

Lastenheft TRuDI

Arbeitskreis Bundesdisplay

Autor: Arbeitskreis Bundesdisplay

Datum: 28.07.2017

Seiten: 48

Einstufung: Eingeschränkt

Versionshistorie

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Name | Änderungen |
| 0.1 | 2017-06-12 | Janosch  Wagner | Dokument erstellt |
| 0.2 | 2017-06-14 | AKBD | Weitere Kommentare zu den einzelnen Kapiteln, Aufteilung der Kapitelerarbeitung. |
| 0.3 | 2017-06-26 | Janosch  Wagner | Beiträge von Hr. Exner, Hr. Giebel, Hr. Weingardt und Hr. Magin eingefügt.  Redaktionelle und inhaltliche Überarbeitungen. |
| 0.4 | 2017-06-27 | Sven Krause | Ergänzung Sektion 5. |
| 0.5 | 2017-06-28 | AKBD | Überarbeitung in der gemeinsamen Telko |
| 0.6 | 2017-06-28 | René Giebel | Kapitel 4 überarbeitet |
| 0.7 | 2017-06-28 | Janosch  Wagner | Formatierung überarbeitet, Redaktionelle Änderungen, Anforderungsnummern eingefügt |
| 0.8 | 2017-06-29 | Janosch  Wagner | Rückmeldung Hr. Weingardt, Hr. Blümer aus AKBD Kommentierung eingepflegt |
| 0.9 | 2017-06-30 | Gerald Exner | Formatierung und Typos korrigiert |
| 1.0 | 2017-07-03 | Janosch  Wagner | Rückmeldungen Landis&Gyr eingepflegt |

Tabelle 1: Versionshistorie

Inhalt

1 Zielbestimmung 5

2 Produkteinsatz 7

2.1 Beteiligte Systemkomponten 8

2.2 Schnittstellen 8

2.3 Grundliegende Funktionen 9

2.3.1 Visualisierungsfunktion 9

2.3.2 Transparenzfunktion 10

3 Produktfunktionen 11

3.1 Mehrspartenbetrieb 11

3.2 Unterstützte Registrierperiode 11

3.3 Herstellerspezifischer Adapter 11

3.3.1 Beschreibung 11

3.3.2 Integration 11

3.4 Tarifierungsmodule 12

3.4.1 Beschreibung 12

3.4.2 Integration 12

3.4.3 Tarifierungsmodul 1: zeitvariable Tarife (TAF-2) 12

3.4.4 Prüfsumme der Tarifierungsinformationen 13

3.5 Vorbereitung WAN Lösung 13

3.6 Prüfung innere Signatur 14

4 Schnittstelleninformationen 15

4.1 IF\_Adapter\_TRuDI 15

4.2 IF\_Lieferant\_TRuDI 16

5 Produktcharakteristiken 18

5.1 Systemumgebung 18

5.1.1 Software 18

5.1.1.1 Plattform 18

5.1.1.2 Komponenten 18

5.1.1.3 Sicherheit 18

5.1.2 Hardware 19

5.1.2.1 Mindestanforderungen Installationsbasiertes System (Windows 7) 19

5.1.2.2 Mindestanforderungen Livesystem (Heise Meterix) 19

5.2 Nicht-funktionale / sonstige Anforderungen 19

5.2.1 Anforderungen aus PTB-A50.8 19

5.2.2 Schulung 19

5.2.3 Dokumentation 19

5.2.4 Lizenzmodell 20

5.2.5 Testumgebung 20

6 Verzeichnisse 21

6.1 Abbildungsverzeichnis 21

6.2 Tabellenverzeichnis 21

6.3 Anforderungsverzeichnis 21

7 Anhang 1: Regeln zur Anwendung des Datenmodells der VDE AR 2418-6 22

7.1 Basisdaten 23

7.2 Prüfungsdaten 34

# Zielbestimmung

Mit TRuDI (Transparenz- und Display-Software) stellt die Intitiative Bundesdisplay eine herstellerübergreifende, standardisierte Visualisierungslösung bereit, die die Anforderungen des MsbG (insbesondere §35, §62), der PTB-A50.8 erfüllt und die im Rahmen der Vorgaben des Bundesamts für Sicherheit in der Informationstechnik nutzbar ist.

TRuDI bietet dabei eine Display Funktion, mit der Messwerte, die im SMGW vorhanden sind für den Letztverbraucher angezeigt werden. Darüber hinaus steht eine sogenannte Transparenz Funktion zur Verfügung. Im Rahmen dieses funktionalen Merkmals ist der Letztverbraucher mit Hilfe der Software in der Lage, Tarifrechnungen, die auf Basis der Messwerte des SMGWs in der Systemlandschaft des Lieferanten durchgeführt hat lokal nachzuvollziehen und damit seine Rechnung zu überprüfen.

Die TRuDI Software Lösung bietet die Möglichkeit einer Nutzung über die HAN oder über die WAN Schnittstelle des SMGWs.

Die Visualisierung über einen online zugänglichen Display-Datendienst (d.h. WAN Schnittstelle) ist ein Komfortangebot für den Letztverbraucher, der die eichrechtlich geforderte lokale Sichtanzeige nicht ersetzt aber im Sinne des GDEW dem Endverbraucher bei dessen Zustimmung einen einfachen Zugang zu seinen Verbrauchswerten ermöglicht.

Die eichrechtlich relevante Anzeige kann mit TRuDI über die HAN Schnittstelle des SMGWs erfolgen. TRuDI ist dabei durch ein integriertes Adapter Konzept mit allen gängigen SMGWs einsetzbar und erlaubt dem Letztverbraucher die geforderte Rechnungsprüfung.

Das Verfügbarmachen der Hilfsmittel zur Rechnungskontrolle ist Pflicht des Messwertverwenders (MessEG § 33 (3)). Mit TRuDI sorgt die Initiative BundesDisplay für ein als Rechnungsprüfungshilfsmittel geignetes Software-Produkt. TRuDI kann dazu auf dem Computer des Letztverbrauchers genutzt werden. Die Lösung ist grundsätzlich Betriebssystem unabhängig, zur Rechnungsprüfung kann auch die Live-Linux Distribution c’t Meterix eingesetzt, die speziell als Systemumgebung zur Rechnungsprüfung konzipiert wurde.

Die TRuDI Software Lösung der Initiative Bundesdisplay kann von den Herstellern im Rahmen der Baumusterprüfbescheinigungsverfahren genutzt werden. Durch die Nutzbarkeit mit allen SMGWs reduziert sie den Prüfungsaufwand im Rahmen der Verfahren.

Das vorliegende Lastenheft enthält neben erläuterndem Fließtext auch dezidierte Anforderungen. Die Anforderungen sind gesondert markiert und müssen vom Auftragnehmer im Pflichtenheft hinsichtlich ihrer Umsetzung beschrieben werden. Der Auftragnehmer muss die im Pflichtenheft beschriebenen Maßnahmen in seiner Implementierung berücksichtigen und deren korrekte Funktion durch eine entsprechende Testabdeckung gewährleisten. Auch in der Dokumentation muss die Implementierung der Maßnahmen beschrieben werden.

Die im vorliegenden Lastenheft beschriebenen Anforderngen gliedern sich in KANN-, MUSS- und SOLL-Anforderungen.

MUSS Anforderungen sind Anforderungen, die der Auftragnehmer bei der Umsetzung des Produkts zwingend berücksichtigen muss.

SOLL Anforderungen sollten vom Auftragnehmer bei der Umsetzung berücksichtigt werden. Wenn eine SOLL Anforderung nicht umgesetzt werden kann, dann ist das im Rahmen der Pflichtenhefterstellung und unter Angabe der dafür ursächlichen Gründe mit dem Auftraggeber abzustimmen.

KANN Anforderungen können vom Auftragnehmer umgesetzt werden, sind jedoch optionaler Natur. Der Auftraggeber hat keinen Anspruch auf eine Umsetzung dieser Anforderungen.

Grundsätzlich ist die Umsetzung der PTB-A50.8 vorragig zu behandeln. Die TRuDI Software Lösung muss dazu die Anforderungen aus PTB-A 50.8 Kapitel 5 und 7 direkt oder sicherheitsniveau-gleich erfüllen.

# Produkteinsatz

In der nachfolgenden Abbildung wird der Produkteinsatz schematisch erläutert.

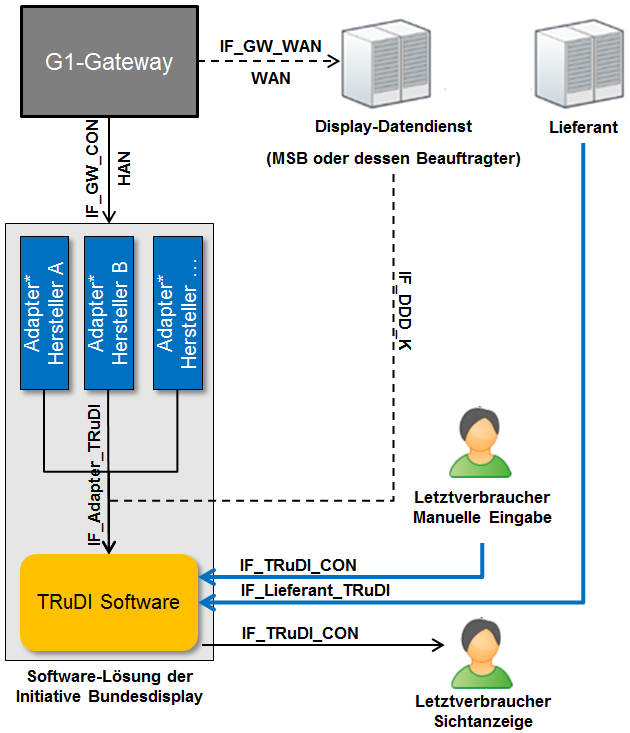


Abbildung 1: Übersicht Systemarchitektur TRuDI

**Durchgezogene schwarze Linien/Pfeile** stellen für die eichrechtliche Zulassung verpflichtende Schnittstellen / Datenflüsse dar.

**Durchgezogene blaue dicke Linien/Pfeile** kennzeichnen alternative Schnittstellen/Datenflüsse, wobei eine der beiden Alternativen umgesetzt werden muss.

**Gestrichelte schwarze Linien/Pfeile** werden für optionale Schnittstellen/Datenflüsse verwendet, die den Visualisierungskomfort des Endkunden erhöhen können.

Die TRuDI Software Lösung der Initiative Bundesdisplay wird an die HAN Schnittstelle des SMGWs angebunden und kann die Verbrauchswerte des jeweiligen Letztverbrauchers über diese Schnittstelle auslesen.

Über den herstellerspezifischen Adapter werden die Verbrauchswerte in ein einheitliches Datenformat (VDE AR 2418-6) transformiert, das von der TRuDI Software verarbeitet werden kann.

TRuDI kann die Verbrauchswerte eichrechtskonform für den Kunden visualisieren. Darüber hinaus können Tarifierungsoperationen lokal nachgerechnet und damit überprüft werden, die in den Systemen des Lieferanten durchgeführt wurden.

Dazu können die Tarifierungsinformationen, die benötigt werden, um die Rechnung des Lieferanten lokal zu überprüfen in die TRuDI Software importiert werden.

Eine Übermittlung der Verbrauchswerte über den Daten-Display-Dienst (d.h. indirekt über die WAN Schnittstelle des SMGWs) ist ebenfalls optional vorgesehen und dient als Komfort Angebot für den Letztverbraucher, nicht jedoch als eichrechtlich relevante Sichtanzeige.

TRuDI\_001\_MUSS Eine Software-Instanz muss zeitgleich immer nur ein SMGW bedienen.

## Beteiligte Systemkomponten

Folgende Systemkomponenten sind in Abbildung 1 dargestellt und interagieren in unterschiedlicher Form mit der TRuDI Software:

* **G1-Gateway:** Smart Meter Gateway mit Zertifizierung nach Schutzprofil (PP-0073), das abrechnungsrelevant mit den Tarifanwendungsfällen TAF-1, TAF-2, TAF-6 und TAF-7 oder einer Untermenge dieser TAF eingesetzt werden soll.
* **Herstellerspezifischer Software Adapter:** Softwarekomponenten, die das herstellerspezifische HAN Protokoll auf VDE AR E 2418-6 (Greenbutton/OFFIS) wandelt. Der Adapter wird von jedem Hersteller in einer einheitlich vorgegebenen Form zugeliefert und vom Hersteller der TRuDI Software in die Software-Lösung der Initiative Bundesdisplay integriert. Die Software-Lösung erkennt automatisch, welcher Adapter bei der Anbindung eines SMGWs eines spezifischen Hersteller zu verwenden ist. Details zum herstellerspezifischen Software-Adapter sind in Kapitel 3.3 enthalten.
* **TRuDI:** Softwarekomponente der Software-Lösung, die für die einheitliche Visualisierungs- und Transparenzfunktion zuständig ist. Diese Softwarekomponente ist für alle Hersteller identisch.
* **Display-Datendienst:** Der Display-Datendienst ist die Online-Portallösung im Sinne MsbG, §35, (3) und kann die Verbrauchsdaten an die TRuDI Software bereitstellen und dazu ebenfalls das AR 2418-6-Format verwenden. Die Bereitstellung erfolgt mit Zustimmung des Letztverbrauchers und erhöht den Visualisierungskomfort desselben. Eine Visualisierung aktueller Informationen im Sinne der PTB-A 50.8, 4.1.3.2.1 ist derzeit noch nicht möglich.

## Schnittstellen

Die folgenden Schnittstellen sind bei der Interaktion von TRuDI mit den umgebenden Systemkomponenten beteiligt:

* **IF\_GW\_CON:** Herstellerspezifisches Protokoll WAN Schnittstelle (Webservice, Webserver, etc.)
* **IF\_GW\_WAN:** Protokoll zur Übermittlung von Messwerten über das WAN an EMTs
* **IF\_Adapter\_TRuDI:** Die Schnittstelle folgt der VDE AR 2418-6, siehe auch Detailspezifikation in Kapitel 4.1
* **IF\_DDD\_K:** Die Schnittstelle folgt der VDE AR 2418-6, siehe auch Detailspezifikation in Kapitel 4.1
* **IF\_Lieferant\_TRuDI:** Die Schnittstelle folgt der VDE AR 2418-6, siehe auch Detailspezifikation in Kapitel 4.2.
* **IF\_TRuDI\_CON:** Visualisierungsschnittstelle, über die dem Letztverbraucher die zugehörigen Verbrauchsinformationen angezeigt werden (schwarze Linie). Alternative zu IF\_Lieferant\_TRuDI, über die der Letztverbraucher seine Vertragsdaten zur Rechnungsprüfung manuell über eine definierte Eingabemaske eingeben kann (blaue Linie).

## Grundliegende Funktionen

In den nachfolgenden Unterkapiteln werden die grundliegenden Funktionen beschrieben, die durch TRuDI abgebildet werden müssen. Teilweise erfolgt eine vertiefende Spezifikation in Kapitel 3, daher sind die beiden Unterkapitel eher als Übersicht konzipiert.

### Visualisierungsfunktion

Für TAF-1/2/6 stellt TRuDI eine Anzeige-Funktion (gemäß PTB-A50.8) zur Verfügung, d.h. die Messwerte, die vom SMGW bereitgestellt werden, werden lediglich visualisiert. Eine Kontrolle oder weiterführende Verrechnung (außer der im Folgenden angegebenen Operationen) dieser Werte findet nicht statt.

TRuDI\_002\_MUSS FÜR TAF-1, TAF-2 und TAF-6 müssen die Messwerte spartenübergreifend, die vom SMGW bereitgestellt werden entsprechend der Anforderungen aus PTB-A50.8 visualisiert werden.

Für TAF-6 handelt es sich dabei um die Werte, die jeweils zu Beginn des abrechnungstechnischen Kalendertages erfasst werden und für einen Zeitraum von 42 Tagen verfügbar sind.

TRuDI\_003\_MUSS TRuDI muss aus den bereitgestellten Verbrauchswerten historische tages-, wochen-, monats- und jahresbezogene Energieverbrauchswerte für die letzten 36 Monate berechnen und für den Letztverbraucher darstellen (d.h.: „kWh pro Tag 1..1095“, „kWh pro Woche 1.. 156“ ,kWh pro Jahr 1..3“), sofern die entsprechenden Daten über IF\_Adapter\_TRuDI verfügbar sind.

Die historischen tages-, wochen-, monats- und jahresbezogenen Energieverbrauchswerte sind nur in soweit zu berechnen und darzustellen, wie auch Verbrauchswerte über den herstellerspezifischen Adapter zugeliefert werden. D.h. wenn nur Verbrauchswerte über einen Zeitraum von 12 Monaten zugeliefert werden, muss TRuDI die historischen Werte ebenfalls nur für 12 Monate berechnen und darstellen.

TRuDI\_004\_MUSS Soweit vorhanden muss TRuDI Zählerstandsgänge für die letzten 36 Monate darstellen.

TRuDI\_005\_MUSS TRuDI muss die Logdaten des Letztverbraucher-Logs, die über die Schnittstelle IF\_Adapter\_TRuDI bereitgestellt werden für den Letztverbraucher entsprechend der Anforderungen aus PTB-A50.8 darstellen.

TRuDI\_006\_MUSS TRuDI muss keine weitere Verarbeitung der Messwerte durchführen sondern diese lediglich visualisieren. Lücken in Messwertreihen, das Fehlerregister in TAF-2 und invalide Messwerte müssen entsprechend gekennzeichnet für den Letztverbraucher dargestellt werden.

TRuDI\_007\_MUSS Der Zeitbereich zur Abfrage der Messwerte über IF\_Adapter\_TRuDI wird bei der Visualisierungsfunktion durch den Letztverbraucher vorgeben. TRuDI muss daher eine Schnittstelle vorsehen, über den der Letztverbraucher den Zeitbereich eingeben kann.

### Transparenzfunktion

Für TAF-7 und eine nachgelagerte Tarifierung beim Lieferanten stellt TRuDI eine Transparenz-Funktion (gemäß PTB-A50.8) zur Verfügung, d.h. die vom SMGW bereitgestellten Messwerte werden mit einer vom Lieferanten bereitgestellten Tarifierungsinformation verrechnet. Das Ergebnis wird dem Letztverbraucher zur Rechnungsprüfung angezeigt.

Die Tarifierungsinformationen des Lieferanten werden in Form eines ausgerollten Kalenders bereitgestellt. Die Tarifierungsinformation enthält daher zu jeder Registrierperiode die Information, in welches Register die Energiemenge dieser Registrierperiode zu zählen ist. Die genaue Datenstruktur wird in Kapitel 7 und in VDE AR 2418-6 beschrieben.

TRuDI\_008\_MUSS Für TAF-7 muss eine Transparenzfunktion bereitgestellt werden, die in der Lage ist, aus dem Zählerstandsgang, der über den herstellerspezifischen Adapter übermittelt wird und den Tarifierungsinformationen, die vom Lieferanten zugeliefert werden, Tarifregister zu berechnen und damit dem Letztverbraucher eine Rechnungsprüfung zu ermöglichen.

TRuDI\_009\_MUSS Falls keine Tarifierungsinformationen zur Verfügung stehen, muss TRuDI den TAF-7 Zählerstandsgang, der vom SMGW bereitgestellt wird, entsprechender der Anforderungen aus PTB-A50.8 visualisieren.

TRuDI\_010\_MUSS Der Zeitbereich zur Abfrage der Messwerte über IF\_Adapter\_TRuDI wird bei der Transparenzfunktion über die Tarifierungsinformation vorgegeben. Messwerte müssen über den Zeitbereich abgerufen werden, für den auch die Tarifierungsinformation (ausgerollter Kalender) vorhanden ist.

# Produktfunktionen

Die Software TRuDI kann durch den Letztverbraucher mit der Transparenz-Funktion zur Prüfung von Rechnungen bei nachgelagerter Tarifierung verwendet werden.Darüber hinaus können über die Visualisierungsfunktion von TRuDI die Messwerte aus dem SMGW für den Letztverbraucher eichrechtskonform angezeigt werden. Ergänzend zur Rechnungsprüfung und Messwertvisualisierung zeigt TRuDI die Logdaten des Letztverbrauchers an.

## Mehrspartenbetrieb

Prinzipiell ist der Mehrspartenbetrieb vorzusehen.

TRuDI\_011\_MUSS Die Transparenz-Funktion wird nur für die Elektrizitätsmessung bereitgestellt.

TRuDI\_012\_MUSS Die Darstellung eines Zählerstandsgangs ist mit den Daten der Originären Messwerteliste möglich.

TRuDI\_013\_MUSS Die Darstellung eines Zählerstandsgangs ist mit den Daten der Abrechnungsturnusliste möglich.

TRuDI\_014\_MUSS Der Umfang einer Darstellung eines Zählerstandsgangs ist zeitlich einschränkbar (Filter ab einschließlich bis einschließlich).

## Unterstützte Registrierperiode

TRuDI\_015\_MUSS Die kleinste unterstützte Registrierperiode für die Transparenz-Funktion beträgt 15 Minuten.

TRuDI\_016\_MUSS In der Parametrisierung des Tarifanwendungsfalls erlaubt Registrierperioden, die ein ganzzahliges Vielfaches von 15 Minuten sind. Die Transparenz-Funktion muss diese Registrierperioden unterstützen.

## Herstellerspezifischer Adapter

Die folgenden Unterkapitel enthalten Anforderungen an den herstellerspezifischen Adapter.

### Beschreibung

Ein herstellerspezifischer Adapter normiert die die durch die verschiedenen Ausprägungen des SMGw vorhandenen Datenformate (siehe Abbildung 1: Übersicht Systemarchitektur TRuDI).

### Integration

Die Integration der durch den Hersteller gelieferter Adapter muss durch die Software TRuDI in geeigneter Weise erfolgen.

TRuDI\_017\_MUSS Die durch die Hersteller zu benutzende Infrastruktur und Techniken der Schnittstelle (z.B. API, Programmaufruf, separate DLL o.ä.) sind zu beschreiben.

TRuDI\_018\_MUSS Die benutzten Datenstrukturen sind zu beschreiben.Die Anforderungen TRuDI\_042\_MUSS, TRuDI\_043\_MUSS, TRuDI\_044\_MUSS müssen eingehalten werden.

TRuDI\_019\_MUSS Für den Adapter wird ein Rumpf in Code-Form zur Verfügung gestellt.

TRuDI\_020\_MUSS Für den Adapter wird ein Testprogramm zur Verfügung gestellt, dass die eingehenden und ausgehenden Daten simulieren kann.

## Tarifierungsmodule

### Beschreibung

Tarifierungsmodule sind Softwareteile, die aus einem eingehenden Messdatenstrom einen tarifierten Messdatenstrom ableiten. Dabei wird die Abfolge der eingehenden Registerwerte in ein oder mehrere abgeleitete Registerwerte gewandelt. Das Tarifierungsmodul wird nur benutzt, wenn keine Tarifierung im SMGW erfolgt (TAF-7 im Gateway). In diesem Fall kann das Gateway nur die Originäre Messwerteliste bereitstellen. Die Ableitung der Abrech-nungsturnusliste ist nur möglich, wenn die Tarifierungsregel (das Modell) durch den EMT zur Verfügung gestellt wird. Für Erweiterung durch den Tarifierungsanbieter ist es notwendig, die Berechnung unabhängig von der Darstellung durchzuführen.

### Integration

Die Integration der durch den Anbieter/Hersteller gelieferter Module muss durch die Software TRuDI in geeigneter Weise erfolgen.

TRuDI\_021\_MUSS Die Schnittstelle entspricht technisch dem in Kapitel 3.3.2 beschrieben Verfahren.

TRuDI\_022\_MUSS Die benutzten Datenstrukturen sind zu beschreiben.

TRuDI\_023\_MUSS Für das Modul wird ein Rumpf in Code-Form zur Verfügung gestellt.

TRuDI\_024\_MUSS Für das Modul wird ein Testprogramm zur Verfügung gestellt, dass die eingehenden und ausgehenden Daten simulieren kann.

### Tarifierungsmodul 1: zeitvariable Tarife (TAF-2)

Das Tarifierungsmodul wandelt die durch das Gateway bereitgestellte Originäre Messwer-teliste in eine Liste mit mehreren Tarifstufen um.

TRuDI\_025\_MUSS Die Tarifierungsinformationen für die Transparenzfunktion werden durch einen ausgerollten Kalender zur Verfügung gestellt.

TRuDI\_026\_MUSS Zur Energiemenge jeder Registrierperiode in der Originären Messwerteliste muss eine Bewertung erfolgen, d.h. es muss eine Zuordung dieser Energiemenge zu einer Tarifstufe im ausgerollten Kalender vorhanden sein. Ist kein Punkt vorhanden, liefert die Transparenzfunktion einen Fehler.

TRuDI\_027\_MUSS TRuDI muss in der Lage sein, die benötigten Tarifierungsinformationen über einen Webservice vom EMT einzulesen.

TRuDI\_028\_MUSS TRuDI muss einen aussagekräftigen Fehler anzeigen und die weitere Verarbeitung beenden, wenn der ausgerollte Kalender im angegebenen Zeitraum unvollständig ist.

TRuDI\_029\_MUSS Wenn eine Lücke im Zählerstandsgang exisitert, der über IF\_Adapter\_TRuDI an die TRuDI Software übergeben wird und im Zeitraum, für den keine Messwerte vorliegen, keine Tarifumschaltung stattfindet, muss TRuDI die Energiemenge ins entsprechende Tarifregister zählen. Die Tarifmenge darf nicht ins Fehlerregister gezählt werden.

TRuDI\_030\_MUSS Wenn eine Lücke im Zählerstandsgang exisitert, der über IF\_Adapter\_TRuDI an die TRuDI Software übergeben wird und im Zeitraum, für den keine Messwerte vorliegen, eine Tarifumschaltung stattfindet, muss TRuDI die Energiemenge ins Fehlerregister zählen und diese Energiemenge für den Kunden kenntlich machen.

TRuDI\_031\_MUSS TRuDI muss eine Schnittstelle bereitstellen, über die der Letztverbraucher die URI des Webservices, über den die Tarifierungsinformation vom EMT/Lieferanten bezogen werden kann, eingeben kann.

TRuDI\_032\_MUSS TRuDI muss eine Schnittstelle bereitstellen, über die der Letztverbraucher manuell eine Datei mit Tarifierungsinformationen in die Software importieren kann.

### Prüfsumme der Tarifierungsinformationen

Die Tarifierungsinformationen für die Transparenz-Funktion werden durch den EMT zur Ver-fügung gestellt. Die Tarifierungsinformation wird durch eine Prüfsumme geschützt.

TRuDI\_033\_MUSS Als Prüfsumme der Tarifierungsinformation wird SHA3 oder RIPEMD-160 durch den EMT benutzt. TRuDI muss beide Verfahren unterstützen.

TRuDI\_034\_MUSS TRuDI muss die Prüfsumme der Tarifierungsinformation anzeigen, damit der Letztverbraucher einen Vergleich mit der Prüfsumme durchführen kann, die er vom Lieferanten erhalten hat.

## Vorbereitung WAN Lösung

Die durch die Transparenzfunktion benutzten Daten werden bei der eichrechtlich relevanten Nutzung der Software aus der Letztverbraucherschnittstelle des Gateways geladen. Als Komfortfunktion soll es dem Letztverbraucher darüberhinaus auch ermöglicht werden, die Daten über das Weitbereichsnetz von einem Daten-Display-Dienst zu beziehen.

TRuDI\_035\_SOLL Die Schnittstelle IF\_Adapter\_TRuDI (siehe Kapitel 3.3.2) ist so zu gestalten, dass anstatt des herstellerspezifischen Adapters auch ein externer Webservice benutzt werden kann. Hier bietet sich die Implementierung eines speziellen Adapters, der einen Webservice aufruft, an.

TRuDI\_036\_SOLL Falls die Übermittlung der Messwerte über WAN erfolgt, muss die Prüfung der inneren Signatur gemäß Kapitel 3.6 erfolgen.

## Prüfung innere Signatur

Im SMGW können aufgezeichnete und abgeleitete Messwerte durch eine innere Signatur geschützt. Das Verfahren ist in FNN LH SMGW Funktionen beschrieben.

TRuDI\_037\_KANN TRuDI kann im Auslieferungszustand ein oder mehrere Root-Zertifikate zur Kettenprüfung beinhalten.

Falls kein Root-Zertifikat vorhanden ist, ist die Kettenprüfung nicht möglich.

TRuDI\_085\_KANN TRuDI kann eine Schnittstelle bereitstellen, über die der Letztverbraucher das Signaturzertifikat (GW\_WAN\_SIG\_CRT) des SMGW einschließlich des ausstellenden SubCA Zertifikats in die Software importieren kann.

TRuDI\_086\_KANN TRuDI kann eine Schnittstelle bereitstellen, über die das Signaturzertifikat (GW\_WAN\_SIG\_CRT) des SMGW einschließlich des ausstellenden SubCA Zertifikats vom herstellerspezifischen Adapter über IF\_Adapter\_TRuDI bereitgestellt werden kann.

TRuDI\_038\_KANN Die Prüfung der Signatur kann durch TRuDI ausgeführt werden, wenn die Signatur durch den herstellerspezifischen Adapter übergeben wird und ein ensprechendes Zertifikat einschließlich der entsprechenden Zertifikatskette in TRuDI vorliegt.

TRuDI\_039\_KANN Das Ergebnis der Signatur Prüfung muss dem Letztverbraucher dargestellt werden.

TRuDI\_040\_KANN Die Prüfung der Signatur schließt die Prüfung der Zertifikatskette des Signaturzertifikats mit ein.

TRuDI\_041\_KANN TRuDI kann eine Schnittstelle bereitstellen, die dem Letztverbraucher die Eingabe von einem oder mehreren öffentlichen Schlüsseln ermöglicht, die zur Signaturprüfung verwendet werden. In diesem Fall entfallen die Notwendigkeit eines Zertifikat-Imports und die Prüfung der Zertifikatskette.

# Schnittstelleninformationen

Das Kapitel Schnittstelleninformationen beschreibt die einheitliche Schnittstelle zur Kommunikation mit dem SMGw eines jeweiligen Herstellers.

Darüber hinaus definiert das vorliegende Kapitel die Schnittstelle zum jeweiligen Lieferanten, um die kundenspezifischen Vertragsdaten zum Abgleich der Messdaten mit der entsprechenden Rechnung abzurufen.

## IF\_Adapter\_TRuDI

Der Auftragnehmer (DZG) beschreibt IF\_Adapter\_TRuDI auf der Basis der AR2418-6.

Ziel der Umsetzung ist eine einheitliche herstellerunabhängige Schnittstelle zur Anbindung der Herstellerabhängigen Letztverbraucherlösung.

Die AR2418-6 verweist zusätzlich auf die „Green Button Spezifikation“.

TRuDI\_042\_MUSS Die Schnittstelle IF\_Adapter\_TRuDI muss AR\_2418-6.xsd in der Version 1.0 vom 14.06.2017 entsprechen.

TRuDI\_043\_MUSS Die Schnittstelle IF\_Adapter\_TRuDI muss der VDE AR\_2418-6 Version 1.0 entsprechen, die derzeit noch in redaktioneller Bearbeitung ist. Die relevanten Auszüge für die Schnittstelle sind jedoch abgeschlossen, können für die Implementierung verwendet werden und sind in diesem Dokument in Kapitel 7 eingefügt.

TRuDI\_044\_MUSS Die TRuDI Software muss die Beispiel XML Dateien IF\_Adapter\_TRuDI\_DatenTAF2.xml und IF\_Adapter\_TRuDI\_DatenTAF7.xml verarbeiten können, die diesem Dokument beigefügt sind.

Die Informationen, die über IF\_Adapter TRuDI zugeliefert werden entsprechen den FNN Vorgaben und enthalten beispielsweise Tarifregister, Fehlerregister oder Zählerstände.

Übergabedaten beim Aufruf des Adapters durch TRuDI sind somit:

TRuDI\_045\_MUSS HAN-Zugangsparameter (IP-Adresse / Port)

TRuDI\_046\_MUSS Zugangsdaten des Letztverbrauchers (es sind beide Zugangsmöglichkeiten wie Benutzername und Passwort bzw Zertifikat zu unterstützen)

TRuDI\_047\_KANN Gateway ID

TRuDI\_048\_MUSS Zeitraum bzw. Abrechnungsperiode

TRuDI\_049\_MUSS tafID

Die TRuDI Software muss weiterhin folgende funktionalen Merkmale an der Schnittstelle IF\_Adapter\_TRuDI umsetzen:

TRuDI\_050\_MUSS Herstellerspezifische Eigenschaften sind mittels eines zusätzlichen Datenfeldes zu übertragen.

TRuDI\_051\_MUSS TRuDI muss die Zugangsdaten zum SMGw übergeben, der Herstelleradapter organisiert die Verbindung und Auslesung des SMGw.

TRuDI\_052\_MUSS TruDI bietet eine Möglichkeit, die Daten aus der Schnittstelle IF\_Adapter\_TRuDI in Richtung der TRuDI unverändert als xml-Datei nach AR2418-6 zu exportieren.

TRuDI\_053\_MUSS Der Auftragnehmer (DZG) muss eine Spezifikation der unterlagerten Schichten (Transportschicht) der Schnittstelle IF\_Adapter\_TRuDI bereitstellen, die sich am Stand der Technik orientiert.

TRuDI\_054\_MUSS Vom Adapter werden bei der Abfrage eines Zeitbereichs die Werte von einschließlich des Startzeitpunkts bis einschließlich des Endpunkts wiedergegeben.

## IF\_Lieferant\_TRuDI

Der Auftragnehmer (DZG) beschreibt IF\_Lieferant\_TRuDI auf der Basis der AR2418-6, welche auf die „Green Button Spezifikation“ verweist.

Ziel der Umsetzung ist eine einheitliche lieferantenunabhängige Schnittstelle zur Darstellung der Messergebnisse auf der Grundlage des Kundenvertrages. Die vom Lieferanten über diese Schnittstelle übergebene Tarifinformation wird in Form eines ausgerollten Kalenders übergeben.

TRuDI\_055\_MUSS Die Schnittstelle IF\_Lieferant\_TRuDI muss AR\_2418-6.xsd in der Version 1.0 vom 14.06.2017 entsprechen.

TRuDI\_056\_MUSS Die Schnittstelle IF\_Lieferant\_TRuDI muss der VDE AR\_2418-6 Version 1.0 entsprechen, die derzeit noch in redaktioneller Bearbeitung ist. Die relevanten Auszüge für die Schnittstelle sind jedoch abgeschlossen, können für die Implementierung verwendet werden und sind in diesem Dokument in Kapitel 7 eingefügt.

TRuDI\_057\_MUSS Die TRuDI Software muss die Beispiel XML Dateien IF\_Lieferant\_TRuDI\_example.xml verarbeiten können.

Diese Lastenheft Version beschreibt zwei mögliche Optionen zum Import des ausgerollten Kalenders.

Eine Lösung umfasst das manuelle Herunterladen des ausgerollten Kalenders durch den Letztverbraucher. Der ausgerollte Kalender beinhaltet alle Informationen, die der Letztverbraucher zur Prüfung seiner Rechnung benötigt (🡪TRuDI\_032\_MUSS).

Eine zweite Lösung stellt die Übernahme der Daten mittels WEB-Service dar (🡪 TRuDI\_031\_MUSS).

Übergabedatum beim Abruf des ausgerollten Kalenders durch den Letztverbraucher ist somit:

TRuDI\_058\_MUSS Zum Abruf eines ausgerollten Kalenders über eine vom Kunden eingegebene URI muss TRuDI eine Authentifizierung über Username / Passwort unterstützen.

Als Antwort erhält TRuDI den ausgerollten Kalender als xml – Datei.

TRuDI\_059\_MUSS TRuDI muss in der Lage sein den ausgerollten Kalender einzulesen und zu verarbeiten.

Die TRuDI Software muss weiterhin folgende funktionalen Merkmale an der Schnittstelle IF\_Lieferant\_TRuDI umsetzen:

TRuDI\_060\_MUSS Wenn im dayProfile nachgelagerte Zeitgrößen nicht angegeben werden (second, hundreds) sind diese gleich ‚0‘ zu setzen.

TRuDI\_061\_MUSS Zur Identifikation der zu tarifierenden Messwerte wird (nach Zulieferung der Datei für IF\_Lieferant\_TRuDI) die UsagePointId (siehe Kapitel 7.1) und die tafID verwendet.

# Produktcharakteristiken

## Systemumgebung

### Software

#### Plattform

Die TRuDI Software ist zum Einsatz auf Letztverbraucherhardware vorgesehen. Als Zielsystem ist die Hardware des Letztverbrauchers (minimal Anforderungen siehe Abschnitt 5.1.2) vorzusehen.

TRuDI\_062\_MUSS Die TRuDI Software ist grundsätzlich plattformunabhängig zu gestallten. Dabei sind die Varianten installationsbasierte Software und Livesystem vorzusehen.

TRuDI\_063\_MUSS Als installationsbasierte Variante ist Microsoft Windows 7 als Minimalsystem vorzusehen.

TRuDI\_064\_MUSS Als Livevariante ist das System Heise Meterix vorzusehen.

#### Komponenten

Die Gatewayhersteller liefern ihren herstellerspezifischen Adapter als eigentsändige Komponente. Der Lieferant bestimmt, welche Software Version bereitgestellt wird. Er erhält dieses Software Version als fertig kompiliertes Paket zusammen mit der Information, welche Adapter enthalten sind.

TRuDI\_065\_MUSS Es ist immer genau ein Adapter aktiv im System zu verwenden (🡪 TRuDI\_001\_MUSS).

Die Integration neuer Adapter, etc. ist in der Zukunft abschließend zu klären.

Das Verfügbarmachen der Hilfsmittel zur Rechnungskontrolle ist Pflicht des Messwertverwenders / Lieferanten (MessEG § 33 (3)). Mit TRuDI sorgt die Initiative BundesDisplay für ein als Rechnungsprüfungshilfsmittel geignetes Software-Produkt, wenn die Daten entsprechend dem in Abbidung 1 dargestellten Konzept übermittelt werden.

#### Sicherheit

TRuDI\_066\_MUSS Die Software, sowie deren benötigten Systemkompenten, müssen physisch gegen Manipulation geschütz werden.

TRuDI\_067\_MUSS Die von der Software verarbeiteten Messdaten müssen vor Verfälschung und Manipulation geschützt werden.

TRuDI\_068\_MUSS Die Letztverbraucheridentifikation gegenüber dem Gateway ist gegen Ausspähen, Manipulation und Weiterleitung zu sichern.

TRuDI\_069\_MUSS Die Software muss ohne erhöhte Systemrechte (Administratorrechte in Windows, sudo in Linux) ausführbar sein.

TRuDI\_070\_MUSS Der Messdatenempfang durch die Software vom Gateway muss ohne aktive Internetverbindung möglich sein.

### Hardware

#### Mindestanforderungen Installationsbasiertes System (Windows 7)

TRuDI\_071\_MUSS TRuDI muss mit folgenden, minimalen HW-Systemumgebungsparametern unter Windows 7 lauffähig sein:

* Prozessor 1GHz oder schneller
* Mindestens 2 GB Ram
* Mindestens 5 GB Diskspace

#### Mindestanforderungen Livesystem (Heise Meterix)

TRuDI\_072\_MUSS TRuDI muss mit folgenden, minimalen HW-Systemumgebungsparametern unter Meterix lauffähig sein:

* Prozessor 1GHz oder schneller
* Mindestens 2 GB Ram
* Bootfähigs CD/DVD Medium oder
* Bootfähiger USB Stick

## Nicht-funktionale / sonstige Anforderungen

### Anforderungen aus PTB-A50.8

TRuDI\_073\_MUSS Die Software muss die Anforderungen aus PTB-A 50.8 Kapitel 5 und 7 direkt oder sicherheitsniveau-gleich erfüllen.

### Schulung

Für die Entwicklung der herstellerspezifischen Adaptoren ist für die Gatewayhersteller eine Anleitung zur Benutzung der Schnittstelle IF\_Adapter\_TRuDI bereitzustellen. Die Anleitung kann als Online-Seminar (per Download oder Webkonferenz) bereitgestellt werden. Die Anleitung muss bei herstellerseitigem Bedarf wiederholt werden.

TRuDI\_074\_SOLL Die Schulung sollte eine Referenzimplementierung der Schnittstelle IF\_Adapter\_TRuDI (ohne Anbindung an ein Gateway) enthalten.

TRuDI\_075\_MUSS Die Sprache der Schulung ist deutsch.

TRuDI\_076\_KANN Zusätzliche Schulungen in englischer Sprache sind möglich.

### Dokumentation

TRuDI\_077\_MUSS Zur Software ist ein Benutzerhandbuch in deutscher Spache mit den Inhalten

* Installation/Deinstallation/Aktualisierung
* Konfiguration
* Erläuterung der Funktionen und deren Benutzerführung
* Fehlerbehebungen

beizulegen.

TRuDI\_078\_MUSS Der Sourcecode der Software ist mit Entwicklerkommentaren zu versehen.   
Der Sourcecode ist mit einer Architekturbeschreibung (empfohlen Arc42) zu versehen.

TRuDI\_079\_MUSS Der Sourcecode ist mit einer Build Beschreibung zu versehen.

### Lizenzmodell

TRuDI\_080\_MUSS Die Software ist unter die Lizenz MIT (<https://opensource.org/licenses/MIT>) zu stellen.

TRuDI\_081\_MUSS Der Sourcecode zur Software ist uneingeschränkt bereitzustellen.

TRuDI\_082\_MUSS Die Software muss als ausführbares/verteilbares Paket für uneingeschränkt bereitzustellen.

TRuDI\_083\_MUSS Eine Kopie der Apache Lizenz muss dem SW-Paket bei liegen. Die Lizenz bezieht sich auf das XML Schema espi (orginal GreenButton Format), das bei denInterfaces IF\_Adapter\_TRuDI und IF\_Lieferant\_TRuDI referenziert wird.

### Testumgebung

TRuDI\_084\_MUSS Es ist ein Softwarepaket zur Prüfung der herstellerspezifische Adapter hinsichtlich Einhaltung der Schnittstelle IF\_Adapter\_TRuDI bereitzustellen (🡪 TRuDI\_020\_MUSS).  
  
Die Prüfung muss headless vornehmbar sein. Die Prüfung muss ohne Installation des produktiven Softwarepakets möglich sein.

# Verzeichnisse

## Abbildungsverzeichnis

[Abbildung 1: Übersicht Systemarchitektur TRuDI 7](#_Toc486407188)

## Tabellenverzeichnis

[Tabelle 1: Versionshistorie 2](#_Toc486442945)

[Tabelle 2: Basisdaten und deren Verwendung in TRuDI 33](#_Toc486442946)

[Tabelle 3: Prüfungsdaten und deren Verwendung in TRuDI 47](#_Toc486442947)

## Anforderungsverzeichnis

# Anhang 1: Regeln zur Anwendung des Datenmodells der VDE AR 2418-6

Im Folgenden sind die Regeln zur Anwendung des VDE AR 2418-6 Datenmodells (AR\_2418-6.xsd v1.0) beschrieben. Diese werden ausgehend von der zentralen Klasse UsagePoint aufgeteilt in Basisdaten und Prüfungsdaten.

Zu unterscheiden ist in den Tabellen in Kapitel 7.1 und 7.2 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 gekenn¬zeichnet, 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 aus <OBIS>.<SMGw-ID>.sm und wird für die Bildung der inneren Signatur genutzt.  Jede Instanz der Klasse *readingType* muss ein Datenelement *qualifiedLogicalName* enthalten. | Erforderlich | Erforderlich | muss implementiert werden |
| 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 |
| 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. | Optional | Optional | keine zwingende Implementierung |
| 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. | 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 |
| 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. Dieses Attribut wird als Default auf ‚0‘ gesetzt.  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 |
| 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

