SIMATIC NET

GSDML-Einstieg leicht gemacht

Getting Started

Datum 23.08.2012

SIEMENS

SIMATIC NET

GSDML

Getting Started

(Leichter Einstieg in die
Erstellung
einer GSD-Datei für PROFINET IO)

Version: 1.3
Datum: 08/12

Haftungsausschluß

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenden Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in der Druckschrift werden jedoch regelmäßig überprüft. Notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Für Verbesserungsvorschläge sind wir dankbar.

Copyright

Versionskennungen

Version	Datum	Änderung
V 1.0	04.09.2008	Erste Fassung
V 1.1	02.03.2009	Korrekturen eingefügt
V 1.2	19.04.2012	Auf den aktuellen Stand gebracht
V 1.3	23.08.2012	Auf den aktuellen Stand gebracht

Inhalt

1 Über dieses Dokument

Dieses Dokument soll Ihnen den Einstieg in die Anwendung und den Aufbau der GSDML erleichtern. Es ersetzt nicht das normativ geltende Dokument, sondern baut darauf auf und möchte anhand einiger Beispiele die prinzipielle Wirkungsweise der Elemente und Attribute eines GSDML -Dokumentes näher bringen.

Das Dokument "GSDML-Specification for PROFINET IO" wird über den Webserver der PNO bereitgestellt unter www.profinet.com → Downloads.

Sollten sich Widersprüche zur GSDML-Spezifikation ergeben, ist die GSDML-Spezifikation maßgebend.

Das Dokument wird künftig an die Marktbedürfnisse angepasst. Rückmeldungen sind deshalb willkommen.

Fehlende Passagen oder Ungereimtheiten melden Sie bitte an ComDeC@siemens.com.

Die Syntax von XML selbst und der Aufbau von Schemadateien werden hier nicht erläutert. Zu diesem Zweck werden zahlreiche gute Bücher angeboten.

Einige Links gehen derzeit noch ins Leere. Sobald sie aktualisiert sind, wird das Dokument nachgezogen.

2 Einleitung

Mit Einführung der GSDML (**G**eneral **S**tation **D**escription **M**arkup **L**anguage) wurde die Eigenschaft XML basierter Dokumente beliebige Hierarchiestufen bilden zu können dazu genutzt, das hierarchische Gerätemodell möglichst unverändert abzubilden.

Entstanden ist eine Beschreibungssprache die über mehrere Ebenen die Eigenschaften einer Gerätefamilie beschreiben kann. Bei der Erstellung der XML Schemata wurde versucht, möglichst viel von der Semantik der GSD zu übernehmen, aber gleichzeitig dort, wo es sinnvoll und notwendig erschien, die Abbildung neu zu gestalten.

Hinweis zur Begriffsbezeichnung:

GSDML ist eine Sprache zur Beschreibung von PROFINET IO-Feldgeräten. Durch die Anwendung dieser Sprache entsteht wiederum eine GSD (General Station Description). Es ist deshalb korrekt von einer "GSD-Datei" zu sprechen auch wenn diese in XML-Notation aufgebaut ist.

Wird im folgenden Dokument der Begriff "GSD" oder "GSD-Datei" verwendet, so bezieht sich dieser stets auf die XML basierte Form. Falls die schlüsselwortbasierte GSD gemeint ist, so wird dies ausdrücklich erwähnt.

2.1 Neuerungen

Verglichen mit der schlüsselwortbasierten GSD-Datei (PROFIBUS) ergeben sich neben der syntaktischen Änderung durch XML folgende wesentliche Neuerungen bei der XML basierten GSD-Datei:

- Der "Device Access Point" (DAP) wird explizit bei der GSDML modelliert. Bei der GSDL(PROFIBUS) wird hingegen nur der DP-Slave als Ganzes beschrieben. Das Einführen des DAPs hat zur Folge, dass fast alle Parameter des Devices am DAP Element beschrieben werden.

- Ein GSDML-Dokument kann beliebig viele DAP Definitionen enthalten. Dies ermöglicht es, eine Datei für eine Gerätefamilie (und nicht nur für einzelne Feldgeräte) zu erstellen. Der Vorteil liegt insbesondere bei modular aufgebauten Feldgeräten auf der Hand: Die Beschreibung der Module kann zentral in **einer** GSD-Datei gepflegt werden – der Erstellungs- und Pflegeaufwand wird gegenüber der schlüsselwortbasierten GSD-Datei deutlich reduziert, da dort die Modulbeschreibungen redundant vorliegen und "per Hand" in den verschiedenen Dateien identisch gehalten werden müssen.

- Ein GSDML-Dokument ist in der Lage, beliebig viele Fremdsprachentexte in **einer** Datei zu halten. Dies führt zu einer weiteren Reduzierung der Anzahl notwendiger Dateien, da bei der schlüsselwortbasierten GSD-Datei für jede Sprache eine separate Datei erforderlich ist.

Hinzu kommen die Möglichkeiten, die sich durch Verwendung des XML Standards ergeben: Validierung einer XML Datei mittels Schemabeschreibung und Transformation in andere Formate mittels XLS.

3 Nutzung von Schemadateien zur Validierung

Der Aufbau eines XML Dokuments kann mit Hilfe einer Schemadatei überprüft werden. Man spricht in diesem Zuge von der Validierung eines Dokuments. Bei diesem Vorgang wird unter anderem geprüft, ob die Element-Struktur und die im XML Dokument verwendeten Attribute mit der Schemadefinition übereinstimmen. In der Schemadatei ist beispielsweise hinterlegt, ob ein Attribut vorhanden sein muss oder ob dieses Attribut optionalen Charakter hat.

Die Validierung selbst erfolgt durch einen XML Parser. Eine Übersicht hierzu finden Sie im Abschnitt "Tools".

Eine weitere Möglichkeit, den korrekten Aufbau einer GSD Datei zu überprüfen ist durch das Tool „PROFINET XML Viewer“ gegeben, das über die Webseiten der PNO zum Download angeboten wird. Sollten Sie dieses Tool nutzen, benötigen Sie kein Know-How über die Funktion und Anwendung von Schemadateien da diese vom PROFINET XML Viewer automatisch installiert und zur Validierung herangezogen werden. Sie können in diesem Fall dieses Kapitel überspringen. Mehr Informationen zum Tool „PROFINET XML Viewer“ finden Sie im Kapitel „Tools“.

3.1 Anwendung der Schemadateien im GSDML-Kontext

Die GSDML-Schemadateien werden vom PNO Arbeitskreis erstellt, gepflegt und über den PNO Webserver zum Download bereitgestellt.

Mit Hilfe eines validierenden XML Parsers ist ein Autor einer GSD Datei nun in der Lage, eine vorliegende GSD Datei gegen das Schema zu prüfen und gegebenenfalls zu korrigieren.

Beim Import einer GSD Datei in ein Tool kann dieses als ersten Schritt ebenfalls automatisch gegen die Schemadateien prüfen und im Fehlerfall den Import einer GSD Datei abweisen.

Im folgenden Bild wird der prinzipielle Ablauf beim Schemahandling dargestellt:

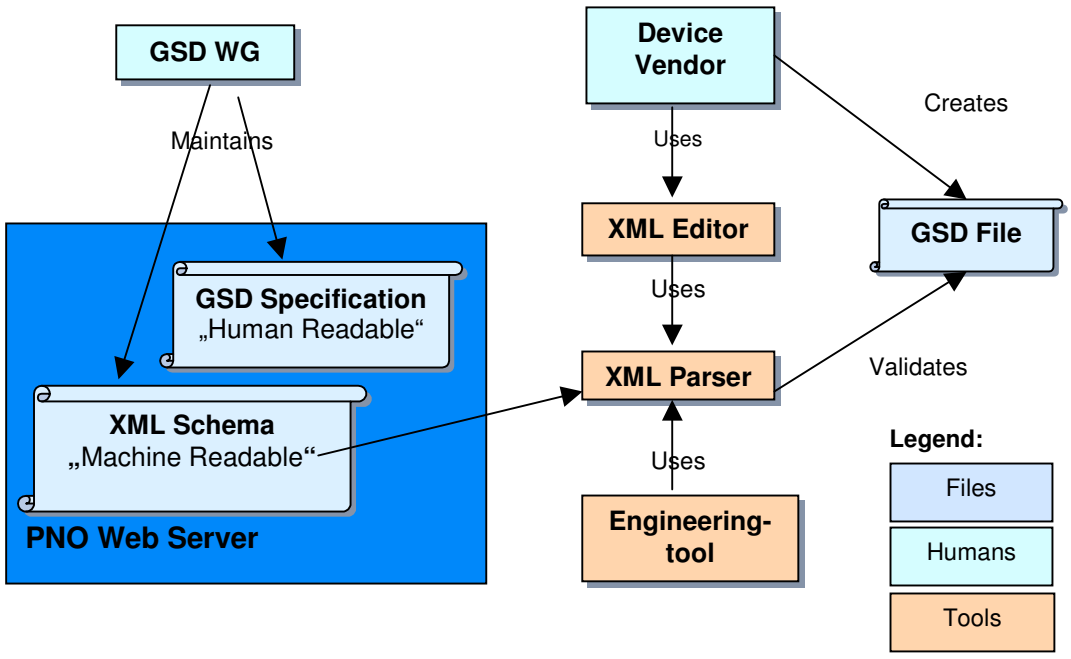


Abbildung 1: Arbeitsfluss der GSDML

3.2 Schemadateien der GSDML

Zur Validierung einer GSD Datei sind vier Schemadateien erforderlich. Diese bauen wie folgt aufeinander auf:

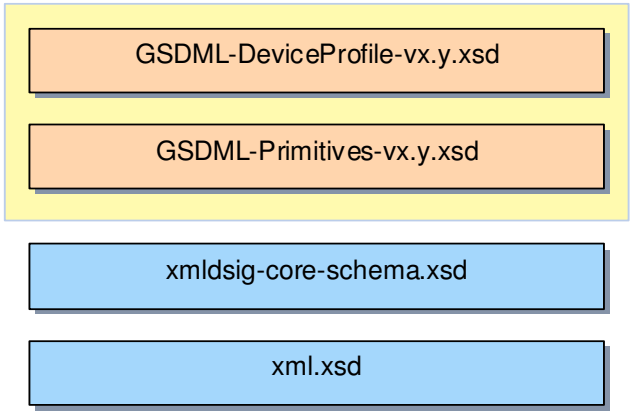


Abbildung 2: Schema-Layer

Die Schemafiles werden von unterschiedlichen Gruppen gepflegt und weiterentwickelt. Die GSDML-Schemata basieren auf den Grundlagen der XML@PROFIBUS Guideline, die ebenfalls über den PNO Webserver geladen werden kann. Hierzu dient die URL zur Namespace-Identifikation gleichzeitig zum Download der Schemadateien.

Die Bedeutung der Schemafiles im Einzelnen:

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

(This schema defines attributes and an attribute group suitable for use by schemas wishing to allow xml:lang or xml:space attributes on elements they define.)

Das World Wide Web Konsortium stellt diese Schemadatei unter der URL http://www.w3.org/2001/xml.xsd zum Download bereit.

xmldsig-core-schema.xsd (Namespace 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.)

Das World Wide Web Konsortium stellt diese Schemadatei unter der URL http://www.w3.org/2000/09/xmldsig# zum Download bereit.

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

Diese Schemadatei enthält grundlegende Strukturdefinitionen.

x.y im Dateinamen steht in diesem Fall stellvertretend für die Version des Schemas. Die erste Version trägt die Versionsbezeichnung 1.0.

GSDML-DeviceProfile-vx.y.xsd (Namespace "http://www.profibus.com/GSDML/2003/11/DeviceProfile")

Dies ist das eigentliche Schemadokument zur Definition der GSDML. Es benutzt das GSDML-Primitives Schema und kann nur zusammen mit diesem verwendet werden.

Die Schemadateien können unter Verwendung der Namespace URLs vom entsprechenden Webserver geladen werden. Sie gelangen bei Eingabe der URL zu einer HTML Seite auf der Sie die entsprechende Schemadatei

auf Ihre Festplatte laden können. Legen Sie hierfür ein Verzeichnis "xsd" an in dem sie diese vier Schemadateien legen.

4 Regeln zur Bildung des GSD Dateinamens

Anders als bei der schlüsselwortbasierten GSD ist die Länge des Dateinamens nicht auf acht Zeichen begrenzt. Der Dateiname muss nachfolgenden Regeln entsprechen:

GSDML-*Schemaversion-Herstellername-Name der Gerätefamilie-Datum*[-*Uhrzeit*].xml

Der Namensbestandteil in eckigen Klammern ist optional.

Für die kursiv beschriebenen Schlüsselwörter gelten die folgenden Regeln:

Schemaversion: Versionskennung des referenzierenden Schemas. Diese Versionskennung muss der Versionskennung im Dateinamen der GSDML-DeviceProfile-[Schemaversion].xsd entsprechen.

Herstellername: Name des Geräteherstellers. Bindestriche und Leerzeichen im Namen sind zulässig. Um zu vermeiden dass sich gleiche Dateinamen unterschiedlicher Hersteller bilden können, wird zudem empfohlen, neben dem Namen auch die PNO-ID (VendorID) als Namensbestandteil zu integrieren.

Name der Gerätefamilie: Definiert, welche Gerätefamilie in der GSDML beschrieben wird. Bindestriche und Leerzeichen im Namen sind zulässig. Der Name sollte dem Inhalt von DeviceFunction/Family/@ProductFamily entsprechen.

Datum: Datum der Freigabe der GSD Datei im Format yyyyMMdd.

Uhrzeit: Uhrzeit der Freigabe der GSD Datei im Format hhmmss (optional).

Durch die Angabe von Datum und Uhrzeit wird eine Versionszählung der GSD Datei selbst überflüssig und es ist gewährleistet, dass unterschiedliche Versionen eine bestehende Datei nicht überschreiben.

Wird eine GSD Datei mit einem neueren Datum/Uhrzeit für die gleiche Produktfamilie installiert, so werden in einem Engineering-Tool in der Regel nur die Feldgeräte der neuesten GSD Datei im Auswahlkatalog angezeigt.

Beispiel für einen gültigen Dateinamen:

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

5 Grobstruktur einer GSD Datei

5.1 Verarbeitungshinweise für den XML Parser

Wie jedes XML Dokument müssen zu Beginn einer GSD Datei die Processing Instructions definiert werden. Diese sehen in der Regel wie folgt aus:

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

Hier kann lediglich die Definition des Encodings variiert werden. In der GSDML gibt es hierfür keine Restriktionen – alle dem XML Standard entsprechenden Codierungen können angegeben werden. Es gibt allerdings Editoren, die mit Unicode Angaben Probleme haben und insbesondere die Byte-Order Mark nicht korrekt interpretieren. Ansonsten ist die Verwendung von UTF-8 in der Regel empfehlenswert da damit sämtliche länderspezifischen Sonderzeichen dargestellt werden können ohne dass dies zu einer signifikanten Vergrößerung der Dateigröße führt (wie bei UTF-16 oder UTF-32).

5.2 Root Element ISO15745Profile

Das Root Element einer GSD Datei ist "ISO15745Profile". Zwingend anzugeben sind die verwendeten Namespaces und die Importanweisung für die Schemadatei.

```
<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-v2.3.xsd">
```

Die Schemadatei sollte in einem parallelen Verzeichnis "xsd" zu der GSD Datei liegen! Diese Regel ist erforderlich, damit ein Engineering-Tool oder eine Zertifizierungsstelle (bei denen die GSD Dateien der unterschiedlichen Hersteller zusammen abgelegt werden) die identischen Schemafiles nicht mehrfach ablegen muss, um eine Validierung durchführen zu können.

Wie in der ISO 15745 vorgegeben, besteht ein ISO15745Profile aus einem ProfileHeader und einem ProfileBody.

Folgende Abbildung zeigt den prinzipiellen Aufbau einer GSD Datei:

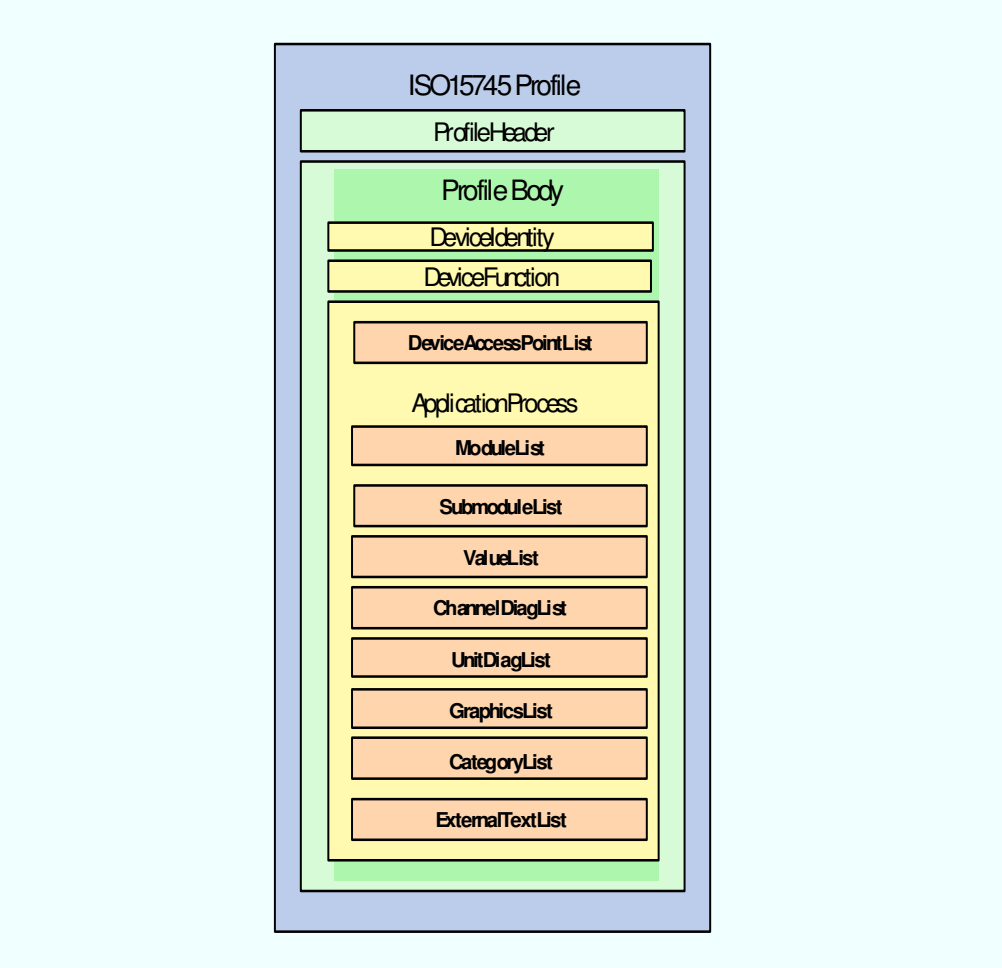


Abbildung 3: Prinzipielle Struktur einer GSD Datei

5.2.1 ProfileHeader

Der ProfileHeader ist wie folgt zu belegen:

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

5.2.2 ProfileBody

Der ProfileBody enthält die eigentlichen Daten eines Feldgeräts und ist in drei Teile gegliedert:

- "DeviceIdentity" enthält Informationen zur Identifizierung eines Feldgeräts,

- "DeviceFunction" enthält Daten, die die Funktion beschreiben.

- "ApplicationProcess". Dies ist der Hauptteil der Beschreibungsdatei. Die wichtigsten Abschnitte des ApplicationProcess Blocks sind in den folgenden Unterkapiteln aufgelistet.

5.2.2.1 DeviceAccessPointList

Dieser Abschnitt enthält die Beschreibung der einzelnen Device Access Points (Netzwerk-Zugangspunkt). Wie bereits erwähnt, kann eine GSD Datei die Beschreibung für beliebig viele Interfacemodule enthalten. Die Summe aller Device Access Points bildet dann eine Gerätefamilie. Der Ausbaugrad eines DAPs ist bei modularen Feldgeräten konfigurierbar. Jedem DAP können unterschiedliche Module (Baugruppen) zugeordnet werden.

5.2.2.2 ModuleList

Dieser Abschnitt enthält die Beschreibung der einzelnen Module eines Feldgeräts. Diese können steckbar (bei modular aufgebauten Feldgeräten) oder fest in ein Feldgerät integriert sein.

5.2.2.3 SubmoduleList

Dieser Abschnitt enthält die Beschreibung der einzelnen Submodule eines Feldgeräts. Diese können steckbar (bei modular aufgebauten Feldgeräten) oder fest in ein Feldgerät integriert sein.

5.2.2.4 ValueList

Dieser Abschnitt enthält die *ValueList* für einzelne Parameter eines Feldgeräts, neben dem Parameternamen auch die Zuweisung zwischen einem konkreten Wert und einem zugehörigem Text.

5.2.2.5 ChannelDiagList

Dieser Abschnitt enthält die *ChannelDiagList*. Sie stellt die Zuordnung zwischen einem Kanalfehler eines Feldgeräts und den entsprechenden Texten dar.

5.2.2.6 UnitDiagTypeList

Dieser Abschnitt enthält die *UnitDiagTypeList* und beschreibt den strukturellen Aufbau der generischen Diagnosemeldungen eines Feldgeräts.

5.2.2.7 LogBookEntryList

Die *LogBookEntryList* beschreibt die Bedeutung der Logbuch-Einträge. ...

5.2.2.8 GraphicsList

Dieser Abschnitt enthält Referenzen auf grafische Repräsentationen eines Feldgeräts.

5.2.2.9 CategoryList

Dieser Abschnitt enthält die Zuweisung eines Moduls zu einer bestimmten Kategorie (z.B.. Digital Input, Analog Output, etc.). Diese dient zur Gliederung und besseren Auffindbarkeit innerhalb des Modulkatalogs eines Engineering-Tools.

5.2.2.10 ExternalTextList

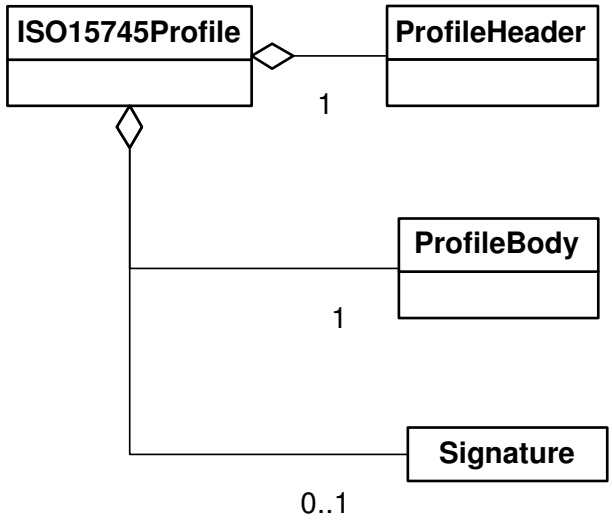
Dieser Abschnitt enthält die „*ExternalTextList*“ der GSDML. Hier sind sämtliche Texte hinterlegt, die von den anderen Abschnitten referenziert werden können. Mehr dazu im Kapitel "Einbindung von Fremdsprachen".

6 Erstellung einer GSD Datei

Zur Erstellung einer GSD Datei "from Scratch" kann natürlich das GSDML-Schema dienen. Die dort enthaltene Vorschrift zur Syntax einer GSDML-basierten Datei ist ausreichend, um eine korrekte und validierende GSD Datei zu erstellen.

Einfacher ist es jedoch, von einer bestehenden GSD Datei auszugehen und diese an die eigenen Feldgeräte anzupassen. Eine Ausgangsbasis stellen z.B. die Beispieldateien der PNO dar, die sowohl der GSDML-Spezifikation als auch dem PROFINET XML Viewer beigefügt sind.

Die Grafiken in der GSD-Datei sind in UML-Notation dargestellt und erleichtern die Lesbarkeit und das Verständnis für die Hierarchie der einzutragenden Definitionen. Hier ein kleines Beispiel:



Der Einfachheit halber lesen Sie bitte das UML-Schema von links nach rechts. Somit eröffnet sich Ihnen auch gleichzeitig die Hierarchie der Datenstruktur. Die angegebenen Zahlenwerte haben folgende Bedeutungen:

1	Diese Definition muss immer angegeben werden
0 … 1	diese Definition kann optional angegeben werden
0 … *	Der Wertebereich bedeutet, dass nichts angegeben werden muss, aber auch beliebig viele Definitionen angegeben werden können
1 … *	Der Wertebereich bedeutet, dass mindestens eine Definition angegeben werden muss. Es können aber auch beliebig viele Definitionen angegeben werden.

Wenn Sie von einer bestehenden GSD Datei ausgehen, müssen Sie – neben den Anpassungen an Ihre Feldgeräte - auf jeden Fall noch folgende Anpassungen durchführen:

- Der Dateiname muss gemäß Kapitel 4 strukturiert sein.
- Tragen Sie als „VendorID“ (Herstellerkennung) die Zahl ein, die für Ihre Firma bei der PNO definiert ist. Eine Übersicht hierzu finden Sie unter http://www.profibus.com/IM/Man_ID_Table.xml. Beachten Sie hierbei, dass Sie die Zahl in der GSDML hexadezimal eingeben.
- Ändern Sie die DeviceID zur Kennung Ihrer Gerätefamilie

- Passen Sie gegebenenfalls die „MainFamily“ an, die eine grobe technologische Einordnung Ihres Feldgeräts darstellt. Dieses Attribut wird zur Strukturierung des HW-Katalogs im Engineering-Tool verwendet. Siehe hierzu auch Kapitel 15.2.

- Tragen Sie im Attribut „ProductFamily“ die Bezeichnung Ihrer Produktfamilie ein.

Im folgenden Bild sehen Sie die Änderungsstellen in der GSD Datei

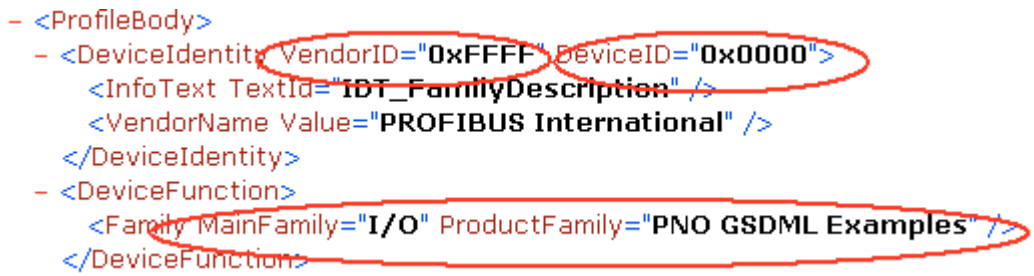


Abbildung 4: VendorID, DeviceID und Produktinformation in der GSD

Beispiel für den Aufbau einer Device Identification:

Der Aufbau der DeviceID kann firmenintern festgelegt werden, sollte aber innerhalb derselben Firma einheitlich sein (bitte mit Marketingabteilung und evtl. Vertrieb absprechen).

Hier ein Beispiel, um die Idee zu verdeutlichen

VendorID		DeviceID	
		Geräteklasse	Gerätefamilie
00	**2A**	Byte	Byte
Beispiel für	Siemens	00= DevelopmentKit	00= reserviert 01= ERTEC 200 DevKit 02= ERTEC 400 DevKit 03= …
		01= Steuerungen (Controller)	01= S7 300 02= S7 400 03= …

7 Device Access Point List (DAP)

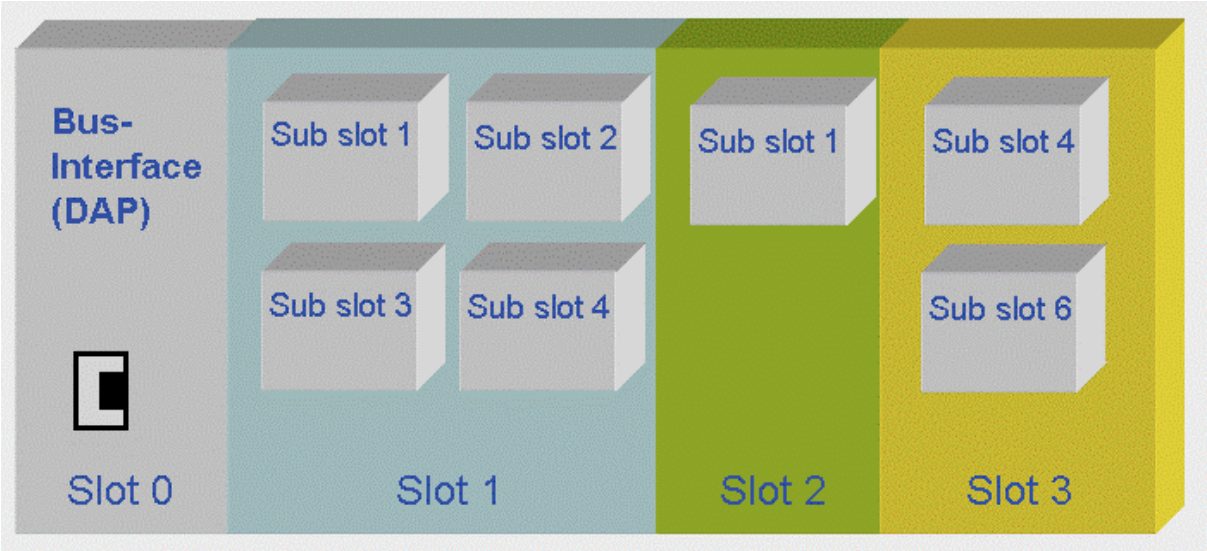


Abbildung 4: Aufbau eines modularen Feldgeräts mit den zugehörigen Modulen.

Innerhalb des ApplicationProcess beginnen die Definitionen mit dem DAP. Ein DAP repräsentiert das Businterface und bestimmt damit die wesentlichen Eigenschaften des IO-Devices. In der GSD-Datei sind somit die technischen Möglichkeiten des IO-Devices und die möglichen Module in ihren Eigenschaften zu beschreiben. In einer GSD-Datei können mehrere DAPs definiert werden.

Zu einem DAP gehören auch immer

- das Physical Device Management (PDEV), das alle hardwarenahen Definitionen beinhaltet (siehe hierzu auch Kapitel 8 „das Physical Device Management).
- Die unterstützten Real Time Klassen
- Redundanz-Definitionen
- Die Port-Definitionen für jeden Switchport

Beispiel: Hier ist ein DAP, basierend auf dem ERTEC 200 definiert. Da der ERTEC 200 2 Switchports besitzt, müsssen diese auch innerhalb des DAPs mit den zugehörigen Subslot-Nummern definiert werden.

```
<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>  <!—hier beginnen die Definitionen für die Adressierung des 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>

Hier stehen die PDev-Definitionen, siehe Kapitel 8

</SystemDefinedSubmoduleList>
```


8 Definition des Physical Device Managements (PDev)

Bei PROFINET werden die physikalischen Schnittstellen eines Feldgerätes und deren Switchports sowie das Verhalten als PDev bezeichnet. Eine Ethernet-Schnittstelle wird dabei eindeutig durch ihren Namen, IP- und MAC-Adresse identifiziert und kann aus mehreren Switchports bestehen.

In einem Feldgerät sind alle hardwarenahen Portinformationen und Übertragungsparameter im PDev remanent abgelegt. Diese sind folgendermaßen organisiert:

Interface-Submodule entspricht einer Instanz. Hier sind beispielsweise die Planungsdaten für die IRT-Kommunikation, Clock-Synchronisation, Medienredundanz, etc. abgelegt.

Port-Submodule Die Daten zu Porteinstellungen, Nachbarschaftsinformationen, Fiber Optic-Parameter, etc. sind hier gespeichert.

Hierzu sind für das jeweilige DAP die Subslot-Nummern gemäß Eintrag in der GSD-Datei definiert. Diese Nummern beginnen ab 0x8000 für den Interface-Subslot. Die Aufteilung ist folgendermaßen gegliedert:

Subslot-Nummer 0x8i00 → i = Nummer des Interface-Moduls (0 ..0x0F) und

Subslot-Nummer 0x8ipp → pp = Nummer des jeweiligen Ports (1 ... 0x0FF)

Die Adressierung des PDev ist in der GSD-Datei des IO-Devices definiert. Nachfolgend sehen Sie, wie so eine Definition in der GSD-Datei beispielhaft aussehen kann.

Beispiel

Die PDev-Definition befindet sich innerhalb der SystemDefinedSubmoduleList und gehört zum jeweiligen DAP. Zur Adressierung des PDev wurde im folgenden Beispiel der Subslot 0x8000 (32768 dez) gewählt. Hier finden Sie auch die unterstützten Real Time Klassen, die IRT-Definitionen und Definitionen für den Fast Startup. Auf diese Definitionen wird in den entsprechenden Kapiteln näher eingegangen. Der Übersicht halber sind sie aber in dem Beispiel enthalten.

```
<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 der Real Time Klassen

PROFINET bietet ein kaskadierbares Real Time Konzept. Um die Kommunikation optimal auf die Bedürfnisse der Automatisierungsanlage abstimmen zu können, ist es notwendig, in der GSD-Datei anzugeben, welche Real Time Klassen von dem jeweiligen Feldgerät unterstützt werden. Diese Definitionen sind DAP-spezifisch und somit im PDev (InterfaceSubmodule) anzugeben.

In der GSD-Datei könnte die Definition der Real Time Klassen folgendermaßen aussehen:

```
<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"
</InterfaceSubmoduleItem>
```

9.1 Isochronous Real Time

Eine Isochronous, oder taktsynchrones Real Time gehört zur synchronisierten Kommunikation (RT_Class_2 und RT_Class_3, synchronisiert). Hierzu sind die entsprechenden Definitionen zum jeweiligen DAP innerhalb des PDev vorzunehmen. Zur Synchronisation verwendet PROFINET das PTCP-Protokoll. Da ein Synchronisationsframe über den gesamten Bus hinweg an alle Teilnehmer gesendet wird, sind auch die Verzögerungszeiten beim Durchlauf des Frames durch einen Switchport in Sende- und Empfangsrichtung anzugeben.

Beispiel

```
<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" <!-- Verzögerung im Switch in Sende- und Empfangsrichtung -->
   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 Module, Submodule, Slots und Subslots

In einer GSD-Datei können mehrere unterschiedliche Module und Submodule definiert sein. Diese werden nur einmal innerhalb einer GSD-Datei definiert und können bei Bedarf unterschiedlichen DAPs zugewiesen werden. Die Module-/Submodule definieren den physikalischen Ausbaugrad einer elektronischen Baugruppe. Sie sind jeweils durch eine achtstellige Identnummer eindeutig referenziert. Deshalb müssen die Module- und Submodule-Identnummern eindeutig (einmalig sein). Für den realen Betrieb werden die Module/Submodule den physikalischen Slots eines Baugruppensystems zugewiesen. Dabei kann ein Module/Submodule auch mehreren Slots zugewiesen werden. Die I/O-Daten werden immer über eine Slot-Subslot-Kombination adressiert.

Die Kombination aus *DeviceID, ModuleIdentnumber und SubmoduleIdentnumber* muss innerhalb des Herstellers eindeutig sein.

Da eine Hardware sowohl aus Sicht des Engineering Tools als auch aus Sicht des prozessnahen Feldgerätes (in der Regel ein IO-Device) eindeutig und unverwechselbar identifizierbar sein muss, gelten folgende Regeln für die Benutzung der Identnummern:

- Module sind durch eindeutige, unverwechselbare Identnummern innerhalb eines Feldgerätes gekennzeichnet.

- Submodule sind durch eindeutige Submodule-Identnummern innerhalb eines Moduls gekennzeichnet.

Weiterhin gilt es zu beachten:

Wenn ein Modul in unterschiedlichen Feldgeräten aus derselben oder einer anderen Gerätefamilie eingesetzt werden kann, muss die ModuleIdentnumber einheitlich über diese Feldgeräte hinweg sein.

Wenn ein Submodul in unterschiedlichen Modulen einsetzbar ist, muss die SubmoduleIdentnummer über diese Module hinweg einheitlich sein.

Für eine solche *SubmoduleIdentnumber* gilt:

Diese Submodule identifizieren nicht nur die physikalischen Eigenschaften, sondern auch die Darstellung der zyklischen I/O-Daten. Wenn nun solche Submodule mehrere Repräsentationen der Datenlänge zulassen, so sind für unterschiedliche Repräsentationen (Länge und Datenformat) auch unterschiedliche *SubmoduleIdentnumber* zu verwenden. Nur dadurch sind physikalisch identische Submodule innerhalb eines Moduls durch unterschiedliche Repräsentation der Daten auch eindeutig identifizierbar.

Die *VendorID, DeviceID* und die *ModuleIdentnumber* stellen eine 1:1 Beziehung dar (d.h. es gibt nie mehr als eine Identnummer für dieselbe Hardware). Ein Submodule hingegen kann für unterschiedliche Repräsentationen der I/O-Daten auch mehrere unterschiedliche *SubmoduleIdentnumber* haben.

Von diesen Regeln darf nur abgewichen werden, wenn

- existierende modulare Feldgeräte zu PROFINET IO Devices werden, da bei diesem Übergang nur das Kopfmodul getauscht werden muss.

- Gateways (Links) zu anderen Netzwerken eingesetzt werden, da normalerweise eine Abbildung der Feldgeräte des unterlagerten Netzwerkes in den PROFINET IO-Adressraum erfolgt.

Submodul-Definitionen können an unterschiedlichen Stellen innerhalb einer GSD-Datei auftauchen. Diese Stellen können sein:

- Interface- und PortSubmodule innerhalb eines DAPs in der *SystemDefinedSubmoduleList*

- PortSubmodule als sogenannte Module in der *SystemDefinedSubmoduleList,* oder *SubmoduleList (steckbare PortSubmodule)*

- „normale“ Module innerhalb der *VirtualSubmoduleList*

- „normale“ Submodule innerhalb der globalen *SubmoduleList*

All diese *IdentificationNumbers* können auf folgende Weise ausgelesen werden:

- RPCObject-UUID (Vendor- und DeviceID)
- I&M-Funktionen (Vendor- und DeviceID)
- Azyklisches Lesen mit RealIdentificationData (ModuleIdentNumber und SubmoduleIdentNumber)

Beispiel:

Nachfolgend sind drei Module definiert, die anhand ihrer ID (hier z.B. ID_Mod_01) referenziert sind. Die Definitionen erfolgen in zwei Schritten. Innerhalb der „*UseableModules*“ werden alle Module mit ihren eindeutigen IDs definiert.

1. Definition der Module:

```
<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 der Eigenschaften der Module innerhalb der ModuleList mit Referenz zur Module_ID. Innerhalb der VirtualSubmodule-Definition wird die eigentliche Datenlänge angegeben.

```
<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>
Hier folgen die weiteren Module
```


10.1 Definition von IRT-fähigen Modulen

Gemäß vorherigem Beispiel wird angenommen, dass das Modul mit der ID_03 IRT-fähig sein soll. Die Definition eines IRT-fähigen Moduls hat beispielhaft folgenden Aufbau.

Beispiel:

```
<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="0x0001" 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 Diagnose-Definitionen

Alarme werden bei PROFINET IO genutzt, um kritische Zustände aus dem Prozess oder dem Feldgerät in Form von Diagnosen an das Leitsystem melden zu können. Diese können

- Herstellerspezifisch
- Profilspezifisch oder
- Systemdefiniert sein.

Damit sie im Engineering-Tool lesbar angezeigt werden können, müssen sie in einer bestimmten Struktur in der GSD-Datei definiert sein. Die herstellerspezifischen und profilspezifischen Alarme müssen in der GSD-Datei sowohl vom Wertebereich als auch von den Texten beschrieben sein. Die systemspezifischen Alarme kann man im Engineering-Tool sowohl von der Struktur und den Texten als bekannt voraussetzen.

Die folgende Tabelle zeigt, welche Alarmtypen bei PROFINET in der GSD-Datei beschrieben werden können:

UserStructureIdentifier	ChannelErrorType	ExtChannelErrorType	GSDML entry
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 entry
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	–	–	–

Tabelle 1: Anwenderspezifische Diagnosen

Beispiel:
Bei der Definition der ***ChannelDiagList*** gibt der Hersteller innerhalb des Keywords ***ChannelDiagItem*** unter der Definition ***ErrorType*** die Fehlernummer an, die durch das Schlüsselwort ***TextId*** näher beschrieben wird. Die Definition beginnt auf derselben Ebene wie die DeviceAccessPointList. Es ist sicher sinnvoll, firmenintern oder zu mindest für ein Baugruppensystem alle möglichen Fehlernummern im Vorfeld einheitlich zu definieren, da die Anwendersoftware im PROFINET IO-Device dann einheitliche Fehlernummern verwenden kann.

```
<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>
```

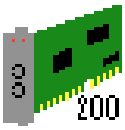
Dieser Parametrierfehler muss dann in der Sprachenliste für alle anderen verwendeten Sprachen ebenfalls definiert werden.

```
<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 Grafiksymbole

Beim Projektieren von PROFINET-Anlagen werden die Feldgeräte anhand von Grafiksymbolen dargestellt. Die Definition solcher Grafiksymbole könnte folgendermaßen aussehen. Sie sehen das Beispiel aus dem Development Kit für den ERTEC 200.

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



13 Texte in GSDML

13.1 Allgemeines

Die meisten Texte in der GSDML sind sprachabhängig definierbar. Zu diesem Zweck werden im Beschreibungsteil lediglich “Ids“ verwendet (bezeichnet als Attribut mit dem Namen "TextId") die auf die eigentlichen Texte in der "ExternalTextList" verweisen. Für die Texte selbst gibt es keine Längenbeschränkung, da in der Regel in den Engineering-Tools mit Hilfe von Scrollbars u.Ä. auf beliebig lange Texte reagiert werden kann. Trotzdem sei darauf hingewiesen, dass es vorkommen kann, dass bei "überlangen" Texten nicht der vollständige Text angezeigt wird.

Es wird deshalb empfohlen sich im Rahmen der folgenden Längendefinitionen zu bewegen:

Verwendung	Elementname	Empfohlene maximale Länge in Octets
Kategorienamen	CategoryItem	30
Parameternamen	ValueItem	30
Parametertexte	Assign	
Bezeichnung der zyklischen E/A Daten	DataItem	30
Hilfeinformationen	Help	1024
Herstellerspezifische Diagnose: Bezeichnung	ChannelDiagItem/Name	30
Herstellerspezifische Diagnose: Hilfetext	ChannelDiagItem/Text	1024
Bezeichnung eines Parametrierdatensatzes	ParameterRecordDataItem	30
Name einer Baugruppe	ModuleInfo/Name	30
Allgemeine Informationen zu einer Baugruppen	ModuleInfo/InfoText	250
Allgemeine Informationen zu einem IO-Device	DeviceIdentity/InfoText	250

Tabelle 2: Empfohlene Textlängen

13.2 Einbindung von Fremdsprachen

Alle sprachabhängigen Texte sind im Abschnitt "ExternalTextList" einer GSD Datei hinterlegt. Die ExternalTextList enthält mindestens den Abschnitt "PrimaryLanguage" der alle Texte in **englischer** Sprache enthalten muss.

Darüber hinaus ist eine beliebig hohe Anzahl weiterer Abschnitte "Language" möglich, in denen jeweils die Texte einer anderen Sprache hinterlegt werden. Die Sprache selbst ist durch das Attribut "xml:lang" definiert und folgt der ISO 639-1 Codes for the representation of names of languages – Part 1: Alpha-2 code.

Die Kodierungen für die gängigsten Sprachen lauten wie folgt:

Abkürzung	Sprache
de	Deutsch
es	Spanisch
fr	Französisch
it	Italienisch

Tabelle 3: Sprachcodierungen in der GSDML

Die Abschnitte "PrimaryLanguage" und "Language" selbst sind identisch strukturiert: Sie enthalten eine beliebige Anzahl von "Text" Einträgen bei denen das Attribut "TextId" zur Identifizierung dient und das Attribut "Value" den eigentlichen Text enthält.

Eine "ExternalTextList" mit zwei Sprachen kann also wie folgt aussehen:

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

Neben diesem Verfahren besteht noch die Möglichkeit, Texte in sogenannte "Satellitendateien" auszulagern. Damit kann eine Sprache ergänzt werden ohne dass eine (evtl. bereits zertifizierte) GSD Datei verändert werden muss.

Eine Satellitendatei hat folgenden Aufbau (ab GSDML V2.2):

```
<?xml version="1.0" encoding="UTF-8"?>
<ExternalTextDocument xml:lang="fr" xmlns="http://www.profibus.com/GSDML/2003/11/Primitives"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.profibus.com/GSDML/2003/11/Primitives ..\..\xsd\GSDML-Primitives-
v2.3.xsd">
    <Text TextId=" IDT_REF_WIREBREAK" Value="rupture de fil"/>
</ExternalTextDocument>
```

Für GSDML vor Version V2.2 ist der Namespace http://www.profibus.com/Common/2003/11/Primitives und das Schema Common-Primitives-v1.0.xsd zu verwenden.

Damit eine Satellitendatei vom Engineering-Tool gefunden werden kann muss sie bestimmten Namenskonventionen entsprechen. Vorgeschrieben ist, dass sich die Datei in einem Unterverzeichnis befindet, dessen Name der Sprachcode ist, und der gleiche Name wie bei der entsprechenden GSD Datei, ergänzt um "-Text-*Sprachcode*" verwendet wird.

Beispiel:

GSD-Dateiname lautet: **GSDML-V1.0-Siemens-ET200X-20030818.xml**

Satellitendatei in italienischer Sprache muss dann lauten:
it\GSDML-V1.0-Siemens-ET200X-20030818-Text-it.xml

14 Verwendung von Referenzen

In einer GSD werden zahlreiche Referenzen verwendet, um eine Mehrfachbeschreibung gleicher Daten zu vermeiden. Beispiel: Zwei Baugruppen unterstützen einen gleichen Parameter "Messart". Zu diesem Parameter sind neben der textuellen Erläuterungen noch zahlreiche Parameter in der GSD hinterlegt (z.B. Codierung). Mit Hilfe von Referenzen ist es jetzt möglich, den Parameter "Messart" an einer Stelle in der GSD zu beschreiben und von allen Baugruppen, die diesen Parameter unterstützen, zu referenzieren. Andernfalls müssten alle Texte in allen Sprachen für jede Baugruppe separat angegeben werden, was zum einen die Pflege der Datei erschwert und zum anderen die Größe einer GSD Datei deutlich steigern würde.

14.1 Überprüfung durch das Schema

Um Elemente in der GSD referenzieren zu können, werden IDs verwendet. In folgendem Beispiel wird die ID eines Moduls dargestellt:

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

Um dieses Modul zu referenzieren, reicht es aus die ID des Moduls anzugeben:

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

Eine Referenzierung funktioniert natürlich nur, wenn die IDs eindeutig gehalten werden. Dies wird bereits durch die Schemavalidierung überprüft und sichergestellt. Folgendes Beispiel zeigt die Fehlermeldung von XMLSpy:

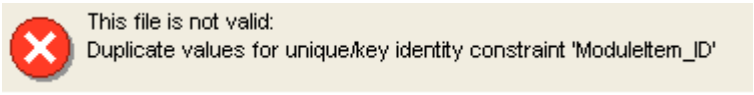


Abbildung 5: Fehlermeldung bei doppelten IDs

Außerdem wird geprüft, ob bei Referenzen eine gültige ID angegeben wurde:

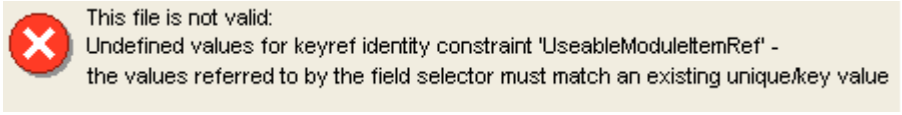


Abbildung 6: Fehlermeldung bei ungültiger Referenz

14.2 Tipps zur sinnvollen Bezeichnung der IDs

Eine GSD Datei kann sehr umfangreich werden. Entsprechend viele IDs sind zu vergeben und konsistent zu halten. Das GSDML-Schema selbst prüft bei der Vergabe der IDs nur, ob der String führende oder endende Blanks besitzt. Diese werden durch eine Schemavalidierung erkannt und abgelehnt. Ebenso sind in einem String keine Zeilenumbrüche erlaubt.

Ansonsten können beliebige Strings für die ID verwendet werden.

Um die Übersicht zu wahren empfiehlt es sich, die IDs mit einem typspezifischen Prefix beginnen zu lassen. Dies hat folgende Vorteile:

- Es ist aus dem String zu erkennen, welches Objekt die ID referenziert
- Die eindeutige Vergabe der ID wird erleichtert
- In einem Editor können die IDs sortiert in einer Liste angezeigt werden.

In folgender Tabelle ist eine Empfehlung für die Vergabe der ID-Prefixe enthalten:

Objekt	**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_

Tabelle 4: Prefixverwendung der IDs

15 Kataloginformation in der GSD

Die bisherige (schlüsselwortbasierte) GSD enthielt keine Informationen darüber, wie die Modulauswahl von modularen Feldgeräten strukturiert dargestellt werden können. Dies führte bei Feldgeräten mit hoher Anzahl von Modulvarianten dazu, dass sich die Auswahl des gewünschten Moduls sehr schwierig gestaltete, da das einzige Kriterium zur Unterscheidung der Modulname selbst war.

In der GSDML-Spezifikation wurde diese Problematik durch Einführung von Kategorien gelöst. Jedes Modul lässt sich einer Kategorie und einer Subkategorie zuordnen. Diese Zuordnung hat **keine** Auswirkung auf die Runtime-Eigenschaften des Moduls sondern dient nur zur verbesserten Darstellung der Modulauswahl, z.B. in Form eines Katalogs des Engineering-Tools.

Mit Hilfe einer Kategorie lassen sich technisch verwandte Module gruppieren. So können beispielsweise alle Analog-Eingabebaugruppen einer Kategorie "Analog Input" zugeordnet werden.

15.1 Kategorie Zuordnung

Jedes Modul (einschließlich dem DAP) besitzt in der GSD ein Pflichtelement "ModuleInfo". Neben weiteren Unterelementen enthält dieses zwei optionale Attribute "CategoryRef" und "Subcategory1Ref". Durch diese Attribute erfolgt die Zuordnung zu einer Haupt- und einer Unterkategorie. Der Inhalt der Attribute muss eine ID des Elements "CategoryItem" der globalen "CategoryList" enthalten.

Ein CategoryItem selbst ist sehr einfach aufgebaut – es enthält neben der ID zur Referenzierung nur noch eine TextId für einen (sprachabhängigen) Text.

Folgendes Beispiel verdeutlicht die Zusammenhänge:

```
<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 Darstellung der Kategorien

Wie Kategorien im Engineering-Tool dargestellt werden, bleibt dem Toolhersteller überlassen. Am Beispiel des Baugruppenkatalogs des Siemens Tools "HW-Konfig" lässt sich die Anwendung der Kategorien aber einfach verdeutlichen:

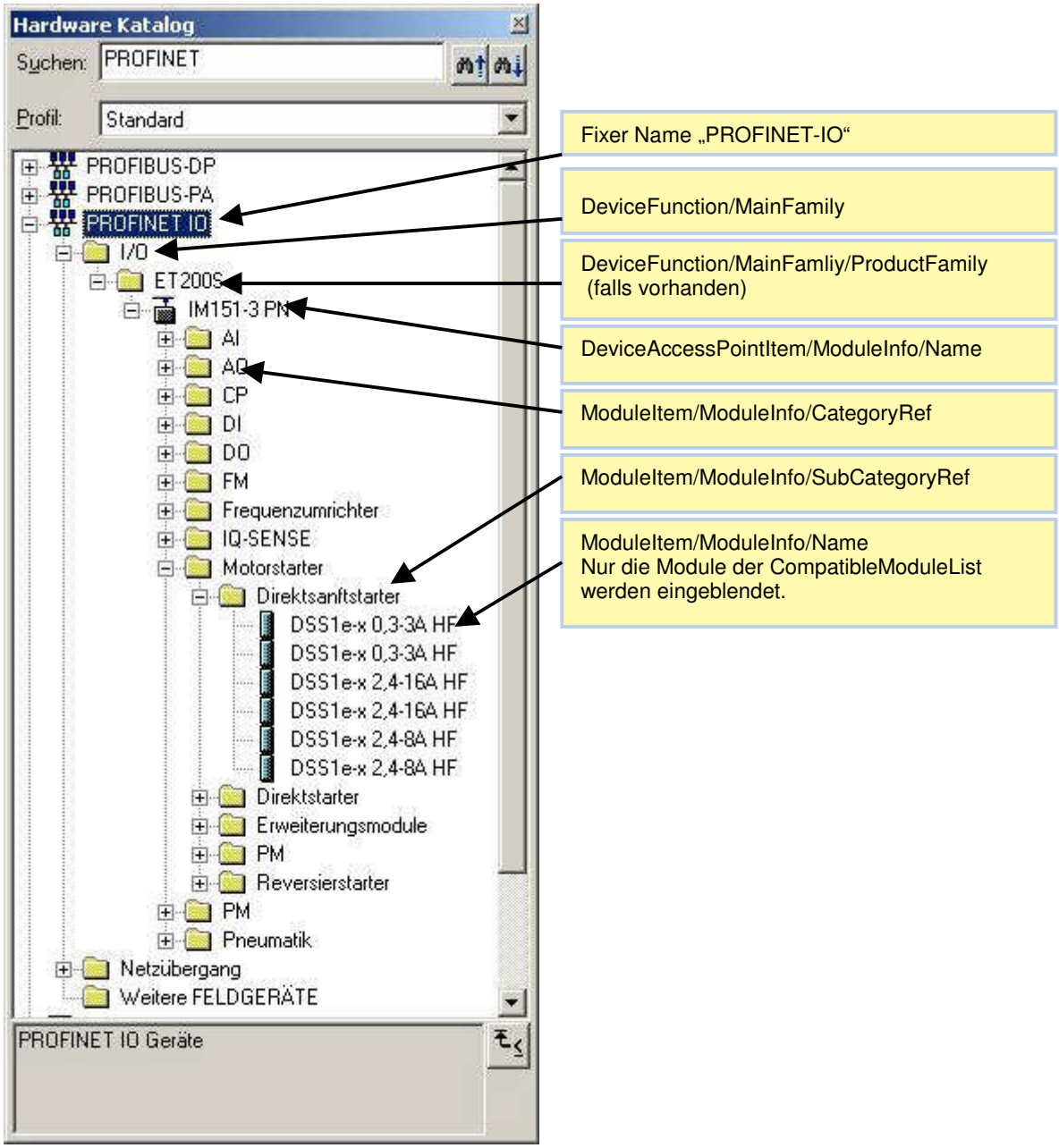


Abbildung 7: Katalogdarstellung am Beispiel STEP7

16 Beschreibung von ParameterRecordDataObjects

Die GSDML ermöglicht die Beschreibung von Parametern, die im Anlauf vom IO-Controller an ein Submodul des IO-Devices übertragen werden. Bei PROFINET IO werden diese Parameter in Form von RecordDataObjects (vergleichbar mit Datensätzen) abgelegt.

Zur Beschreibung dieser Daten wird in der GSDML das Element "ParameterRecordDataItem" verwendet, das optional innerhalb eines Submoduls zu finden ist.

Die einzelnen Parameter eines Submoduls können über mehrere RecordDataObjects verteilt sein. Jedes RecordDataObject ist (innerhalb eines Submoduls) eindeutig über das Attribut "Index" adressierbar.

Über das Attribut "TransferSequence" kann festgelegt werden, in welcher zeitlichen Reihenfolge die einzelnen ParameterRecordDataObjects übermittelt werden.

Weitere Eigenschaften eines ParameterRecordDataItems in der GSDML sind.

- Länge des RecordDataObjects in Octets (Attribut "Length")
- Name des RecordDataObjects (Childelement "Name"). Dieser wird nur für Anzeigezwecke im Engineering-Tool verwendet.
- Initialisierung der Datenstruktur (Childelement "Const")
- Definition der einzelnen Parameter (Childelement "Ref")

16.1 "Const" Definition eines ParameterRecordDataItems

Die Datenstruktur eines ParameterRecordDataItems kann mit Hilfe des "Const" Elements initialisiert – d.h. mit Defaultwerten gefüllt – werden.

Hierbei kann ein ParameterRecordDataItem beliebig viele „Const“ Definitionen enthalten.

Ein „Const“ Element besitzt zwei Attribute:

<Const ByteOffset="1" Data="0x01,0x00"/>
Das Attribut "Data" enthält die durch Komma getrennten Bytewerte des RecordDataObjects in Hexdarstellung (beginnend mit "0x").

Das optionale Attribut "ByteOffset" beschreibt, ab welchem Byte die folgende Bytes aus dem "Data" Attribut einzufügen sind. Fehlt dieses Attribut, wird von ByteOffset "0" ausgegangen.

Allgemein gilt die Regel dass die ByteOffset + die Anzahl der beschriebenen Bytes des „Data“ Elements die Länge des RecordDataObjects nicht übersteigen darf.

Folgendes Beispiel zeigt die Initialisierung eines 4 Byte langen RecordDataObjects mittels der „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

Durch Const Definition beschriebene Felder

Abbildung 8: Wirkung der Const Definition in einem Record Data Object

In diesem Beispiel sind die Werte des Bytes mit Offset "2" nicht durch die „Const"-Definition festgelegt. In diesem Fall ist durch die GSDML-Spezifikation festgelegt dass diese Werte mit "0" gefüllt werden.

16.2 "Ref" Definition eines ParameterRecordDataItems

Die einzelnen Parameter eines ParameterRecordDataItems werden mit Hilfe des "Ref"-Elements beschrieben. Dieses Element besitzt zahlreiche Attribute die die Eigenschaften des Parameters festlegen.

Anhand eines Beispiels soll die Anwendung des „Ref"-Elements verdeutlicht werden:

Oben beschriebenes *RecordDataObject* soll einen Parameter "Messart" enthalten der, mit Hilfe von zwei Bits codiert, folgende Werte annehmen können soll:

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

Das Objekt könnte wie folgt aussehen:

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

Abbildung 9: Wirkung der Ref Definition in einem Record Data Object

In der GSD würde dieser Parameter wie folgt beschrieben werden:

```
<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"/>

</ParameterRecordDataItem>


<ValueList>
    <ValueItem ID=" IDV_Messbereich">
        <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"/>
        <Text TextId="IDT_REF_Messbereich" Value="Messbereich"/>
        <Text TextId="IDT_ASSIGN_0-20mA " Value="Strom 0..20 mA"/>
        <Text TextId="IDT_ASSIGN_4-20mA " Value="Strom 4..20 mA"/>
        <Text TextId="IDT_ASSIGN_0-10V " Value="Spannung 0..10 Volt"/>
        <Text TextId="IDT_ASSIGN_+-10V " Value="Spannung + - 10 Volt"/>
    </PrimaryLanguage>
</ExternalTextList>
```

Folgende Besonderheiten sind zu beachten:

- Über "ValueItemTarget" kann jedem konkretem Wert ein Text zugewiesen werden. Dieser Text wird dem Anwender im Engineering-Tool als Auswahlkriterium angeboten. Fehlt *ValueItemTarget*, so wird das Tool diesen Parameter als reines Editierfeld anbieten d.h. in obigem Fall könnten lediglich die Werte "0" bis "3" eingegeben werden.

- Über das Attribut *AllowedValues* des „Ref“ Elements sind Einschränkungen des Wertebereichs möglich. Eine ähnliche Baugruppe, die lediglich in der Lage ist Strom zu messen, könnte dies über das Attribut "*AllowedValues*="0 1" einschränken.

- Fehlt sowohl "ValueItemTarget" als auch "AllowedValues" so ist der eingebbare Wertebereich nur durch den Datentyp beschränkt.

Eine mögliche Darstellung des Parameters im Engineering-Tool zeigt folgender Dialog:

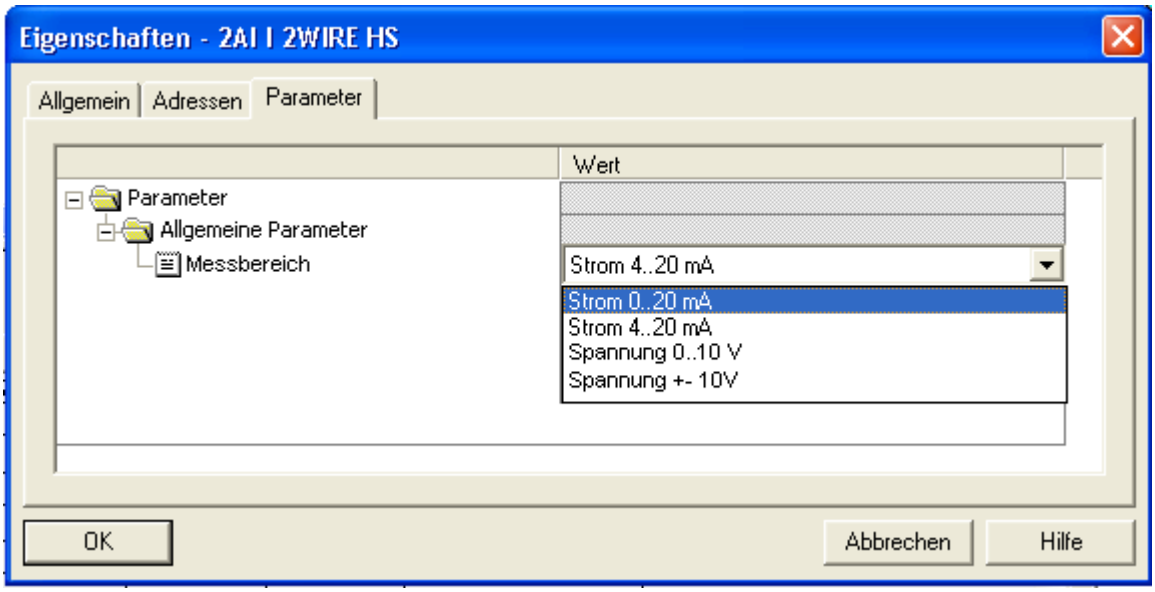


Abbildung 10: Darstellung eines Parameters im Engineering-Tool

17 SNMP und MIB2

SNMP steht für **S**imple **N**etwork **M**anagement **P**rotocol. Wenn ein Feldgerät (genauer gesagt der *DeviceAccessPoint*) das SNMP-Protokoll unterstützt, kann beispielsweise ein Engineering Tool damit die MIB2 und LLDP-Library auslesen. Diese Informationen sind sehr nützlich beim Feldgerätetausch oder zum Anzeigen der Anlagentopologie. Anschließend sehen Sie ein Beispiel wie eine Definition aussehen kann.

```
<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

Ethernet Teilnehmer benötigen nach Power On normalerweise mehrere Sekunden, um in den Datenaustausch zu gelangen. In einigen Bereichen der Automatisierungsttechnik ist aber ein wesentlich schnellerer Hochlauf eines Feldgerätes erwünscht. Diese Variante des Hochlaufs nennt man Fast Start Up (FSU). Hierbei wird das Feldgerät nach Spannungswiederkehr selbst aktiv und meldet sich mit dem DCP-Dienst DCP.Hello. Diese Definition ist DAP-spezifisch und ist deshalb innerhalb des Interface Submoduls angesiedelt. Zur Definition von FSU sind 3 Schritte notwendig:

1 PowerOnToCommReady innerhalb des DAPs

2.Die PDev-Definition (normalerweise SubslotItem SubslotNumber="32768") und

3.Die Definition dass DCP.Hello unterstützt wird DCP_HelloSupported="true"

Eine Definition des optionalen FSU-Dienstes könnte beispielsweise folgendermaßen aussehen:

1. PowerOnToCommReady innerhalb des DAPs

```
<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">  <!--Zeit in ms bis zum ersten Datenaustausch -->
```

2.Die PDev-Definition (normalerweise SubslotItem SubslotNumber="32768") und das zugehörige 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.Die Definition dass DCP.Hello unterstützt wird (DCP_HelloSupported="true")

Im entsprechenden Subslot des PDev ist folgendes anzugeben:

```
<InterfaceSubmoduleItem SubslotNumber="32768" SubmoduleIdentNumber="0x0002"
    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 Medien-Redundanz

Die Medien-Redundanz besteht aus einem **M**edia **R**edundancy **M**anager und einem **M**edia **R**edundancy **C**lient. Gewöhnlich übernimmt der IO-Controller auch die Funktion des „Managers“, so dass im folgenden Beispiel innerhalb des InterfaceSubmodules nur die „Client-Funktionalität“ definiert ist:

```
<InterfaceSubmoduleItem ID="IDS_3I" 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="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" /> <! MRP-Client Definition.>

</InterfaceSubmoduleItem>
```


20 PROFIsafe-Definitionen in der GSD-Datei

I/O-Module, die PROFIsafe-Funktionalität unterstützen, müssen in der GSD-Datei innerhalb der *VirtualSubmoduleList* entsprechend definiert werden. Nachfolgend sehen Sie beispielhaft, wie solche Definitionen aussehen können:

```
<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 Kompatibilität zwischen verschiedenen GSDML-Versionen

Wie auch bei PROFIBUS wirken sich Weiterentwicklungen der PROFINET-Technologie meist auf die GSD aus. Diese werden durch eine neuere Schemaversion der GSDML ausgedrückt. Bei der Weiterentwicklung der GSDML wird darauf geachtet, dass bestehende GSD-Dateien stets mit einer neueren Schemaversion kompatibel sind. (Aufwärtskompatibilität)

Die andere Richtung („Neuere“ GSD trifft auf „älteres“ GSDML-Schema) bedarf allerdings einer genaueren Betrachtung:

Sobald in der GSD Strukturen verwendet werden, die im vorliegenden Schema noch nicht definiert sind, wird eine Schemavalidierung Fehler melden. Dies hätte die Konsequenz, dass entwender immer die neuesten Engineering-Tools eingesetzt werden müssen (die eine neue Schemaversion „kennen“) oder der Feldgerätehersteller mehrere Versionen der GSD weiterentwickelt und pflegt. Beide Varianten sind aber nicht erwünscht.

Eine Alternative wäre es, auf die Schemavalidierung in den Tools zu verzichten und alle unbekannten GSD Strukturen im GSD Interpreter zu ignorieren. Dies hätte aber den Nachteil, dass dadurch die korrekte Funktion des Feldgeräts nicht gewährleistet werden kann da es für das Feldgerät eventuell notwendig ist, die neu in der GSD beschriebenen Daten auch zu erhalten.

Aus diesen Gründen wurde in GSDML-Version 2.0 das Attribut „RequiredSchemaVersion“ eingeführt mit dem der Feldgerätehersteller steuern kann, welche GSD-Strukturen von einem Tool überlesen werden dürfen ohne dass dies zu einem Fehlerverhalten führt.

Die Wirkungsweise ist wie folgt: Über dieses Attribut kann auf Ebene eines einzelnen Moduls oder DAP angegeben werden, welche Schemaversion ein Engineering-Tool mindestens beherrschen muss damit ein sinnvoller Betrieb des Moduls möglich ist. Folgendes Beispiel zeigt den Eintrag in einer GSD der Schemaversion 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>
   </ VirtualSubmoduleList >
</ VirtualSubmoduleList > .
[..]
</ModuleItem>
```

Hier ist beim Modul angegeben, dass dieses auch mit der Version 1.0 der GSDML betreibbar ist. In dem Beispiel besteht die Erweiterung gegenüber der Version 1.0 aus der Beschreibung der Bitstruktur der Ausgangsdaten (Element BitDataItem). Da diese Beschreibung keine Auswirkung auf die Funktion des Moduls selbst hat, können diese Elemente überlesen werden.

Ein Engineering-Tool dass nur eine GSDML-Version 1.0 beherrscht wird also folgende Daten lesen:

…

```
<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>

  </ VirtualSubmoduleList >

</ VirtualSubmoduleList > .

[..]
</ModuleItem>
```

In anderen Fällen ist es aber aus Sicht des Feldgerätes notwendig, die in der GSD beschriebenen Parameter zu erhalten. Ein Beispiel hierfür kann z.B. ein Feldgerät sein, das sinnvoll nur in einer taktsynchronen Umgebung eingesetzt werden kann. Da in der GSDML V1.0 hierzu keine Beschreibungsmittel zur Verfügung stehen macht es keinen Sinn, dieses Feldgerät mit einem Tool zu projektieren, das lediglich eine Version 1.0 der GSDML interpretieren kann.

Aus Sicht des Engineering-Tools werden Module oder DAPs, die als „RequiredSchemaVersion“ eine nicht unterstützte Version eingetragen haben, nicht zur Konfiguration angeboten.

22 Tools

22.1 PROFINET XML Viewer

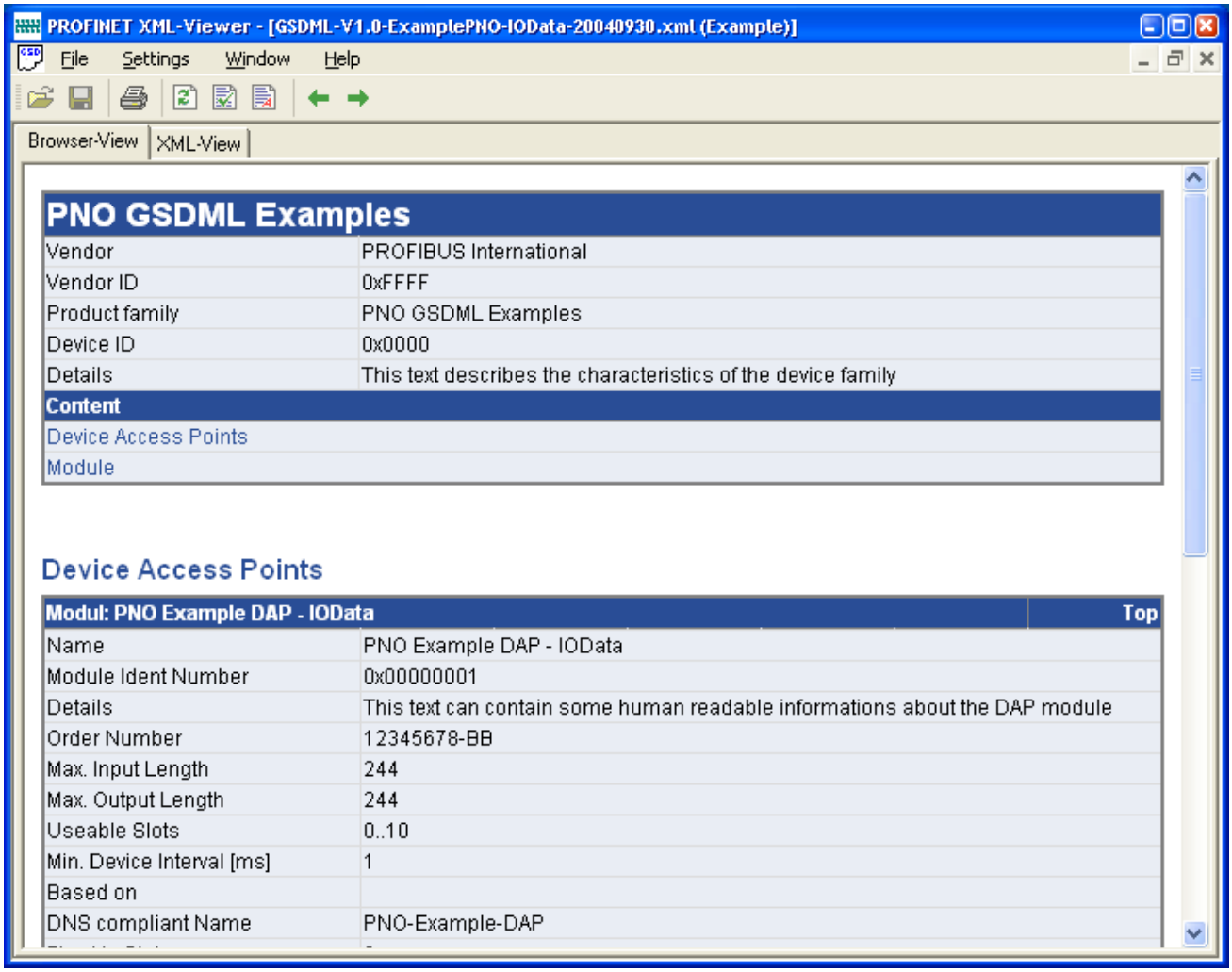
Dieses Tool wird kostenlos von der PNO bereitgestellt und bietet im Wesentlichen zwei Funktionen:

- Anzeige des Inhalts einer GSD in übersichtlicher HTML Darstellung
- Validierung einer GSD gegen die Schemadateien. GSD Dateien können darüber hinaus noch mit dem „GSD Checker“ geprüft werden, der über die im Schema hinterlegten Regeln weitere Prüfungen durchführt und dafür sorgt, dass der Inhalt korrekt gestaltet ist.

Darüber hinaus stellt der Viewer noch die Dokumentation zur GSD bereit und ermöglicht die Einbindung eines beliebigen XML Editors zur Erstellung von GSD Dateien.

Da gleichzeitig einige Beispieldateien mitinstalliert werden, bietet das Tool eine gute Basis zur Erstellung neuer GSD Dateien.

Folgendes Bild zeigt einen Screenshot des Tools:



22.1.1 Systemvoraussetzungen

Der PROFINET XML Viewer 2.3 ist eine .NET basierte Anwendung und erfordert deshalb das Mircosoft .NET Framework V2.0 Bei der Installation des PROFINET XML-Viewers wird geprüft, ob dies bereits auf Ihrem PC vorhanden ist und falls nicht werden Sie auf den Downloadbereich von Microsoft weitergeleitet.

Aus lizenzrechtlichen Gründen wird der Xerces Parser der Apache Software Foundation nicht automatisch mitinstalliert. Wie dieser Parser eingebunden wird ist im Kapitel 22.1.7 „Einstellungen“ beschrieben.

22.1.2 Darstellungsform

Der PROFINET XML Viewer bietet zwei unterschiedliche Darstellungsformen an:

In der XML-Ansicht „XML-View“ wird eine GSD Datei so angezeigt, wie sie auch durch Öffnen des Internet Exploreres erscheinen würde und entspricht im Wesentlichen der Darstellungsweise eines ASCII Editors.

In der HTML-Ansicht „Browser-View“ werden die Inhalte einer GSD in einer übersichtlichen Tabellenform dargestellt. Da gleichzeitig die Textreferenzen in Texte umgewandelt werden, ist die Lesbarkeit deutlich höher als in der XML Darstellung. Darüber hinaus kann damit bei selbst erstellten GSD Dateien sehr schnell geprüft werden, ob die Textreferenzen korrekt gesetzt sind.

Die Umschaltung zwischen den beiden Darstellungsformen erfolgt durch Klick auf die jeweilige Lasche im oberen Bereich des Fensters.

22.1.3 Beispieldateien

Bei der Installation des Tools werden automatisch einige Beispieldateien mitinstalliert. Sie können diese Dateien über den Menüpunkt „File Open GSD Example..“ öffnen und als Basis für eine selbst erstellte GSD Datei verwenden.

22.1.4 Kopie der angezeigten Datei speichern

Diese Funktion – aufgerufen über den Menüpunkt „Save As…“- ermöglicht es, eine Kopie der angezeigten GSD Datei zu erstellen. Hierbei wird geprüft, ob im Zielordner die zur Validierung notwendigen Schemadateien vorliegen. Ist dies nicht der Fall, werden die Schemadateien automatisch ins Unterverzeichnis „..\xsd“ kopiert, so dass anschließend eine Validierung mit einem Standard XML Editor möglich ist ohne dass Sie sich um die Schemadateien kümmern müssen.

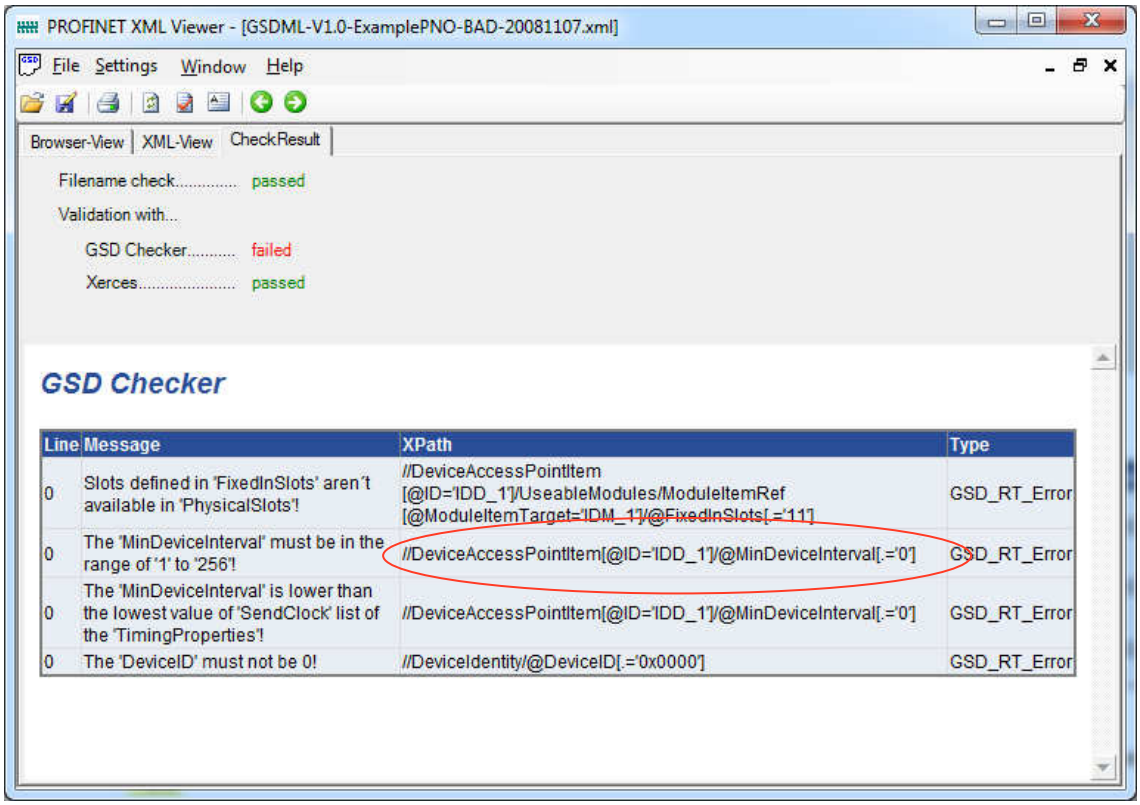
22.1.5 Editor einbinden

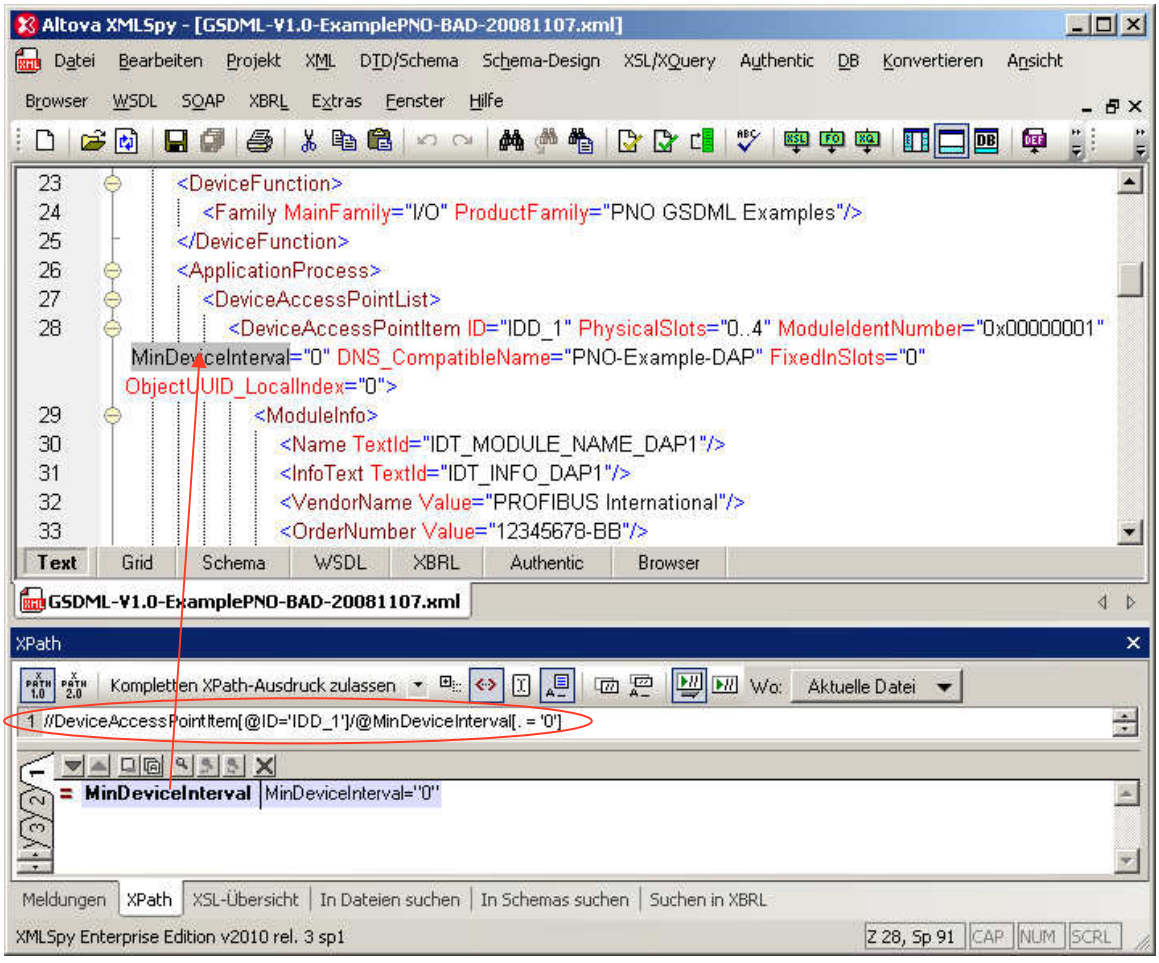
Aus dem PROFINET XML Editor können Sie einen beliebigen Editor starten. Welcher Editor gestartet werden soll, kann unter „Settings“ eingestellt werden. Haben Sie noch keine Wahl getroffen wird der Windows-Editor „Notepad“ verwendet. Nachdem Sie Änderungen im Editor durchgeführt haben, können Sie mit Hilfe der „Reload“ Funktion die Datei im XML-Viewer neu laden so dass Sie die Änderungen sehen und neu prüfen können.

22.1.6 GSD Datei prüfen

Der PROFINET XML Viewer prüft mit Hilfe von drei gängigen XML Parsern, ob der Inhalt der GSD Datei korrekt aufgebaut ist. Darüber hinaus ist für GSD Dateien ein separater Checker eingebaut, der weitere Prüfungen durchführt, deren Regeln nicht im Schema beschrieben werden können.

Treten Fehler auf, so wird bei den XML Parsern die jeweilige Zeilennummer des Fehlerorts angezeigt. Der GSD Checker zeigt stattdessen den exakten Fehlerort in Form eines XPath Ausdrucks an. XPath ist eine standardisierte Form, um einzelne Elemente in einer XML Datei adressieren zu können. Sie können diesen XPath Ausdruck verwenden um in einem XML Editor zum Fehlerort zu springen. Der XML-Editor „XMLSpy“ bietet dies z.B. mit Hilfe der Funktion „Evaluate XPath“ an:



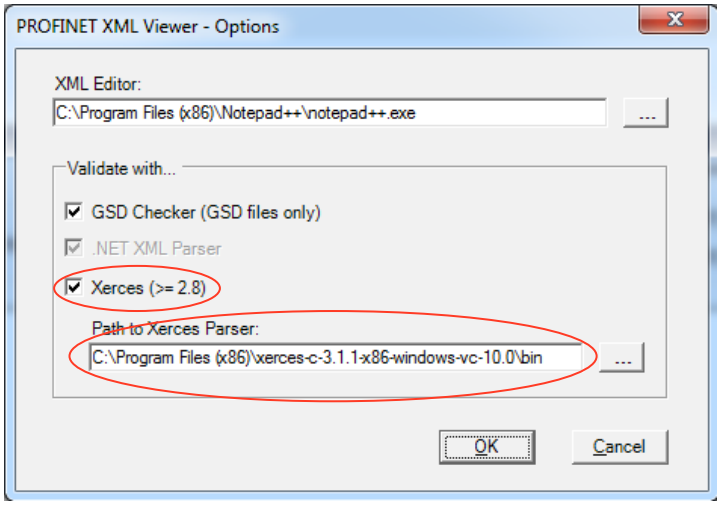


22.1.7 Einstellungen

Über den Dialog „Einstellungen“ können Sie festlegen, welcher Editor verwendet werden soll und welche XML Parser zur Validierung herangezogen werden sollen.

Aus lizenzrechtlichen Gründen wird der Xerces Parser der Apache Software Foundation nicht automatisch mitinstalliert. Sie können dies aber sehr einfach selbst durchführen. Hierzu laden Sie die Software von der Webseite der Apache Software Foundation http://xml.apache.org/xerces-c/.

Nach Entpacken des ZIP Files reicht es nun aus das so entstandene „bin“ Verzeichnis im Feld „Path to Xerces Parser“ einzutragen.



22.1.8 Dokumentation

Die GSDML-Dokumenation wird ebenfalls automatisch mit installiert und kann über den Menüpunkt „Help – GSMDL Dokumenation“ aufgerufen werden.

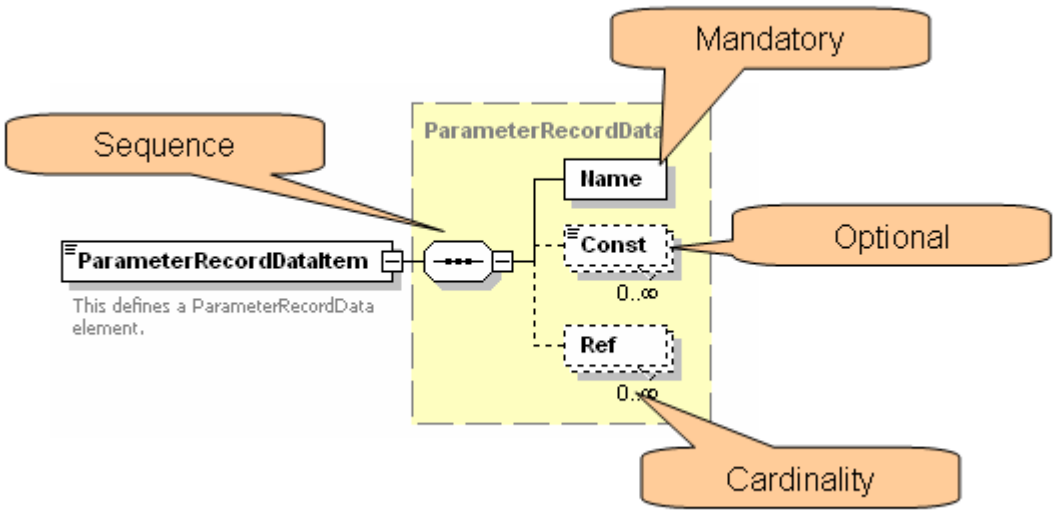
Hierbei wird die von XMLSpy benutzte Schemanotation angewendet:

Eine einzuhaltende Reihenfolge wird dabei durch das „Sequence“ Symbol ausgedrückt (siehe nachfolgendes Bild).

Elemente, die enthalten sein müssen (Mandatory) sind als umrandetes Kästchen gezeichnet.

Ist ein Element optional, so ist das Kästchen gestrichelt umrandet.

Unterhalb eines Elements kann noch hinterlegt sein, wie häufig es angewendet werden kann. Im unteren Beispiel ist das „Ref“ Element optional und kann bebliebig häufig angewendet werden.



22.2 Validierende XML Parser

Die XSD Dateien für GSDML benutzen zahlreiche Features der XML-Schemasprache. Ein XML-Parser muss in der Lage sein, diese verwendeten Konstrukte zu interpretieren.

Folgende Freeware Parser können zur Validierung einer GSD Datei verwendet werden. (Die Liste erhebt keinen Anspruch auf Vollständigkeit!)

MSXML 6.0

Zu beziehen unter

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

Bitte nutzen Sie Microsoft Update um die neuesten Sicherheitsupdates zu erhalten.

XmlReader Klasse von Microsoft .NET Framework Version 2.0

mit eingeschalteter Schemavalidierung, d.h. mit Settings.ValidationType = ValidationType.Schema.

Zu beziehen unter

- 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

Bitte nutzen Sie Microsoft Update um die neuesten Sicherheitsupdates zu erhalten.

Xerces XML- Parser

Zu beziehen unter

http://xerces.apache.org/xerces-c/

Altova AltovaXML Community Edition

Zu beziehen unter

http://www.altova.com/de/altovaxml.html

22.3 Tools zur GSD Erstellung

Eine GSD Datei ist eine XML-Datei und lässt sich damit mit einem einfachen Texteditor erstellen. Allerdings wird hierbei keine Unterstützung bei der Eingabe geboten. Somit ist die Gefahr groß, eine nicht wohlgeformte oder gemäß einem Schema ungültige Datei zu erstellen.

Durch die Verwendung von XML können zahlreiche Tools zur Erstellung einer GSD verwendet werden. Zu unterscheiden sind hier im Wesentlichen zwei Gruppen:

Validierende XML-Parser überprüfen, ob ein vorliegendes XML-Dokument gemäß den Regeln eines Schemas aufgebaut ist.

XML-Editoren bieten eine Unterstützung beim Editieren eines XML-Dokuments und besitzen in der Regel einen integrierten XML-Parser, oder einen Schnittstelle externen XML-Parser.

Zur Erstellung von XML-Dokumenten gibt es eine unüberschaubar große Anzahl von Editoren. Zur Erstellung der GSDML-Schemadateien selbst wurde das Tool "XMLSpy" verwendet, mit dem auch sehr einfach GSD Dateien editiert werden können. XMLSpy besitzt einen "build-In" XML-Parser, mit dem eine GSD Datei validiert werden kann.

Zu beziehen ist der XMLSpy unter

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

Weitere XML-Editoren:

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

