GS1 EPCglobal Tag Data Translation (TDT) 1.6

Ratified Standard

October 12, 2011

Latest version: 1.6 Issue 2

Previous versions: 1.0, 1.4

Disclaimer

GS1 AISBL (GS1) is providing this document as a free service to interested industries. This document was developed through a consensus process of interested parties in developing the Standard. Although efforts have been made to assure that the document is correct, reliable, and technically accurate, GS1 makes NO WARRANTY, EXPRESS OR IMPLIED, THAT THIS DOCUMENT IS CORRECT, WILL NOT REQUIRE MODIFICATION AS EXPERIENCE AND TECHNOLOGY DICTATE, OR WILL BE SUITABLE FOR ANY PURPOSE OR WORKABLE IN ANY APPLICATION, OR OTHERWISE. Use of this document is with the understanding that GS1 DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF NON-INFRINGEMENT OF PATENTS OR COPYRIGHTS, MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE, THAT THE INFORMATION IS ERROR FREE, NOR SHALL GS1 BE LIABLE FOR DAMAGES OF ANY KIND, INCLUDING DIRECT, INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES, ARISING OUT OF USE OR THE INABILITY TO USE INFORMATION CONTAINED HEREIN OR FROM ERRORS CONTAINED HEREIN.

Table of Contents

Terminology

Within this specification, the terms SHALL, SHALL NOT, SHOULD, SHOULD NOT, MAY, NEED NOT, CAN, and CANNOT are to be interpreted as specified in Annex G of the ISO/IEC Directives, Part 2, 2001, 4th edition [ISODir2]. When used in this way, these terms will always be shown in ALL CAPS; when these words appear in ordinary typeface they are intended to have their ordinary English meaning.

The `Courier` font is used to indicate the names of XML elements and attributes and names of variable fields within the Tag Data Translation markup.

All sections of this document are normative, except where explicitly noted as non-normative.

Status of this document

This document is a Ratified Standard as of September 9, 2011 and is based on the previous version which was ratified by the EPCglobal Board of Governors on June 10th, 2009. It now has been updated to reflect the current functionality in TDS v1.6. The previous version of the TDT was version 1.4. We have jumped to version v1.6 for this version to imply compatibity with TDS v1.6.

This version of the GS1 EPC Tag Data Translation Standard is the Ratified version of the standard and has completed all process steps under the new GSMP. Please note that in Issue 2 of this document, the “Unratified Standard” watermark has been removed and this section has been updated. The technical content remains the same as the first issue of September 9th, 2011.

Comments on this document should be sent to the GSMP@gs1.org .

Changes from previous versions

This version of the specification supports the latest TDS version 1.6. The following changes are made to this specification:

- Added new TDT definition file for ADI-var scheme to support variable-length EPC identifier construct for Aerospace & Defence, for the unique identification of aircraft parts
- Relaxed schema restrictions for the `tagLength` and `optionKey` attributes of the <scheme> element in EpcTagDataTranslation.xsd, in order to accommodate the variable-length EPC identifiers; `tagLength` and `optionKey` are not required attributes of <scheme> for variable-length EPC schemes such as ADI-var.
- Provided clarification in flowcharts (Figures 9a, 9b in section 3.10.0) regarding the padding and stripping of characters or bits when converting between binary and non-binary levels; the term 'NON-BINARY' is replaced with 'TAG-ENCODING URI' since only the tag-encoding URI representation has a 1-1 correspondence with the binary encoding for each of the structural elements.

  Note that when encoding from any level other than the BINARY level, it is necessary to examine the corresponding fields within the TAG-ENCODING URI and BINARY levels in order to make use of the flowcharts in Figures 9a and 9b. (The previous version of these flowcharts did not make this sufficiently clear - and for example, a field such as itemref might be defined within the BINARY and TAG-ENCODING levels but not defined in the LEGACY level (if it cannot be unambiguously parsed from the input (an element string or GS1 Application Identifier notation) without first applying rules as defined in `rule` elements)

- Errata corrections to TDT definition files defined in TDT 1.4 (typically missing LEGACY_AI levels in some schemes derived from GS1 identifier keys)
- Updates to Figures 3 through 10 and Table 2 to mention additional levels of representation introduced in TDT 1.4 and TDT 1.6.
- XML comments used throughout the XSD schema files for TDT to provide helpful annotation and explanation.

Previous changes introduced with TDT 1.4 relative to TDT 1.0:

- Modified tagLength attribute in schema definition to remove tagLength restriction (EpcTagDataTranslation.xsd)
- Added three new schema definition to support GSRN-96, GDTI-96 and GDTI-113
- Added example string format for GSRN and GDTI in Table 3
- Added bitPadDir attribute to the schema definition to specify padding direction for binary output. Added bitPadDir description to section 3.10 (Padding of fields) and replace existing table in this section with flow chart to provide more clarity
- Added support for additional functions to the schema definition to support arithimetic and added these functions to section 3.14 (Core Function)
- Added table entry for bitPadDir to section 4.6 (Attributes)
- Added GSRN and GDTI to section 9 (Glossary)
- Added GSRN and GDTI to the section 10 (References)

1 Introduction

1.1 Overview

The Electronic Product Code (EPC) is a globally unique identifier that is designed to allow the automatic identification of objects anywhere.

The EPC Tag Data Standard (TDS) indicates how existing coding systems such as the GS1 (formerly EAN.UCC) family of codes (GTIN, GLN, SSCC, GRAI, GIAI, GSRN, GDTI) and a small number of other identifier constructs should be embedded within the Electronic Product Code (EPC).

By providing a machine-readable framework for validation and translation of EPC identifiers, Tag Data Translation is designed to help to future-proof the EPC Network and in particular to reduce the pain / disruption in supporting additional EPC identifier schemes that may be introduced in the future, as the EPC Network is adopted by additional industry sectors and new applications. The EPC Tag Data Standard (TDS) also describes in terms of human-readable encoding and decoding rules for each coding scheme, how to translate between three representations of the electronic product code (EPC), namely the binary format and two formats of uniform resource identifiers (URI), one for tag-encoding and another for pure identity.

The canonical representation of an EPC is the pure-identity URI representation, which is intended for communicating and storing EPCs in information systems, databases and applications, in order to insulate them from knowledge about the physical nature of the tag, so that although 64 bit tags may differ from 96 bit tags in the choice of literal binary header values and the number of bits allocated to each element or field within the EPC, the pure-identity URI format does not require the information systems to know about these details; the pure-identity URI can be just a pure identifier.

The binary format is used to store the EPC identifier in the EPC/UII memory of the RFID tag. The binary format consists of a header (which indicates the coding scheme and version - usually the first 8 bits, although a 2-bit header was defined for SGTIN-64), a fast filter value (which can be used for distinguishing between different packaging levels), as well as fields indicating the company responsible for the object, the object class and a unique serial number.

The tag-encoding URI provides a 1-1 mapping with the binary number recorded in the physical tag and as such indicates the bit-length of the tag (for fixed-length EPCs) and usually also includes an additional field (usually 3 bits) which is reserved for fast filtering purposes, e.g. to distinguish between various packaging levels for trade items. The tag-encoding URI is therefore intended for low-level applications which need to write EPCs to tags or physically sort items based on packaging level.

The pure-identity URI format isolates the application software from details of the bit-length of the tags or any fast filtering values, so that tags of different bit-lengths which code for the same unique object will result in an identical pure-identity URI, even though their tag-encoding URIs and binary representations may differ. This means that when a manufacturer switches from using 64-bit tags to 96-bit tags or longer for tagging a particular product, the pure-identity URI representation of the EPC will appear the same (except for different serial numbers for different instances of the product).

Section E.3 of Appendix E of Tag Data Standard v1.6 provides examples of the pure-identity URI, tag-encoding URI and binary encoding for all current EPC schemes.

The EPC Tag Data Translation (TDT) standard is concerned with a machine-readable version of the EPC Tag Data Standard rules for formatting and translation of EPC identifiers. The machine-readable version can be readily used for validating EPC formats as well as translating between the different levels of representation in a consistent way.

This standard describes how to interpret the machine-readable version. It contains details of the structure and elements of the machine-readable markup files and provides guidance on how it might be used in automatic translation or validation software, whether standalone or embedded in other systems.

1.2 Tag Data Translation Charter

The three objectives in the original charter of the Tag Data Translation working group were:

- To develop the necessary specifications to express the current TDS encoding and decoding rules in an unambiguous machine-readable format; this will allow any component in the EPC Network technology stack to automatically translate between the binary and tag-encoding URI and pure-identity URI formats of the EPC as appropriate. The motivation is to allow components flexibility in how they receive or transmit EPCs, to reduce potential 'impedance mismatches' at interfaces in the EPC Network technology stack. Reference implementations of software that demonstrate these capabilities will also be developed.
- To provide documentation of the TDS encodings in such a way that the current prose based documentation can be supplemented by the more structured machine-readable formats.
- To ensure that automated tag data translation processes can continue to function and also handle additional numbering schemes, which might be embedded within the EPC in the future. By aiming for a future-proof mechanism which allows for smooth upgrading to handle longer tags (e.g. 256 bits) and the incorporation of additional encoding/decoding rules for other coding systems, we expect to substantially reduce the marginal cost of redeveloping and upgrading software as the industry domains covered by the EPC expand in the future. We envisage that data which specifies the new rules for additional coding schemes will be readily available for download in much the same way as current anti-virus software can keep itself up to date by periodically downloading new definition files from an authoritative source.

1.3 Tag Data Translation Concept

The Tag Data Translation process translates one representation of EPC into another representation, within a particular coding scheme. For example, it could translate from the binary format for a GTIN on a 96-bit tag to a pure-identity URI representation of the same identifier, although it could not translate a SSCC into a SGTIN or vice versa.

The Tag Data Translation concept is illustrated in Figure 1.

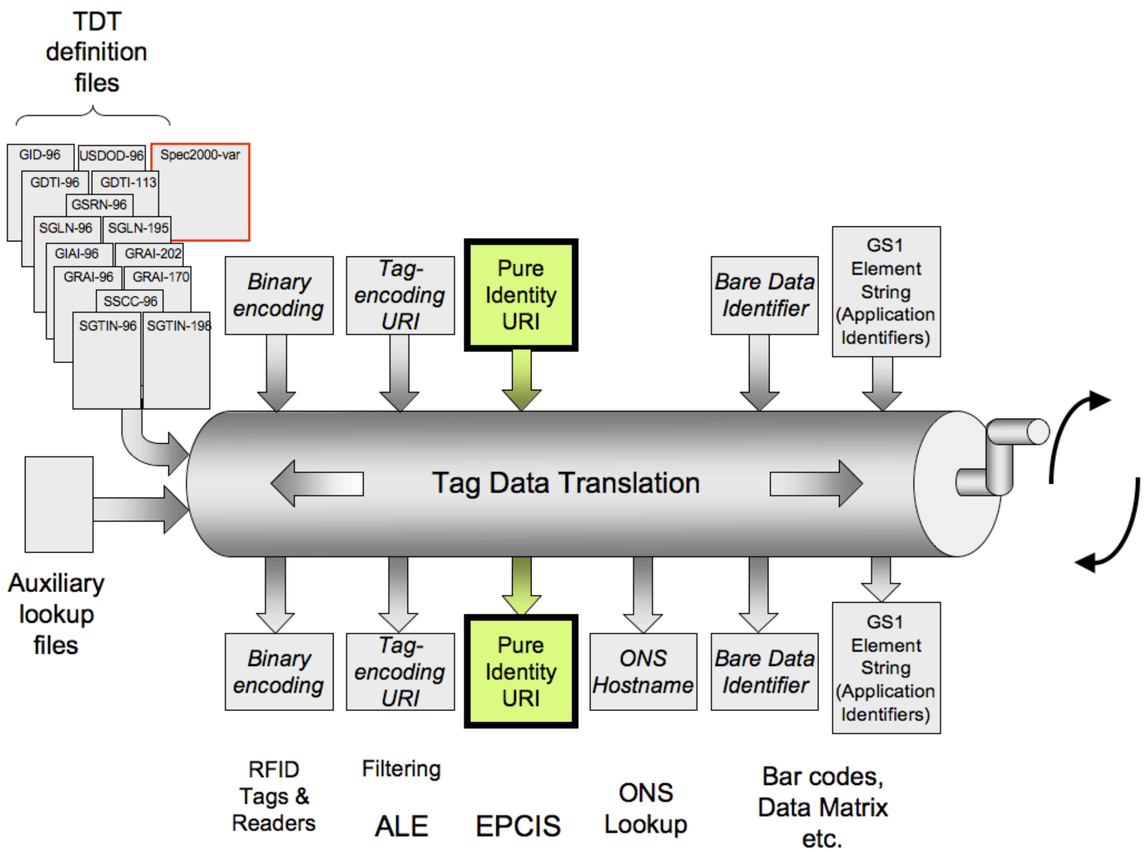

Figure 1 - Tag Data Translation - Concept

Tag Data Translation capabilities may be implemented at any level of the EPC Network stack, from readers, through filtering middleware, as a pre-resolver to the Object Name Service (ONS), as well as by applications and networked databases complying with the EPCIS interface. Tag Data Translation converts between different levels of representation of the EPC and may make use of external tables, such as the GS1 Company Prefix Index lookup table for 64-bit tags. It is envisaged that Tag Data Translation software will be able to keep itself up-to-date by periodically checking for and downloading TDT markup files, although a continuous network connection should not be required for performing translations or validations, since the TDT markup files and any auxiliary tables can be cached between periodic checks; in this way a generic translation mechanism can be extensible to further coding schemes or variations for longer tag lengths, which may be introduced in the future.

Although the TDT markup files are made available in XML format, this does not impose a requirement for all levels of the EPC Network technology stack to implement XML parsers. Indeed, TDT functionality may be included within derived products and services offered by solution providers and the existence of additional or updated TDT definition files may be reflected within software/firmware updates released by those providers.

Authoritative TDT definition files and schema are made freely available for anyone to download from the standards section of the GS1 EPCglobal website. For example, the manufacturer of an RFID reader may regularly check for and obtain the current TDT markup files, then use data binding software to convert these into hierarchical software data objects, which could be saved more compactly as serialized objects accessible from the particular programming language in which their reader software/firmware is written. The reader manufacturer could make these serialized objects available for download to owners of their products – or bundle them with firmware updates, thus eliminating the need for either embedded or real-time parsing of the TDT markup files in their original XML format at the reader level.

1.4 Role within the EPC Network Architecture

In the EPC Network Architecture [EPC Network Architecture Framework document] as depicted in Figure 2 below, the green bars denote interfaces governed by EPCglobal standards, while the blue boxes denote roles played by hardware and/or software components of the system.

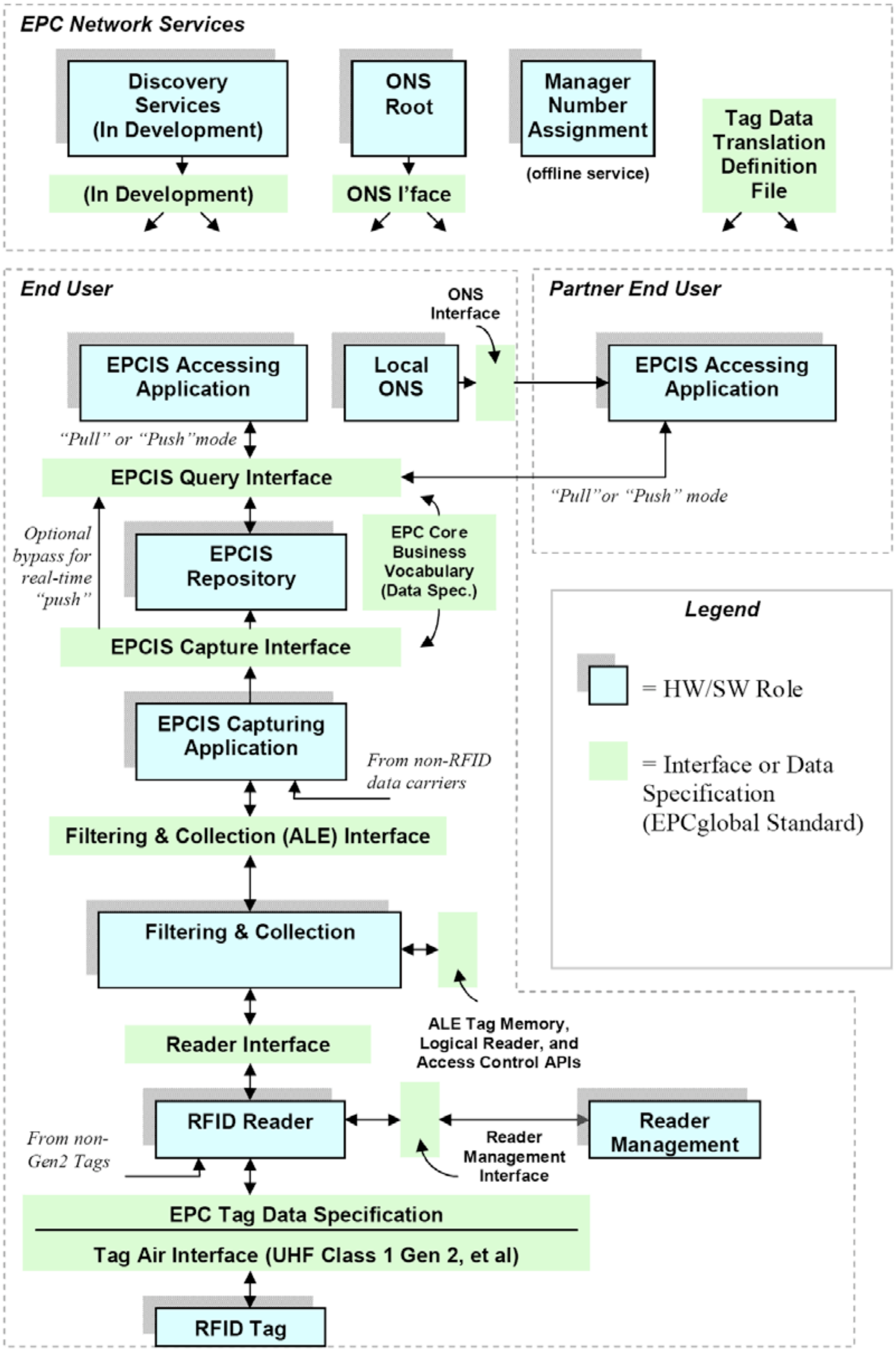

Figure 2 - EPC Network Architecture diagram

Table 1 describes the key elements of the EPC Network and the potential usages for the Tag Data Translation process for encoding and decoding Tag Data Standard.

EPC Network Standards	Description	TDT Role	Potential TDT Usage
Lower Level Reader Protocol (LLRP)	Defines the control and delivery of raw tag reads from Readers to F&C Middleware	Yes	Conversion upon 'impedance mismatch' of EPC representation
Application Level Events (ALE) Filtering & Collection	API for software that filters and collects raw tag reads, over time intervals delimited by event cycles as defined by applications such as the EPCIS Capturing Application	Yes	Conversion of other EPC representations into URI format for reports Assistance with converting declarative URI patterns into combinations of bit-mask
EPCIS Capturing Application	Software that supervises the operation of the lower EPC network elements and coordinates with enterprise level business events	Yes	Conversion upon 'impedance mismatch' of EPC representation
ONS	ONS is a network service layered over the existing Domain Name System that is used to lookup authoritative pointers to EPCIS-enabled Repositories and other EPC-related information services, given an EPC Manager Number or full Electronic Product Code	No	TDT provides an output-only format which is the hostname for DNS type 35 lookup, in order to perform an ONS query
EPCIS Service Repository	Networked database or information system providing query/update access to EPC-related data	No	In underlying databases, EPCs might be stored in other formats (e.g. GTIN+serial, separately – or hexadecimal). TDT can convert these to URI formats
EPCIS Enabled Application	Application software responsible for carrying out overall enterprise business processes, such as warehouse management, shipping and receiving	No	Conversion upon 'impedance mismatch' of EPC representation
Trading Partner Application	Trading Partner software that performs the role of an EPCIS Accessing Application.	No	Conversion upon 'impedance mismatch' of EPC representation

Table 1 – Potential role for Tag Data Translation throughout the EPC Network

The majority of the EPC Network components require the ability to consistently translate between binary data on tags and URI formats for information systems. However, it should be noted that levels of the stack above the Low Level Reader Protocol interface should normally be using the URI representation rather than the binary representation. This also enforces a need for a standard translation mechanism across the entire EPC network so that the translation process and resulting data is consistent and valid.

1.5 Tag Data Translation Process

The fundamental concept of Tag Data Translation is to automatically convert one representation of an EPC (whether binary, tag-encoding URI, pure-identity URI) or a serialized element string – and convert it into another representation as required.

This is illustrated in Figure 3

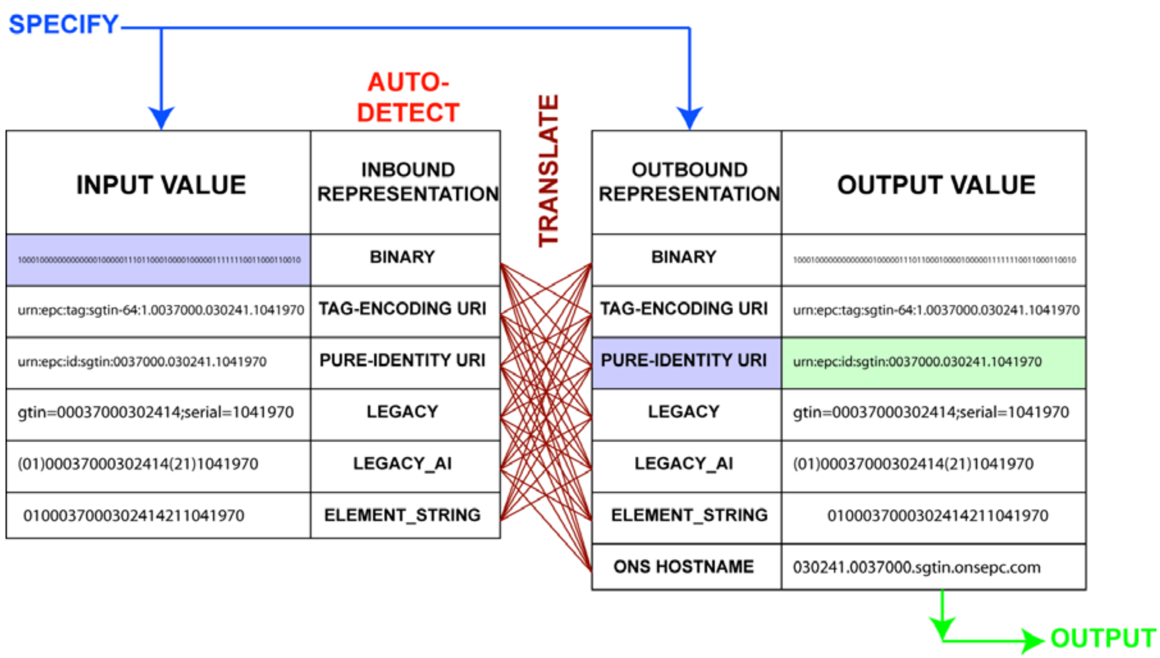

Figure 3 - Tag Data Translation process with examples of different representations.

The Tag Data Translation process takes an input value in a particular representation (binary / tag-encoding URI / pure-identity URI). We refer to the representation in which the input value is expressed as the inbound representation. In the conversion process, the desired outbound representation is also specified by the client requesting the translation. The Tag Data Translation process then returns an output value that is the input value after translation from the inbound representation to the outbound representation.

In practice, some representations contain more information than others. For example, the binary and tag-encoding URI representations also contain information about the number of bits used to store the EPC identifier on a physical RFID tag. They also contain information about a fast filter value, which can be used to discriminate between different packaging levels for trade items.

The serialized element strings contain the essential information (company, [object class,] serial number) for an EPC – but often they do not clearly indicate the boundary between the company identifier and the object class identifier – so additional information needs to be supplied, such as the length of the company identifier, from which the boundary can be determined..

This means that as well as providing an input value and a required outbound representation, there are cases where additional parameters need to be supplied. This is illustrated in Figure 4

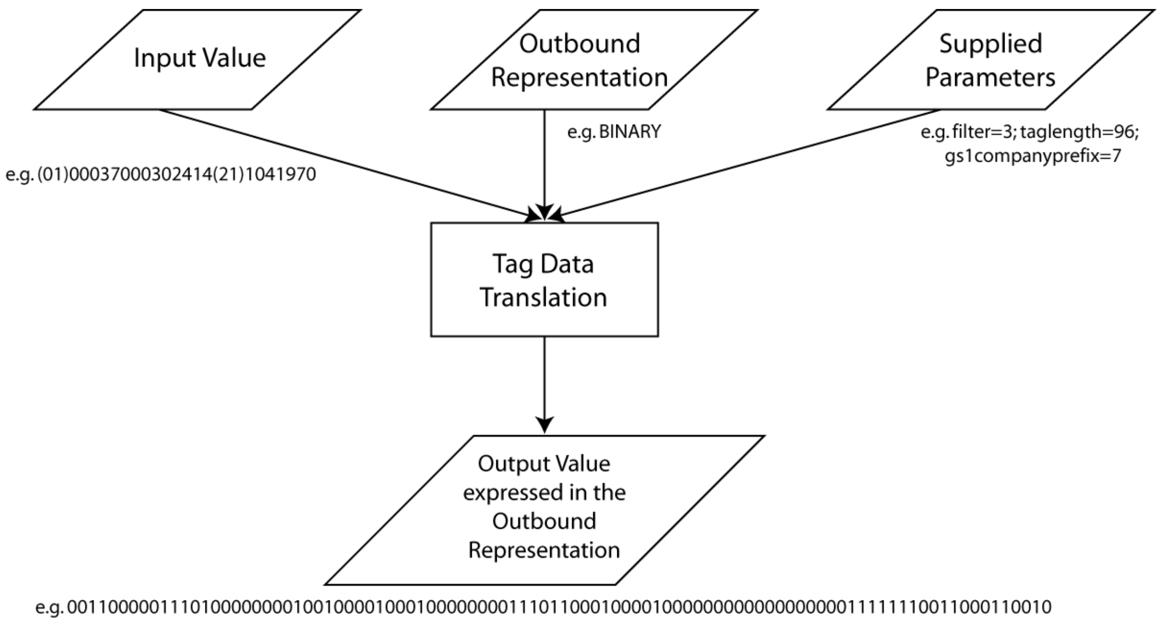

Figure 4 - Flowchart showing input and output parameters to a Tag Data Translation process.

In the context of Tag Data Translation, we refer to encoding as any conversion of the format in the direction of the binary representation, whereas decoding is any conversion away from the binary representation. This is illustrated in Figure 5.

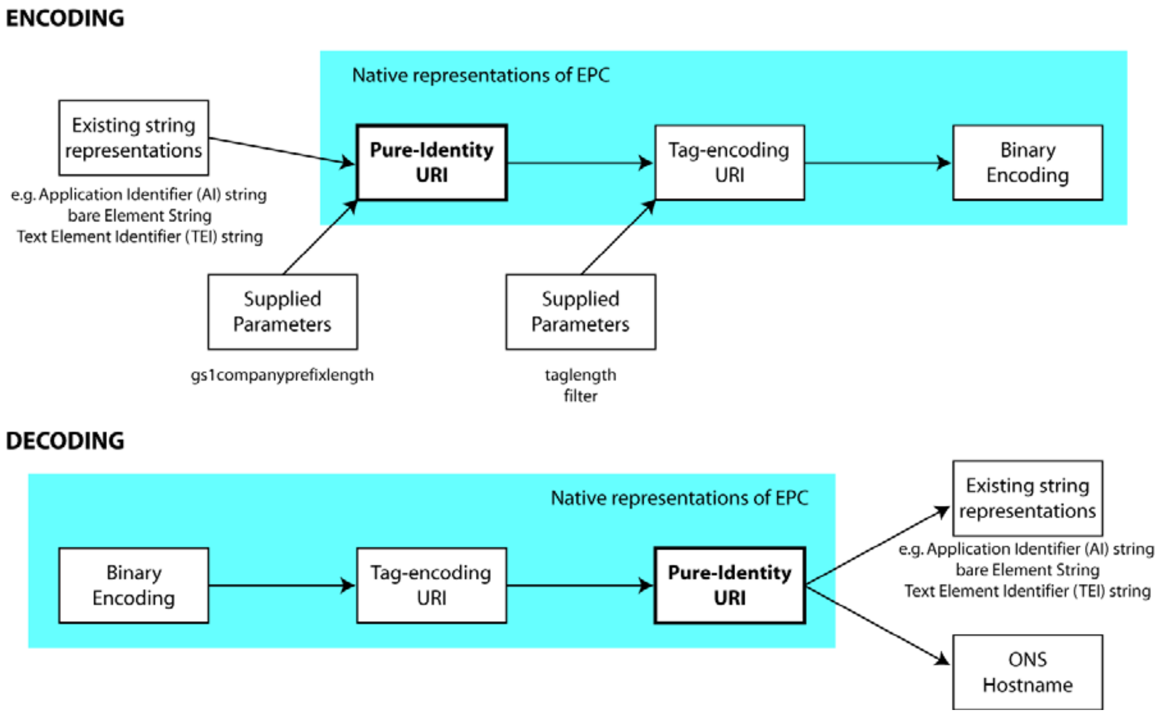

Figure 5 - Encoding and Decoding between different representations of an EPC. Note that when encoding, additional parameters need to be supplied.

In Figure 5 above, there are actually two distinct groups of supplied parameters – those such as `gs1companyprefixlength` which are required for parsing the input value when it is an element string or expressed in GS1 Application Identifier representation – and others such as `filter` and `tagLength`, which are required to format the output for certain levels of representation, such as binary or tag-encoding URI. In order to assist Tag Data Translation software in checking that all the required information has been supplied to perform a translation, the <level> elements of the Tag Data Translation markup files may contain the attribute `requiredParsingParameters` to indicate which parameters are required for parsing input values from that level and `requiredFormattingParameters` to indicate which parameters are required for formatting the output at that outbound representation level. Further details on these attributes appear in Chapter 4, which describes the TDT markup files. Note that tagLength is not a required formatting parameter nor a required parsing parameter for levels other than binary or tag-encoding URI and this means that there can be situations where more than one TDT definition file has a pattern matching the input (e.g. if converting an SGTIN with an all-numeric serial number from pure-identity URI representation to any level of representation except binary or tag-encoding URI). In such situations, it does not matter which of the matching definition files is selected.

A list of GS1 Company Prefixes of EPCglobal subscribers (without attributions) is available at the website http://www.onsepc.com in either XML or plain text format. From this list, it is possible to identify a suitable GS1 Company Prefix and therefore to determine its length in characters. This can then be passed as the value of the parameter `gs1companyprefixlength`, which should be supplied when translating from GS1 identifier keys to binary, tag-encoding URI or pure-identity URI representations. For the appropriate choice of filter value to use with a particular identifier scheme, please refer to the filter tables defined in EPCglobal Tag Data Standard. The tagLength parameter is used to help an implementation of Tag Data Translation to select the appropriate TDT definition file among EPC schemes that correspond to the same identifier but differ in length, e.g. to choose between GRAI-64, GRAI-96, GRAI-170 depending on whether the value of tagLength is set to 64, 96 or 170. For the value of the tagLength parameter, please also consider the available size (in bits) for the EPC identifier memory in the RFID tag (e.g. 96 bits) - and whether this is sufficient. [Non-normative example: For example, a GRAI-170 supports alphanumeric serial codes but cannot be encoded within a 96-bit tag.]

A desirable feature of a Tag Data Translation process is the ability to automatically detect both the coding scheme and the inbound representation of the input value. This is particularly important when multiple tags are being read – when potentially several different coding schemes could all be used together and read simultaneously.

For example, a shipment arriving on a pallet may consist of a number of cases tagged with SGTIN identifiers and a returnable pallet identified by a GRAI identifier but also carrying an SSCC identifier to identify the shipment as a whole. If a portal reader at a dock door simply returns a number of binary EPCs, it is helpful to have translation software which can automatically detect which binary values correspond to which coding scheme, rather than requiring that the coding scheme and inbound representation are specified in addition to the input value.

1.6 Expressing different representations of EPC

Patterns (Regular Expressions)

Given an input value, regular expression patterns may be used to match and extract groups of characters, digits or bits from the input value, in order that their values may later be used for constructing the output value in the desired outbound representation, after suitable manipulation, such as binary – decimal conversion, padding etc. We refer to these variable parts as 'fields'. Examples of fields include the GS1 Company Prefix (which usually identifies the manufacturer), the Serial Number, Fast Filter value etc.

Grammar (Augmented Backus-Naur Form [ABNF])

An Augmented Backus-Naur Form (ABNF) grammar may be used to express how the output is reassembled from a sequence of literal values such as URN prefixes and fixed binary headers with the variable components, i.e. the values of the various fields. For the `grammar` attributes of the TDT markup files, in accordance with the ABNF grammar conventions, fixed literal strings SHALL be single-quoted, whereas unquoted strings SHALL indicate that the value of the field named by the unquoted string SHOULD BE inserted in place of the unquoted string.

Rules for obtaining additional fields

However, not all fields that are required for formatting the output value are obtained directly from pattern-matching of the inbound representation. Sometimes additional fields are required to be known. For example, when translating a SGTIN-64 from binary to element strings, it will be possible to extract a GS1 Company Prefix Index, Item Reference and Serial Number from pattern-matching on the binary input – but the outbound representation needs other fields such as GS1 Company Prefix, Check Digit, Indicator Digit, which SHOULD be derived from the fields extracted from the inbound representation. For this reason, the TDT markup files also include sequences of rules, mainly within the element strings and binary levels. The rules express how such additional fields may be calculated or obtained via functions operating on fields whose values are already known.

Furthermore, there are some fields that cannot even be derived from fields whose values are already known and which SHALL therefore be specified independently as supplied parameters. For example, when translating a GTIN value together with a serial number into the binary representation, it is necessary to specify independently which length of tag to use (e.g. 64 bit or 96 bit) and also the fast filter value to be used. Such supplied parameters would be specified in addition to specifying the input value and the desired outbound representation. As illustrated in Figure 5, additional parameters SHOULD be supplied together with the input value when performing encoding. For decoding, it is generally not necessary to supply any additional parameters.

1.7 Translation Process Steps

There are five fundamental steps to a translation:

1. Use of the prefix matches and regular expression patterns to automatically detect the inbound representation and coding scheme of the supplied input value
2. Using the regular expression pattern to extract values of fields from the input value
3. Manipulation, (string manipulation, binary – decimal/alphanumeric conversion, padding etc.) of values of those fields in order to translate from the inbound representation to the outbound representation
4. Using the rules to calculate any additional fields required for the output
5. Using the ABNF grammar to format the required fields in the appropriate output representation

Note that the `prefixMatch` attribute in the TDT markup files is provided to allow optimization of software implementations to perform auto-detection of input representation more efficiently. Where multiple `option` elements are specified within a particular `level` element, each will generally have a `pattern` attribute with a subtly different regular expression as its value. The `prefixMatch` attribute of the enclosing `level` element expresses an initial prefix of these patterns which is common to all of the nested options. Optimized software need not test each nested option for a pattern match if the value of the `prefixMatch` attribute fails to match at the start of the input value. Only for those levels where the `prefixMatch` attribute matches at the start of the string should the patterns of the nested options be considered for matching.

Note that in the TDT markup files, the `prefixMatch` attribute SHALL be expressed as a substring to match at the beginning of the input value. The `prefixMatch` attribute SHOULD NOT be expressed in the TDT markup files as a regular expression value, since a simple string match should suffice. Software implementations MAY convert the prefixMatch attribute string value into a regular expression, if preferred, for example by prefixing with a leading caret ['^'] symbol (to require a match at the start of the string) and by escaping certain characters as required, e.g. escaping the dot character as '`\.`' or '`\\.`'.

2 Tag Data Standard

2.1 Overview

In the EPC Tag Data Standard, the canonical representation of an Electronic Product Code (EPC) is as a pure-identity URI. This is to be used when an EPC is communicated within software applications and information systems, EPCIS and when information about EPCs is exchanged between organizations, since these systems should not be concerned with the nature of the physical tag in which the EPC was encoded - or indeed whether the EPC was encoded within an RFID tag, a barcode or DataMatrix symbol. When an EPC is encoded within the EPC/UII memory bank of an RFID tag, a binary encoding is used. This binary encoding includes additional information such as an indication of the length of the EPC (for fixed-length EPCs) and a filter value. A tag-encoding URI format is also defined, which provides a faithful representation of all of the information contained within the binary encoding of an EPC. We therefore have to concern ourselves with three representations of the Electronic Product Code, namely binary encoding, tag-encoding URI and pure-identity URI.

Furthermore, the EPC Tag Data Standard specification (v1.6) describes how a number of the GS1 (formerly EAN.UCC) coding schemes (GTIN, SSCC, GLN, GRAI, GIAI, GSRN and GDTI) should be embedded within the EPC for 64-bit, 96-bit and larger tags for GTIN, GRAI, GIAI and GDTI to support alpha-numeric serial number. The Electronic Product Code (EPC) is intended to enable unique identification of any object anywhere automatically. Many of the existing GS1 identifier keys (SSCC, GRAI and GIAI) are already fully serialised. Others, such as the GTIN represent a product class rather than an individual fully serialized object. For use with the EPC, some GS1 identifiers (e.g. GTIN, GLN) may be accompanied with an additional serial number and referred to as SGTIN, SGLN.
Although technically the serialised GS1 codes are not themselves a representation of the EPC, they can be encoded into- and decoded from the three representations of EPC, as

described in the EPC Tag Data Standard specification – so for this reason we consider various representation levels for a EPC Tag Data Translation process as illustrated in Table 2.

	7	Hostname for DNS type 35 query in order to perform an ONS lookup	Output-only format	
ENCODE	6	Text Element Identifier string (where appropriate, e.g. ADI)	Constrained by specifications of existing coding schemes Does not express tag length, filter value	DECODE
	5	Application Identifier string or bare Element String		
	4	Serialised human-readable representation (SGTIN,SSCC,SGLN,GRAI,GIAI)		
	3	Pure-identity URI format of EPC		
	2	Tag-encoding URI format of EPC	Constrained by number of bits available in physical tag. Expresses tag length, filter value	
	1	Binary representation of EPC		

Table 2 - Levels of representation involved in the Translation Process

As Table 2 indicates, the various 'levels' involved in the translation process are not completely equivalent. There is a one-to-one mapping between the pair of levels numbered 1 and 2 (binary and tag-encoding URI) and between the pair of levels numbered 3 and 4 (pure-identity URI and serialized element string). The levels 3 and 4 lack the information present in levels 1 and 2 about tag length and fast filtering value. This is illustrated in more detail in Figure 6 below. Levels 5 and 6 shown in Table 2 are simply additional string representations of the main elements contained within the Pure-identity URI representation, to support ease of integration with identifiers encoded within linear barcodes or 2-dimensional barcodes. Note that for convenience, TDT 1.6 provides a further 'level' of representation, corresponding to the hostname for which a DNS Type 35 (NAPTR) query should be performed in order to effect an ONS lookup. This is not strictly an equivalent level of representation of EPC, since ONS v1.0 does not currently provide serial-level pointers for all coding schemes. It is therefore an output-only format and not a valid input format for encoding purposes. For this reason, only an ABNF grammar is defined for formatting the output in the ONS hostname representation – and no regular expression is defined for parsing the ONS hostname representation as input. i.e. in the TDT markup files, the `pattern` attribute SHALL always be absent from the `level` element representing the ONS hostname format. This SHALL indicate to translation software that any auto-detection of the inbound representation SHALL NOT consider the ONS hostname representation as a valid input.

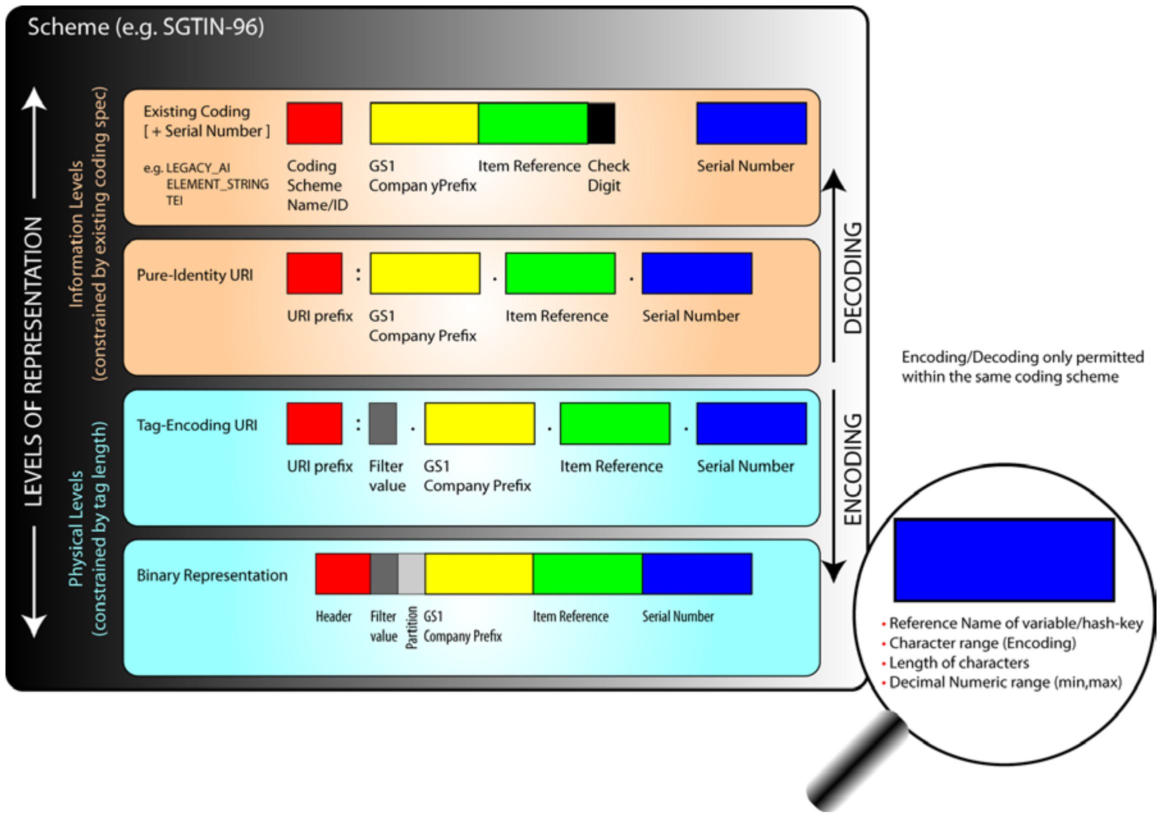

Figure 6 - Comparison of the data elements present in each level of each scheme. Note that the level marked as 'Existing Coding' in Figure 6 corresponds to levels 4-6 of Table 2.

2.2 Many Schemes, Multiple Levels within each scheme and multiple options within each level

We refer to each EPC coding system (SGTIN, SSCC, SGLN, GRAI, GIAI, GSRN, GDTI, USDOD, ADI and GID) as a scheme. The GS1 EPC Tag Data Standard defines the structure and encoding/decoding rules for each EPC scheme. Note that Tag Data Translation provides separate definition files for each EPC scheme and for each permitted length of the binary encoding for fixed-length EPC schemes. e.g. TDT provides separate definition files for SGTIN-64, SGTIN-96, SGTIN-198. Within each scheme, there are various levels of representation (binary, tag-encoding URI, pure-identity URI as well as string representations and ONS hostname).

Furthermore, the GS1 identifier keys use a GS1 Company Prefix of variable length, between 6 and 12 decimal digits. The TDS specification takes two different approaches to handling this in the 64-bit and 96-bit schemes. For the 64-bit schemes, an integer-based GS1 Company Prefix Index is encoded into the binary representation, in order to accommodate a larger range of numbers for the Item Reference and Serial Number partitions. The GS1 Company Prefix is obtained from the encoded Company Prefix Index by lookup in a table and it is always the GS1 Company Prefix that appears in the

URI formats. For the 96-bit schemes, a 3-bit field (the partition value) following the fast filter value within the binary representation is used to indicate the length of the GS1 Company Prefix, in the range 6-12 digits, denoted by binary partition values 000 – 110. The bit-length partitions allocated to the GS1Company Prefix and Item Reference fields varies accordingly as described in the EPC Tag Data Standard.

One option would be to use a separate lookup table for the partition values as described in the TDS specification. However, since the correspondence between the partition value and the length of the GS1 Company Prefix is common to all the GS1 schemes and the partition table is static in nature, we propose a more pragmatic approach and instead embed 7 variants ('Options') of the coding structure within each level, with the appropriate Option being selected either by matching a hard-coded partition value from the inbound data (where this is supplied in binary representation) – or from the length of the GS1 Company Prefix (which SHALL be supplied independently if encoding from the GS1 identifier key). This approach also allows the TDT markup files to specify the length and minimum and maximum values for each field, which will often vary, depending on which Option was selected – i.e. depending on the length of the GS1 Company Prefix used.

In TDT 1.6, different Option elements are also used within the TDT definition file for the variable-length EPC ADI-var to support the permitted alternative variations within that EPC regarding how the unique identifier is constructed.

For each option, the representation of the EPC is expressed as both a regular expression pattern to match the inbound representation against, and as an Augmented Backus-Naur Form (ABNF) grammar for formatting the outbound representation.

The regular expression patterns and ABNF grammar are therefore subtly different for each of the options within a particular level – usually in the literal values of the bits for the partition value and lengths of digits or bits for each of the subsequent partitions (where delimiters such as a period '.' separate these partitions) – or in the case of the element strings and binary representation, the way in which groups of digits or bits are grouped within the regular expression pattern. This approach facilitates the automatic detection of the boundary between GS1 company prefix and item reference simply by regular expression pattern matching, although care should be taken to ensure that only one option has a pattern that matches any valid input for that EPC scheme. Negative lookahead constructs within regular expressions can be helpful for ensuring this.

Within each option, the various fields matched using the regular expression are specified, together with any constraints which may apply to them (e.g. maximum and minimum values), as well as information about how they should be properly formatted in both binary and non-binary (i.e. information about the number of characters or bits, when a certain length is required, as well as information about any padding conventions which are to be used (e.g. left-pad with '0' to reach the required length of a particular field). The concept of multiple options within each level of each scheme is illustrated in Figure 7.

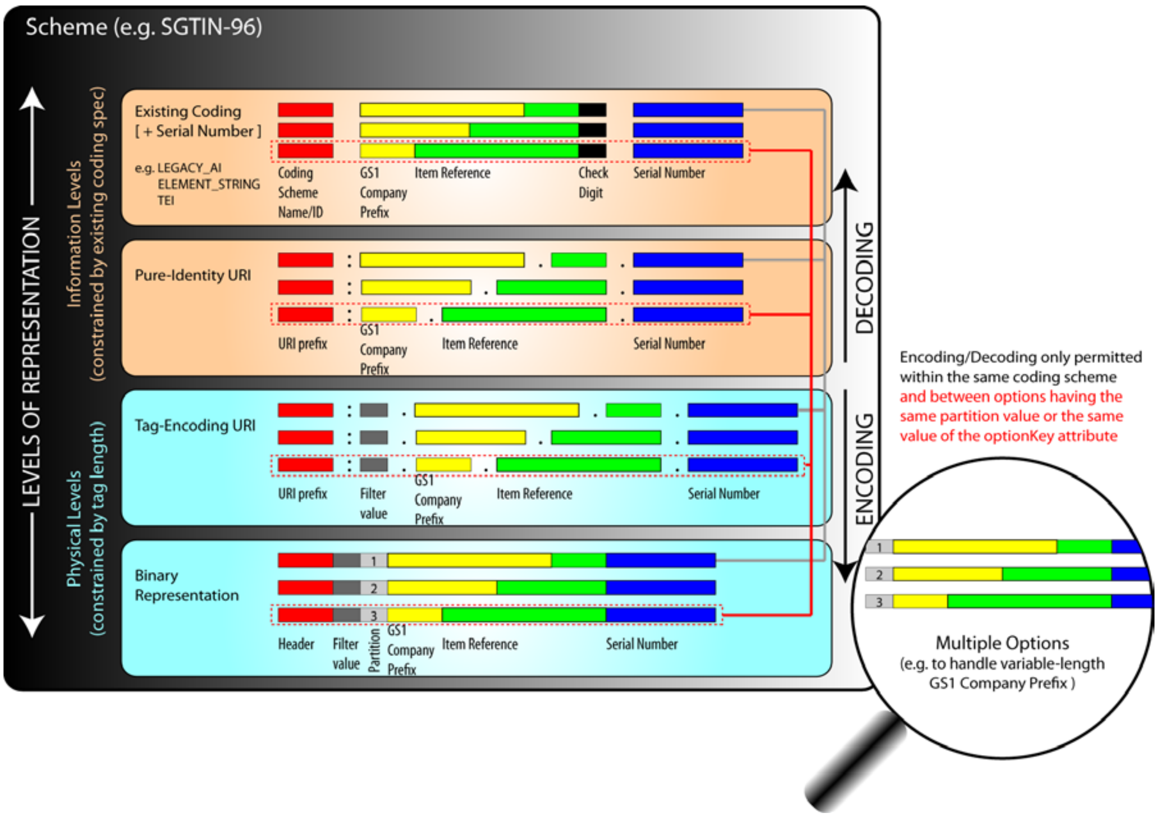

Figure 7 - Depiction of multiple options within each level to handle variable-length GS1 Company Prefixes. Note that the level marked as 'Existing Coding' within Figure 7 corresponds to levels 4-6 of Table 2.

3 TDT Markup and Logical Process

The key element of the above architecture is the collection of TDT markup files, which enables encoding and decoding between various levels of representation for each particular coding scheme. This generic design requires open and highly flexible representation of rules for translation software to encode/decode based on the input value. The TDT markup language is a machine-readable XML format expressing the encoding/decoding and validation rules for various identifiers / coding schemes defined in the TDS specification. The TDT markup SHALL be created and maintained by EPCglobal for all the identities defined by the EPC Tag Data Standard specification.

This chapter provides a descriptive explanation of how to interpret the TDT Markup files in the context of a Tag Data Translation process. Chapter 4 provides a formal explanation of the elements and attributes of the TDT markup files.

3.1 TDT Artifacts

Individual TDT definition files are provided for each coding scheme (i.e. separate files for SGTIN-64, SGTIN-96, SSCC-64, SSCC-96, GID-96, etc.) and are made freely available for public download from the EPCglobal web page for the TDT standard

[http://www.gs1.org/gsmp/kc/epcglobal/tdt]. Also available are the corresponding XSD schema files.

Version control is achieved within each artifact file via version numbers and timestamps of updates.

3.2 TDT Markup

The key elements of the TDT markup are defined in the XSD schema files and shown in Figure 8.

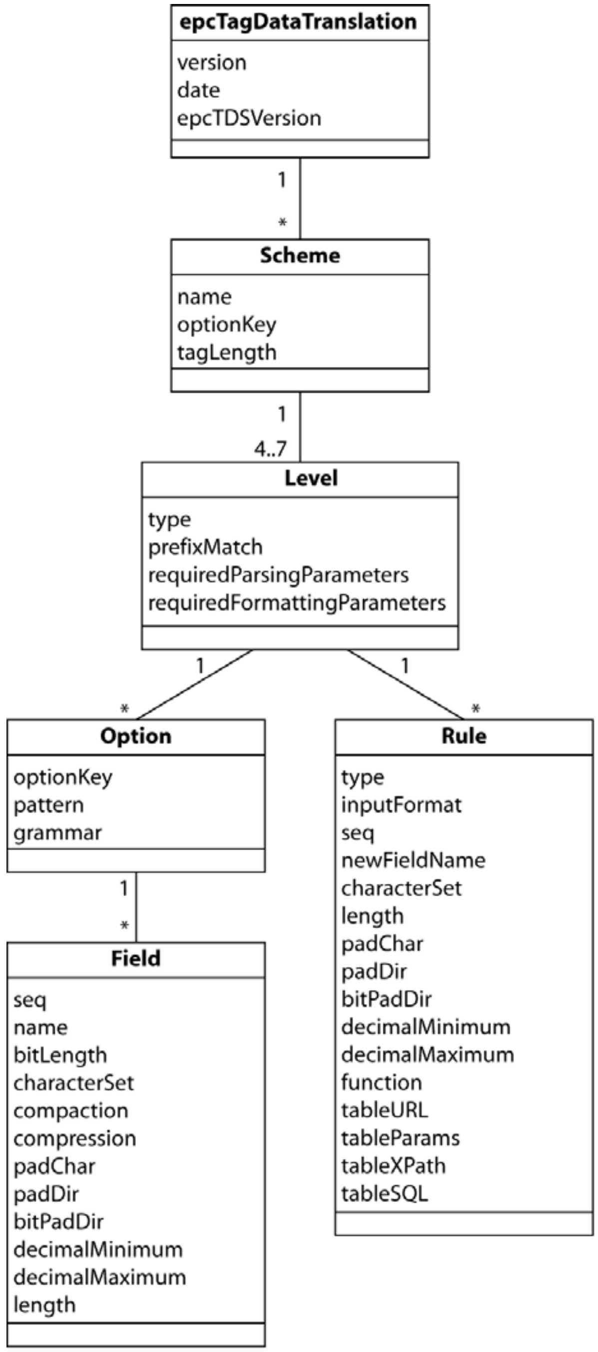

Figure 8 - Tag Data Translation Markup Language schema as a UML class diagram

3.3 Definition of Formats via Regular Expression Patterns and ABNF Grammar

The TDT specification uses regular expression patterns and Augmented Backus-Naur Form (ABNF) grammar expressions to express the structure of the EPC in various levels of representation.

The regular expression patterns are primarily intended to be used to match the input value and extract values of particular fields via groups of bits, digits and characters which are indicated within the conventional round bracket parentheses used in regular expressions.

The regular expression patterns provided in the TDT markup files SHALL be written according to the Perl-Compliant Regular Expressions, with support for zero-length negative lookahead.

It is not sufficient to use the XSD regexp type as documented at `http://www.w3.org/TR/xmlschema-2/` *because it is sometimes useful to be able to use a negative lookahead '*`?!`*' construct within the regular expressions. The implementations of regular expressions in Perl, Java, C#, .NET all allow for negative lookahead. Note that the TDT definition file for ADI-var makes use of the negative lookahead construct in the patterns at the BINARY level in order to make the patterns more restrictive and avoid the situation where a valid binary string might match more than one option.*

The ABNF grammar form allows us to express the outbound string as a concatenation of fixed literal values and fields whose values are variables determined during the translation process. In the ABNF grammar, the fixed literal values are enclosed in single quotes, while the names of the variable elements are unquoted, indicating that their values should be substituted for the names at this position in the grammar. All elements of the grammar are separated by space characters. We use the Augmented Backus-Naur Form (ABNF) for the grammar rather than simple Backus-Naur Form (BNF) in order to improve readability because the latter requires the use of angle brackets around the names of variable fields, which would need to be escaped to < and > respectively for use in an XML document.

The child 'Field' elements within each option allow the constraints and formatting conventions for each individual field to be specified unambiguously, for the purposes of error-checking and validation of EPCs.

The use of regular expression patterns, ABNF grammar and separate nested (child) field elements with attributes for each of the fields allows for the constraints (minimum, maximum values, character set, required field length etc.) to be specified independently for each field, providing flexibility in the URI formats, so that for example an alphanumeric serial number field could co-exist alongside a decimal GS1 Company Prefix field, as would be required to support the full range of possible GRAI codes for a future tag with a larger number of bits devoted to the EPC identifier.

3.4 Determination of the inbound representation

A desirable feature of any Tag Data Translation software is the ability to automatically detect the format of the inbound string received, whether in binary, tag-encoding URI, pure-identity URI, element strings or GS1 identifier keys expressed using Application Identifier (AI) representation, together with additional serialization, where required. Furthermore, the coding scheme should also be detected. The tag-length SHALL either be determined from the input value (i.e. given a binary string or tag-encoding URI), – or otherwise, where the input value does not indicate a particular tag-length (e.g. pure-identity URI, element strings or GS1 identifier keys expressed using Application Identifier (AI) representation, together with additional serialization, where required), the intended tag-length of the output SHALL be specified additionally via the supplied parameters when the input value is either a pure-identity URI, an element string or GS1 identifier key expressed using Application Identifier (AI) representation, together with additional serialization, where required, none of which specify the tag-length themselves. It is important that this initial matching can be done quickly without having to try matching against all possible patterns for all possible schemes, tag lengths and lengths of the GS1 Company Prefix.

For this reason the Tag Data Translation markup files specify a prefix-match for each level of each scheme, which SHALL match from the beginning of the input value. If the prefix-match matches, then the translation software can iterate in further detail through the full regular expression patterns for each of the options to extract parameter values – otherwise it should immediately skip to try the next possible prefix-match to test for a different scheme or different level of representation, without needing to try all the options nested within each of these, since all of the nested regular expression patterns share the same prefix-match.

3.5 Specification of the outbound representation

The Tag Data Translation process only permits encoding or decoding between different representations of the same scheme. i.e. it is neither possible nor meaningful to translate a GTIN into an SSCC – but within any given scheme, it is possible to translate between multiple levels of representation, namely binary, tag-encoding URI, pure-identity URI, human-readable string and GS1 identifier keys expressed using Application Identifier (AI) representation, with or without parentheses. Translation to/from Text Element Identifier strings is also possible for the Aerospace & Defence Identifier (ADI).

With this constraint, it should be possible for Tag Data Translation software to perform a conversion so long as the input value and the outbound representation level are specified.

In addition, Tag Data Translation 1.6 provides for each EPC scheme an output format which is the hostname for which a type 35 ('NAPTR') DNS lookup should be made in order to effect an ONS query. Note that this is an output-only representation, as indicated in Table 2.

3.6 Specifying supplied parameter values

Decoding from the binary level through the tag-encoding URI, pure-identity URI and finally to the element strings or GS1 identifier keys in AI representation only ever involves a potential loss of information. With the exception of the lookup table mapping GS1 Company Prefix Index to GS1 Company Prefix for the 64-bit tags, it is not necessary to specify supplied parameters when decoding, since the binary and tag-encoding formats already contain more information than is required for the pure-identity URI, element string or Application Identifier (AI) formats.

Encoding often requires additional information to be supplied independently of the inbound string. Examples of additional information include:

- Independent knowledge of the length of the GS1 Company Prefix
- Intended length of the physical tag (64-bit, 96-bit …) to be encoded
- Fast filter values (e.g. to specify the packaging type – item/case/pallet)

It should be possible to provide these supplied parameters to Tag Data Translation software. In all the cases above, this may simply populate an internal key-value lookup table or associative array with parameter values additional to those that are automatically extracted from parsing the inbound string using the matching groups of characters within the appropriate matching regular expression pattern.

Note that two specific GS1 identifier keys, namely GTIN and GLN are extended with serial numbers for EPC use. In this situation, the serial number SHALL NOT be passed via the supplied parameters. Instead, the serial number SHALL be passed as part of the input value. For the element string representation, this is achieved by appending the GTIN or GLN with with '`;serial=`' followed by the serial number or serialized extension. For Application Identifier representation, this is achieved through the use of AI 21 for the Serial Number associated with the GTIN - or AI 254 for the serial extension field used in conjunction with the GLN for EPC purposes.

In this way, either the GTIN or GLN and the serial number CAN be obtained as the output value because the same grammar is used for both input and output. This is important because the Tag Data Translation Application Programming Interface (API) defined in Chapter 6 of this document provides no direct access to the private values of intermediate variables or fields used within the translation process. Table 3 shows examples of how the input value should be formatted for serialized identifiers. Note that SSCC, GRAI and GIAI, GDTI and GSRN are already intrinsically serialized and should therefore not be appended with '`;serial=`…' in the element string representation and in the Application Identifier representation, the Application Identifiers (21) or (254) should not be used in conjunction with these GS1 identifier keys.

Coding Scheme	**Example format for input GS1 identifier keys, showing element**

	string or Application Identifier (AI) representation
SGTIN	`gtin=00037000302414;serial=10419703` `(01)00037000302414(21)10419703`
SSCC	`sscc=000370003024147856` `(00)000370003024147856`
SGLN	`gln=0003700030241;serial=1041970` `(414)0003700030241(254)1041970`
GRAI	`grai=00037000302414274877906943` `(8003)00037000302414274877906943`
GIAI	`giai=00370003024149267890123` `(8004)00370003024149267890123`
GSRN	`gsrn=061414123456789012` `(8018)061414123456789012`
GDTI	`gdti=0073796100001` `(253)0073796100001`
GID	`generalmanager=5;objectclass=17;serial=23` [No corresponding AI representation]
USDOD	`cageordodaac=AB123;serial=3789156` [No corresponding AI representation]

ADI	`ADI CAG 359F2/PNO PQ7VZ4/SEQ M37GXB92`
	`ADI CAG 3Y302/SER JK23M895`
	`ADI CAG 3Y302/serial=#284957MH`
	`ADI DAC 4987JK/PNO PQ7VZ4/SEQ M37GXB92`
	`ADI DAC 294HMX/SER JK23M895`
	`ADI DAC 4987JK/serial=#284957MH`
	[TEI strings prefixed with 'ADI' and space character, no corresponding AI representation]

Table 3 Example formats for supplying existing identifier formats as the input value.

Note: Definition files in TDT 1.6 also allow for an alternative representation for EPC identifiers based on GS1 keys for which numeric Application Identifiers are defined in the GS1 General Specifications. This additional level is denoted in the TDT definition files as 'LEGACY_AI' and accepts/returns EPC identifiers in GS1 Application Identifier (AI) notation, such as the prefix (8003) before a GRAI, rather than the construct 'grai='. The human readable representation that was introduced in TDT 1.0 is still denoted 'LEGACY' in the TDT definition files and is available for all EPC identifier schemes, including those which are not based on GS1 keys. TDT 1.6 also introduces two new *representations, 'ELEMENT_STRING' (which is identical to 'LEGACY_AI' except that it contains no parentheses around Application Identifiers) and 'TEI' for Text Element Identifier representation of ADI-var.*

Note that in Tag Data Translation implementations, the values extracted from the inbound EPC representation SHALL always override the values extracted from the supplied parameters; i.e. the parameter string may specify '`filter=5`' – but if the inbound EPC representation encodes a fast filter value of 3, then the value of 3 shall be used for the output since the value extracted from the input value overrides any values supplied via the supplied parameters.

Although many programming languages support the concept of an associative array as a data type, these are not generally portable across different languages in the way that data types such as integer and string are. For this reason, the associative array of key-value pairs for the supplied parameters SHALL be passed as a string format, using a semicolon [;] as the delimiter between multiple key=value pairs. A string in this format can be readily converted into an associative array in most modern programming languages, while remaining portable and language-unspecific.

3.7 Validation of values for fields and fields derived via rules

The `field` element and the `rule` element contain several attributes for validating and ensuring that the values for particular fields fall within valid ranges, both in terms of numeric ranges, as well as lengths of characters, allowed character ranges and the use of padding characters.

TDT markup files use such an explicit markup of the format and constraints of each field in order to provide for a great deal of future extensibility, particularly for encoding alphanumeric characters.

3.8 Restricting and checking decimal ranges for values of fields

In some cases, the numeric range which can be expressed using the specified number of bits exceeds the maximum decimal value permitted for that identifier in its formal specification.

For example, the serial number of an SSCC may be up to ten decimal digits – permitting the decimal numbers 1-9,999,999,999. This requires 34 bits to encode in binary. However, 34 bits would allow numbers in the range 0-17,179,869,183, although those between 10,000,000,000 and 17,179,869,183 are deemed not valid for use as the serial reference of an SSCC – and should result in an error if an attempt is made to encode these into an SSCC.

In order to prevent encoding of numbers outside the ranges permitted by the formal coding specifications, the decimal minimum and decimal maximum limits of each field are indicated via the field attributes `decimalMinimum` and `decimalMaximum`. Where these attributes are omitted, no numeric (minimum,maximum) limits are specified and checking of numeric range NEED NOT be performed by TDT implementations.

Otherwise, where numeric values are specified, the software should check that the value of the field lies within the inclusive range, i.e.

`decimalMinimum` <= field <= `decimalMaximum`

Values which fall outside of the specified range should throw an exception.

3.9 Restricting and checking character ranges for values of fields

The `characterSet` attribute of the `field` element indicates the allowed range of characters which may be present in that field. The range is expressed using the same square-bracket notation as for character ranges within regular expressions. The asterisk symbol following the closing square bracket indicates that 0 or more characters within this range are required to match the field in its entirety. Implementations may find it useful to add a leading caret ('^') and a trailing dollar symbol ('$') to ensure that the characterSet matches the entire field. e.g. for [0-7]* in the TDT markup, TDT implementations may use ^[0-7]*$ as the regular expression pattern.

For example,

[01] permits only characters '0' and '1'*

[0-7] permits only characters '0' thru '7' inclusive*

[0-9] permits only characters '0' thru '9' inclusive*

[0-9 A-Z\-] permits digits '0' thru '9', the SPACE character (ASCII 32) and upper-case letters 'A' thru 'Z' inclusive and the hyphen character.*

The `characterSet` attribute allows checking that all of the characters fall within the permitted range. For example, if a user specifies a serial number for GRAI containing characters that are not wholly numeric, although the character ranges for GRAI-96 and GRAI-64 only permit wholly numeric serial numbers, i.e. characters in the range [0-9], this should result in an error. Note however that an error might not be reported in the situation where a user attempts to encode an alphanumeric GRAI serial code onto a 96-bit tag in the case where the serial code supplied fortuitously happens not to contain any alphabetic characters.

Furthermore, a GRAI can be encoded using two alternative two headers – one for wholly numeric serial numbers (GRAI-96), the other for alphabetic serial numbers (GRAI-170). The presence of the `compaction` attribute SHALL indicate that a particular field is to be interpreted as the binary encoding of a character string; its absence SHALL indicate that the field should be interpreted as an integer value or all-numeric integer string, with leading pad characters if the `padChar` attribute is also present and the integer has fewer digits than the `length` attribute specifies.

Tag Data Translation software SHOULD NOT rely upon particular values of the `characterSet` attribute as an alternative to taking notice of the `compaction` attribute; certain coding schemes, such as the US DOD's CAGE code omit certain characters, such as the letter 'I' in order to reduce confusion with the digit '1', when the CAGE code is communicated in human-readable format – in this case, the `characterSet` attribute may look like '[0-9A-HJ-NP-Z]*', in which case a naïve search for 'A-Z' in the `characterSet` attribute would fail to match, even though the binary value SHOULD BE converted to a character string because the `compaction` attribute was present.

3.10 Padding of fields

Changes since TDT v1.0

Certain fields within either the binary representation, the URI representations and also the element string and AI representations require the padding of the value to a particular number of characters, digits or bits, in order to reach a particular length for that field.

In TDS v1.3, additional EPC identifier schemes were introduced to support GS1 identifiers that have alphanumeric serial codes. Examples of these include the SGTIN-198, SGLN-195, GRAI-170 and GIAI-202. In such schemes, TDS specifies that the alphanumeric serial codes should be encoded using 7 bits per character (7-bit compacted ASCII). In some situations, the alphanumeric serial codes are allowed to have variable length in the GS1 general specifications. This in turn means that the total number of bits required to encode the alphanumeric serial field varies, depending on its length. For the GRAI-170 and GIAI-202 in particular, TDS requires the result of such 7-bit compaction of the serial number to be appended to the right with zero bits to reach a specified total number of bits. This is in marked contrast with the practice of prepending binary padding bits to the left for binary-encoded all-numeric serial numbers, such as those in SGTIN-96.

In version 1.4 of TDT, we took the opportunity to make the rules for padding of fields less ambiguous, both before and after encoding to binary or before and after decoding from binary. The attributes padDir, padChar and length continue to have the same meanings as in TDT v1.0 – but we also explicitly introduced a new `bitPadDir` attribute at the binary level to indicate whether padding with bits is required – and if so, in which direction. This is necessary because since TDS v1.3, it became necessary to also allow for padding with bits to the right, in the case of alphanumeric fields. This was not anticipated in TDT v1.0. The `bitPadDir` attribute is therefore intended to avoid confusion or overloading of meaning on the role of the `padDir` and `padChar` attributes, which continue to play an important role in the padding or stripping of pad characters from the corresponding non-binary field.

When encoding to binary from any other level (hereafter referred to as 'non-binary'), the field itself may be padded (prior to any conversion to binary) with characters such as '0' or space if the `padChar` and `padDir` attributes are present in the binary level.

An example of where this occurs is the CAGE code field in USDOD-96, where the 5-character CAGE code is prepended with a space character to the left before these six characters are encoded in binary as 48 bits. (The reason for this is so that the USDOD-

96 could also accommodate a 6-character DODAAC code instead of a 5-character CAGE code).

After converting to binary, some fields need to be padded either to the left or to the right with leading/trailing zero bits respectively, depending on the value of the new `bitPadDir` attribute.

For example, the serial number in SGTIN-96 has `bitPadDir="LEFT"` to indicate that the binary field should be prepended to the left with zero bits when encoding. In contrast, the serial code of a GRAI-170 or GIAI-202 has `bitPadDir="RIGHT"` to indicate that the binary field should be appended to the right with zero bits when encoding.

When decoding from the binary level to any other non-binary level, there is sometimes a need to strip the leading/trailing bits from a particular direction prior to conversion from binary to integer or character string (depending on the presence/absence and value of the compaction attribute).

An example of this is the stripping of the trailing zeros from the serial field of a GRAI-170 or GIAI-202 upon decoding from binary, before converting to a character string.

After conversion from binary, the field value may need to be padded with characters such as '0' if the padChar and padDir attributes are present in the non-binary level.

An example of where this occurs is the GS1 Company Prefix, which may have significant leading zeros. For example, the GS1 Company Prefix 0037000 would require this.

Alternatively, the sequence of characters decoded from the binary may contain a pad character that needs to be stripped in order to produce the corresponding field inn the non-binary level.

An example of where this occurs is the CAGE code field in USDOD-96, where the 48-bit binary encoding consists of six characters consisting of the 5-character CAGE code, prepended with a space character to the left, which should not appear in the URI representations nor as part of the 5-character CAGE code. (The reason for this is so that the USDOD-96 could also accommodate a 6-character DODAAC code instead of a 5-character CAGE code within the same field).

Because TDS allows bits to be padded either to the left or to the right, depending on the field and EPC identifier scheme, TDT allows the attributes `bitPadDir` and `bitLength` to appear within the `field` or `rule` elements but only when those `field` or `rule` elements are nested within a `level` element that has attribute `type="BINARY"`.

padChar and padDir

The `padChar` attribute SHALL consist of a single character to be used for padding. Typically this is the '0' digit (ASCII character 48 [30 hex]). Other coding schemes MAY specify the space character (ASCII character 32 [20 hex]) or a different character to use.

The `padChar` attribute indicates the non-binary character to be used for padding. If a `field` or `rule` element contains a `padChar` attribute, then within the same level, the field SHALL be padded with repetitions of the character indicated by the `padChar` attribute, in the direction indicated by `padDir` attribute so that the padded value of the field has the length of characters as specified by the `length` attribute. This applies at the validation, parsing, rule execution and formatting stages of the translation process.

The `padDir` attribute SHALL take a string value of either '`LEFT`' or '`RIGHT`', indicating whether the padding characters should appear to the left or right of the unpadded value.

The attributes `length`, `padDir` and `padChar` MAY appear within any `field` or `rule` element of the TDT markup files. Within each `field` element, all three SHALL either be present together – or all three SHALL be absent together. Within `rule` elements, there is no requirement for the `padDir` and `padChar` attributes to be present, even if the `length` attribute is specified; functions defined in rules may return a value which does not require further padding – in this case, the `length` attribute may be specified, merely in order to verify that the result is of the correct length of characters.

When `padChar`, `padDir` and `length` appear as attributes within a `field` or `rule` element within the tag-encoding `level` element, this indicates that the corresponding field in all non-binary levels may need to be padded with the padding character `padChar` within this level of representation.

When `padChar` and `padDir` and `length` appear within a `field` or `rule` within the binary `level` element, this indicates that the field should be padded with the non-binary padding character `padChar` in the direction `padDir` only immediately prior to conversion to binary and that when decoding away from the binary level, such non-binary padding characters should be stripped if the attributes `padChar` and `padDir` are absent from the tag-encoding level.

For example, for a GS1 Company Prefix, all non-binary levels should have `padChar=”0”` and `padDir=”LEFT”` because the leading zeros are significant and should appear in the URI representations, element strings and AI representation.

In contrast, for the CAGE code in USDOD-96, `padChar=” ”` and `padDir=”LEFT”` and these attributes only appear in the binary level, because any leading space padding should be stripped before the CAGE code or DODAAC code is inserted in a URI representation.

For any EPC identifier scheme, the attributes `padChar` and `padDir` should not appear within a field or rule within the binary level if they also appear within the same field or rule within the non-binary levels. If `padChar` and `padDir` are specified in a field or rule within the binary level and also in the corresponding field or rule in any non-binary level, the TDT definition file should be considered invalid. Note that some fields that appear within the binary level do not appear in all non-binary levels. For example, the filter value never appears in the pure-identity URI level. For this reason, in section 3.10.1, the flowchart advises checking of the tag-encoding URI representation to see whether or not padChar and padDir are defined for each field corresponding to the fields defined within the binary level.

bitPadDir and bitLength

For `field` or `rule` elements contained within a `level` element that has attribute `type="BINARY"`, the additional attributes `bitPadDir` and `bitLength` may also appear. The `bitPadDir` attribute may either be absent or if present, must take a string value of either `'LEFT'` or `'RIGHT'`

For the serial number field of SGTIN-96, `bitPadDir='LEFT'`, whereas for the serial code field of GRAI-170, `bitPadDir='RIGHT'`

3.10.1 Summary of padding rules

Figure 9a is a flowchart summary of the rules about whether or not to pad a field (or strip padding characters) when encoding a non-binary field to binary encoding.

Figure 9b is a flowchart summary of the rules about whether or not to pad a field (or strip padding characters) when decoding a binary encoding of a field to a non-binary representation (e.g. to be used in the URI representations, element strings or AI representation).

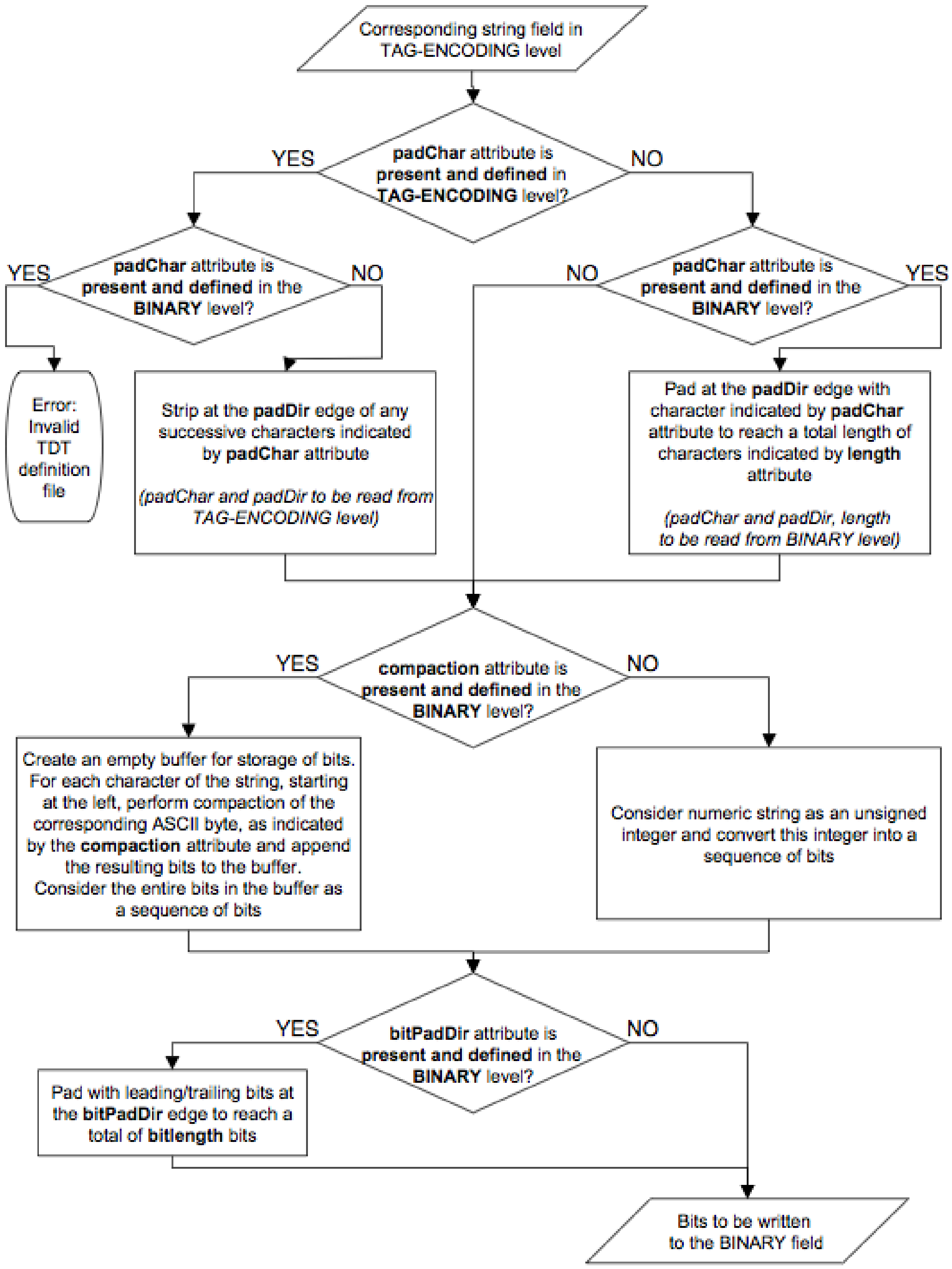

Figure 9a – Summary of rules about whether or not to pad or strip a field when encoding from non-binary representation to binary encoding

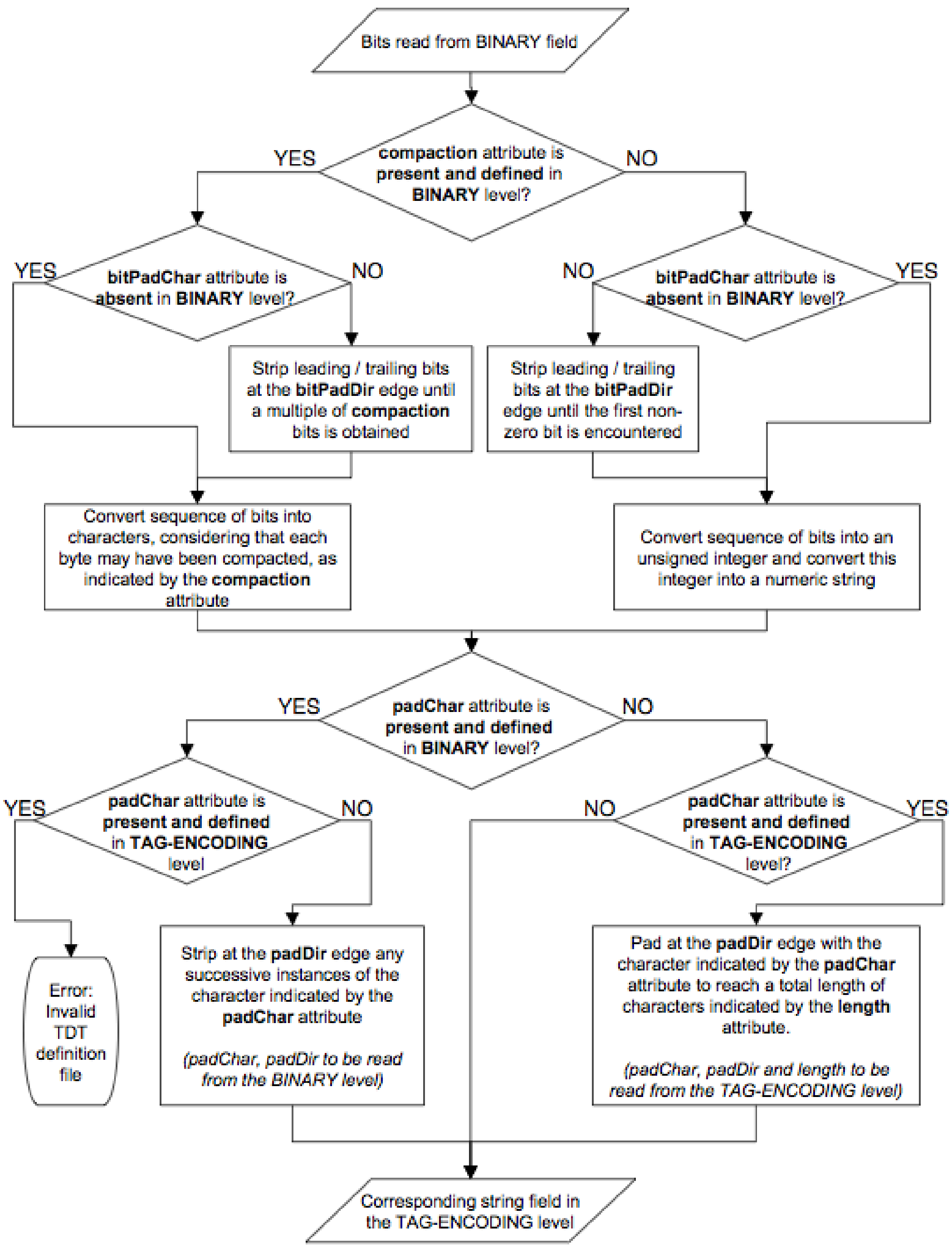

Figure 9b – Summary of rules about whether or not to pad or strip a field when decoding from binary encoding to non-binary representation

For example, for a 96-bit SGTIN, for the field whose `name="companyprefix"`*, the non-binary levels define a* `length` *attribute of 7, a* `padChar` *of '0' and the* `padDir` *as 'LEFT' for the option where* `optionKey` *= 7. For the corresponding binary level where* `optionKey` *=7,* `bitLength` *=24,*`bitpadChar` *='LEFT' and* `compaction,` `padDir` *and* `padChar` *are all absent. This means that when decoding, a 24-bit binary value of '000000001001000010001000' read from the tag for the field named companyprefix should be stripped off its leading zero bits at the LEFT edge, then converted to the integer 37000, then padded to the LEFT with the pad character '0' to 7 characters, yielding '0037000' as the numeric string value for this field.*

For a SGLN where the length of the companyprefix is 12 digits, the location reference is a string of zero characters length. This may result in URIs which look strange because there is an empty string between two successive delimiters, e.g. '..' in a URL which looks like urn:epc:id:sgln:123456789012..12345

This is however correct – and it is incorrect to render the zero-length field as '0' between the period (.) delimiters because '0' is of length 1 character – not zero characters length as required by the `length` *attribute of the appropriate* <field> *element.*

3.11 Compaction and Compression of fields

When strings other than purely numeric strings are to be encoded in the binary level of representation, the `field` element contains two additional attributes, `compaction` and `compression`. Absence of the `compaction` attribute SHALL indicate that the binary value represents an integer or all-numeric string. Presence of the `compaction` attribute SHALL indicate that the binary value represents a character string encoded into binary using a per-character compaction method for economizing on the number of bits required. Allowed values are '`5-bit`', '`6-bit`', '`7-bit`' and '`8-bit`', referring to the compaction methods described in ISO 15962, in which the most significant 3/2/1/0 bits of the 8-bit ASCII byte for each character are truncated.

Note that a `compaction` value of '`8-bit`' SHALL be used to indicate that each successive eight bits should be interpreted as an 8-bit ASCII character, even though there is effectively no compaction or per-byte truncation involved, unlike the other values of the `compaction` attribute. The `compaction` values '16-bit' and '32-bit' are not used in the markup files for this version of the TDT specification – but are reserved in the TDT XSD schema and SHALL indicate 16-bit and 32-bit UNICODE representation where this is required in the future.

The `compression` attribute is intended for future use, to indicate a compression technique to be applied to the value as a whole, rather than on a per-character basis. Permitted values for the `compression` attribute are not currently defined in this version of the Tag Data Translation specification but those values defined in future may indicate compression techniques such as zip / gzip compression, Huffman encoding etc.

3.12 Names of fields used within the TDSv1.6 schemes

The names of fields appearing in the TDT markup files are completely arbitrary but by convention SHALL consist of lower case alphanumeric words with no spaces or hyphens. There are no reserved words and the use of a name within one coding scheme does not imply any correlation with an identically named field within a different coding scheme; each coding scheme effectively has its own namespace for field names. Table 5 lists some field names that are used in the EPC schemes defined in EPC Tag Data Standard v1.6

`filter`	fast filter value – decimal range 0-7
`serial`	serial number – decimal or alphanumeric
`gs1companyprefix`	GS1 company prefix
`gs1companyprefixlength`	length of a GS1 company prefix as a number of characters – decimal integer e.g. for `gs1company prefix` = '0037000' → `gs1companyprefixlength`=7
`tagLength`	64/96/256 etc. – number of bits for the EPC identifier
`gs1companyprefixindex`	an integer-based lookup key for accessing the real gs1Company Prefix – for use with 64-bit tags
`itemref`	Identifies the Object Type or SKU within a particular company for a GTIN
`locationref`	Identifies the Location within a company for a GLN
`assetref`	A serialised asset reference – for use with the GIAI
`serialref`	A serialised reference – e.g. for use with the SSCC
`serviceref`	Identifies the service relation within a particular company for a GSRN
`documenttype`	Identifies the Document Type within a company for a GDTI
`cageordodaac`	Either a Commercial And Government Entity or a Department of Defense Activity Address Code (used with DOD-96 scheme)
`cage`	A Commercial And Government Entity (CAGE) code (also including a NATO CAGE

	(NCAGE) code) - used within the ADI-var scheme)
`dodaac`	A Department of Defense Activity Address Code (used within the ADI-var scheme)
`originalpartnumber`	The original part number (PNO) for an aircraft part (used in ADI-var in the situation where a company serializes uniquely only within the original part number)

Table 5 – Names of fields used within Tag Data Standard v1.6

3.13 Rules and Derived Fields

Certain fields required for formatting the outbound representation are not obtained simply from pattern matching of the inbound representation. A sequence of rules allows the additional fields to be derived from fields whose values are already known.

The reason why this is necessary is that there is often some manipulation of the original identifier codes required in order to translate them into the pure-identity URI representation. Examples include string manipulation such as the relocation of the initial indicator digit or extension digit to the front of the item reference field – or for decoding, the re-calculation of the GS1 checksum – and appending this as the last digit of the GS1 identifier key, where appropriate. Likewise, replacement of the GS1 Company Prefix Index integer by the corresponding GS1 Company Prefix is something that is not readily expressed simply via regular expressions. By working through an example for the GTIN, it is clear that although the processing steps are reversible between encoding into the pure-identity URI and decoding into the GS1 identifier key, the way in which those steps are defined takes on an unsymmetrical appearance in the sequence of rules. An example illustrates this point:

Decoding the GTIN (i.e. translating from pure-identity URI into an element string or Application Identifier representation)

- `indicatordigit = SUBSTR(itemref,0,1);`
- `itemrefremainder = SUBSTR(itemref,1);`
- `gtinprefix = CONCAT(indicatordigit,companyprefix,itemrefremainder);`
- `checkdigit = GS1CHECKSUM(gtinprefix);`

The above are all examples of rules to be executed at the 'EXTRACT' stage, i.e. immediately after parsing the input value.

Encoding the GTIN (i.e. translating from element string or Application Identifier representation into pure-identity URI)

(assumes `gs1companyprefixlength` is passed as a supplied parameter)

- `gtinprefixremainder=SUBSTR(gtin,1,12);`
- `indicatordigit=SUBSTR(gtin,0,1);`
- `itemrefremainder=SUBSTR(gtinprefixremainder,gs1companyprefixlength);`
- `itemref=CONCAT(indicatordigit,itemrefremainder);`
- `gs1companyprefix=SUBSTR(gtinprefixremainder,0,gs1companyprefixlength);`

The above are all examples of rules to be executed at the 'FORMAT' stage, i.e. when constructing the output value.

As the above examples show, the definitions of particular fields (e.g. itemrefremainder) depends upon whether encoding or decoding is being performed (or equivalently, whether the field is required for formatting the output value – or being extracted from the input value), since each successive definition depends on prior execution of the definitions preceding it, in the correct order, in order that all the required fields are available.

The rules in the example above apply generally, with minor modifications to all of the GS1 coding schemes covered in the TDS Specification v1.6. It is worth noting that each of the above rule steps contains only one function or operation per step, which means that even a very simple parser can be used, without needing to deal with nesting of functions in parentheses.

3.14 Core Functions

The core functions which SHALL be supported by Tag Data Translation software in order to encode/decode the GS1 coding schemes are described in Table 6.

`SUBSTR (string, offset)`	the substring starting at `<offset>` (offset=0 is the first character of string)
`SUBSTR (string, offset, length)`	the substring starting at `<offset>` (offset=0 is the first character of string) and of `<length>` characters

`CONCAT (string1, string2, string3,…)`	concatenation of string parameters
`LENGTH(string)`	number of characters of a string
`GS1CHECKSUM (string)`	Computes the GS1 checksum digit given a string containing all the preceding digits
`TABLELOOKUP (inval, tablename, incol, outcol)`	Performs a lookup in table called tablename. Given an input value `<inval>`, look in table `<tablename>` to find a match in column names `<incol>` and return the corresponding value for the same row from output column `<outcol>`. The `TABLELOOKUP` function only indicates the logical lookup – not any bindings. The table URL is specified via a separate attribute `tableURL` and bindings to XPath or SQL expressions are specified via separate attributes `tableXPath` and `tableSQL`.
`add(String, int)`	Converts the String value to integer and adds increment to the converted value. Returns result as a String value
`multiply(String, int)`	Converts the String value to integer and multiplies the converted String with the integer value supplied. Returns the result as a String value
`divide(String, int)`	Converts the String value to integer and divides the converted String by the integer value supplied. Returns the result as a String value
`subtract(String,int)`	Converts the String value to integer and subtracts the supplied integer value from the converted value. Returns result as a String value
`mod(String, int)`	Converts the String to integer and returns the result of the remainder of the converted String after integer division by the integer value supplied. i.e. returns (String **mod** int)

Table 6 - Basic built-in functions required to support encoding and decoding within the GS1 schemes currently covered by the TDS specification

In order to make full use of the Tag Data Translation markup files, implementations of translation software should provide equivalent functions in the programming language in which they are written, either by the use of native functions or custom-built methods, functions or subroutines.

In this version of Tag Data Translation, the requirement that implementations should be able to recalculate check digits only applies to the GS1 coding schemes, when output in the element string or GS1 Application Identifier representation is required. Further details on calculation of the GS1 checksum can be found at `http://www.gs1.org`. It should be noted that ISO 7064 provides a standard for more general-purpose calculation of check digits and that this may be considered in future versions of this specification.

It is important to note that modern programming languages (including Java, C++, C#, Visual Basic, Perl, Python) do not all share the same convention in the definitions of their native functions, especially for string functions. In some languages the first character of the string has an index 0, whereas in others, the first character has an index 1. Furthermore, many of the languages provide a substring function which takes two additional parameters as well as the string itself. Usually, the first of these is the start index, indicating the starting position where the substring should be extracted. However, some languages (e.g. Java, Python) define the last parameter as the end index, whereas others (C++, VB.Net, Perl) define it as the length of the substring, i.e. number of characters to be extracted. Table 7 indicates a number of language-specific equivalents for the three-parameter SUBSTR function in Table 6.

	`SUBSTR(string,offset,length)`	Notes
C++	`String.substr(offset, length);`	
C#	`String.Substring(offset, length);`	
Perl	`substr($stringvariable, offset, length);`	
Visual Basic	`String.Substring(offset,length)`	
Java	`Java.lang.String` `String.substring(beginIndex, endIndex)`	beginIndex = offset endIndex = offset+length
Python	`String[start:end]`	start = offset end = offset+length

Table 7 – Comparison of how substring functions are defined in a number of modern programming languages. The parameters offset and length are of integer type.

Note that for the case of rules which use the `TABLELOOKUP` function, additional attributes `tableURL` and `tableXPath` or `tableSQL` are provided. Tables may be provided in XML format or as comma-separated values (CSV) or tab-separated values

(TSV), even though any Tag Data Translation software MAY internally store the table values in a different format altogether. For this reason, the binding to the original format is handled separately via the `tableURL` and `tableParams and` either `tableXPath` or `tableSQL` attributes, while the `TABLELOOKUP` function expresses the logical lookup, irrespective of the format in which any table is actually supplied.

As an example, consider the GS1 Company Prefix Index lookup tables for use with 64-bit tags. An XML version and a comma-separated values (CSV) version are provided at `http://www.onsepc.com`

For the XML version, `tableURL="http://www.onsepc.com/ManagerTranslation.xml"` and `tableXPath` and `tableParams` are one of the following pairs:

```
tableXPath="/GEPC64Table/entry[@index='$1']/@companyPrefix"
tableParams="companyprefixindex"
```

for the case where

```
function="TABLELOOKUP(companyprefixindex,'GEPC64Table',comp
anyprefixindex,companyprefix)"
```

OR

```
tableXPath="/GEPC64Table/entry[@companyPrefix='$1']/@index"
tableParams="companyprefix"
```

for the case where

```
function="TABLELOOKUP(companyprefix,'GEPC64Table',companypr
efix,companyprefixindex)"
```

The first example pair is used to obtain the value of `companyprefix` given the value of `index` (e.g. retrieve `companyprefix='0037000'` given `companyprefixindex='1'`).

The second example pair is used to obtain the value of `companyprefixindex` given the GS1 `company prefix` (e.g. retrieve gs1`companyprefixindex='1'` given that gs1`companyprefix='0037000'`).

Note that `tableParams` may be a comma-separated string of either fieldnames (if unquoted) or fixed literal values, if wholly numeric or single-quoted strings. The `$1` in the `tableXPath` expressions indicates that the actual value of the field named by the first parameter of `tableParams` string should be substituted into the `tableXPath` expression at this point before passing the XPath expression to an XML DOM parser.

For example, if the value of companyprefix is '0037000', then for the second example pair, the value of '0037000' would be substituted in place of '`$1`' in `tableXPath` so that it would be the following XPath expression:

```
"/GEPC64Table/entry[@companyPrefix='0037000']/@index"
```

which is actually passed to the XML DOM parser.

Where more than one parameter is listed in `tableParams`, `$2` indicates where to substitute the second parameter, while `$3` indicates where to substitute the third parameter, and so on.

A table supplied as comma-separated values (CSV) or tab-separated values (TSV), can be readily converted to a relational database table with the appropriate column headings.

For the example of the GS1 Company Prefix Index table for 64-bit tags, the CSV version is available from `http://www.onsepc.com/ManagerTranslation.csv`

In this case, the attribute

```
tableURI= "http://www.onsepc.com/ManagerTranslation.csv"
```

and the attributes `tableSQL` and `tableParams` may be one of the following pairs:

```
tableSQL="SELECT companyPrefix from GEPC64Table WHERE
index='$1'"
tableParams="companyprefixindex"
```

for the case where

```
function="TABLELOOKUP(companyprefixindex,'GEPC64Table',comp
anyprefixindex,companyprefix)"
```

OR

```
tableSQL="SELECT index from GEPC64Table WHERE
companyPrefix='$1'"
tableParams="companyprefix"
```

for the case where

```
function="TABLELOOKUP(companyprefix,'GEPC64Table',companypr
efix,companyprefixindex)"
```

Each of the two example pairs above corresponds to the respective pairs in the previous examples for the `tableXPath` attributes. Likewise, the notation `$1, $2,` etc. indicates where values of fields named by parameters from the `tableParams` string should be substituted into the `tableSQL` expression before passing to the relational database engine for execution.

4 TDT Markup - Elements and Attributes

4.1 Root Element

The epcTagDataTranslation element is the root element of the TDT definition.

Attributes

Name	Description	Example Values
`version`	TDT Definition version number	1.6
`date`	Creation Date	2011-01-18T11:33Z
`epcTDSVersion`	TDS Specification version	1.6

Elements

Name	Description
`scheme`	Please see scheme definition below for more details

4.2 Scheme Element

For every identifier / coding scheme as defined in the TDS specification, the Scheme element provides details of encoding/decoding rules and formats for use by Tag Data Translation software. In this version of the TDT specification, markup files are provided for the following identifiers: GDTI-96, GDTI-113, GIAI-64, GIAI-96, GIAI-202, GID-96, GRAI-64, GRAI-96, GRAI-170, GSRN-96, SGLN-64, SGLN-96, SGLN-195, SGTIN-64, SGTIN-96, SGTIN-198, ADI-var, SSCC-64, SSCC-96, USDOD-64, USDOD-96.

Attributes

Name	Description	Example Values
`name`	Name of the coding scheme	GDTI-96, GDTI-113, GIAI-64, GIAI-96, GIAI-202, GID-96, GRAI-64, GRAI-96, GRAI-170, GSRN-96, SGLN-64, SGLN-96,

		SGLN-195, SGTIN-64, SGTIN-96, SGTIN-198, ADI-var, SSCC-64, SSCC-96, USDOD-64, USDOD-96
`optionKey`	The name of a variable whose value determines which one of multiple options to select. Note that as of TDT 1.6, `optionKey` is no longer a required attribute within the `<scheme>` element, although it is still specified for fixed-length EPC constructs. Even if the `optionKey` value is not specified within the `<scheme>` element, nested `<option>` elements are nevertheless numbered with an `optionKey` attribute and translation is performed between `<option>` elements having the same value of `optionKey` attribute present within the `<option>` element.	`companyprefixlength`
`tagLength`	This refers to the length of the EPC identifier itself (e.g. the bits encoded from position 20h in the EPC/UII memory bank of a Gen2 tag). The `tagLength` attribute shall not be specified for a variable-length EPC identifier, although it shall be specified for all fixed-length EPC identifiers.	64, 96 or larger values. (The `tagLength` attribute shall not be specified for a variable-length EPC identifier)

Elements

Name	Description

`level`	Contains option elements expressing a pattern, grammar and encoding/decoding rules for each level of representation

4.3 Level Element

This element provides a prefix match for each level of representation. Nested within the `level` element are `option` elements (which provide the `pattern` regular expressions for parsing the input into fields and ABNF `grammar` for formatting the output) and `rule` elements used for obtaining additional fields from functional operations on known fields.

Attributes

Name	Description	Example Values

`type`	Indicates level of representation	`BINARY`
		`TAG_ENCODING`
		`PURE_IDENTITY`
		`LEGACY`
		`LEGACY_AI`
		`ONS_HOSTNAME`

`prefixMatch`	Prefix value required for each encoding/decoding level	00001010 uri:epc:tag:sscc-64 uri:epc:id:sscc sscc= (00)
`requiredParsingParameters`	Comma-delimited string listing names of fields whose values SHALL be specified in the list of suppliedParameters in order to parse the fields of an input value at this level	`gs1companyprefixlength`
`requiredFormattingParameters`	Comma-delimited string listing names of fields whose values SHALL be specified in the list of suppliedParameters in order to format the outbound value at this	`filter,tagLength`

	level	

Elements

Name	Description
`option`	Contains patterns and grammar
`rule`	Contains rules required for determining values of additional variables required

4.4 Option Element

Attributes

Name	Description	Example Values
`optionKey`	A fixed value which the `optionKey` attribute of the `<scheme>` element SHALL match if this option is to be considered, provided that the `optionKey` attribute is specified within the `<scheme>` element. For variable-length EPCs, the `optionKey` attribute might not be specified within the `<scheme>` element but is still used for ensuring that the `<option>` element for the outbound representation is appropriate for the `<option>` element for the inbound representation. For all EPCs, translation shall always be between two `<option>` elements having the same value of their `optionKey` attribute	Any string value but for GS1 identifier keys, the values '6','7','8','9','10','11','12'. In the case of ADI-var, the `optionKey` is used to distinguish between six recognized variations in the way in which the unique identifier may be constructed. In this situation, the `optionKey` is simply a number to represent a particular variation but has no specific correspondence to a particular field.
`pattern`	A regular expression pattern to be used for parsing the input string and extracting the values for variable fields	00101111([01]{4})00100000([01]{40})([01]{36})
`grammar`	An ABNF grammar indicating how the output can be reassembled from a combination	'00101111' filter cageordodaac serial

	of literal values and substituted variables (fields)	*N.B. single quoted string indicate fixed literal strings, unquoted strings indicate substitution of the correspondingly named field values*

Elements

Name	Description
`field`	Provides information about each of the variables, e.g. (min, max) values, allowed character set, length, padding etc.

4.5 Field Element

4.6 Attributes

Name	Description	Example Values
`seq`	The sequence number for a particular sub-pattern matched from a regular expression – e.g. 1 denotes the first sub-pattern extracted	1, 2, 3…
`name`	The name of the variable (or field) – just a reference used to ensure that each field may be used to construct the output format	`filter, companyprefix, itemref, serial,` …
`decimalMinimum`	Decimal minimum value allowed for this field	0
`decimalMaximum`	Decimal maximum value allowed for this field	9999999
`length`	Required length of this field in string characters.	7
`bitLength`	Required length of this field in bits. Omitted for all levels except for the BINARY encoding level	24
`bitPadDir`	Direction to insert '0' to the binary value	`'LEFT', 'RIGHT'`
`characterSet`	Allowed character set for this field, expressed in	[0-9]*,[01]*, [0-9A-HJ-NP-Z]*

	regular expression character range notation	
`padChar`	Character to be used to pad to required value of fieldlength. Omitted if no padding is required for the corresponding field outside of the BINARY level (e.g. within the TAG-ENCODING level)	'0', ' ' (ASCII space character)
`padDir`	Direction to insert pad characters.	`'LEFT', 'RIGHT'`

4.7 Rule Element

Attributes

Name	Description	Example Values
`type`	Indicates at which stage of the process the definition should be evaluated	`'EXTRACT', 'FORMAT'`
`inputFormat`	Indicates whether the input parameter to the definition is in binary format or non-binary ('string') format	`'STRING', 'BINARY'`
`seq`	A sequence number to indicate the running order for definitions sharing the same mode value. The definitions should be run in order of ascending 'seq' value	1,2,3,4,5…
`newFieldName`	A name for the new field or variable whose value is determined by evaluating the definition	Any string consisting of alphanumeric characters and underscore
`function`	An expression indicating how the new field can be determined from a function of already-known fields	e.g. `SUBSTR(itemref,0,1)`

`decimalMinimum`	For numeric fields, the decimal minimum value allowed for this field	e.g. 0
`decimalMaximum`	For numeric fields, the decimal maximum value allowed for this field	e.g. 9999999
`length`	Required length of this field in string characters.	7
`padChar`	Character to be used to pad to required value of fieldlength. Omitted if no padding is required. Present if padding is required.	'0', ' '
`padDir`	Direction to insert pad characters	`'LEFT', 'RIGHT'`
`bitLength`	Required length of this field in bits. Omitted for all levels except for the BINARY encoding level	e.g. 24
`bitPadDir`	Direction to insert '0' to the binary value	`'LEFT', 'RIGHT'`
`characterSet`	Allowed character set for this field, expressed in regular expression character range notation	[0-9],[01]
`tableURL`	A URL where the data table can be obtained	http://www.onsepc.com/ManagerTranslation.xml
`tableXPath`	An XPath expression for obtaining a particular attribute or element value from an XML table. The inline notation '$1', '$2' etc. indicates where the values of the first, second, etc. elements of the tableParams list should be substituted before passing to an XML parsing engine.	/GEPC64Table/entry[@index='$1']/@companyPrefix
`tableSQL`	A SQL expression for obtaining a particular field from a relational database	SELECT companyPrefix FROM GEPC64Table WHERE index='$1'

	table. The inline notation '$1', '$2' etc. indicates where the values of the first, second, etc. elements of the tableParams list should be substituted before passing to a relational database query engine.	
`tableParams`	A comma-delimited string list of fieldsnames whose actual values should be substituted into the tableXPath or tableSQL expressions	e.g. companyprefixindex

5 Translation Process

The execution of the rules in the TDT process takes place at two distinct processing stages, denoted `'FORMAT'` and `'EXTRACT'`, as explained in Table 8:

Stage	Description
`EXTRACT`	Operates on fields after parsing of the inbound value
`FORMAT`	Operates on fields in order to prepare additional fields required by the grammar for formatting the output value.

Table 8 – The two stages for processing rules in Tag Data Translation

The rules for each scheme are within the context of a particular level of representation. The first block of rules, `'EXTRACT'` are tied to the inbound representation level. The last block of rules, `'FORMAT'` is tied to the outbound representation level. Each block may consist of zero or more `rule` elements. The rules within each block are executed in a strict order, as specified by an ascending integer-based sequence number, indicated by the attribute '`seq`' of the `rule` element.

The translation process is described by the following steps:

1. Setup

Read the input value and the supplied extra parameters.

Populate an associative array of key-value pairs with the supplied extra parameters.

During the translation process, this associative array will be populated with additional values of extracted fields or fields obtained through the application of rules of type 'EXTRACT' or 'FORMAT'

Note the desired outbound level.

2. Determine the coding scheme and inbound representation level.

To find the scheme and level that matches the input value, consider all schemes and the `prefixMatch` attribute of each `level` element within each scheme.

If the `prefixMatch` string matches the input value at the beginning, the scheme and level should be considered as a candidate for the inbound representation. If the scheme element specifies a `tagLength` attribute, then if the value of this attribute does not match the value of the `tagLength` key in the associative array, then this scheme and level should no longer be considered as a candidate for the inbound representation.

3. Determine the option that matches the input value

To find the option that matches the input value, consider any scheme+level candidates from the previous step. For each of these schemes, if the `optionKey` attribute is specified within the scheme element in terms of the name of a supplied parameter (e.g. `gs1companyprefixlength`), check the associative array of supplied parameters to see if a corresponding value is defined and if so, select the `option` element for which the `optionKey` attribute of the `option` element has the corresponding value.

e.g. if a candidate scheme has a scheme attribute `optionKey`="`gs1companyprefixlength`" and the associative array of supplied extra parameters has a key=value pair `gs1companyprefixlength`=7, then only the `option` element having attribute `optionKey="7"` should be considered.

If the `optionKey` attribute is not specified within the `scheme` element or if the corresponding value is not present in the associative array of supplied extra parameters, then consider each `option` element within each scheme+level candidate and check whether the `pattern` attribute of the `option` element matches the input value at the start of the string.

When a match is found, this option should be considered further and the corresponding value of the `optionKey` attribute of the `option` element should be noted for use in step 6.

4. Parse the input value to extract values for each field within the option

Having found a scheme, level and option matching the input value, consider the `field` elements nested within the `option` element.

Matching of the input value against the regular expression provided in the `pattern` attribute of the `option` element should result in a number of backreference strings being extracted. These should be considered as the values for the `field` elements, where the `seq` attribute of the field element indicates the sequence in which the fields are extracted as backreferences, from the start of the input value, e.g. the value from the first backreference should be considered as the value of the `field` element with `seq="1"`, the value of the second backreference is the value of the `field` element with `seq="2"`.

For each `field` element, if a `characterSet` attribute is specified, check that the value of the field falls entirely within the specified character set.

For each `field` element, if the `compaction` attribute is null, treat the field as an integer. If the `type` attribute of the input level was `"BINARY"`, treat the string of 0 and 1 characters matched by the regular expression backreference as a binary string and convert it to a decimal integer.

If the `decimalMinimum` attribute is specified, check that the value is not less than the decimal minimum value specified.

If the `decimalMaximum` attribute is specified, check that the value is not greater than the decimal maximum value specified.

If the inbound representation was binary, perform any necessary stripping, conversion of binary to integer or string, padding, referring to the procedure described in the flowchart Figure 9b.

5. Perform any rules of type EXTRACT within the inbound option in order to calculate additional derived fields

Now run the rules that have attribute `type="EXTRACT"` in sequence, to determine any additional derived fields that must be calculated after parsing of the input value.

Store the resulting key-value pairs in the associative array after checking that the value falls entirely within the permitted `characterSet` (if specified) or within the permitted numeric range (if `decimalMinimum` or `decimalMaximum` are specified) and performing any necessary padding or stripping of characters.

6. Find the corresponding option in the outbound representation

To find the corresponding option in the outbound representation within the same scheme, select the `level` element having the desired outbound representation and within that, select the `option` element that has the same value of the `optionKey` attribute that was noted at the end of step 3

7. Perform any rules of type FORMAT within the outbound representation in order to calculate additional derived fields

Run any rules with attribute `type="FORMAT"` in sequence, to determine any additional derived fields that must be calculated in order to prepare the output format.

Store the resulting key-value pairs in the associative array after checking that the value falls entirely within the permitted `characterSet` (if specified) or within the permitted numeric range (if `decimalMinimum` or `decimalMaximum` are specified) and performing any necessary padding or stripping of characters.

8. Use the grammar string and substitutions from the associative array to build the output value

Consider the `grammar` string for that `option` as a sequence of fixed literal strings (the characters between the single quotes) interspersed with a number of variable elements, whose key names are indicated by alphanumeric strings without any enclosing single quotation marks.

Perform lookups of each key name in the associative array to substitute the value of each variable element, substituting the corresponding value in place of the key name.

Note that if the outbound representation is binary, it is necessary to convert values from decimal integer or string to binary, performing any necessary stripping or padding, following the method described in the flowchart Figure 9a.

Concatenate the fixed literal strings and values of variable together in the sequence indicated by the `grammar` string and consider this as the output value.

5.1 Tag Data Translation Software - Reference Implementation

A reference implementation may be a package / object class or subroutine, which may be used at any part of the EPC Network technology stack and integrated with existing software. Additionally, for educational and testing purposes, it will be useful to make a Tag Data Translation capability available as a standalone service, with interaction either via a web page form for a human operator or via a web service interface for automated use, enabling efficient batch conversions.

6 Application Programming Interface

There are essentially two interfaces to consider for Tag Data Translation software, namely a client-side interface, which provides conversion methods for users and a maintenance interface, which ensures that the translation software is kept up-to-date with the latest encoding/decoding definitions data.

6.1 Client API

`public String translate(String epcIdentifier, String parameterList, String outputFormat)`

Translates `epcIdentifier` from one representation into another within the same coding scheme.

Parameters:

`epcIdentifier` – The epcIdentifier to be converted. This should be expressed as a string, in accordance with one of the grammars or patterns in the TDT markup files, i.e. a binary string consisting of characters '0' and '1', a URI (either tag-encoding or pure-identity formats), or a serialized identifier expressed as in Table 3.

`parameterList` – This is a parameter string containing key value pairs, using the semicolon [';'] as delimiter between key=value pairs. For example, to convert a GTIN code the parameter string would look like the following:

filter=3;companyprefixlength=7;tagLength=96

`outputFormat` – The output format into which the epcIdentifier SHALL be converted. The following are the formats supported:

1. `BINARY`
2. `LEGACY`
3. `LEGACY_AI`
4. `TAG_ENCODING`
5. `PURE_IDENTITY`

6. `ONS_HOSTNAME`

Returns:

The converted value into one of the above formats as String.

Throws:

`TDTTranslationException` – Throws exceptions due to the following reason:

1. `TDTFileNotFound` – Reports if the engine could not locate the configured definition file to compile.
2. `TDTFieldBelowMinimum` - Reports a (numeric) Field that fell below the `decimalMinimum` value allowed by the TDT markup
3. `TDTFieldAboveMaximum` - Reports a (numeric) Field that exceeded the `decimalMaximum` value allowed by the TDT markup
4. `TDTFieldOutsideCharacterSet` - Reports a Field containing characters outside the `characterSet` range allowed by the TDT markup
5. `TDTUndefinedField` - Reports a Field required for the output or an intermediate rule, whose value is undefined
6. `TDTSchemeNotFound` - Reported if no matching Scheme can be found via `prefixMatch`
7. `TDTLevelNotFound` - Reported if no matching Level can be found via `prefixMatch`
8. `TDTOptionNotFound` - Reported if no matching Option can be found via the `optionKey` or via matching the `pattern`
9. `TDTLookupFailed` - Reported if lookup in an external table failed to provide a value – reports table URI and path expression.
10. `TDTNumericOverflow` – Reported when a numeric overflow occurs when handling numeric values such as `serial number`.

6.2 Maintenance API

`public void` **`refreshTranslations()`**

Checks each subscription for any update, reloading new rules where necessary and forces the software to reload or recompile its internal representation of the encoding/decoding rules based on the current remaining subscriptions.

7 TDT Schema and Markup Definition

See `http://www.gs1.org/gsmp/kc/epcglobal/tdt` for the latest version of the TDT schema and TDT definition files for each EPC scheme.

8 Glossary (non-normative)

This section provides a non-normative summary of terms used within this specification. For normative definitions of these terms, please consult the relevant sections of the document.

Term	Meaning
[Numbering/Coding] Scheme	A well-defined method of assigning an identification code to an object / shipment / location / transaction
Serialised	Provides a unique serial number for each unique object referenced using that coding scheme
GTIN	Global Trade Item Number – used to identify traded objects and services.
SSCC	Serial Shipping Container Code – provides a globally unique reference number for each shipment
GLN	Global Location Number – used to identify physical locations but also legal and organizational entities and departments
GRAI	Global Returnable Asset Identifier – used to identify returnable assets such as pallets and crates, gas cylinders, etc.
GIAI	Global Individual Asset Identifier – used to identify assets owned by an organisation, which are not being traded – often used for tracking inventory of high value equipment
GSRN	The Global Service Relation Number (GSRN) may be used to identify the recipient of services in the context of a service relationship.
GDTI	The Global Document Type Identifier is the Identification Key for a document type combined with an optional serial number
GID	General Identifier – original hierarchical structure proposed for EPC by Auto-ID Centre. GID is a generic scheme, not specifically aligned with any particular GS1 identifier key or other existing identifier scheme.
GS1 identifier keys	Fundamental identifiers used for distinct purposes and defined in the GS1 General Specifications. Examples include GTIN, SSCC, GLN, GRAI, GIAI, GDTI, GSRN. Tag Data Standard 1.6 defines EPC representations of these.

Term	Meaning
Levels of Representation	The way in which the identifier is represented. Examples of different types of representation include sequences of binary digits (bits), sequences of numeric or alphanumeric characters, as well as Uniform Resource Identifiers (URIs)
Input Value	The identifier to be translated. The format is which it is expressed is the Inbound Representation.
Inbound representation	The way in which the identifier is supplied to the translation software. This may be auto-detected from the input value.
Outbound representation	The way in which the output from the translation software should be expressed. This must be specified by the client.
Binary	A sequence of binary digits or bits, consisting of only the digits '0' or '1'
Non-Binary Form	An integer, numeric or alphanumeric character string when not expressed in the corresponding binary format
URI / URN	A Uniform Resource Identifier / Uniform Resource Name – a string that uniquely identifies any particular object. Unlike a URL (Uniform Resource Locator) which may change when a web page moves from one website to another, the URI is intended to be a permanent reference, fixed for all time – even if the underlying binding to a particular website address changes. The URI is therefore at a higher level of abstraction than a URL. Currently most web browser technology will only resolve URLs – but not URIs.
Tag-Encoding URI	A URI format which encodes the physical tag length and fast-filter values in addition to the information encoded in the pure-identity URI. Intended for low-level applications – e.g. sorting machines, tag writers, etc.

Term	Meaning
Pure-Identity URI	A more abstract URI format that provides each object with a unique identity but conveys no information regarding the physical limitations of the tag used to deliver that EPC. *If an object is tagged with either a 64-bit tag or a 96-bit tag, then although the binary representation and tag-encoding URIs will differ, the pure-identity URI will be the same. Intended for use by high-level applications which are not concerned with writing to tags nor sorting on packaging level.*
Physical Level[s]	Representations where the encoding conveys information about the physical tag length (number of bits) and/or the packaging/classification level of the object. Specifically, the binary representation and tag-encoding URI.
Identity Level[s]	Higher-level representations that say nothing about the physical tag length, nor include explicit information about the packaging/classifcation level. Specifically the pure-identity URI, element string and Application Identifier (AI) representation
Supplied parameters	Parameters that shall be supplied in addition to the input value, mainly because the input value itself lacks specific information required for constructing the output.
Options	Variations to handle variable-length data partitions, such as those resulting from the variable-length GS1 Company Prefix in the GS1 family of coding schemes. Where multiple options are specified, the same number of options should be specified for each level of representation and translation should always translate from the matching option within the inbound level to the corresponding option within the outbound level.
Regular Expression Pattern	A notation for representing sub-patterns of particular groups of characters to match

Term	Meaning
ABNF Grammar	Augmented Backus-Naur Form. Defined in RFC 2234. [http://www.ietf.org/rfc/rfc2234.txt] Notation indicating how the result can be expressed through a concatenation of fixed literal values and values of variable fields, whose values are previously determined.
[Fast] Filter	A number which is used to conveniently select only EPCs of a particular packaging level or classification – e.g. a filter within a smart reader may be configured to report only the cases and pallets – but not all of the items within those cases. The fast filter value may also be used for filtering and sorting.
Header	A binary EPC prefix which indicates the coding scheme and usually also the tag length. Headers of 2 bits and 8 bits are defined in the EPC Tag Data Standard specification
Field	The variable elements of the EPC in any of its representations – each partition or field has a logical role, such as identifying the responsible company (e.g. the manufacturer of a trade item) or the object class or SKU. Tag Data Translation software uses the regular expression pattern to extract values for each field. These may be temporarily stored in variables or an associative array (key-value lookup table) until they are later required for substitution into the outbound format.
Rules	There are already a number of requirements to perform various string manipulations and other calculations in order to comply with the current TDS specification. Neither the regular expression patterns nor the ABNF grammar contain any embedded inline functions. Instead, additional fields are embedded and a separate list of rules are provided, in order to define how their values should be derived from fields whose values are already known. The rules also indicate the context and running order in which they should be executed, namely by specifying the scheme, level and stage of execution (Extract or Format) and the running order as an integer index, with functions executed in ascending order of the sequence number indicated by the `seq` attribute

Term	Meaning
Prefix Match	The Prefix Match is a substring which is used to determine the scheme of the inbound string. This is merely a method of optimizing the performance of translation software by limiting the number of pattern-match tests that are required, since the translation software only attempts full pattern matching and processing for the options of those schemes/levels whose Prefix Match matches at the start of the input value.
OptionKey	The OptionKey is used to identify the appropriate option to use where multiple variations are specified to deal with partitions of variable length. A default strategy may be to simply iterate through all the possible options and find only one where the format string matches the inbound string. However, this approach fails when multiple options match the inbound value. In this case, the translation software can use the enumerated value of the OptionKey to select the appropriate option to use. Each option entry is numbered – and each level specifies (via the name of a field) the appropriate option to choose. For example for the GS1 codes, the level element always specifies that the OptionKey="companyprefixlength" , so for a GS1 Company Prefix of '0037000', then field "companyprefixlength" would be specified as 7 via the supplied parameters and therefore Option #7 would be chosen for both the inbound and outbound levels.
Encoding	A conversion process towards the binary representation, i.e in the direction: Application Identifier (AI) representation or Element String → Pure-identity URI → Tag-encoding URI → Binary
Decoding	A conversion process away from the binary representation, i.e in the direction: Binary → Tag-encoding URI → Pure-identity URI → Application Identifier (AI) representation or Element String → ONS hostname

Term	Meaning
Built-In Functions	Functions that should be supported by all implementations of the tag data translation software, irrespective of the programming language in which the software was actually written. See Table 6.
TDT XML Markup / Definition files	A well-defined machine-readable structured packet of data that represents the patterns, grammar, rules, and field constraints for each identifier coding scheme. Tag data translation software should periodically receive updated versions or patches of the XML markup tables, which it can then use to update its own internal set of rules for performing the conversions, whether this is done at run-time or compile-time. The TDT XML definition files are freely downloadable.
[EPC] [Tag Data] Translation Software	A piece of software that performs conversions between different representations of the EPC within any given coding scheme. The translation software may be a library module or object which may be accessed by / embedded within any technology component in the EPC Network technology stack. It may also be implemented as a standalone service, such as an interactive web page form or a web service for automated batch-processing of conversions.
EPC Tag Data Validation Software	Software which need not perform any transalation but may nevertheless make use of the Tag Data Translation markup files in order to validate that an EPC in any of its representations conforms to a valid format.
EPC Network [Technology] Stack	This consists of several architectural building blocks in order to connect physical objects with information systems. The technology stack includes: EPC – the Electronic Product Code Tags and Readers Filtering and Collection middleware Object Name Service (ONS EPC Information Service (EPCIS).

Term	Meaning
Checksum / Check Digit	A number that is computed algorithmically from other digits in a numerical code in order to perform a very basic check of the integrity of the number; if the check digit supplied does not correspond to the check digit calculated from the other digits, then the number may have been corrupted. The check digit is in a way analogous to a message digest of a data packet or software package – except that message digests tend to be more robust since they consist of strings of several characters and hence many more possible permutations than a single check digit 0-9, with the result that there is a much smaller probability that a corrupted number or data packet will product the same message digest than that it will fortuitously produce a valid check digit. The algorithm for computing the check digit for GS1 coding schemes is specified at http://www.gs1.org/productssolutions/barcodes/support/check_digit_calculator.html ISO 7064 is a standard specifying a generic framework for check digit calculations.
GS1 Company Prefix	A number allocated by GS1 which uniquely specifies a unique company – often the manufacturer of a trade item
GS1 Company Prefix Index	An integer used to obtain the full GS1 Company Prefix via a lookup table, keyed on the smaller integer number of the GS1 Company Prefix Index. This is used with the 64-bit schemes in order to allocate a larger range of bits for the remaining data partitions. The GS1 Company Prefix Index is tabulated in XML and comma-separated value formats at http://www.onsepc.com

9 References

EPCglobal Architecture Framework Document

http://www.gs1.org/gsmp/kc/epcglobal/architecture

TDS - EPCglobal Tag Data Standard

See EPCglobal, “EPC Tag Data Standard”, v1.6 ratified on *** http://www.gs1.org/gsmp/kc/epcglobal/tds/

ONS- Object Naming Service

See EPCglobal, “EPCglobal Object Naming Service (ONS), Version 1.0.1,” Ratified Standard, May 2008, http://www.gs1.org/gsmp/kc/epcglobal/ons/

GTIN – Global Trade Item Number

GLN – Global Location Number

SSCC – Serial Shipping Container Code

GRAI – Global Returnable Asset Identifier

GIAI – Global Individual Asset Identifier

GSRN – Global Service Relation Number

GDTI – Global Document Type Identifier

GS1 (formerly EAN UCC Company Prefix)

GS1 Check Digit Calculation

See http://www.gs1.org under 'The EAN.UCC System' > 'Identification'

US DOD / CAGE and DODAAC codes in passive tags

See http://www.acq.osd.mil/log/rfid/ under 'Passive Tag Data'

NAPTR – Naming Authority Pointer records

See RFC2915 at http://www.ietf.org/rfc/rfc2915.txt?number=2915

PCRE – Perl-Compliant Regular Expressions

See http://www.pcre.org

ABNF – Augmented Backus-Naur Form

See RFC2234 at http://www.ietf.org/rfc/rfc2234.txt?number=2234

URI – Uniform Resource Identifiers

See RFC2396 at http://www.ietf.org/rfc/rfc2396.txt?number=2234

CGI – Common Gateway Interface

See http://hoohoo.ncsa.uiuc.edu/cgi/

UML – Unified Modelling Language

See http://www.uml.org/

ISO AFI – Application Family Identifier

See ISO/IEC 15693 and ISO/IEC 15961 and 15962

UHF Generation 2 Protocol

See EPCglobal, “EPC™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz Version 1.2.0,” EPCglobal Specification, May 2008, http://www.gs1.org/gsmp/kc/epcglobal/uhfc1g2/uhfc1g2_1_2_0-standard-20080511.pdf.

XML DOM (Document Object Model) and XPath

See http://www.w3.org/TR/xpath

10 Acknowledgement of Contributors and Companies Opted-in during the Creation of this Standard (Informative)

Disclaimer

Whilst every effort has been made to ensure that this document and the information contained herein are correct, GS1 EPCglobal and any other party involved in the creation of the document hereby state that the document is provided on an “as is” basis without warranty, either expressed or implied, including but not limited to any warranty that the use of the information herein with not infringe any rights, of accuracy or fitness for purpose, and hereby disclaim any liability, direct or indirect, for damages or loss relating to the use of the document.

Below is a list of active participants and contributors in the development of TDT 1.6. This list does not acknowledge those who only monitored the process or those who chose not to have their name listed here. Active participants status was granted to those who generated emails, attended face-to-face meetings and conference calls that were associated with the development of this standard.

Member	Company	Member Type or WG Role
Dr. Mark Harrison	Auto-ID Labs	Editor of TDT 1.6
Rajiv Singh	Garud Technology Services Inc	Member
Ms. Sue Schmid	GS1 Australia	Member
Kevin Dean	GS1 Canada	Member
Mr. Han Du	GS1 China	Member
Mr. Lionel Willig	GS1 France	Member
Mr. Ralph Troeger	GS1 Germany	Member
Ian Robertson	GS1 Global Office	GS1 GOStaff
Mark Frey	GS1 Global Office	GSMP Group Facilitator/Project Manager/ISO Liaison WG4/SG1
Frank Sharkey	GS1 Global Office	GS1 GO Staff
KK Suen	GS1 Hong Kong	Member
Mr. Koji Asano	GS1 Japan	Member
Ms. Reiko Moritani	GS1 Japan	Member
Ms. Yuko Shimizu	GS1 Japan	Member
Mrs. Sylvia Stein	GS1 Netherlands	Member/Facilitator
Ms. Alice Mukaru	GS1 Sweden	Member
Mr. Heinz Graf	GS1 Switzerland	Member
Ray Delnicki	GS1 US	Member
Mr. James Chronowski	GS1 US	Co-chair
Ken Traub	Ken Traub Consulting LLC	Editor of TDS 1.6
Denton Clark	Lockheed Martin	Member
Dr. Patrick King	Manufacture francaise des Pneumatiques Michelin	Member
Mr. Rick Schuessler	Motorola	Co-chair
Mr. Henk Dannenberg	NXP Semiconductors	Member
Kevin Ung	The Boeing Company	Member
Steve Lederer	The Goodyear Tire & Rubber Co.	Member

*Prior to this version of TDT 1.6 being created in the TDTS WG, previous versions were created in the SAG TDT WG where Mark Harrison and Vijay Sundhar presided as co-Chairs.

The following list in corporate alphabetical order contains all companies that were opted-in to the Tag Data and Translation Standard Working Group and have signed the EPCglobal IP Policy as of June 24, 2011

Company Name
Auto-ID Labs
Garud Technology Services Inc
GS1 Australia
GS1 Austria
GS1 Canada
GS1 China
GS1 France
GS1 Germany
GS1 Global Office
GS1 Hong Kong
GS1 Ireland
GS1 Japan
GS1 Korea
GS1 Netherlands
GS1 New Zealand
GS1 Poland
GS1 Sweden
GS1 Switzerland
GS1 UK
GS1 US
Impinj, Inc
INRIA
Ken Traub Consulting LLC
Lockheed Martin
Manufacture francaise des Pneumatiques
Michelin
Motorola
NXP Semiconductors
QED Systems
The Boeing Company
The Goodyear Tire & Rubber Co.