PNO and essentially offers two functions:

- Display of the contents of a GSD-file in clear HTML representation
- Validation of a GSD-file against the schematic files. In addition, GSD files can be checked with a “GSD Checker“ that performs additional checks beyond the rules stored in this schematic, and ensures that the content is structured correctly.

Moreover, the Viewer makes the documentation available for the GSD and allows for the integration of any XML Editor for generating GSD files.

Since a few sample files are installed at the same time, the tool provides a good basis for generating new GSD files.

The figure below shows a screen shot of the tool:



22.1.1 System Prerequisites

The PROFINET XML Viewer V 2.3 is a.net based application and therefore requires the Microsoft .net Framework V2.0. When installing the PROFINET XML Viewer, a check is made whether it already exists on your PC and if not, you will be guided to the download area of Microsoft.

For licensing reasons, the Xerces Parser of the Apache Software Foundation is not automatically installed also. How this parser is integrated is described in the chapter “Settings“.

22.1.2 Form of Representation

The PROFINET XML Viewer offers two different forms of representation:

In the XML view “XML View“, a GSD file is displayed in the way it would appear if it were opened with the Internet Explorer and essentially corresponds to the representation of an ASCII Editor.

In the HTML view “Browser View“, the contents of a GSD are displayed in clearly arranged table form. Since at the same time, the text references are converted into texts, readability is considerably better than in XML representation. In addition, if you generated the GSD files yourself, you can check very quickly whether the text references are set correctly.

You can switch between the two types of representation by clicking on the respective tab in the upper area of the window.

22.1.3 Sample Files

When the tools are installed, a few sample files are installed also. You can open these files with the menu option “File Open GSD Example..“ and use them as a basis for a GSD file you generate yourself.

22.1.4 Saving a Copy of the Displayed File

This function -called under the menu option “Save As…“- makes it possible to generate a copy of the displayed GSD file. In the process, a check is made whether the schematic files needed for validation are available in the target folder. If this is not the case, the schematic files are copied automatically to the subdirectory “..\xsd“ so that afterwards, the copy can be validated with a standard XML Editor and you don’t have to concern yourself with schematic files.

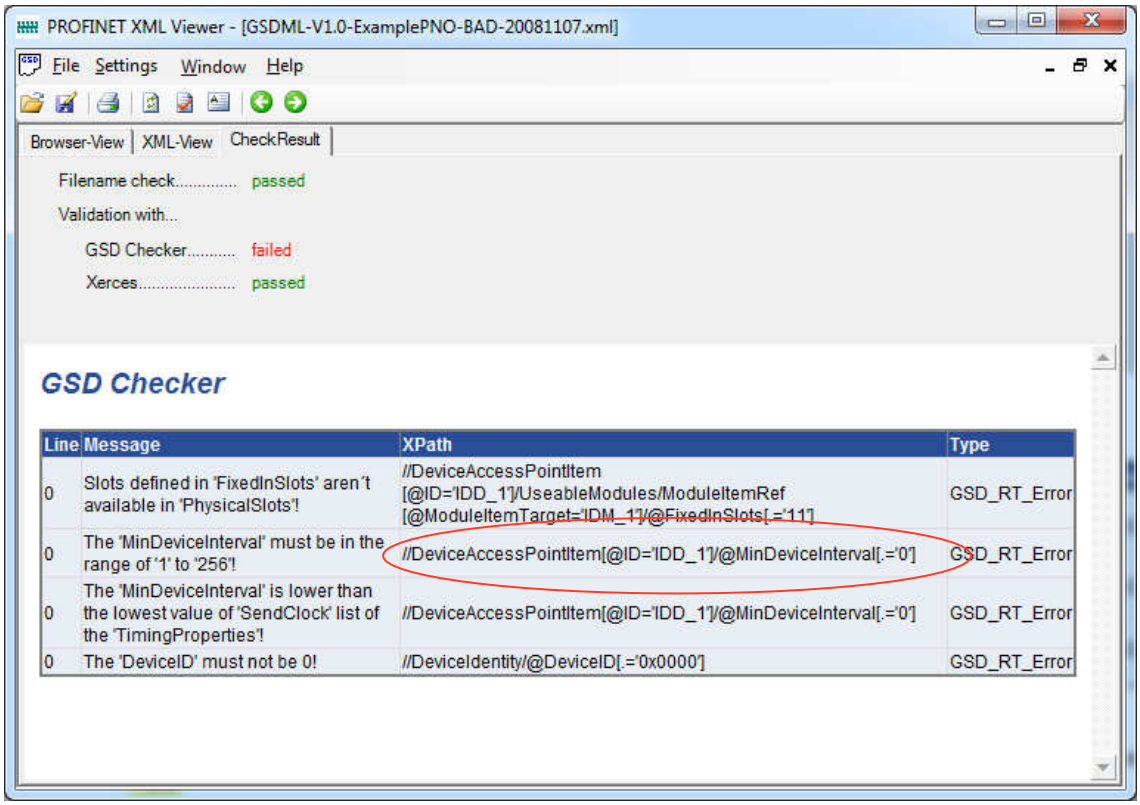
22.1.5 Integrating the Editor

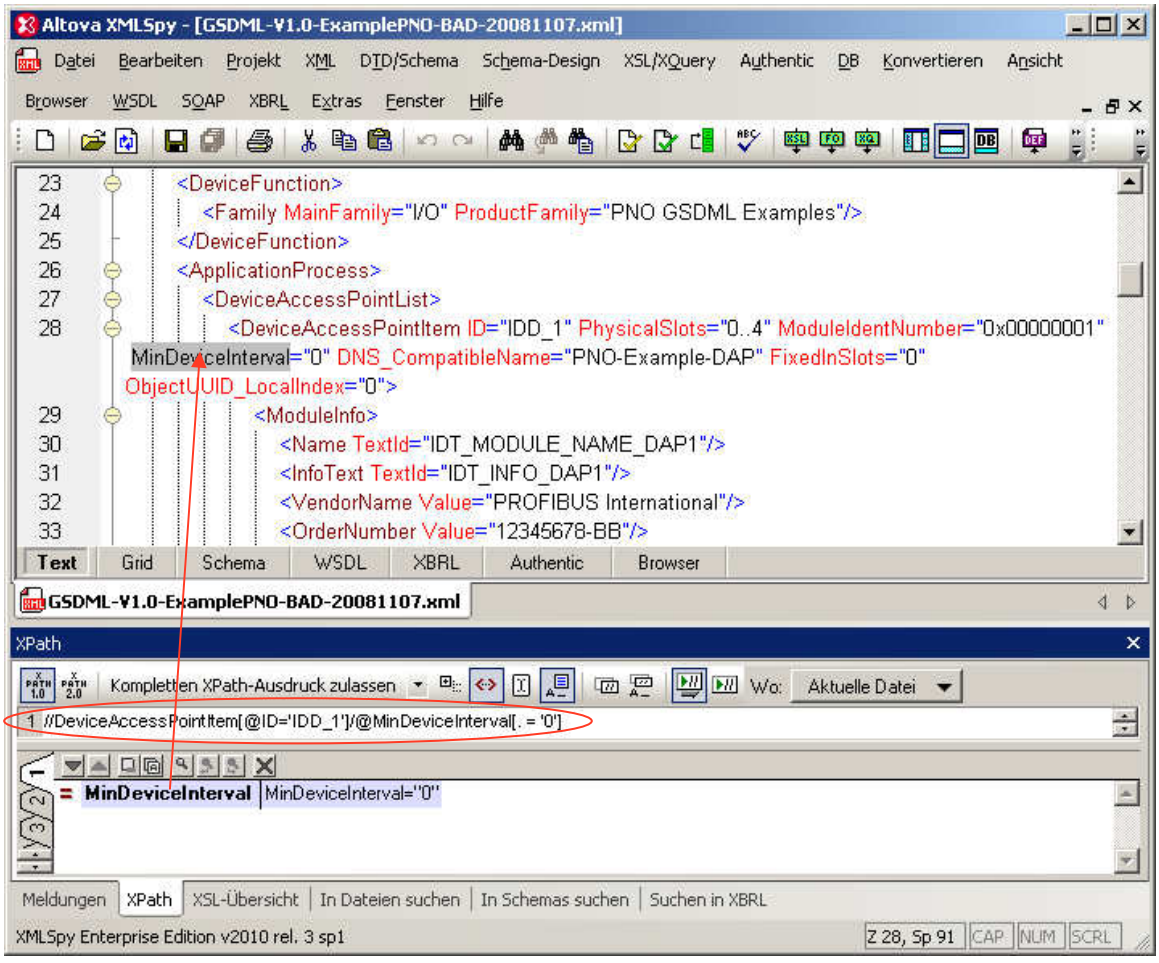
You can start any editor from the PROFINET XML Editor. Under “Settings“ you can set which editor you want to start. If you have not made a choice yet, the Windows editor “Notepad“ is used. After you have made changes in the editor, you can reload the file in the XML Viewer with the “Reload“ function to view and check the changes again.

22.1.6 Checking the GSD File

With the aid of three common XML parsers, the PROFINET XML Viewer checks whether the content of the GSD file is set up correctly. In addition, a separate checker is installed for GSD files that performs additional checks whose rules can not be described in the schematic.

If errors occur, the respective line number of the error location is displayed in the case of XML parsers. The GSD Checker indicates the exact error location in the form of an XPath expression instead. XPath is a standardized form for addressing individual elements in an XML file. You can use this XPath expression to jump in an XML editor to the error location. The XML editor “XML Spy“ provides this with the function “Evaluate XPath“, for example:



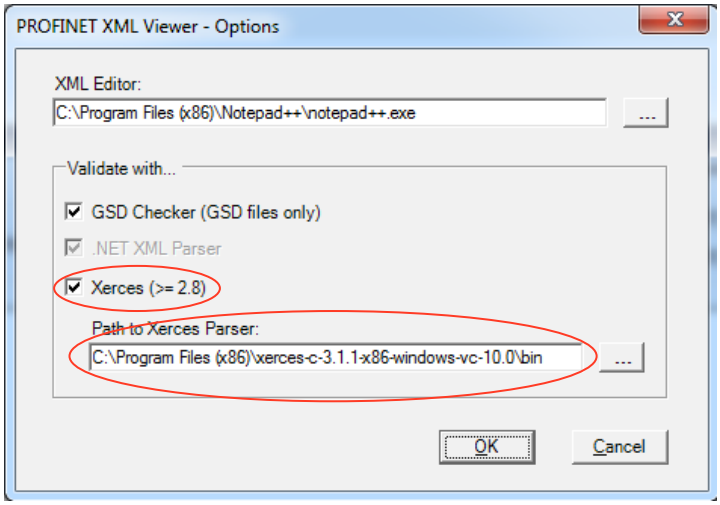


22.1.7 Settings

In the dialog “Settings“, you can specify which editor is to be used and which XML parsers are to be used for validation.

For licensing reasons, the Xerces Parser of the Apache Software Foundation is not installed automatically also. However, you can do this yourself without a problem. To this end, load the software from the Web Page of the Apache Software Foundation http://xml.apache.org/xerces-c/.

After unzipping the ZIP file, all you have to do is enter the “bin“ directory that was created in the field “Path to Xerces Parser“.



22.1.8 Documentation

The GSDML documentation is automatically installed also and can be called with the menu option “Help – GSMDL Documentation“.

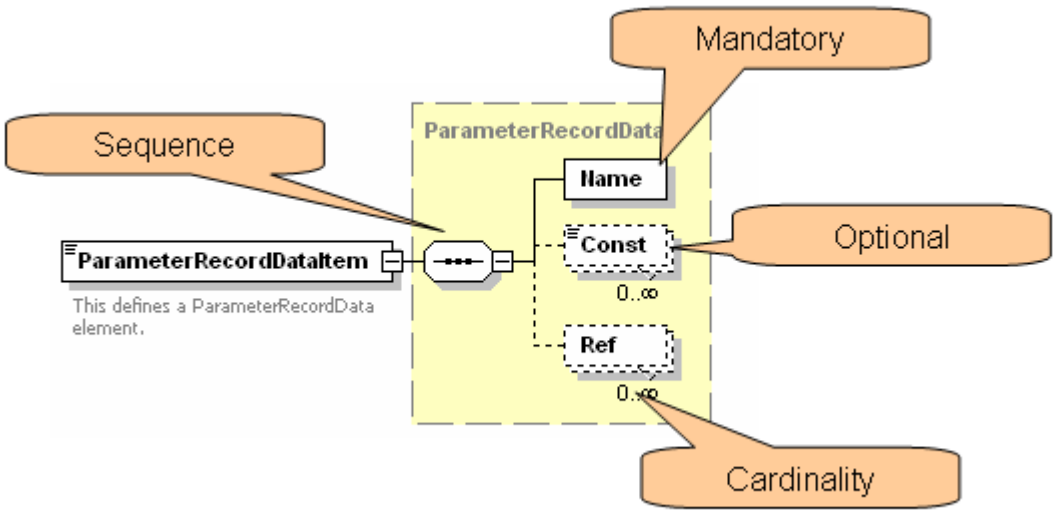
The XMLSpy schematic notation is used:

A sequence that has to be adhered to is indicated with the “Sequence“ symbol (refer to figure below).

Elements that have to be included (mandatory) are drawn as a framed box.

If an element is optional, the box is framed with dashes.

Below an element you can store how often it can be used. In our example, the “Ref“ element is optional and can be used as often as required.



22.2 Validating XML Parsers

The XSD files for GSDML use many features of the XML schematic language. An XML parser has to be able to interpret the constructs that are used.

The following Freeware Parsers can be used for validating a GSD file. (No claim is made that the list is complete!)

MSXML 6.0

To be obtained under:

Microsoft Core XML Services (MSXML) 6.0 Service Pack 1 – http://www.microsoft.com/downloads/de-de/details.aspx?FamilyID=d21c292c-368b-4ce1-9dab-3e9827b70604

Please use the Microsoft Update to obtain the latest security updates.

XmlReader Klasse von Microsoft .NET Framework Version 2.0

With limited scheme valisation, i. ewith Settings.ValidationType = ValidationType.Schema.

To be obtained under:

- Microsoft .NET Framework Version 2.0 Redistributable Package – Date Published: 1/22/2006, http://www.microsoft.com/downloads/details.aspx?familyid=0856EACB-4362-4B0D-8EDD-AAB15C5E04F5&displaylang=de
- Microsoft .NET Framework 2.0 Service Pack 2 – Date Published: 1/16/2009, http://www.microsoft.com/downloads/details.aspx?FamilyID=5b2c0358-915b-4eb5-9b1d-10e506da9d0f&displaylang=de

Please use the Microsoft Update to obtain the latest security updates.

Xerces XML Parser

To be obtained under http://xerces.apache.org/xerces-c/

Altova AltovaXML Community Edition

To be obtained under http://www.altova.com/de/altovaxml.html

22.3 Tools for GSD Generation

A GSD file is an XML file and can therefore be generated with a simple text editor. However, no support is provided when the input is made. Thus, the risk is great that a file is generated that is not well designed, or not valid according to a schematic.

When using XML, numerous tools can be employed to generate a GSD file. Two groups have to be essentially differentiated:

Validating XML parsers check whether the present XML document is structured according to the rules of a schematic.

XML editors provide support when editing an XML document and as a rule possess an integrated XML parser, or an interface for an external XML parser.

There is a huge number of editors for generating XML documents. To generate the GSDML schematic files themselves the tool "XMLSpy" was used. It can also be used to easily edit GSD files. XML Spy has a "built in" XML parser that can be used for validating GSD files.

The XMLSpy can be obtained under

http://www.altova.com/xmlspy.html

Further XML-editors:

XML Notepad 2007 (free of charge, see
http://www.microsoft.com/downloads/details.aspx?familyid=72d6aa49-787d-4118-ba5f-4f30fe913628&displaylang=de)
Xopus (commercial, see http://xopus.com/)
XMLFox (kostenlos, siehe http://xmlfox.com/)
XML Copy Editor (free of charge, see http://xml-copy-editor.sourceforge.net/)
<oXygen/> XML Editor (commercial, see http://www.oxygenxml.com/)
Easy XML Editor (commercial, see http://easy-xml-editor.de/)

