



The future is open
Peppol

Peppol ViDA model for Tax Data Document

Table of Contents

Link to main site of documentation	2
Introduction	3
Scope	3
Audience	3
Benefits	3
1. Business processes	5
1.1. Parties and roles	5
1.2. Tax Data Document process	6
1.3. Tax Data Document functionality	7
2. Technical details	8
2.1. BIS Identifiers	8
2.2. Invoice UUID calculation	8
2.3. Datatypes	13
2.4. XML schemas and namespaces	16
2.5. Glossary	17

Link to main site of documentation

[Main documentation site](#)

Introduction

The purpose of this document is to describe a Tax Data Document (TDD), which may be used to report issued and received invoices, together with related metadata, to a relevant authority for the purpose of VAT reporting using Continuous Transaction Control (CTC) methods.

Statement of copyright

OpenPeppol AISBL holds the copyright of this Peppol specification. This Peppol BIS document may not be modified, re-distributed, sold, or repackaged in any other way without the prior consent of OpenPeppol AISBL.

Scope

This document is concerned with clarifying the structure and functionality of the TDD transaction and how it can be applied in different use cases.

Audience

The audience for this document consists of organisations wishing to be Peppol-enabled for CTC tax reporting, and/or their ICT suppliers. These organisations may include:

- Service providers
- Contracting Authorities (CA)
- Economic Operators (EO)
- Software Developers
- Government Authorities

More specifically, the roles addressed include the following:

- ICT Architects
- ICT Developers
- Business Experts

For further information on Peppol/OpenPeppol, see <https://peppol.org>

Benefits

The TDD transaction may be used by either a seller or a buyer, or by a service provider acting on their behalf, to report the sending or receipt of an invoice to a relevant government authority, such as a tax authority, for the purpose of tax declaration.

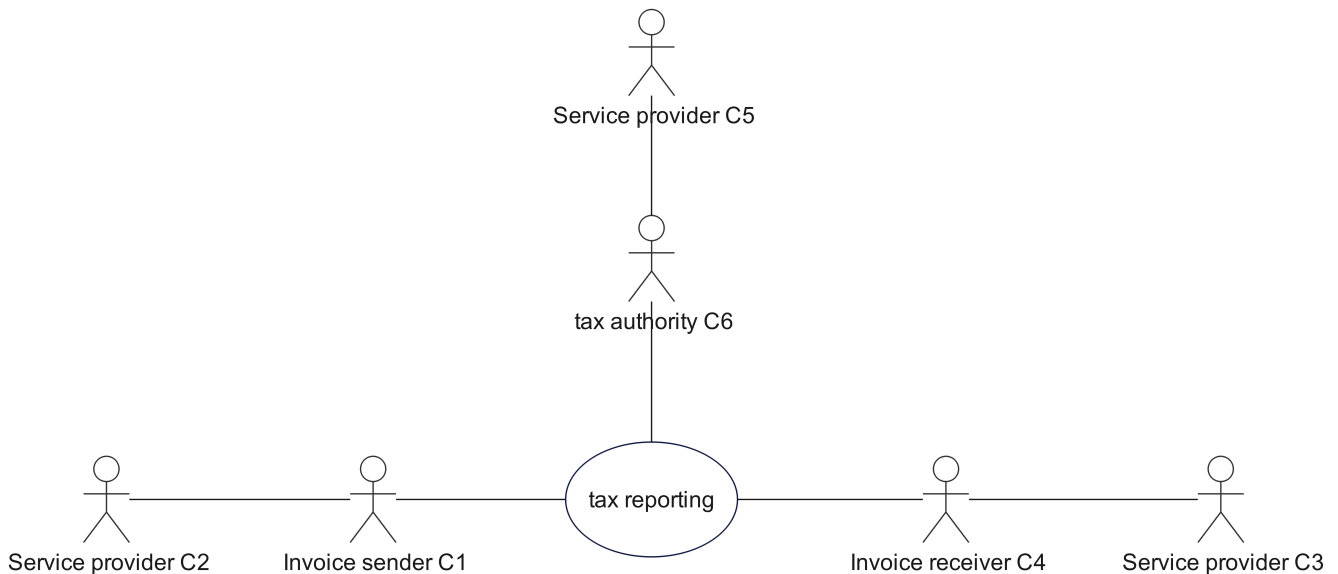
- The TDD can be used by either the issuer or the receiver of an invoice.
- It provides information about the TDD message itself.

- It supports management of the TDD message, such as **Submit**, **Resubmit**, and **Disregard**, which may be independent of the invoice exchange between the seller and the buyer.
- It provides different levels of detail about the invoice, ranging from a simple extract of basic invoice information to full inclusion of the invoice.
- It allows for full inclusion of an invoice, either as an attachment or in expanded form.

Chapter 1. Business processes

1.1. Parties and roles

The diagram below illustrates the roles involved in invoice and credit note transactions. The customer and the invoice receiver are the same entity, as are the supplier and the invoice sender.



1.1.1. Parties

Invoice sender C1

The party who, in the role of a supplier, issues an invoice to an invoice receiver. The invoice may contain tax, which the invoice sender, in the role of a taxable person, is responsible for reporting to the relevant tax authority.

Service provider C2

A party that provides electronic messaging services, including preparing and transmitting invoices and TDD transactions on behalf of the invoice sender, C1.

Service provider C3

A party that provides electronic messaging services, including preparing and transmitting TDD transactions on behalf of C4, and receiving and forwarding invoices to the invoice receiver, C4.

Invoice receiver C4

The legal person or organisation who, in the role of a buyer, demands a product or service and, in the role of a taxable person, is responsible for reporting to the relevant authority. Examples of invoice receiver roles include buyer, consignee, debtor, and contracting authority.

Service provider C5

A party that provides electronic messaging services, including receiving and managing TDD transactions on behalf of a tax authority, C6.

Tax authority C6

A public authority that is responsible for collecting taxes and enforcing relevant legislation.

1.1.2. Roles

Taxable person

An organization or an individual who is responsible for reporting and settling taxes in accordance with applicable legislation.

Service provider

A party who assists another party in carrying out its responsibilities.

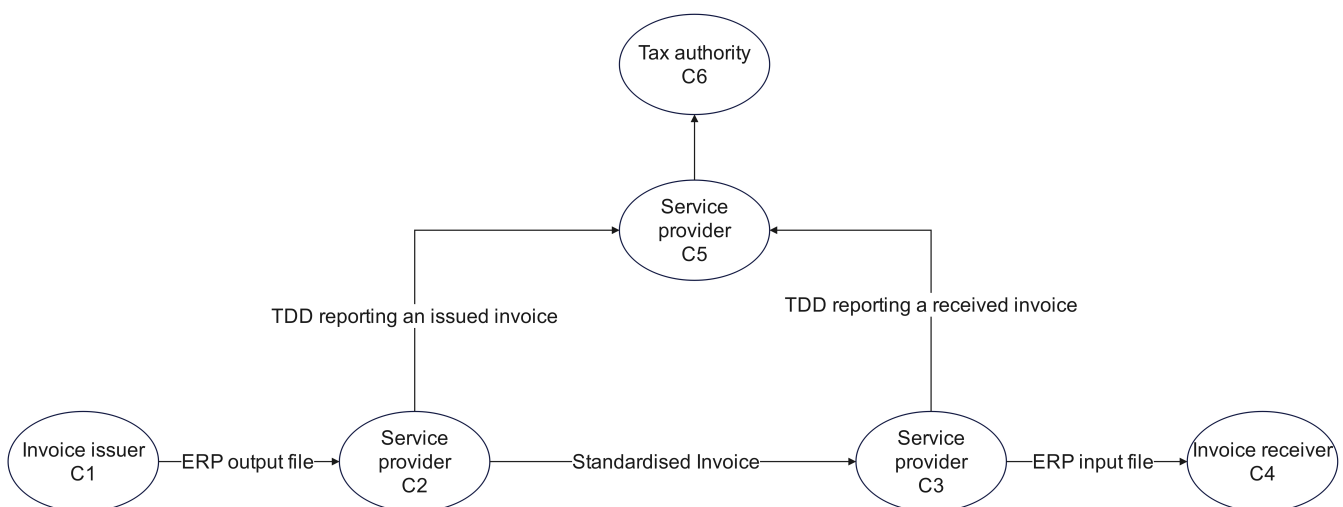
Tax authority

A public body or government organization responsible for administering tax legislation, receiving tax data documents, and assessing, collecting, and enforcing taxes.

1.2. Tax Data Document process

The TDD process depends on the legislation of the jurisdiction in which it is used and on the objectives the process is intended to achieve in that jurisdiction. This affects the content, sequence, and timing of the TDD within the overall process.

The generic process can be described using the following diagram.



Depending on the implementation of TDD in the relevant jurisdiction, process variations may include the following:

- The invoice receiver may not be required to report the receipt of the invoice.
- The timing of forwarding the invoice from C2 to C3 may occur either before or after the sending of the TDD. In a clearance model, the TDD must be sent and a response received from the tax authorities before the invoice can be forwarded. In a reporting model, the sequence may be more flexible.
- The exchange of the TDD may be independent of the invoice, in the sense that correction and resending of the TDD may or may not require a corresponding exchange of the invoice.

1.3. Tax Data Document functionality

The information provided in a TDD must comply with a defined set of rules governing both the content of business terms and the relationships between them.

The TDD document contains two main sections:

- TDD document metadata
- Data extracts from the reported invoice

1.3.1. TDD metadata

The TDD is a report based on its own specification. It is exchanged with tax authorities, rather than between a buyer and a seller, and it may be sent at a different time.

The first part of the TDD document contains metadata that identifies the specification, the relevant parties, and the applicable dates and times. It also provides codes that define the role of the parties involved in the exchange and the function of each TDD instance.

- A given party may send TDDs either in the role of an invoice issuer or an invoice receiver.
- There may be more than one TDD instance for the same invoice, with different functions such as **Submit**, **Resubmit**, and **Disregard**.

1.3.2. Data extracts

A single TDD instance may be used to report more than one invoice, depending on the implementation in the relevant jurisdiction.

For each reported invoice, key data values may be extracted into the TDD, depending on the applicable requirements.

Chapter 2. Technical details

Following section provide technical details.

2.1. BIS Identifiers

Peppol has defined policies that specify how identifiers are to be used both within its transport infrastructure and in the documents exchanged across that infrastructure. These policies also introduce principles for any identifiers used in the Peppol environment. The policies applicable to this BIS are described below.

2.1.1. Profiles and messages

All messages contain a Business process type (IBT-23) and a Specification identifier (IBT-24). The Business process type (IBT-23) identifies the business process to which a given message belongs, and the Specification identifier (IBT-24) identifies the type of message and the rules that apply.

Valid document instances shall contain matching Business process type (IBT-23) and Specification identifier (IBT-24).



The Specification identifier (IBT-24) is a string without spaces. The list below includes spaces in the Specification identifier (IBT-24) values to improve readability. **Ensure that all spaces are removed before use.**

In the table below you will find the values to be used as the specification identifier (TDT-001) and the business process type (TDT-002) for this profile:

Type	Element <code>cbc:CustomizationID</code>	Element <code>cbc:ProfileID</code>
Tax Data Document	urn:peppol:taxdata:vida-1	urn:peppol:taxreporting

UBL example of profile identifier

```
<cbc:CustomizationID>urn:peppol:taxdata:vida-1</cbc:CustomizationID>  
<cbc:ProfileID>urn:peppol:taxreporting</cbc:ProfileID>
```

2.1.2. Document type identifier scheme

The `busdoux-docid-qns` Document Type Identifier scheme shall be used in accordance with the current version of the Peppol Policy for the use of identifiers.

2.2. Invoice UUID calculation

The Invoice UUID (TDT-017) identifies the invoice that a Tax Data Document reports. It is not a free identifier: it is **derived from the invoice itself**, so that every party that handles the same invoice arrives at the same value without coordinating.

This matters because the sender side (C2) and the receiver side (C3) each create their own Tax Data Document for the same underlying invoice. A Tax Authority receiving both must be able to recognise them as reporting one and the same invoice. A calculated identifier achieves this without either party transmitting an identifier to the other.

It follows that every detail of the calculation below is normative. Two implementations that differ in any detail result in differing identifiers, and the mechanism fails silently as validation rules cannot detect deviations.

2.2.1. Method

The Invoice UUID is a **version 5 (name-based, SHA-1) UUID** as defined in [RFC 4122 §4.3](#), calculated from two inputs: a fixed namespace and a name derived from four invoice fields.

Namespace

```
e0bc4ac8-b025-46e5-a76d-0c893fc3027e
```

This namespace is fixed for ViDA tax reporting and MUST NOT be substituted with any other value, including the RFC 4122 predefined namespaces.

The name is derived from these four invoice fields, in this order:

BT-31 Seller VAT identifier

`cac:AccountingSupplierParty/cac:Party/cac:PartyTaxScheme/cbc:CompanyID`, where the sibling `cac:TaxScheme/cbc:ID` is VAT

BT-03 Invoice type code

`cbc:InvoiceTypeCode`, or `cbc:CreditNoteTypeCode` on a credit note

BT-01 Invoice number

`cbc:ID`

BT-02 Invoice issue date

`cbc:IssueDate`

2.2.2. Preparing the field values

Take each of the four values as it appears in the invoice, then:

1. **Remove all leading and trailing whitespace.** Whitespace means one or more of space (`#x20`), tab (`#x9`), carriage return (`#xD`) or line feed (`#xA`). An XML parser preserves such characters in element content, so an invoice that has been pretty-printed or re-serialised can carry them; without this step, two senders holding the same invoice would calculate different identifiers.
2. **Do not otherwise alter the values.** In particular:
 - Do **not** collapse whitespace inside a value. This is the difference between trimming and the XPath `normalize-space()` function, which also replaces internal runs of whitespace with a single space. Where an invoice number carries a run of two or more whitespace characters,

that run is preserved unchanged. Specifically, using `normalize-space()` will result in a wrong Invoice UUID.

- Do **not** change case. Values are used exactly as the invoice carries them; in particular the seller VAT identifier is not upper-cased and its country prefix is not stripped.
- Do **not** reformat the issue date. BT-02 is used in the `YYYY-MM-DD` form in which the invoice carries it.

3. A value that is empty after trimming counts as absent. See [When BT-31 is absent](#).

2.2.3. Building the name

Join the four prepared values in the order below, separated by a **single space character** (`' '`, `#x20`, ASCII 32):

```
name = BT-31 ' ' BT-03 ' ' BT-01 ' ' BT-02
```

There is exactly one space between adjacent values, and none at either end. Encode the name as **UTF-8** before hashing.

Note that the name is a hash input only: it is never parsed back into fields, so a value that does itself contain a space needs no escaping.

Render the resulting UUID in **lower-case canonical form** (`8-4-4-4-12` hexadecimal digits). Comparison is case-insensitive, but lower case is what implementations should emit.

The same method applies to credit notes; only the value of BT-03 differs.

2.2.4. Full sample calculation

This example reproduces the `base-example.xml` sample document. The seller VAT identifier is shown pretty-printed onto its own line, as a re-serialised invoice may well carry it, so that the trimming step can be seen doing its work.

Step 1 — Read the four values from the invoice

```
<cbc:ID>INV-2025-0046</cbc:ID>
<cbc:IssueDate>2025-02-10</cbc:IssueDate>
<cbc:InvoiceTypeCode>380</cbc:InvoiceTypeCode>
<cac:AccountingSupplierParty>
  <cac:Party>
    <cac:PartyTaxScheme>
      <cbc:CompanyID>
        DE811569869
      </cbc:CompanyID>
      <cac:TaxScheme>
        <cbc:ID>VAT</cbc:ID>
      </cac:TaxScheme>
    </cac:PartyTaxScheme>
  </cac:Party>
</cac:AccountingSupplierParty>
```

Step 2 — Remove leading and trailing whitespace from each value

Only BT-31 is affected here. As parsed it is a line break, eight spaces, **DE811569869**, a line break and six spaces; after trimming it is **DE811569869**. Without this step it would hash to an entirely different identifier. The other three values carry no surrounding whitespace and are unchanged.

Table 1. Prepared values, in order

Order	Term	Prepared value
1	BT-31 Seller VAT identifier	DE811569869
2	BT-03 Invoice type code	380
3	BT-01 Invoice number	INV-2025-0046
4	BT-02 Invoice issue date	2025-02-10

Step 3 — Join them with a single space between each

```
DE811569869 380 INV-2025-0046 2025-02-10
```

Step 4 — Calculate the version 5 UUID over the fixed namespace

```
uuidv5(e0bc4ac8-b025-46e5-a76d-0c893fc3027e, "DE811569869 380 INV-2025-0046 2025-02-10")
= 62fd54e5-8689-57a6-95a7-600c86734ea0
```

The Invoice UUID (TDT-017) carried in the Tax Data Document is therefore **62fd54e5-8689-57a6-95a7-600c86734ea0**.

2.2.5. Test vectors

Implementations should reproduce these values exactly. Each corresponds to a sample document

published with this specification.

BT-31	BT-03	BT-01	BT-02	Invoice UUID (TDT-017)
DE811569869	380	INV-2025-0046	2025-02-10	62fd54e5-8689-57a6-95a7-600c86734ea0
BE0477472701	380	INV-2025-0001	2025-01-10	81270c84-bfb1-5a12-8314-8887ee7d3e27
DE811569869	389	INV-2025-0046	2025-02-10	936d1327-0863-5632-ac82-30001c004e01
FR40303265045	261	CN-2025-0003	2025-02-05	9a787cd0-ef99-5bd1-ad5d-daf020699cf2

The first and third vectors share the seller, the invoice number and the issue date and differ only in the invoice type code, yet produce unrelated identifiers — a property of the hash, and the reason all four fields are required.

2.2.6. When BT-31 is absent

BT-01, BT-02 and BT-03 are mandatory in the Peppol BIS invoice; BT-31 is not. For an invoice to be reportable for tax purposes, however, the seller must be VAT registered, so BT-31 is expected to be present on every invoice in scope of ViDA tax reporting. This gives three cases:

- **All four values present** — calculate the Invoice UUID and create the Tax Data Document as normal.
- **BT-01, BT-02 or BT-03 missing or empty** — the invoice is not valid for tax reporting. Reject it; no Tax Data Document is created.
- **BT-31 missing or empty** — the invoice is still accepted and exchanged onward to C4, but it cannot be reported: no Invoice UUID is calculated and no Tax Data Document is created.

2.2.7. Relationship to the Invoice Transmission UUID

The Invoice UUID (TDT-017) identifies the **invoice**. If the same invoice is exchanged more than once, every exchange yields the same Invoice UUID — which is the intended behaviour, and the reason a second identifier exists.

The Invoice Transmission UUID (TDT-018) identifies **one exchange** of that invoice across the network. It is not calculated from the invoice content; it is the identifier of the transmission itself, captured by C2 and C3 at the point the Tax Data Document is created. This is why it is carried in the Report Details (TDG-01) alongside the reported document, rather than inside it. Together the two identifiers let a Tax Authority answer both "which invoice is this?" and "which delivery of it?".

2.2.8. Validation

Rule **ibr-tdd-87** checks that TDT-017 is a well-formed version 5 UUID. The Schematron cannot verify that the value was derived from the correct invoice fields — that remains an implementation responsibility, and the test vectors above are the means to confirm it.

2.3. Datatypes

Semantic data types are used to bridge the gap between the semantic concepts expressed by the information elements defined in the semantic model and the technical implementation. The semantic data types define the allowed value domain for the content, and any additional information components (attributes) needed in order to ensure its precise interpretation.

2.3.1. Primitive types

Semantic data type content may be of the following primitive types. These primitive types were taken from ISO15000, Annex A.

Primitive type	Definition
Date	Time point representing a calendar day on a time scale consisting of an origin and a succession of calendar ISO8601.
Decimal	A subset of the real numbers, which can be represented by decimal numerals.
String	A finite sequence of characters.

2.3.2. Semantic data types

The different semantic data types are described in the tables below, where various features such as attributes, format, and decimals as well as the basic type are defined for each semantic data type. They are based on {ISO15000}.

When used in an instance of an invoice, each data element will contain data. In the below tables this is identified as the “content”. Whenever a business term is used this term shall **always** have content and therefore the content is always mandatory.

Amount

An amount states a numerical monetary value. The currency of the amount is defined as a separate business term.

Component	Use	Primitive Type	Example
Content	Mandatory	Decimal	10000.25

Unit Price Amount

A unit price amount states a numerical monetary amount value for data elements that contain item prices that may be multiplied by item quantities. The currency of the amount is defined as a separate business term.



Unit price amount does not set restrictions on number of decimals, as contrast to the Amount type

Component	Use	Primitive Type	Example
Content	Mandatory	Decimal	10000.1234

Percentage

Percentages are given as fractions of a hundred (per cent) e.g. the value 34.78 % in percentage terms is given as 34.78.



No restriction on number of decimals for percentages.

Component	Use	Primitive Type	Example
Content	Mandatory	Decimal	34.7812

Quantity

Quantities are used to state a number of units such as for items. The code for the Unit of Measure is defined as a separate business term.



No restriction on number of decimals for quantities.

Component	Use	Primitive Type	Example
Content	Mandatory	Decimal	10000.1234

Code

Codes are used to specify allowed values in elements as well as for lists of options. Code is different from Identifier in that allowed values have standardized meanings that can be known by the recipient.



Codes shall be entered exactly as shown in the selected code list of the applicable syntax.

Component	Use	Primitive Type	Example
Content	Mandatory	String	Abc123

Indicator

Indicators are used to give boolean values to state whether something is (true) or is not (false).



Indicators shall be used in lower case.



Default value is "false" and applies if the relevant business term is not used.

Component	Use	Primitive Type	Allowed values
Content	Mandatory	String	false
			true

Identifier

Identifiers (IDs) are keys that are issued by the sender or recipient of a document or by a third party.



The use of the attributes is specified for each information element.

Component	Use	Primitive Type	Example
Content	Mandatory	String	abc:123-DEF
Scheme identifier	Optional	String	GLN
Scheme version identifier	Optional	String	1.0

Date

Dates shall be in accordance to the “Calendar date complete representation” as specified by {ISO8601}, format **YYYY-MM-DD**.



Dates shall not include timezone information.

Table 2. EN 16931_ Date. Type

Component	Use	Primitive Type	Example
Content	Mandatory	Date	2017-12-01

Time

Time shall be according to UBL allowed format.



Time shall include time zone information.

Table 3. EN 16931_ Date. Type

Component	Use	Primitive Type	Allowed forms
Content	Mandatory	Date	13:20:00 (1:30 PM)
			13:20:30.55 (30.55 sec)
			13:20:00Z (UTC)
			13:20:00-05:00 (UTC-5)
			00:00:00 (midnight)
			24:00:00 (midnight)

Time formats without time zone information (i.e. other than hh:mm:ssz and hh:mm:ss-h) shall be interpreted as being in the time zone of the sellers address and according to daylight saving status on the issue date of the invoice.

Text

Text is the actual wording of anything written or printed. Line breaks in the text may be present, and any line breaks should be preserved and respected by the receiver's system

Component	Use	Primitive Type	Example
Content	Mandatory	String	5% allowance when paid within 30 days

2.4. XML schemas and namespaces

The primary XML schema used is:

- OpenPeppol TDD, with the target namespace `urn:peppol:schema:vida-taxdata:1.0`

As this XML Schema is defined by OpenPeppol itself, it is handled slightly differently than, for example, the schema for the UBL Invoice. The TDD XML Schema reuses as many common components from OASIS UBL 2.1 as possible, while maintaining its own primary identity.

The XML namespaces used in the TDD XML Schema are:

Prefix	URL
pxs	<code>urn:peppol:schema:vida-taxdata:1.0</code>
cac	<code>urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-2</code>
cbc	<code>urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-2</code>
cec	<code>urn:oasis:names:specification:ubl:schema:xsd:CommonExtensionComponents-2</code>
udt	<code>urn:oasis:names:specification:ubl:schema:xsd:UnqualifiedDataTypes-2</code>

For technical reasons, there are no deep integrations with the required XML schemas. Instead, it is

expected that an [XML Catalog](#) is used to resolve the required XML schemas. The referenced XML schemas can be downloaded from the [OASIS UBL 2.1](#) publication.

2.5. Glossary

Business term - label assigned to a given information element which is used as a primary reference

Compliant - some or all features of the invoice model are used and all rules of the invoice model are respected Note 1 to entry: Based on TOGAF definition of a compliant specification

Conditional - Whether the option is used or not and in what way is dependent on other data in the message.

Conformant - all rules of the invoice model are respected and some additional features not defined in the invoice model are also used

Electronic invoice - invoice that has been issued, transmitted and received in a structured electronic format which allows for its automatic and electronic processing

Identification scheme - collection of identifiers applicable for a given type of object governed under a common set of rules

Identifier - character string used to establish the identity of, and distinguish uniquely, one instance of an object within an identification scheme from all other objects within the same scheme Note 1 to entry: An identifier may be a word, number, letter, symbol, or any combination of those, depending on the identification scheme used.

Information element - semantic concept that can be defined independent of any particular representation in a syntax

Mandatory - The option must be used in all messages.

May - is truly optional.

Optional - Whether the option is used or not is the choice of the users independently from other data in the message.

Semantic data model - structured set of logically interrelated information elements

Shall - the definition is an absolute requirement of the specification.

Shall not - the definition is an absolute prohibition of the specification.

Should - there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

Should not - there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

Structured information element - information element that can be processed automatically

Syntax - machine-readable language or dialect used to represent the information elements contained in an electronic document (e.g. an electronic invoice)