Datenformat TRuDI und PrüDi (AR 2418-6)

Arbeitskreis Bundesdisplay

Autor:

Arbeitskreis Bundesdisplay

IVU Softwareentwicklung GmbH

Versionshistorie

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Name | Änderungen |
| 0.1 | 2018-07-27 | Thomas Müller (IVU) | Dokument erstellt, Änderungen für PRüDI aufgenommen |
|  |  |  |  |

Tabelle 1: Versionshistorie

Offene Punkte:

* Cosem-Datenmodel: season\_profiles.season\_start 🡪 TYPE\_date\_var: ist ggf. nicht abbildbar auf dateTime (wurde geändert in Version 0.98)

Inhalt

[1 Regeln zur Anwendung des Datenmodells 4](#_Toc520456392)

[1.1 Basisdaten 5](#_Toc520456393)

[1.2 Prüfungsdaten 15](#_Toc520456394)

# Regeln zur Anwendung des Datenmodells

Im Folgenden sind die Regeln zur Anwendung des VDE AR 2418-6 Datenmodells (AR\_2418-6.xsd v1.2) beschrieben. Diese werden ausgehend von der zentralen Klasse UsagePoint aufgeteilt in Basisdaten und Prüfungsdaten.

Zu unterscheiden ist in den Tabellen in Kapitel 1.1 und 1.2 zwischen Angaben zu Klassen und zu Attributen. Attribute, die als „Erforderlich“ gekennzeichnet sind, deren Klasse aber als „Optional“ gekennzeichnet ist, sind erforderlich sofern die Klasse in der entsprechenden Nachricht genutzt wird. Wird die Klasse nicht genutzt, so finden die Attribute keine Anwendung. Es ist ebenfalls zu unterscheiden, für welchen Anwendungsfall das Datenformat verwendet wird. Bei der Verwendung für die Schnittstelle „IF\_Adapter\_TRuDI“ sind andere Attribute/Klassen erforderlich als an der Schnittstelle „IF\_Lieferant\_TruDi“.

Zur Aufrechterhaltung der (Abwärts-)Komptabilität zwischen dem ESPI REQ.21 (Green Button) und dem Datenmodell dieser VDE-Anwendungsregel, sind in dem Datenmodell einige Elemente als optional gekennzeichnet, die im Folgenden als „Erforderlich“ beschrieben werden. Führend sind hier die Angaben in den nachfolgenden Tabellen.

Regelungen des Green Button Datenmodells, wie zum Beispiel der Festlegung von Referenzierungen über den Atom Syndication Format oder Inhaltsvorgaben für einzelne Datenelemente, wurden soweit es für Kompatibilitätszwecke notwendig ist, übernommen.

Alle Zeitangaben erfolgen in xs:dateTime (RFC 3339) mit zwingender Angabe der Zeitzone. Dies erfolgt abweichend zur „Green Button“-Lösung aus Kompatibilität zur Klassendefinition zu BSI TR-03109 nach COSEM (Stand 28.10.2016).

## Basisdaten

Datenelemente, deren Verwendung mit „Erforderlich\*“ beschrieben ist, sind erforderlich, sofern die Daten für eine eichrechtlich relevante Überprüfung übermittelt werden (Abrechnungsdaten).

| **Nr.** | **Beschreibung** | **Verwendung bei IF\_Adapter\_ TRruDI** | **Verwendung bei IF\_Lieferant\_TRuDI** | **Realisierung für TRuDI v1.0** |
| --- | --- | --- | --- | --- |
| 1.01 | **UsagePoint**  Die Klasse *UsagePoint* repräsentiert den Zählpunkt und stellt das zentrale Datenelement einer Nachricht dar. Jede Nachricht **muss** mindestens einen Zählpunkt beinhalten.  Eine Instanz der Klasse UsagePoint:   * **muss** auf eine Instanz der Klasse *InvoicingParty* verweisen * IF\_Adapter\_TRuDI: Usagepoint **muss** auf eine Instanz der Klasse *Customer* verweisen * IF\_Lieferant\_TRuDI: Usagepoint **kann** auf eine Instanz der Klasse *Customer* verweisen * **muss** auf eine Instanz der Klasse *SMGW* verweisen * **muss** auf eine Instanz der Klasse *ServiceCategory* verweisen * **muss** auf mindestens eine Instanz der Klasse *MeterReading* verweisen * **kann** auf Instanzen der Klasse *LogEntry* verweisen * **kann** auf Instanzen der Klasse *MathOperation* verweisen * **kann** auf eine Instanz der Klasse *AnalysisProfile* verweisen | Erforderlich | Erforderlich | muss implementiert werden |
| 1.02 | **UsagePoint – roleFlags**  Das Datenelement *roleFlags* spezifiziert die Rolle eines Zählpunkts. Die Werte des Datenelements sind nach ESPI REQ.21 spezifiziert. Das Datenelement roleFlags ist **optional.** | Optional | Optional | keine zwingende Implementierung |
| 1.03 | **UsagePoint – status**  Das Datenelement *status* beschreibt den aktuellen Status des Zählpunktes. Nach ESPI REQ.21 kann der Status die Werte 0 (off) oder 1 (on) annehmen. Das Datenelement *status* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.04 | **UsagePoint – usagePointId**  Die *usagePointId* entspricht der Zählpunktbezeichnung nach dem aktuellen MeteringCode. Eine Instanz der Klasse *UsagePoint* **muss** genau eine Zählpunktbezeichnung enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.05 | **UsagePoint – tariffName**  Das Datenelement *tariffName* beinhaltet eine Identifikation des Tarifs.  Jede Instanz der Klasse *UsagePoint* **muss** ein Datenelement vom Typ *tariffName* beinhalten. | Optional | Optional | muss implementiert werden |
| 1.06 | **ServiceCategory**  Die Klasse *ServiceCategory* repräsentiert nach ESPI REQ.21 die Sparte des Produkts (Service), welches dem Letztverbraucher am Zählpunkt zur Verfügung gestellt wird.  Die Klasse ServiceCategory enthält keine Verweise auf weitere Klassen. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.07 | **ServiceCategory – kind**  *kind* beschreibt als Datenelement die konkrete Sparte des Zählpunktes.  Gültige Werte nach ESPI REQ.21 sind:  0 – electricity (Elektrizität)  1 – gas (Gas)  2 – water (Wasser)  4 – pressure (Druck)  5 – heat (Wärme)  6 – cold (Kälte)  7 – communication (Kommunikation)  8 – time (Zeit)  Eine Instanz der Klasse *ServiceCategory* **muss** genau einen der genannten Werte für das Datenelement *kind* beinhalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.08 | **MeterReading**  Die Klasse *MeterReading* repräsentiert den Kopf einer Messwertliste. Die Klasse enthält untergeordnet die entsprechenden Werte und Zusatzinformationen einer Messwertliste. Eine Instanz der Klasse *MeterReading*:   * **Muss** auf eine Instanz der Klasse *ReadingType* verweisen * **Muss** auf mindestens eine Instanz der Klasse *IntervalBlock* verweisen * **Muss** auf mindestens eine Instanz der Klasse *Meter* verweisen * **Kann** auf eine Instanz der Klasse *CO2Indicator* verweisen | Erforderlich | Optional | muss implementiert werden |
| 1.09 | **MeterReading – meterReadingId**  Die *meterReadingId* identifiziert eine Messwertliste eindeutig. Die ID kann zum Beispiel aus der Zählpunktbezeichnung, der Gerätenummer und der OBIS-Kennziffer zusammengesetzt werden.  Eine Instanz der Klasse *MeterReading* muss ein Datenelement vom Typ *meterReadingId* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.10 | **CO2Indicator**  Die Klasse *CO2Indicator* kann genutzt werden, um einer Messwertliste Informationen über damit verbundene CO2-Werte zu übermitteln. Die Nutzung der Klasse *CO2Indicator* ist **optional**.  Die Klasse *CO2Indicator* verweist auf keine weiteren Klassen. | Optional | Optional | keine zwingende Implementierung |
| 1.11 | **CO2Indicator – powerOfTenMultiplier**  Das Datenelement *powerOfTenMultiplier* repräsentiert den Einheitenvorsatz des übermittelten CO2-Wertes. Gültige Werte nach ESPI REQ.21 sind:  0 = None  1 = deca=x10  2 = hecto=x100  –3 = mili=x10–3  3 = kilo=x1000  6 = Mega=x106  –6 = micro=x10–3  9 = Giga=x109  Bei Instanziierung der (optionalen) Klasse CO2Indicator **muss** das Datenelement *powerofTenMultiplier* mit einem entsprechenden Wert gefüllt werden. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 1.12 | **CO2Indicator – uoi**  Das Datenelement *uoi* repräsentiert die Maßeinheit des übermittelten CO2-Wertes. Gültige Werte entsprechen ESPI REQ.21. Eigene Maßeinheiten können hier ergänzt werden.  Bei Instanziierung der (optionalen) Klasse CO2Indicator **muss** das Datenelement *uoi* mit einem entsprechenden Wert gefüllt werden. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 1.13 | **CO2Indicator – value**  Das Datenelement *value* repräsentiert den eigentlichen CO2-Wert, dessen Maßeinheit durch die weiteren Datenelemente der Klasse bestimmt wird.  Bei Instanziierung der (optionalen) Klasse CO2Indicator **muss** das Datenelement *value* mit einem ent­sprechenden ganzzahligen Wert gefüllt werden. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 1.14 | **ReadingType**  Die Klasse *ReadingType* spezifiziert die Inhalte einer Messwertliste.  Jede Messwertliste **muss** eine Instanz der Klasse *ReadingType* beinhalten.  Die Klasse *readingType* verweist auf keine weiteren Klassen. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.15 | **ReadingType – accumulationBehaviour**  Das Datenelement *accumulationBehaviour* beschreibt, wie die Werte über die Zeit kumuliert werden. Die Liste der gültigen Werte ist nach ESPI REQ.21 im *AccumulationBehaviourType* definiert. Beispiele sind:  0 = nicht anwendbar  3 = kumuliert  Die Nutzung des Datenelements *accumulationBehaviour* ist optional. Eine Verwendung sollte nur bei auf­summierten Werten in der Messwertliste stattfinden. | Optional | Optional | keine zwingende Implementierung |
| 1.16 | **ReadingType – commodity**  Das Datenelement *commodity* beschreibt das gelieferte Produkt genauer. Gültige Werte nach ESP REQ.21 sind:  0 = Not Applicable (nicht anwendbar)  1 = Electricity metered value (Elektrizität)  4 = Air (Luft)  7 = NaturalGas (Erdgas)  8 = Propane (PropanGas)  9 = PotableWater (Trinkwasser)  10 = Steam (Dampf)  11 = WasteWater (Abwasser)  12 = HeatingFluid (Wärmflüssigkeit)  13 = CoolingFluid (Kühlflüssigkeit)  Die Nutzung des Datenelements *commodity* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.17 | **ReadingType – consumptionTier**  Code für die Stufe des Verbrauchs, welcher mit der Messung verbunden wird Beispiele nach ESPI REQ.21 sind:  0 = Nicht anwendbar  1 = Block Tier 1  2 = Block Tier 2  Das Datenelement *consumptionTier* wird im Sinne dieser VDE-Anwendungsregel nicht genutzt. | Nicht genutzt | Nicht genutzt | keine zwingende Implementierung |
| 1.18 | **ReadingType – currency**  Das Datenelement *currency* codiert die Währung, welche mit den Kostenangaben der einzelnen Mess­werte verbunden ist. Beispiele für gültige Werte nach ESPI REQ.21 sind:  0 – Not Applicable  840 – US Dollar  978 – Euro  Die Nutzung des Datenelements *currency* ist **optional**. Bei Nutzung des Datenelements *cost* in der Klasse *IntervalReading* wird die Nutzung des Datenelements *currency* empfohlen | Optional | Optional | keine zwingende Implementierung |
| 1.19 | **ReadingType – dataQualifier**  Das Datenelement dataQualifier codiert bestimmte Eigenschaften der Messwerte der Messwertliste. Gültige Werte nach ESPI REQ.21 sind:  Valid values include:  0 = Not Applicable  2 = Durchschnitt  8 = Maximum  9 = Minimum  12 = Normal  Die Nutzung des Datenelements *dataQualifier* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.20 | **ReadingType – defaultQuality**  Das Datenelement *defaultQuality* beschreibt den Standardwert für den Messwertestatus. Dieser kann genutzt werden, falls für einzelne Messwerte in der Klasse *ReadingQuality* das Datenelement *quality* nicht gefüllt wurde. Gültige Werte nach ESPI REQ.21 sind:  0 – valid (gültig)  7 – manually edited (manuell geändert)  8 – estimated using reference day (geschätzt basierend auf einem Referenztag)  9 – estimated using linear interpolation (geschätzt mittels linearer Interpolation)  10 – questionable (unplausibel)  11 – derived (abgeleited)  12 – projected (forecast) (hochgerechnet (Prognose))  13 – mixed (gemischt)  14 – raw (roh/unverarbeitet)  15 – normalized for weather (normalisiert im Bezug zum Wetter)  16 – andere (other)  17 – validated (gültig)  18 – verified (überprüft)  Die Nutzung des Datenelements *defaultQuality* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.21 | **ReadingType – flowDirection**  Das Datenelement flowDirection beschreibt die Fließrichtung, die mit den Messwerte verbunden ist. Gültige Werte nach ESPI REQ.21 sind:  0 = nicht anwendbar  1 = Vorwärts  19 = Rückwärts  Die Nutzung des Datenelements flowDirection ist **optional.** | Optional | Optional | keine zwingende Implementierung |
| 1.22 | **ReadingType – intervalLength**  Das Datenelement intervalLength beschreibt die Standard-Intervalllänge die für die einzelnen Messwerte zugrunde liegt. Die Intervalllänge wird nach ESPI REQ.21 in Sekunden spezifiziert.  Ist kein Datenelement intervalLength in einer Instanz der Klasse *ReadingType* beschrieben, so wird von einer Standard-Intervallänge von 15min ausgegangen. Im Falle einer originären Messwertliste muss dieser Wert der Registrierperiode entsprechen. | Optional | Optional | keine zwingende Implementierung |
| 1.23 | **ReadingType – kind**  Das Datenelement *kind* codiert eine generelle Klassifikation der in der Messwertliste beinhalteten Werte dar. Beispiele nach ESPI REQ.21 sind:  0 = Not Applicable (nicht anwendbar)  3 = Currency (Währung)  4 = Current (Stromstärke, Ampere)  5 = CurrentAngle (Strömungswinkel)  7 = Date (Datum)  8 = Demand (Abruf)  12 = Energy (Energie)  15 = Frequency (Frequenz)  37 = Power (Leistung)  38 = PowerFactor (Leistungsfaktor)  40 = QuantityPower (Leistungsmenge)  54 = Voltage (Spannung)  55 = VoltageAngle (Spannungswinkel)  64 = DistortionPowerFactor (Klirrfaktor)  155 = VolumetricFlow (Volumenstrom)  Die Nutzung des Datenelements *kind* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.24 | **ReadingType – phase**  Das Datenelement *phase* codiert die Phase, die mit den Messwerten der Messwertliste in Verbindung steht. Beispiele nach ESPI REQ.21 sind:  0 = Not Applicable (nicht anwendbar)  129 = Phase AN  128 = Phase A  132 = Phase AB  Die Nutzung des Datenelements *phase* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.25 | **ReadingType – powerOfTenMultiplier**  Das Datenelement *powerOfTenMultiplier* repräsentiert den Einheitenvorsatz der Maßeinheit der in der Messwertliste übermittelten Werte. Gültige Werte nach ESPI REQ.21 sind:  0 = None  1 = deca=x10  2 = hecto=x100  –3 = mili=x10–3  3 = kilo=x1000  6 = Mega=x106  –6 = micro=x10–3  9 = Giga=x109  Das Datenelement *powerofTenMultiplier* **muss** mit einem entsprechenden Wert gefüllt werden. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.26 | **ReadingType - scaler**  Das Datenelement *scaler* repräsentiert den Skalierungsfaktor der ganzahligen Messwerte (IntervalReading – value) in der Messwertliste. Durch diesen kann eine Kommaverschiebung für den Messwert dargestellt werden. | Optional | Optional | muss implementiert werden |
| 1.27 | **ReadingType – timeAttribute**  Das Datenelement *timeAttribute* wird genutzt, um bestimmte Intervalltypen zu spezifizieren, mit denen die Messwertlisten beschrieben werden können. Beispiele für gültige Werte nach ESPI REQ.21 sind:  0 = Not Applicable  1 = 10-Minuten  2 = 15-Minuten  11 = täglich  24 = wöchentlich  Die Nutzung des Datenelements *timeAttribute* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.28 | **ReadingType – tou**  Das Datenelement *tou* kann genutzt werden, um bestimmte Time-Of-Use-Blöcke zu codieren. Beispiele nach ESPI REQ.21 sind:  0 = Nicht anwendbar  1 = TOU A  2 = TOU B  Die Nutzung des Datenelements *tou* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.29 | **Readingtype – uom**  Das Datenelement *uom* codiert die Maßeinheit, welche für alle Messwerte der Messwertliste gilt. Gültige Werte nach ESPI REQ.21 sind:  0 = Not Applicable  5 = A (Current)  29 = Voltage  31 = J (Energy joule)  33 = Hz (Frequency)  38 = Real power (Watts)  42 = m3 (Cubic Meter)  61 = VA (Apparent power)  63 = VAr (Reactive power)  65 = CosPhi (Power factor)  67 = V2 (Volts squared)  69 = A2 (Amp squared)  71 = VAh (Apparent energy)  72 = Real energy (Watt-hours)  73 = VArh (Reactive energy)  106 = Ah (Ampere-hours / Available Charge)  119 = ft3 (Cubic Feet)  122 = ft3/h (Cubic Feet per Hour)  125 = m3/h (Cubic Meter per Hour)  128 = US gl (US Gallons)  129 = US gl/h (US Gallons per Hour)  Das Datenelement *uom* **muss** mit einem entsprechenden Wert gefüllt werden. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.30 | **ReadingType – converterFactor**  Das Datenelement *converterFactor* kann genutzt werden um einen Wandlerfaktor mit aufzunehmen.  Die Nutzung des Datenelements *converterFactor* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.31 | **ReadingType – description**  Das Datenelement *description* kann genutzt werden um eine Beschreibung der Messwertliste einzufügen. Diese Beschreibung kann 256 Zeichen beinhalten und sollte den OBIS-Code für den Letztverbraucher näher spezifizieren.  Die Nutzung des Datenelements *description* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.32 | **ReadingType – obisCode**  Das Datenelement *obisCode* codiert die Messwerte der Messwertliste wie im Object Identification System (OBIS) nach DIN EN 62056‑6‑1 und für die Nutzung in den EDIFACT-Nachrichtentypen des deutschen Energiemarktes beschrieben.  Das Datenelement *obisCode* **muss** mit einem entsprechenden Wert (OBIS-Kennzahl) gefüllt werden.  **Dastellung als** HexCode ohne Trennzeichen im einem String-Datenformat. | Erforderlich | Erforderlich | Imuss implementiert werden |
| 1.33 | **ReadingType – qualifiedLogicalName**  Das Datenelement *qualifiedLogicalName* ist ein eindeutiger Bezeichner einer Messreihe, die von einem Smart Meter Gateway übertragen werden kann.  Er setzt sich nach der COSEM-Definition zusammen:   * <OBIS>.<Meter-ID>.sm für originäre Messwertelisten * <OBIS>.<TAF-ID>.sm für abgeleitete Messwertelisten   Jede Instanz der Klasse *readingType* muss ein Datenelement *qualifiedLogicalName* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
|  | **ReadingType – measurementPeriod**  Messperiode in Sekunden. | Erforderlich | Optional | Implementiert |
| 1.34 | **IntervalBlock**  Die Klasse *IntervalBlock* enthält die einzelnen Intervallwerte. Eine Nachricht **muss** mindestens eine Instanz der Klasse *IntervalBlock* enthalten.  Jede Instanz der Klasse IntervalBlock:  **• muss** auf mindestens eine Instanz der Klasse *IntervalReading* verweisen | Erforderlich | Erforderlich | muss implementiert werden |
| 1.35 | **IntervalBlock – interval**  Das Datenelement *interval* beschreibt die gesamte Zeitperiode, für die die nachfolgenden Messwerte in der Messwertliste enthalten sind.  Die Zeitperiode wird durch einen Startzeitpunkt und eine Dauer definiert.  Der Startzeitpunkt wird als Xs:dateTime beschrieben. Die Dauer wird als ganzzahliger Sekundenwert beschrieben.  Jede Instanz der Klasse *IntervalBlock* **muss** ein Datenelement vom Typ *interval* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.36 | **IntervalReading**  Die Klasse *IntervalReading* repräsentiert die Daten zu einem konkreten Messwert. Jede Nachricht **muss** mindestens eine Instanz der Klasse *IntervalReading* enthalten.  Jede Instanz der Klasse *IntervalReading*:  **• muss** auf eine Instanz der Klasse *ReadingQuality* verweisen. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.37 | **IntervalReading – cost**  Das Datenelement *cost* kann genutzt werden, um die Kosten, die mit diesem konkreten Messwert ver­bunden sind, darzustellen.  Die Nutzung des Datenelements *cost* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.38 | **IntervalReading – timePeriod**  Das Datenelement *timePeriod* beschreibt das Intervall für das der angegebene Messwert gültig ist.  Das Intervall wird durch einen Startzeitpunkt und eine Dauer definiert.  Der Startzeitpunkt wird als Xs:dateTime beschrieben. Für eichrechtlich relevante Daten muss hier zwingend die „capture\_time“ eingetragen werden, also der sekundengenaue Zeitpunkt der Messwert­erfassung. Dieser wird zur Überprüfung der inneren Signatur benötigt. Die Dauer wird als ganz­zahliger Sekundenwert beschrieben.  Bei einem Zählerstandsgang ist die Dauer 0s, da es sich um einen Zeit*punkt* handelt.  Jede Instanz der Klasse *IntervalReading* **muss** ein Datenelement vom Typ *timePeriod* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
|  | **IntervalReading – targetTime**  Das Datenelement *targetTime* beschreibt den Zeitpunkt, für den der angegebene Messwert aufgezeichnet wurde. | Erforderlich | Optional | Implementiert |
|  | **IntervalReading – measurementTimeMeter**  Das Datenelement *measurementTimeMeter* beschreibt den Zeitpunkt, zu dem das Messgerät den Wert aufgezeichnet hat. | Optional | Optional | (ggf. relevant für PrüDi wegen Signaturprüfung) |
| 1.39 | **IntervalReading – value**  Das Datenelement *value* repräsentiert den Wert der Messung. Dieser wird als ganzzahliger Wert definiert.  Jede Instanz der Klasse *IntervalReading* **muss** ein Datenelement *value* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.40 | **IntervalReading – certId**  Das Datenelement *certId* enthält die ID des Zertifikats der signierenden Instanz des MesswertesDas entsprechende Zertifikat wird in der Klasse *Certificate* übertragen.  Eine Instanz der Klasse *IntervalReading* **muss** ein Datenelement *certId* beinhalten, sofern eine - Prüfung der Signatur durchgeführt werden soll. | Optional | Optional | keine zwingende Implementierung |
| 1.41 | **IntervalReading – meterSig**  Das Datenelement *meterSig* repräsentiert die Zählersignatur des Messwertes. Diese kann versendet werden, sofern der Zähler eine entsprechende Signaturfunktion unterstützt. Die Zählersignatur kann mit dem *publicKey* der Klasse *Meter* überprüft werden.  Die Nutzung des Datenelements *meterSig* ist optional. | Optional | Optional | keine zwingende Implementierung |
| 1.42 | **IntervalReading – signature**  Das Datenelement *signature* repräsentiert die innere Signatur des Messwertes. Diese wird vom Smart Meter Gateway gebildet und den Messwerten der Messwertliste zugeordnet.  Eine Instanz der Klasse IntervalReading **kann** ein Datenelement *signature* enthalten. | Erforderlich | Optional | PrüDi |
| 1.43 | **IntervalReading – statusFNN**  Das Datenelement *statusFNN* repräsentiert das Statuswort, welches aus dem Statuswort des Zählers und des Smart Meter Gateways zusammensetzt. Dieses Statuswort wird laut FNN Lastenhaft Smart Meter Gateway [FNN SNGW] als octet-String übertragen und wird für die Überprüfung der inneren Signatur benötigt.  Datenformat: Hex-Darstellung gespeichert als String256.  Eine Instanz der Klasse *IntervalReading* **muss** ein Datenelement *statusFNN oder statusPTB* beinhalten. Wenn eine Prüfung der Signatur durchgeführt werden soll, muss das Datenelement zwingend enthalten sein.  Die beiden Statusworte sind Least-Significant-Bit-First-Codiert.  Beispiel (nur die Statuswort-Identifikation-Bits sind gesetzt):  <SMGW-Status> + <Zähler-Status>, z.B. A000000020000000 | Erforderlich  **(**bedingt**)** | Erforderlich  **(**bedingt**)** | muss implementiert werden |
| 1.44 | **IntervalReading – statusPTB**  Das Datenelement *statusPTB* repräsentiert das Statuswort des Zählers nach PTB 50.8. Dieses Statuswort wird laut FNN Lastenhaft Smart Meter Gateway [FNN SNGW] als octet-String übertragen.  Eine Instanz der Klasse *IntervalReading* **muss** ein Datenelement *statusFNN oder statusPTB* beinhalten. | Erforderlich  **(**bedingt**)** | Erforderlich  **(**bedingt**)** | muss implementiert werden |
| 1.45 | **ReadingQuality**  Die Klasse *ReadingQuality* repräsentiert den Status eines Messwertes. Jeder Messwert **kann** mit einer entsprechenden Statusinformation gekennzeichnet werden.  Die Klasse *ReadingQuality* verweist auf keine weiteren Klassen. | Optional | Optional | keine zwingende Implementierung |
| 1.46 | **ReadingQuality – quality**  Das Datenelement *quality* repräsentiert die eigentliche Statusinformation zu einem Messwert. Gültige Werte nach ESPI REQ.21 sind:  0 – valid (gültig)  7 – manually edited (manuell geändert)  8 – estimated using reference day (geschätzt basierend auf einem Referenztag)  9 – estimated using linear interpolation (geschätzt mittels linearer Interpolation)  10 – questionable (unplausibel)  11 – derived (abgeleited)  12 – projected (forecast) (hochgerechnet (Prognose))  13 – mixed (gemischt)  14 – raw (roh/unverarbeitet)  15 – normalized for weather (normalisiert im Bezug zum Wetter)  16 – andere (other)  17 – validated (gültig)  18 – verified (überprüft)  Jede Instanz der der Klasse *ReadingQuality* **muss** ein Datenelement *quality* mit einem entsprechenden Wert enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 1.47 | **Meter**  Die Klasse *Meter* identifiziert den Zähler über die enthaltene Zählerkennung. Eine originäre Messliste **muss** eine Zählernummer enthalten.  Die Klasse *Meter* verweist auf keine weiteren Klassen. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.48 | **Meter – meterId**  Das Datenelement *meterId* repräsentiert die Zählernummer des Smart Meters.  Eine Instanz der Klasse *Meter* **muss** ein Datenelement *meterId* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.49 | **Meter – publicKey**  Das Datenelement *publicKey* der Klasse *Meter* kann genutzt werden um den öffentlichen Schlüssel des Zählers zu übertragen. Dies ist insbesondere der Fall, wenn das Smart Meter die Zähler mit Zähler­signatur versendet. Der Wert *publicKey* ermöglicht damit die Überprüfung der Zählersignatur.  Die Nutzung des Datenelements *publicKey* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 1.50 | **SMGW**  Die Klasse *SMGW* repräsentiert Informationen zum Smart Meter Gateway, von welchem die beinhalten­den Messwertlisten stammen. Eine Nachricht **muss** eine Instanz von *SMGW* enthalten.  Die Klasse *SMGW* verweist auf keine weiteren Klassen. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.51 | **SMGW – certId**  Das Datenelement *certId*enthält die ID eines Zertifikats, das inhaltlich zu diesem S gehört. Das Zertifikat selbst und um welchen Typ (HAN Zertifikat des SMGW, WAN SIG Zertifikat des SMGW, SubCA Zertifikat der ausstellenden SubCA) wird in der Klasse *Certificate* übertragen.  Jede Instanz von *SMGW* **muss** bei lokalem Zugriff auf die Messwerte über HAN an der Schnittstelle IF\_Adapter\_TRuDI mind ein Datenelement vom Typ *certId* beinhalten, das auf ein HAN Zertifikat verweist. Sofern eine Signaturprüfung durchgeführt werden soll, müssen sowohl das WAN SIG als auch das zugehörige SubCA Zertifikat referenziert werden. | Erforderlich | Optional | muss implementiert werden |
| 1.52 | **SMGW – smgwId**  Das Datenelement *smgwId* repräsentiert die Kennung des Smart Meter Gateways, von denen die Daten der Nachricht und der öffentliche Schlüssel stammen. Jede Instanz der Klasse *SMGW* **muss** ein Daten­element vom Typ *smgwId* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
|  | **SMGW – FirmwareVersion**  Das Datenelement *FirmwareVesion* enthält die Firmware-Version des SMGW. | Erforderlich |  | PrüDi |
|  | **SMGW – FirmwareComponent**  Das Datenelement *FirmwareComponent* enthält Version und/oder eine Checksumme einer einzelnen Firmware-Komponente. |  |  | PrüDi (optional) |
|  | **SMGW – FirmwareComponent -name**  Das Datenelement *name* enthält den Namen der konkreten Firmware-Komponente. **Muss** immer vorhanden sein. | Erforderlich |  |  |
|  | **SMGW – FirmwareComponent -version**  Das Datenelement *version* enthält die Version der konkreten Firmware-Komponente. | Erforderlich  (bedingt) |  | PrüDi (optional) |
|  | **SMGW – FirmwareComponent -checksum**  Das Datenelement *checksum* enthält die Prüfsumme der konkreten Firmware-Komponente. | Erforderlich  (bedingt) |  | PrüDi (optional) |
|  | **VendorConfig**  Das Datenelement *VendorConfig* enthält Herstellerspezifische Datenstrukturen, welche von PrüDi benötigt werden. |  |  |  |
|  | **tafProfile**  Das Datenelement *tafProfile* enthält das TAF-Profil möglichst 1:1 wie es im Gateway hinterlegt ist. Das TAF-Profil muss vom SMGW signiert geliefert werden.  Die Daten werden im Element *tafProfile* als xs:hexBinary abgelegt.  Die Verwendung eines CMS-Containers wäre das empfohlene Format für die Übertragung des TAF-Profils inkl. einer Signatur. |  |  |  |
| 1.53 | **Certificate**  Die Klasse *Certificate* repräsentiert das Zertifikat, welches für die Inhaltsdatensignierung (WAN SIG), für die TSL-Verschlüsselung am HAN oder für die Signierung von Zertifikaten durch eine SubCA genutzt wird.Wird von einer anderen Rolle als dem SMGW ein Messwert signiert (z. B. bei manueller Änderung eines Messwerts), so ist das entsprechende Zertifikat hier zusätzlich einzufügen.  Für eine eichrechtlich-konforme Überprüfung der Daten **muss** die Nachricht mindestens eine Instanz der Klasse *Certificate* beinhalten.  Die Klasse *Certificate* verweist auf keine weiteren Klassen. | Erforderlich | Optional | muss implementiert werden |
| 1.54 | **Certificate – certId**  Das Datenelement certId identifiziert ein Zertifikat eindeutig. Das Zertifikat des SMGW hat per Default die ID mit der Nummer 1.  Jede Instanz der Klasse *Certificate* muss ein Datenelement vom Typ *CertId* enthalten. | Erforderlich\* | Erforderlich | muss implementiert werden |
| 1.55 | **Certificate – certType**  Das Datenelement certType klassifiziert ein Zertifikat eindeutig als:  1 = Signatur-Zertifikat  2 = SubCA-Zertifikat  3 = SMGW-HAN-Zertifikat  Jede Instanz der Klasse *Certificate* **muss** ein Datenelement vom Typ *certType* enthalten | Erforderlich | Erforderlich | muss implementiert werden |
| 1.56 | **Certificate – parentCertId**  Das Datenelement *parentCertId* enthält die ID des Zertifikats der nächst höheren SubCA in der Zertifikatskette.  Jede Instanz der Klasse *Certificate* **kann** ein Datenelement vom Typ *parentCertId*enthalten. | Optional | Optional | muss implementiert werden |
| 1.57 | **Cerificate – certContent**  Das Datenelement **certContent**enthält das eigentliche Zertifikat.  Jede Instanz der Klasse *Certificate* muss ein Datenelement vom Typ **certContent**enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.58 | **InvoicingParty**  Die Klasse *InvoicingParty* repräsentiert den Rechnungssteller bzw. die Marktrolle, die für die Tarifierung der Messdaten für den Letztverbraucher verantwortlich ist. Jede Nachricht **muss** eine Instanz der Klasse *InvoicingParty* beinhalten.  Die Klasse *InvoicingParty* verweist auf keine weiteren Klassen. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.59 | **InvoicingParty – invoicingPartyId**  Das Datenelement *invoicingPartyId* wird genutzt, um die eindeutige Kennung des Marktteilnehmers, welcher für die Tarifierung der Messdaten für den Letztverbraucher verantwortlich ist. Dies kann zum Beispiel die Kennung des Logical Devices des EMT im SMGW ohne die Endung „.sm“ sein. Die eindeutige Kennung muss dem Letztverbraucher vorab bekannt gemacht werden. Jede Instanz der Klasse *InvoicingParty* **muss** ein Datenelement vom Typ *invoicingPartyId* beinhalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 1.60 | **Customer**  Die Klasse *Customer* repräsentiert den Letztverbraucher, welcher über eine je Rechnungsstellende ein­deutige Kennung beschrieben wird. Jede Nachricht an der Schnittstelle IF\_Adapter\_TRuDI **muss** eine Instanz der Klasse *Customer* beinhalten.  Die Klasse *Customer* verweist auf keine weiteren Klassen. | Erforderlich | Optional | muss implementiert werden |
| 1.61 | **Customer – customerId**  Das Datenelement *customerId* beinhaltet eine eindeutige Identifikation des Letztverbrauchers. Diese Identifikation sollte mindestens je Marktteilnehmer eindeutig sein. Genutzt werden kann hier die Kennung des Logical Devices des Letztverbrauchers im SMGW ohne die Endung „.sm“.  Jede Instanz der Klasse *Customer* muss ein Datenelement vom Typ *customerId* beinhalten. | Erforderlich | Erforderlich | muss implementiert werden |

Tabelle 2: Basisdaten und deren Verwendung in TRuDI

## Prüfungsdaten

Die Prüfungsdaten werden zur eichrechtlich konformen Rechnungsprüfung benötigt. Zentrales Datenelement der Prüfungsdaten ist die Klasse AnalysisProfile.

| **Nr.** | **Beschreibung** | **Verwendung bei IF\_Adapter\_TRuDI** | **Verwendung bei IF\_Lieferant\_TRuDI** | **Realisierung für TRuDI v1.0** |
| --- | --- | --- | --- | --- |
| 2.01 | **AnalysisProfile**  Die Klasse *AnalysisProfile* repräsentiert das Auswerteprofil. Bei der Übertragung von Daten zu Zwecken der eichrechtskonformen Rechnungsüberprüfung **muss** die Nachricht eine Instanz der Klasse enthalten. Bei der Übertragung von Daten für das Alltagsdisplay ist die Nutzung der Klasse *AnalysisProfile* **optional**.  Eine Instanz der Klasse *AnalysisProfile*:  **• muss** **auf** mindestens eine Instanz der Klasse *TariffStage* verweisen  **• kann** auf eine Klasse *RuleSystem* verweisen | Optional | Erforderlich | muss implementiert werden |
| 2.02 | **AnalysisProfile – billingPeriod**  Das Datenelement *billingPeriod* spezifiziert den Abrechnungszeitraum für den dieses Auswerte­profil Gültigkeit hat.  Eine Instanz der Klasse *AnalysisProfile* **muss** genau ein Datenelement *billingPeriod*enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.03 | **AnalysisProfile – description**  Das Datenelement *description* kann zur Beschreibung des Tarifs genutzt werden, für den das Aus­werteprofil gültig ist.  Das Datenelement *description* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.04 | **AnalysisProfile – tariffUseCase**  Das Datenelement *tariffUseCase* spezifiziert den Tarifanwendungsfall für den diese Tarifstufe gültig ist. Die angegebene Nummer entspricht dabei dem Anwendungsfall wie er in der TR-03109-1 des BSI definiert wurde.  Eine Instanz der Klasse *AnalysisProfile* **muss** genau ein Datenelement der Klasse *tariffUseCase* bein­halten. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.05 | **AnalysisProfile – tariffId**  Das Datenelement *tariffId* entspricht der Tarifkennbezeichnung des Lieferanten.  Eine Instanz der Klasse *AnalysisProfile* **muss** genau ein Datenelement *tariffId* enthalten | Erforderlich | Erforderlich | muss implementiert werden |
| 2.06 | **AnalysisProfile – defaultTariffNumber**  Das Datenelement *defaultTariffNumber* ist eine Referenz auf die Standardtarifstufe, die zu Beginn eines Abrechnungszeitraums gültig ist.  Eine Instanz der Klasse *AnalysisProfile* **muss** genau ein Datenelement *defaultTariffNumber* ent­halten. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.07 | **RuleSystem**  Die Klasse *RuleSystem* repräsentiert das Regelwerk.  Eine Instanz der Klasse *RuleSystem* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.08 | **RuleSystem – rules**  Das Datenelement *rules* spezifiziert eine URI/URL als Link zum Ablageort des Regelwerks. Dieses kann dann von der Basissoftware dort heruntergeladen werden. Eine Instanz der Klasse RuleSystem **muss** einen Link zum Regelwerk enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.09 | **TariffStage**  Die Klasse *TariffStage* spezifiziert einzelnen Tarifstufen des Auswerteprofils. Ein Auswerteprofil **muss** mindestens eine Instanz der Klasse *TariffStage* enthalten.  Eine Instanz der Klasse *TariffStage*:  **• kann** auf eine Instanz der Klasse *DatetimePeriod* verweisen  **• kann** auf eine oder zwei Instanzen der Klasse *MinMaxCounter* verweisen  **• kann** auf Instanzen der Klasse *TariffChangeTrigger* verweisen  **• kann** auf eine Instanz der Klasse *Prepaid* verweisen | Erforderlich | Erforderlich | muss implementiert werden |
| 2.10 | **TariffStage – tariffNumber**  Das Datenelement *tariffNumber* spezifiziert die Tarifnummer der Tarifstufe. Eine Instanz der Klasse *TariffStage* **muss** genau ein Datenelement *tariffNumber* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.11 | **TariffStage – description**  Das Datenelement *description* dient zur freien Beschreibung der Tarifstufe. Die Nutzung ist **optional**. | Optional | Optional | muss implementiert werden |
| 2.12 | **TariffStage – obisCode**  Das Datenelement *obisCode* der Klasse *TariffStage* beschreibt den OBIS-Code der Tarifstufe. Mit Hilfe des OBIS-Codes ist wird eine Zuordnung der Tarifstufen zu Messwertlisten ermöglicht.  **Dastellung als** HexCode ohne Trennzeichen im einem String-Datenformat Jede Instanz der Klasse *TariffStage* **muss** genau ein Datenelement *obisCode* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.13 | **DatetimePeriod**  Die Klasse *DateTimeperiod* beschreibt den Gültigkeitszeitraum einer tarifstufe. Eine Instanz der Klasse *TariffStage* **kann** genau auf eine Instanz der Klasse *DatetimePeriod* verweisen.  Eine Instanz der Klasse *DatetimePeriod* muss:  **• entweder** ein Datenelement *datetimeInterval* beinhalten  **• oder** ein Datenelement *datetimevar* beinhalten  Eine Instanz der Klasse DatetimePeriod verweist auf keine weiteren Klassen. | Optional | Optional | keine zwingende Implementierung |
| 2.14 | **DatetimePeriod – datetimeInterval**  Das Datenelement *dateTimeInterval* spezifiziert den Gültigkeitszeitraum, zu dem eine Tarifstufe gültig ist.  **Bedingung**: Entweder **muss** das Datenelement *datetimeInterval* oder es **muss** das Datenelement *dateTimeVar* in der Instanz der Klasse *DatetimePeriod* angegeben werden. | Erforderlich (bedingt) | Erforderlich  **(**bedingt**)** | keine zwingende Implementierung |
| 2.15 | **DatetimePeriod – datetimeVar**  Das Datenelement *dateTimeVar* spezifiziert den Gültigkeitszeitraum unter der Angabe des Start­zeitpunkts.  **Bedingung**: Entweder **muss** das Datenelement *datetimeInterval* oder es **muss** das Datenelement *dateTimeVar* in der Instanz der Klasse Datetimperiod angegeben werden. | Erforderlich  **(**bedingt**)** | Erforderlich  **(**bedingt**)** | keine zwingende Implementierung |
| 2.16 | **MinMaxCounter**  Die Klasse *MinMaxCounter* beschreibt die Anzahl und Eigenschaft der Minimal- bzw. Maximal­werte, die für eine Tarifstufe ermittelt bzw. angegeben werden sollen.  Die Verwendung der Klasse *MinMaxCounter* ist **optional**  Die Klasse *MinMaxCounter* **kann maximal zweimal** instanziiert werden, um so Minimalwerte und Maximalwerte zu ermitteln bzw. anzugeben.  Die Klasse MinMaxCounter verweist auf keine weiteren Klassen. | Optional | Optional | keine zwingende Implementierung |
| 2.17 | **MinMaxCounter – count**  Das Datenelement *count* spezifiziert die Anzahl der zu ermittelnden bzw. anzugebenden Minimal- oder Maximalwerte.  Eine Instanz der Klasse *MinMaxCounter* **muss** ein Datenelement *count* enthalten werden. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.18 | **MinMaxCounter – maxCounter**  Das Datenelement *maxCounter* spezifiziert einen boolschen Wert (Flag). Ist der Wert 1=true, werden Maximalwerte angegeben bzw. ermittelt. Ist der Wert des Datenelements 0=false, werden Minimalwerte angegeben bzw. ermittelt.  Eine Instanz der Klasse *MinMaxCounter* **muss** ein Datenelement vom Typ *maxCounter* enthalten.  Das Datenelement **muss** den Wert 0 oder den Wert 1 annehmen. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.19 | **Prepaid**  Die Klasse *Prepaid* dient zur Auswertung von Prepaid-Tarifen (TAF12).  Eine Instanz der Klasse *Prepaid* ist **optional**.  Eine Instanz der Klasse *Prepaid*:  **• kann** auf Instanzen der Klasse *Breaker* verweisen  **• kann** auf Instanzen der Klasse *ThresholdTrigger* verweisen | Optional | Optional | keine zwingende Implementierung |
| 2.20 | **Prepaid – availableEnergy**  Das Datenelement *availableEnergy* spezifiziert die bezahlte und damit verfügbare Menge an Energie des Prepaid-Tarifs.  Eine Instanz der Klasse Prepaid **muss** ein Datenelement vom Typ *availableEnergy* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.21 | **Prepaid – powerOfTenMultiplier**  Das Datenelement *powerOfTenMultiplier* spezifiziert den Einheitenvorsatz der bezahlten und ver­fügbaren Menge des Prepaid-Tarifs.  Gültige Werte nach ESPI REQ.21 sind:  0 = None  1 = deca=x10  2 = hecto=x100  –3 = mili=x10–3  3 = kilo=x1000  6 = Mega=x106  –6 = micro=x10–3  9 = Giga=x109  Eine Instanz der Klasse Prepaid **muss** ein Datenelement vom Typ *powerOfTenMultiplier* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.22 | **Prepaid – toleranceInterval**  Das Datenelement *toleranceInterval* spezifiziert die Länge des Zeitraums in dem der Letzt­verbraucher nach Verbrauch der verfügbaren Energiemenge noch Energie beziehen kann.  Eine Instanz der Klasse *Prepaid* **muss** ein Datenelement vom Typ *toleranceInterval* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.23 | **Prepaid – uom**  Das Datenelement *uom* spezifiziert die Einheit in der die bezahlte und verfügbare Menge des Prepaid-Tarifs angeben wird.  Eine Instanz der Klasse *Prepaid* **muss** ein Datenelement vom Typ *uom* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.24 | **Breaker**  Die Klasse *Breaker* beschreibt die Unterbrecher, die zum Zeitpunkt des Verbrauches der verfüg­baren Energiemenge, ein Signal erhalten.  Instanzen der Klasse *Breaker* sind **optional**.  Eine Instanz der Klasse *Breaker* verweist auf keine weiteren Klassen. | Optional | Optional | keine zwingende Implementierung |
| 2.25 | **Breaker – breakerId**  Das Datenelement *breakerId* spezifiziert die eindeutige Kennzeichnung des Unterbrechers.  Eine Instanz der Klasse *Breaker* **muss** ein Datenelement vom Typ *breakerId* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.26 | **TariffChangeTrigger**  Die Klasse *TariffChangeTrigger* abstrahiert alle weiteren Triggerformen und damit alle weiteren Tarifumschaltgründe.  Eine Instanz der Klasse *TariffChangeTrigger*:  **• muss** auf genau eine Instanz der folgenden Klassen verweisen:  – *ThresholdTrigger*  – *ExternalTrigger*  – *TimeTrigger*  Eine Instanz der Klasse *TariffChangeTrigger* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.27 | **ThresholdTrigger**  Die Klasse *ThresholdTrigger* beschreibt einen Trigger, der einen Tarifstufenwechsel durch Grenz­wertüberschreitung oder -unterschreitung auslöst.  Eine Instanz der Klasse *ThresholdTrigger* ist **optional**.  Eine Instanz der Klasse *ThresholdTrigger*:  **muss** auf eine Instanz der Klasse *Threshold* verweisen | Optional | Optional | keine zwingende Implementierung |
| 2.28 | **ExternalEventTrigger**  Die Klasse *ExternalEventTrigger* beschreibt einen Trigger, der durch externe Events ausgelöst werden kann, um in die referenzierte Tarifstufe zu wechseln. Eine Instanz der Klasse *ExternalEventTrigger* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.30 | **Threshold**  Die Klasse *Threshold* beschreibt einen mengenbasierten Grenzwert, der aufgrund Grenzwert­überschreitung oder –unterschreitung einen Tarifstufenwechsel auslöst.  Eine Instanz der Klasse **muss** angegeben werden, sofern eine Instanz der Klasse *ThresholdTrigger* existiert.  Eine Instanz der Klasse *ExternalEvent* **kann** Instanzen der Klasse *Threshold* referenzieren, hierbei ist eine Instanz der Klasse *Threshold* **optional**.  **•** Eine Instanz der Klasse *Threshold* verweist auf keine weiteren Klassen. | Erforderlich/Optional | Erforderlich/Optional | keine zwingende Implementierung |
| 2.31 | **Threshold – direction**  Das Datenelement *direction* spezifiziert, ob die Grenzwerte, welche in den Datenelementen *overLimit* oder *underLimit* angegeben werden, über- oder unterschritten werden müssen, um einen Tarifstufenwechsel oder ein externes Event auszulösen. Durch 1=true oder 0=false wird festgelegt, ob ein Über- oder ein Unterschreiten eines Grenzwertes geprüft wird.  Bei Prüfung auf Überschreitung (1=true) tritt das Ereignis bei Überschreitung des ‚over\_limit‘ ein.  Bei Prüfung auf Unterschreitung (0 = false) tritt das Ereignis bei Unterschreitung des ‚under\_limit‘ ein.  Das Datenelement *direction* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.32 | **Threshold – meterReadingId**  Das Datenelement *meterReadingId* spezifiziert die Referenz auf Messwertlisten auf die sich die Grenzwertdefinition beziehen. Das Datenelement *meterReadingId* referenziert damit auf das Datenelement *meterReadingId* der Klasse *MeterReading* in den Basisdaten. Hiermit können auch Grenzwerte in Messwertlisten referenziert werden, die nicht die aktuelle Tarifstufe betreffen (z. B. Grenzwerte der Leistungsmessung).  Eine Instanz der Klasse *Threshold* muss ein Datenelement *meterReadingId enthalten.* | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.33 | **Threshold – overLimit**  Das Datenelement *overLimit* spezifiziert den oberen Grenzwert eines Bandes zwischen den Grenzwerten abgebildet durch *overLimit* und *underLimit*. Die Datenelemente *overLimit* und *under­Limit* können auf den gleichen Wert gesetzt werden.  Das Datenelement *overLimit* ist vom Datentyp *ThresholdLimit*.  Eine Instanz der Klasse *Threshold* **muss** ein Datenelement *overLimit* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.34 | **Threshold – underLimit**  Das Datenelement *underLimit* spezifiziert den unteren Grenzwert eines Bandes zwischen den Grenzwerten abgebildet durch *overLimit* und *underLimit*. Die Datenelemente *overLimit* und *underLimit* können auf den gleichen Wert gesetzt werden.  Das Datenelement *underLimit* ist vom Datentyp *ThresholdLimit*.  Eine Instanz der Klasse *Threshold* **muss** ein Datenelement *underLimit* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.35 | **ThresholdLimit**  Die Klasse *ThresholdLimit* spezifiziert den Datentyp für Grenzwerte inkl. Einheit und Einheiten­vorsatz. Eine Instanz der Klasse ist je nach Verwendung des Datentyps **optional**.  Die Klasse *ThresholdLimit* verweist auf keine weiteren Klassen. | Optional (Datentyp) | Optional (Datentyp) | keine zwingende Implementierung |
| 2.36 | **ThresholdLimit – powerOfTenMultiplier**  Das Datenelement *powerOfTenMultiplier* spezifiziert den Einheitenvorsatz des Grenzwertes.  Gültige Werte nach ESPI REQ.21 sind:  0 = None  1 = deca=x10  2 = hecto=x100  –3 = mili=x10–3  3 = kilo=x1000  6 = Mega=x106  –6 = micro=x10–3  9 = Giga=x109  Das Datenelement **muss** genau einmal angegeben werden. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.37 | **ThresholdLimit – uom**  Das Datenelement *uom* spezifiziert die Einheit des Grenzwertes. Das Datenelement **muss** genau einmal angegeben werden. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.38 | **ThresholdLimit – value**  Das Datenelement *value* beschreibt den Wert des Grenzwertes. Das Datenelement **muss** genau einmal angegeben werden. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.39 | **ExternalEvent**  Eine Instanz der Klasse *ExternalEvent* beschreibt ein externes Ereignis (Event). Ist eine Instanz der Klasse *ExternalEventTrigger* vorhanden, **muss** diese auf genau eine Instanz der Klasse *ExternalEvent* verweisen.  Eine Instanz der Klasse *ExternalEvent*:  **• kann** auf eine oder zwei Instanzen der Klassse *Threshold* verweisen. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.40 | **ExternalEvent – eventId**  Das Datenelement *eventId* spezifiziert die Kennzeichnung eines externen Events und dient zur Zuordnung von Events aus den Logdaten.  Eine Instanz der Klasse *ExternalEvent* **muss** ein Datenelement vom Typ *eventId* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.41 | **ExternalEvent – eventPeriod**  Das Datenelement *eventPeriod* spezifiziert den Zeitraum, in das Event gültig ist.  Eine Instanz der Klasse *ExternalEvent* **muss** ein Datenelement vom Typ *even­tId* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.42 | **ExternalEvent – tariffNumber**  Das Datenelement *tariffNumber* beschreibt die Tarifstufe, die mit dem Eintritt des Events aktiviert wird.  Eine Instanz der Klasse *ExternalEvent* **muss** ein Datenelement vom Typ *tariffNumber* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.43 | **TimeTrigger**  Die Klasse *TimeTrigger* beschreibt einen Trigger, der anhand von zeitlichen Angaben einen Tarif­stufenwechsel auslöst.  Eine Instanz der Klasse *TimeTrigger* ist **optional**.  Eine Instanz der Klasse *TimeTrigger*:  **• muss** auf mindestens eine Instanz der Klasse *DayProfile* verweisen  **• kann** auf Instanzen der Klasse *SpecialDayProfile* verweisen  **• kann** auf Instanzen der Klasse *WeekProfile* verweisen  **• kann** auf Instanzen der Klasse *SeasonProfile* verweisen | Optional | Optional | muss implementiert werden |
| 2.44 | **SeasonProfile**  Die Klasse *SeasonProfile* spezifiziert eine zeitliche Saison in der ein bestimmtes Wochenprofil (Zeit) referenziert wird.  Eine Instanz der Klasse *SeasonProfile*:  **• muss** auf eine Instanz der Klasse *WeekProfile* verweisen | Optional | Optional | keine zwingende Implementierung |
| 2.45 | **SeasonProfile – seasonProfileName**  Das Datenelement *seasonProfileName* dient zur freien Beschreibung des Saisonnamens.  Eine Instanz der Klasse *SeasonProfile* **muss** ein Datenelement vom Typ *seasonProfileName* ent­halten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.46 | **SeasonProfile – seasonStart**  Das Datenelement *seasonStart* spezifiziert den Start der Saison.  Eine Instanz der Klasse *SeasonProfile* **muss** ein Datenelement vom Typ *seasonProfileName* ent­halten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.47 | **SeasonProfile – weekProfileName**  Das Datenelement *weekProfileName* spezifiziert die Kennzeichnung bzw. Identifizierung des referenzierten Wochenprofils.  Eine Instanz der Klasse *SeasonProfile* **muss** ein Datenelement vom Typ *weekProfileName* ent­halten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.48 | **WeekProfile**  Die Klasse *WeekProfile* dient zur Beschreibung eines Wochenprofils. Hierbei kann jeder Tag einer Woche ein eigenes Tagesprofil zugeordnet bekommen. Eine Instanz der Klasse *SeasonProfile* muss auf eine Instanz der Klasse *WeekProfile* verweisen. Eine Instanz der Klasse *TimeTrigger* kann auf eine Instanz der Klasse *WeekProfile* verweisen.  Die Klasse *WeekProfile* verweist auf keine weiteren Klassen. | Erforderlich/Optional | Erforderlich/Optional | keine zwingende Implementierung |
| 2.49 | **WeekProfile – monday**  Das Datenelement *monday* referenziert auf ein Tagesprofil (Datentyp *DayProfile*). Es ist das Datenelement *dayId* des zu referenzierenden Tageprofils einzusetzen.  Das Datenelement ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.50 | **WeekProfile – tuesday**  Das Datenelement *tuesday* referenziert auf ein Tagesprofil (Datentyp *DayProfile*). Es ist das Datenelement *dayId* des zu referenzierenden Tageprofils einzusetzen.  Das Datenelement ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.51 | **WeekProfile – wednesday**  Das Datenelement *wednesday* referenziert auf ein Tagesprofil (Datentyp *DayProfile*). Es ist das Datenelement *dayId* des zu referenzierenden Tageprofils einzusetzen.  Das Datenelement ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.52 | **WeekProfile – thursday**  Das Datenelement *thursday* referenziert auf ein Tagesprofil (Datentyp *DayProfile*). Es ist das Datenelement *dayId* des zu referenzierenden Tageprofils einzusetzen.  Das Datenelement ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.53 | **WeekProfile – friday**  Das Datenelement *friday* referenziert auf ein Tagesprofil (Datentyp *DayProfile*). Es ist das Daten­element *dayId* des zu referenzierenden Tageprofils einzusetzen.  Das Datenelement ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.54 | **WeekProfile – saturday**  Das Datenelement *saturday* referenziert auf ein Tagesprofil (Datentyp *DayProfile*). Es ist das Datenelement *dayId* des zu referenzierenden Tageprofils einzusetzen.  Das Datenelement ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.55 | **WeekProfile – sunday**  Das Datenelement *sunday* referenziert auf ein Tagesprofil (Datentyp *DayProfile*). Es ist das Daten­element *dayId* des zu referenzierenden Tageprofils einzusetzen.  Das Datenelement ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.56 | **WeekProfile – weekProfileName**  Das Datenelement *weekProfileName* spezifiziert die Kennzeichnung bzw. Identifizierung des Wochenprofils.  Eine Instanz der Klasse *WeekProfile* muss ein Datenelement vom Typ *weekProfileName* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.57 | **DayProfile**  Instanzen der Klasse *DayProfile* kapseln ein oder mehrere Instanzen der Klasse *DayTimeProfile*.  Eine Instanz der Klasse *DayProfile:*  **• muss** genau auf mindestens eine Instanz der Klasse *DayTimeProfile* verweisen. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.58 | **DayProfile – dayId**  Das datenelement *dayId* identifiziert ein Tagesprofil eindeutig. Es wird von anderen Klassen genutzt, um auf das entsprechende Tagesprofil zu referenzieren.  Jede Instanz der Klasse *dayProfile* **muss** ein Datenelement vom Typ *dayId* beinhalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.59 | **DayTimeProfile**  Die Klasse *DayTimeProfile* spezifiziert die Tageszeit für die übergeordnete *DayProfile*-Instanz.  Die Klasse *DayTimeProfile* verweist auf keine weiteren Klassen. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.60 | **DayTimeProfile – startTime**  Das Datenelement *startTime* spezifiziert den Startzeitpunkt des referenzierenden Tagesprofiles (Klasse *DayProfile*). Der Startzeitpunkt wird aus der (zeitzonenunabhängigen) Angabe der Stunde (*TimeVarType – hour*) und der Minuten (*TimeVarType – minute*) gebildet. Als Default-Wert sollten für beide Werte 0 genutzt werden.  Jede Instanz der Klasse *DayTimeProfile* **muss** ein Datenelement vom Typ *startTime* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.61 | **DayTimeProfile – tariffNumber**  Das Datenelement *tariffNumber* verweist auf die Tarifstufe, die zu der angegebenen Startzeit (Datenelement *startTime*) gültig wird. Das Datenelement muss auf eine Instanz des Datenelements *tariffNumber* in der Klasse *TariffStage* verweisen.  Jede Instanz der Klasse *DayTimeProfile* **muss** ein Datenelement vom Typ *tariffNumber* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.62 | **SpecialDayProfile**  Die Klasse *SpecialDayProfile* spezifiziert Sondertage, die ein bestimmtes Tagesprofil abbilden müssen.  Instanzen der Klasse *SpecialDayProfile* sind **optional**.  Eine Instanz der Klasse *SpecialDayProfile:*  **• muss** auf eine Instanz der Klasse *DayProfile* verweisen. | Optional | Optional | muss implementiert werden |
| 2.63 | **SpecialDayProfile – dayId**  Das Datenelement *dayId* der Klasse *SpecialDayProfile* referenziert auf das entsprechende Tages­profil, welches die Startzeit definiert.  Jede Instanz der Klasse *SpecialDayProfile* **muss** ein Datenelement vom Typ *dayId* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.64 | **SpecialDayProfile – specialDayDate**  Das Datenelement *specialDayDate* spezifiziert das Datum des Sondertages. Das Datenelement *specialDayDate* spezifiziert das Datum des Sondertages. Das Datum wird über die Klasse *DayVarType* beschrieben.  Eine Instanz der Klasse *SpecialDayProfile* **muss** ein Datenelement vom Typ *specialDayDate* ent­halten. | Erforderlich | Erforderlich | muss implementiert werden |
| 2.65 | **TimeVarType**  Die Klasse *TimeVarType* kann genutzt werden um feste (zeitzonenunabhängige) Zeitangaben als Einzelwerte mit Stunden- und Minutenangabe zu definieren  Die Klasse *TimeVarType* verweist auf keine weiteren Klassen. | Optional | Optional | muss implementiert werden |
| 2.66 | **TimeVarType – hour**  Das Datenelement *hour* der Klasse *TimeVarType* kann genutzt werden um eine ganzzahlige Stundenangabe zu definieren.  Die Nutzung des Datenelements *hour* ist **optional**.  Wenn das Datenelement nicht in einer Instanz der Klasse *TimeVarType* vorhanden ist, so wird der Wert „0“ als gesetzt angesehen. | Optional | Optional | muss implementiert werden |
| 2.67 | **TimeVarType – minute**  Das Datenelement *minute* der Klasse *TimeVarType* kann genutzt werden um eine ganzzahlige Minutenangabe zu definieren.  Die Nutzung des Datenelements *minute* ist **optional**. Wenn das Datenelement nicht in einer Instanz der Klasse *TimeVarType* vorhanden ist, so wird der Wert „0“ als gesetzt angesehen. | Optional | Optional | muss implementiert werden |
| 2.68 | **TimeVarType – second**  Das Datenelement *second* der Klasse *TimeVarType* kann genutzt werden um eine ganzzahlige Sekundenangabe zu definieren.  Die Nutzung des Datenelements *second* ist **optional**. Wenn das Datenelement nicht in einer Instanz der Klasse *TimeVarType* vorhanden ist, so wird der Wert „0“ als gesetzt angesehen. | Optional | Optional | muss implementiert werden |
| 2.69 | **TimeVarType – hundreds**  Das Datenelement *hundreds* der Klasse *TimeVarType* kann genutzt werden um eine ganzzahlige Hunderstel-Sekundenangabe zu definieren.  Die Nutzung des Datenelements *hundreds* ist **optional**. Wenn das Datenelement nicht in einer Instanz der Klasse *TimeVarType* vorhanden ist, so wird der Wert „0“ als gesetzt angesehen. | Optional | Optional | keine zwingende Implementierung |
| 2.70 | **DayVarType**  Die Klasse *DayVarType* kann genutzt werden, um Datumsangaben darzustellen, zum Beispiel um Feiertage zu beschreiben. Ist in der Klasse *DayVarType* keine Jahresangabe enthalten, so gilt das angegebene Datum jährlich. Sind keine Monatsangaben enthalten, so gilt das angegebene Datum monatlich.  • Die Klasse *DayVarType* verweist auf keine weiteren Klassen. | Optional | Optional | muss implementiert werden |
| 2.71 | **DayVarType – dayOfMonth**  Das Datenelement dayOfMonth kann genutzt werden, um den genauen Tag innerhalb eines Monats zu spezifizieren.  Die Nutzung des Datenelements dayOfMonth ist **optional**. | Optional | Optional | muss implementiert werden |
| 2.72 | **DayVarType – dayOfWeek**  Das Datenelement *dayOfWeek* kann genutzt werden, um den Tag innerhalb einer Woche als Wert zwischen 1 und 7 zu spezifizieren. Dabei ist 1 = Montag und 7 = Sonntag, die weiteren Tage ent­sprechend kalendarischer Reihenfolge.  Die Nutzung des Datenelements *dayOfWeek* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.73 | **DayVarType – month**  Das Datenelement *month* kann genutzt werden, um den Monat als ganzzahligen Wert zu beschreiben. Dabei ist 1 = Januar und 12 = Dezember, die weiteren Monate entsprechend kalen­darischer Reihenfolge.  Die Nutzung des Datenelements *month* ist **optional**. | Optional | Optional | muss implementiert werden |
| 2.74 | **DayVarType – year**  Das Datenelement *year* kann genutzt werden, um ein bestimmtes Jahr als ganzzahligen Wert zu beschreiben.  Die Nutzung des Datenelements *year* ist **optional**. | Optional | Optional | muss implementiert werden |
| 2.75 | **LogEntry**  Die Klasse *LogEntry* spezifiziert Logeinträge.  Eine Instanz der Klasse LogEntry:  **• kann** auf eine Instanz der Klasse *LogEvent* verweisen | Optional | Optional | muss implementiert werden |
| 2.76 | **LogEntry – parentRecordNumber**  Das Datenelement *parentRecordNumber d*ient der Markierung verketteter Ereignisse. Enthält die „Parent Record Number“ denselben Wert wie die „Record Number“, so handelt es sich um das Root-Element einer Kette von zusammenhängenden Ereignissen oder um ein Einzel-Ereignis. Ent­hält die „Parent Record Number“ den Wert der „Record Number“ eines zuvor bereits erzeugten Ereignisses, so handelt es sich um ein Kind-Element einer Kette von zusammenhängenden Ereignissen.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.77 | **LogEntry – recordNumber**  Das Datelement *recordNumber* ist der eineindeutige Bezeichner des Logeintrags. Dieser wird mit Ablegen des Eintrags im Logbuch durch die Geräte-Firmware erzeugt. Das Datenelement ist **optional** anzugeben. | Optional | Optional | muss implementiert werden |
| 2.78 | **LogEntry – repetitionCounter**  Das Datenelement *repetitionCounter* beschreibt den Wiederholungszähler und ist bei einem neu auftretenden Ereignis mit „1“ zu besetzen. Bei nachfolgenden Ereignissen kann der „Repetition Counter“ in Verbindung mit Logbüchern benutzt werden, um das mehrfache Einfügen der Ereignisse in ein Logbuch zu vermeiden. Dabei ist zu gewährleisten, dass immer der erste und letzte Logbuch-Eintrag vorhanden ist.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.79 | **LogEvent**  Die Klasse *LogEvent* spezifiziert Logereignisse.  Instanzen der Klasse sind **optional**. | Optional | Optional | muss implementiert werden |
| 2.80 | **LogEvent – evidence**  Das Datenelement *evidence* enthält, falls vorhanden, die Signatur, die im Zusammenhang mit dem Logereignis aufgetreten ist.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.81 | **LogEvent – destination**  Das Datenelement *destination* beinhalten, falls vorhanden die URL des Adressaten, an den die Aktion gerichtet ist, die das Ereignis ausgelöst hat.  Das Datenelement *destination* ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.82 | **LogEvent – type**  Das Datenelement *type* ist die eineindeutige Beschreibung des Ereignisses mit dem Datentyp „Evis.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.83 | **LogEvent – level**  Das Datenelement *level* beschreibt die dem Ereignis zugeordnete Rubrik als ein Element der Enumeration aus {1=INFO, 2=WARNING, 3=ERROR, 4=FATAL, 5=EXTENSION}.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | muss implementiert werden |
| 2.84 | **LogEvent – messageExtension**  Das Datenelement *messageExtension* liefert die Texte aus ‚Message‘, jene Textteile, die in der Information zum ‚type‘ an Stelle der Platzhalter zu verwenden sind. Die Reihenfolge der Texte muss mit der Reihenfolge der Platzhalter übereinstimmen.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.85 | **LogEvent – text**  Das Datenelement *text* liefert die textuelle Beschreibung des Logeintrags.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | muss implementiert werden |
| 2.86 | **LogEvent – outcome**  Das Datenelement *outcome* spezifiziert das Ergebnis der mit dem Ereignis verbundenen Aktion.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | muss implementiert werden |
| 2.87 | **LogEvent – subjectIdentity**  Das Datenelement *subjectIdentity* beschreibt die Identität der Quelle, die das Ereignis ausgelöst hat.  Die Nutzung des Datenelements *subjectIdentity* ist optional. | Optional | Optional | keine zwingende Implementierung |
| 2.88 | **LogEvent – secondsIndex**  Das Datenelement *secondsIndex* beschreibt den monoton steigenden Sekundenindex aus dem SMGW.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.89 | **LogEvent – timestamp**  Das Datenelement *timestamp* beschreibt den Zeitstempel mit Zeitpunkt, wann das Ereignis einge­treten ist.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | muss implementiert werden |
|  | **LogEvent – destination**  Das Datenelement *destination* beschreibt die Adresse des Kommunikationspartners beim Verbindungsauf/-abbau und Datenaustausch.  Die Nutzung des Datenelements *destination* ist optional. |  |  |  |
|  | **LogEvent – userIdentity**  Das Datenelement *userIdentity* beschreibt die Identität des Users, der das Ereignis ausgelöst hat.  Die Nutzung des Datenelements *userIdentity* ist optional. |  |  |  |
|  | **LogEvent – eventKindVendor**  Das Datenelement *eventKindVendor* beschreibt herstellerspezifische Parameter welche ggf. für eine Signaturprüfung benötigt werden.  Die Nutzung des Datenelements *eventKindVendor* ist optional. |  |  | PrüDi |
| 2.90 | **Evis**  Die Klasse *Evis* beschreibt ein LogEvent eineindeutig.  Die Nutzung der Klasse *Evis* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.91 | **Evis – deviceType**  Das Datenelement *deviceType b*eschreibt eine physische Ausprägung eines Gerätes. Im FNN werden diese Einheiten typischerweise über ein Lastenheft beschrieben. Z. B. FNN Lastenheft Basiszähler etc. Hinweis: Gerätetyp 00 beschreibt ein nicht spezifiziertes Gerät.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.92 | **Evis – eventId**  Das Datenelement *eventId* identifiziert ein Ereignis übergreifend über Gerätetyp, Modul und Funktion. Sie wird eindeutig pro vendorId beschrieben. Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.93 | **Evis – eventSubId**  Das Datenelement *eventSubId* identifiziert die Detailmeldung.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.94 | **Evis – length**  Das Datenelement *length* beschreibt die Länge der Datenstruktur.  Das Datenelement *length* ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.95 | **Evis – function**  Das DateneIement *function* identifiziert eine funktionale Einheit innerhalb des Moduls. Die funktionale Einheit kann als Hardware oder Software ausgeprägt sein. Hinweis: Funktion 00 beschreibt eine nicht spezifizierte Funktion.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.96 | **Evis – module**  Das Datenelement *module* identifiziert ein Modul als Quelle innerhalb eines Gerätes. Das Modul kann als Hardware oder Software ausgeprägt sein. Hinweis: Modul 00 beschreibt ein nicht spezifiziertes Modul.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.97 | **Evis – vendorId**  Das Datenelement *vendorId* gibt die Verwendung nach Flag-Kennung „the flag association“ an (siehe FNN COSEM Definition).  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.98 | **Evis – version**  Das Datenelement *version* spezifiziert die Version des Datentyps.  Das Datenelement ist **optional** anzugeben. | Optional | Optional | keine zwingende Implementierung |
| 2.99 | **MessageExtensionType**  Die Klasse *MessageExtensionType* liefert jene Textteile, die in der Information zum ‚event\_type‘ an Stelle der Platzhalter zu verwenden sind. Die Reihenfolge der Texte muss mit der Reihenfolge der Platzhalter übereinstimmen und wird über das Datenelement *id* festgelegt.  Die Nutzung der Klasse *MessageExtensionType* ist **optional**. | Optional | Optional | keine zwingende Implementierung |
| 2.100 | **MessageExtensionType – id**  Das Datenelement *id* der Klasse *MessageExtensionType* identifiziert einen Textteil des Logeintrages und legt die Reihenfolge der Texte fest.  Eine Instanz der Klasse *MessageExtension* **muss** immer ein Datenelement vom Typ *id* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.101 | **MessageExtensionType – text**  Das Datenelement enthält den Text des Logeintrags  Eine Instanz der Klasse *MessageExtension* **muss** immer ein Datenelement vom Typ *text* enthalten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.102 | **MathOperation**  Die Klasse *MathOperation* spezifiziert mathematische Operationen zwischen Messwertlisten, um dadurch Zählerbeziehungen abzubilden.  Eine Instanz der Klasse *MathOperation*:  **• muss** auf eine Instanz der Klasse *Operand1* verweisen  **• muss** auf eine Instanz der Klasse *Operator* verweisen  **• muss** auf eine Instanz der Klasse *Operand2* verweisen | Optional | Optional | keine zwingende Implementierung |
| 2.103 | **Operand1**  Die Klasse *Operand1* spezifiziert den ersten Operand einer mathematischen Operation zwischen Messwertlisten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.104 | **Operand1 – meterReadingId**  Das Datenelement *meterReadingId* spezifiziert den Identifikator der Messwertreihe dessen Werte mathematisch mit Werten einer weiteren Messwertreihe verknüpft werden sollen. Es **muss** eine Messwertliste über das Datenelement *meterReadingId* referenziert werden. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.105 | **Operator**  Die Klasse *Operator* spezifiziert den mathematischen Operator der mathematischen Operation als Container für das Datenelement *operator*. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.106 | **Operator – operator**  Das Datenelement *operator* spezifiziert den mathematischen Operator der mathematischen Operation als Element der Enumeration aus {1=Addition, 2=Subtraktion, 3=Division, 4=Multiplikation}. Für eine mathematische Operation **muss** ein Operator angegeben werden. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.107 | **Operand2**  Die Klasse *Operand2* spezifiziert den zweiten Operand einer mathematischen Operation zwischen Messwertlisten. | Erforderlich | Erforderlich | keine zwingende Implementierung |
| 2.108 | **Operand2 – meterReadingId**  Das Datenelement *meterReadingId* spezifiziert den Identifikator der Messwertreihe dessen Werte mathematisch mit Werten einer weiteren Messwertreihe verknüpft werden sollen. Es **muss** eine Messwertliste über das Datenelement *meterReadingId* referenziert werden. | Erforderlich | Erforderlich | keine zwingende Implementierung |

Tabelle 3: Prüfungsdaten und deren Verwendung in TRuDI