The Global Language of Business

EPC Tag Data Translation Standard (TDT)

Release 2.2, Ratified, Feb 2025

4 Document Summary

Document Item	Current Value
Document Name	EPC Tag Data Translation Standard (TDT)
Document Date	Feb 2025
Document Version	2.2
Document Issue	
Document Status	Ratified
Document Description	

5 Contributors (Participant in TDS/TDT 2.0 MSWG)

Name	Organisation
Dr. Mark Harrison (Chair)	Milecastle Media Limited
Jeanne Duckett (Chair)	Avery Dennison RFID
Jaewook Byun	Auto-ID Labs at KAIST
Jin Mitsugi	Auto-ID Labs at Keio University
HJ Cha	Avery Dennison RFID
John Gallant	Avery Dennison RFID
Akane Mitsui	Avery Dennison RFID
Kevin Berisso	BAIT Consulting
Shi Yu	Beijing REN JU ZHI HUI Technology Co. Ltd.
Tony Ceder	Charmingtrim
François-Régis DOUSSET	DANONE SPA
Olivier Joyez	DECATHLON
Yousuke Okayama	DENSO WAVE Incorporated
Michael Isabell	eAgile Inc.
Jim Springer	EM Microelectronic
Odarci Maia Junior	EMPRESA BRASILEIRA DE CORREIOS E TELEGRAFOS
Julie McGill	FoodLogiQ
Aruna Ravikumar	GS1 Australia
Sue Schmid	GS1 Australia
Jeroen van Weperen	GS1 Australia
Ethan Ward	GS1 Australia
Eugen Sehorz	GS1 Austria
Luiz Costa	GS1 Brasil
Roberto Matsubayashi	GS1 Brasil
Huipeng Deng	GS1 China

Name	Organisation
Zhimin Li	GS1 China
Gao Peng	GS1 China
Yi Wang	GS1 China
Ruoyun Yan	GS1 China
Marisa Lu	GS1 Chinese Taipei
Sandra Hohenecker	GS1 Germany
Roman Winter	GS1 Germany
GSMP Calendar	GS1 Global Office
Steven Keddie	GS1 Global Office
Timothy Marsh	GS1 Global Office
Craig Alan Repec	GS1 Global Office
Greg Rowe	GS1 Global Office
John Ryu	GS1 Global Office
Claude Tetelin	GS1 Global Office
Elena Tomanovich	GS1 Global Office
Wayne Luk	GS1 Hong Kong, China
K K Suen	GS1 Hong Kong, China
Judit Egri	GS1 Hungary
Linda Vezzani	GS1 Italy
Koji Asano	GS1 Japan
Kazuna Kimura	GS1 Japan
Noriyuki Mama	GS1 Japan
Mayu Sasase	GS1 Japan
Yuki Sato	GS1 Japan
Sergio Pastrana	GS1 Mexico
Sarina Pielaat	GS1 Netherlands
Gary Hartley	GS1 New Zealand
Alice Mukaru	GS1 Sweden
Heinz Graf	GS1 Switzerland
Shawn Chen	GS1 US
Norma Crockett	GS1 US
JONATHAN GREGORY	GS1 US
Ned Mears	GS1 US
Andrew Meyer	GS1 US
Gena Morgan	GS1 US

Name	Organisation
Amber Walls	GS1 US
Guilda Javaheri	Golden State Foods
Megan Brewster	Impinj, Inc
Shekhar Nambi	JOHNSON & JOHNSON PTE LTD
Shinichi Ike	Johnson & Johnson
Blair Korman	Johnson & Johnson
Ausias Vives	Keonn Technologies SL
Fabian Moritz Schenk	Lambda ID GmbH
Don Ferguson	Lyngsoe Systems Ltd.
Danny Haak	Nedap
Marisa Campos	PROAGRIND, Lda.
Chris Brown	Printronix Auto ID
Jeffrey Chen	Printronix Auto ID
Akshay Koshti	Robert Bosch GmbH
Mo Ramzan	SML
Jerome Torro	SNCF Rolling Stock Department
Holly Mitchell	Seagull Scientific
Masatoshi Oka	TOPPAN
Taira Wakamiya	TOPPAN
Albertus Pretorius	Tonnjes ISI Patent Holding GmbH
Elizabeth Waldorf	TraceLink

6 Log of Changes

Release	Date of Change	Changed By	Summary of Change
1.0	Jan 2006		Original publication

Release	Date of Change	Changed By	Summary of Change
1.4	Jun 2009		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 3-1 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)
1.6	Sep 2011	Mark Harrison	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 section 3.10) regarding the padding and stripping of characters or bits when translating between the binary level and other levels; the term 'NON-BINARY' is replaced with 'TAG-ENCODING URI' since only the tag-encoding URN format 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 Section 3.10. (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 & Tables to mention additional formats 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.

Release	Date of Change	Changed By	Summary of Change
2.0	May 2023	Mark Harrison Craig Alan Repec	**WR-21-319**: Update to the latest GS1 branding. Update all existing TDT artefacts, add new TDT artefacts for missing EPC schemes (including new EPC schemes introduced in Tag Data Standard 2.0) and supporting tables introduced in TDS 2.0. Provide all current artefacts in XML and JSON and include support for GS1 Digital Link URIs, while dropping support for ONS hostname (no longer of use). Improve support for lookup of length of GS1 Company Prefix for older EPC schemes and improve handling of percent-encoding of symbol characters in URN and Web URI formats. Add chapter regarding encoding of additional AIDC data after the EPC in the EPC/UII memory bank for new EPC schemes. Add new sections for new encoding/decoding methods introduced in TDS 2.0 and to explain the use of 'encodedAI' for encoding GS1 Application Identifiers within the new EPC identifiers introduced in TDS 2.0. Added explanation of new attribute 'valueIfNull' to correctly handle SGLN schemes in which the GLN extension (254) is not expressed in element string, GS1 Digital Link URI or bare identifier.
2.1 draft (unreleased)	Apr 2024	Mark Harrison Craig Alan Repec	**WR-24-019**: Updated TDT artefacts "TDT_TableF.json" and "TDT_TableF.xml" to align with Table F in TDS 2.1 (February 2024), as follows: The entry for AI 37 is corrected from `"f":4, "g":8` to `"g":4, "h":8`, as was already the case for AI 30, to fix incorrect column lettering. Entries for AIs 3900-3909 is corrected from `"f":4, "g":15` to `"g":4, "h":15`, to fix incorrect column lettering. Entries for AIs 4330-4333, 7011, 7241-7242 and 8030 are added to provide support for these new AIs, which are new in Release 24.0 of the GS1 General Specifications [GS1GS] (January 2024). Introduced GS1_AI_JSON input/output format for Tag Data Translation as a less ambiguous alternative to GS1_ELEMENT_STRING – see Section 0.

Release	Date of Change	Changed By	Summary of Change
2.2	Feb 2025	Mark Harrison Craig Alan Repec Nick Porter	Updated TDT_TableF.json and TDT_TableF.xml to support all new GS1 Application Identifiers from ratified GSCNs for the Release 25.0 of the GS1 General Specifications [GS1GS] (January 2025), namely (7041), (716), (7250)-(7259) as well as including errata fixes for some missing/incorrect details for GS1 AIs (20), (242), (30), (3100)-(3695), (37), (3900)-(3953), (402), (421)-(426), (4309), (7004), (7030)-(7039), (8001), (8005), (8011) Added new parameter `aiSequence` within `option` to indicate which GS1 Application Identifiers are encoded within the EPC identifier when using that `option`. This is used for pre-processing the input when the input format is GS1_AI_JSON or GS1_DIGITAL_LINK, in order to ensure that the regular expression `pattern` provided within the TDT definition file can match. Added new parameter `gs1DigitalLinkKeyQualifiers` within `level` for GS1_DIGITAL_LINK only to indicate the permitted sequential order of GS1 Application Identifiers that may appear in the URI path information after the primary identification key. This is used for post-processing the output when GS1_DIGITAL_LINK is selected as the output format, in order to ensure that those GS1 Application Identifiers that should be expressed in the URI path information do so, in the correct sequence. Please see new section 3.4.1 for further details about pre-processing input values and post-processing output values, due to the limitations of regular expression patterns used within the TDT definition file framework, as well as limitations of ABNF grammar, especially regarding correct handling of GS1 Application Identifiers appearing within the URI path information of GS1 Digital Link URIs. Updated UML class diagram to show these new parameters. Updated all TDT definition files to provide the `length` parameter directly within the `field` of each BINARY `level` so that the corresponding length of the value after conversion from binary can be more easily accessed (previously only shown for the corresponding `field` within the TAG_ENCODING `level`).

Disclaimer

GS1 seeks to minimise barriers to the adoption of its standards and guidelines by making the intellectual property required to implement them available, to the greatest extent possible, on a royalty-free basis, or when necessary, under a RAND licence. Such royalty-free and RAND licences are provided pursuant to the GS1 IP Policy (available here: https://www.gs1.org/standards/ip), which governs the work of work group participants who contribute to drafting standards and guidelines, including this document. In addition to licences, the GS1 IP Policy provides various benefits and obligations that apply to all implementers of GS1 standards and guidelines, and all implementations of GS1 standards are subject to those terms.

Nevertheless, please note the possibility that an implementation of one or more features of this standard or guideline may be the subject of a patent or other intellectual property right that is not covered by the licences granted pursuant to the IP Policy. In addition, the licences granted under the IP Policy do not include the IP rights or claims of third parties who were not participants in the corresponding standard development work group.

Table of Contents

0 Changes relative to previous versions 13

1 Introduction 14
1.1 What is an EPC? 14
1.2 Where is an EPC defined? – in the GS1 EPC Tag Data Standard 16
1.3 What is GS1 EPC Tag Data Translation? 18

2 Translation between various formats 21

3 Structure of TDT definition files 23
3.1 Patterns (Regular Expressions) 27
3.2 Grammar (Augmented Backus-Naur Form [ABNF]) 27
3.3 Rules for obtaining additional fields 28
3.4 Using the information in TDT definition files within a translation process 28
3.4.1 Pre-processing of input and post-processing of output 29
3.5 Definition of formats via Regular Expression Patterns and ABNF Grammar 32
3.6 Determination of the input format 33
3.7 Specification of the output format 33
3.8 Specifying supplied parameter values 33
3.9 Validation of values for fields and fields derived via rules 35
3.10 Restricting and checking ranges for values of numeric fields in base 10 35
3.11 Restricting and checking character ranges for values of fields 36
3.12 Padding of fields 37
3.12.1 Changes since TDT v1.0 37
3.12.2 padChar and padDir 38
3.12.3 bitPadDir and bitLength 39
3.12.4 Summary of padding rules 39
3.13 Compaction and Compression of values of fields 42
3.14 Values of the name property of field objects within TDT definition files 42
3.15 Rules and Derived Fields 49
3.15.1 Decoding the GTIN (i.e. translating from pure-identity URN into an element string or Application Identifier format) 50
3.15.2 Encoding the GTIN (i.e. translating from element string or Application Identifier format into pure-identity URN) 50
3.16 Core Functions 50
3.17 Encoded GS1 Application Identifiers in new EPC schemes introduced in TDS 2.0 53

4 Encoding/Decoding of additional AIDC data after the EPC 60
4.1 Encoding additional AIDC data after the EPC 60
4.2 Decoding additional AIDC data after the EPC 61

5 TDT Definition Files – formal definition 64
5.1 Root object 64
5.1.1 Datatype Properties (inline XML attributes) 64
5.1.2 Object Properties (nested XML elements) 64
5.2 Scheme object 64
5.2.1 Datatype Properties (inline XML attributes) 65

5.2.2 Object Properties (nested XML Elements) 65
5.3 Level object 66
5.4 Option object 67
5.5 Field object 69
5.5.1 Rule object 71
5.6 encodedAI object 72

6 Translation Process 73

7 Tag Data Translation Software - Reference Implementation 75

8 Application Programming Interface 75
8.1 Client API 76
8.2 Maintenance API 77

9 TDT Schema, TDT Definition Files and TDT Tables 77

10 Glossary (non-normative) 77

11 References 81

12 Flowcharts to assist encoding and decoding GS1 Application Identifiers (within new EPC schemes and for additional AIDC data) 83
12.1 Encoding GS1 Application Identifiers 84
12.2 Decoding GS1 Application Identifiers 96

Index of figures

Figure 1-1 Overview of EPC schemes and correspondence to GS1 Application Identifiers 14
Figure 1-2 Different formats of EPC as used in different layers of the GS1 System Architecture 15
Figure 1-3 EPC schemes and their various formats..........17
Figure 1-4 Translation between different formats using TDT definition files and tables..........19
Figure 1-5 Tag Data Translation process with examples of different formats. 20
Figure 2-1 Flowchart showing input and output parameters to a Tag Data Translation process. 21
Figure 2-2 Encoding and Decoding between different formats of an EPC. Note that when encoding, additional parameters may need to be specified...........22
Figure 3-1 UML class diagram for TDT definition files 24
Figure 3-2 SGTIN-96 levels of representation..........26
Figure 3-3 SGTIN-96 levels with multiple encoding options..........26
Figure 3-4 SGTIN+ levels of representation 27
Figure 3-5 Summary of rules about whether or not to add or remove padding to a field when encoding from other formats to binary encoding 40
Figure 3-6 Summary of rules about whether or not to pad or strip a field when decoding from binary encoding to any format other than binary..........41
Figure 3-7 Encoding GS1 Application Identifiers..........58
Figure 3-8 Decoding GS1 Application Identifiers 59
Figure 4-1 Encoding AIDC data..........61
Figure 4-2 Decoding AIDC data 63
Figure 12-1 Encoding each additional piece of AIDC data..........84
Figure 12-2 E0 - Encoding +AIDC data after an EPC 85
Figure 12-3 E1 - Decoding one or more elements of encodedAI in new EPC schemes..........86
Figure 12-4 E2- Decoding each element of encodedAI in new EPC schemes 87
Figure 12-5 E3 - Encoding the first component..........88
Figure 12-6 E4 - Encoding the encoding indicator for the first component..........89
Figure 12-7 E5 - Encoding the length indicator for the first component..........90
Figure 12-8 E6 - Encoding the value for the first component..........91
Figure 12-9 E7 - Encoding the second component 92
Figure 12-10 E8 - Encoding the encoding indicator for the second component..........93
Figure 12-11 E9 - Encoding the length indicator for the second component 94
Figure 12-12 E10 - Encoding the value for the second component..........95
Figure 12-13 Decoding each additional piece of AIDC data 96
Figure 12-14 D0 - Decoding +AIDC data after an EPC..........97
Figure 12-15 D1 - Use Table K to determine the length of an AI key..........98
Figure 12-16 D2 - Decoding one or more elements of encodedAI in new EPC schemes..........99
Figure 12-17 D3 - Decoding each element of encodedAI in new EPC schemes..........100
Figure 12-18 D4 - Decoding the first component..........101
Figure 12-19 D5 - Decoding the encoding indicator for the first component..........102
Figure 12-20 D6 - Decoding the length indicator for the first component..........103
Figure 12-21 D7 - Decoding the value for the first component 104
Figure 12-22 D8 - Decoding the second component..........105
Figure 12-23 D9 - Decoding the encoding indicator for the second component..........106
Figure 12-24 D10 - Decoding the length indicator for the second component..........107
Figure 12-25 D11 - Decoding the value for the second component 108

Index of tables

Table 3-1 – Example formats for supplying existing identifier formats as the input value. 34
Table 3-2 Field names used within TDT definition files 43
Table 3-3 Basic built-in functions required to support encoding and deciding within the EPC schemes currently defined in TDS 2.0 50
Table 3-4 Comparison of how substring functions are defined in a number of modern programming languages. The parameters offset and length are of integer type...52
Table 6-1 The two stages for processing rules in Tag Data Translation..73

0 Changes relative to previous versions

Before TDT v2.2, an element string input was supported for both input and output for implementations of GS1 EPC Tag Data Translation. This aligned closely with the format for human-readable information (HRI), in which the GS1 Application Identifier keys are enclosed within round brackets. Multiple GS1 Application Identifiers and their values were specified within a single concatenated string without line break characters. That approach unfortunately had a potential ambiguity, since the round bracket characters were also valid literal characters within the GS1 AI encodable character set 82 (see GS1 Gen Specs Figure 7.11-1). The new scheme DSGTIN+ introduced the possibility that a date value (such as expiration date expressed via GS1 AI (17)) could be specified in addition to the GTIN (01) and Serial Number (21). This led to a potential ambiguity in the interpretation of an input string such as this:

(01)01234567890128(21)ABC123(17)240422

Should this be interpreted as having expiration date of 22nd April 2024 and Serial Number "ABC123" or is the serial number actually "ABC123(17)240422" ?

To avoid this ambiguity, in TDT 2.2, such ambiguous element string syntax is dropped in favour of a JSON object syntax, in which each GS1 Application Identifier key is enclosed in double quotes and separated by a colon from its value (also in double quotes), with a comma separating multiple key:value pairs.

Using the above example, JSON object syntax enables unambiguous distinction between:

{"01":"01234567890128","21":"ABC123","17":"240422"}

vs

{"01":"01234567890128","21":"ABC123(17)240422"}

Such JSON object syntax is supported in TDT 2.2 with the representation level 'GS1_AI_JSON' as a replacement for the former representation level 'GS1_ELEMENT_STRING'.

For each EPC scheme, there is a TDT definition file with a hierarchical structure: `scheme` > `level` > `option` > `field` in which various details are provided at each appropriate layer in the hierarchy. This hierarchical structure is explained in further detail in section 3 of this standard.

The default ordering of all-numeric keys within JSON objects can be somewhat unpredictable or counterintuitive. For this reason, the TDT definition files include a new parameter named `aiSequence`, which appears within each `option` within the `level` for 'GS1_AI_JSON'. The value of this parameter `aiSequence` is a JSON square bracket array of double-quoted numeric strings to specify the order in which the GS1 Application Identifier keys are expected within the `pattern` and `grammar` parameters within each `option` for that `level` for 'GS1_AI_JSON'.

For example, within the 'GS1_AI_JSON' `level` of DSGTIN+, the `option` with `optionKey` equal to '4' has the following values for pattern, grammar and aiSequence:

<table>
<tr><td>pattern</td><td>"^\\{\\s*\"01\"\\s*:\\s*\"([0-9]{14})\"\\s*,\\s*\"21\"\\s*:\\s*\"((?:[!%-?A-Z_a-z]|\\\\\"){1,20})\"\\s*,\\s*\"17\"\\s*:\\s*\"([0-9]{6})\""</td></tr>
<tr><td>grammar</td><td>"'{\"01\":\"' gtin '\",\"21\":\"' serial '\",\"17\":\"' expDate '\"}'"</td></tr>
<tr><td>aiSequence</td><td>"[\"01\",\"21\",\"17\"]"</td></tr>
</table>

1 Introduction

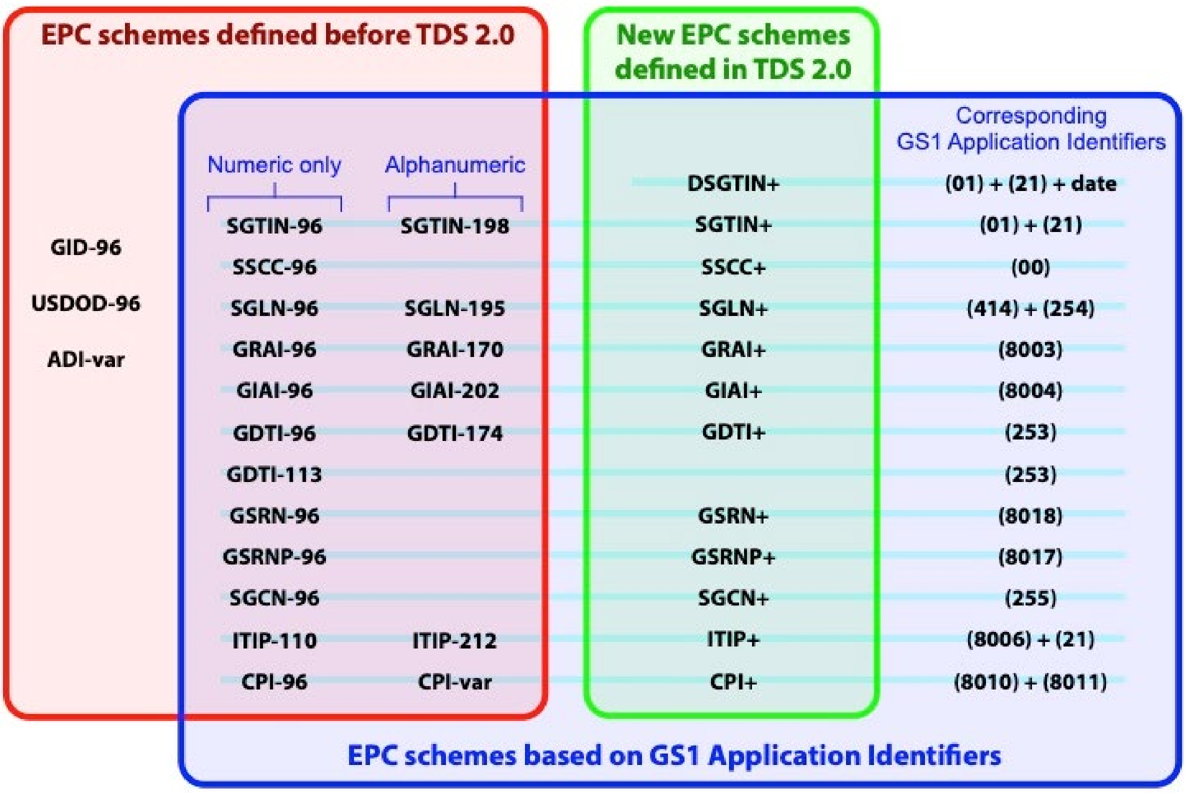
This chapter provides an introduction about the principles of an Electronic Product Code [EPC] and the complementary roles of the GS1 EPC Tag Data Standard [TDS] (that normatively defines the formats and encoding/decoding rules for EPCs) and this standard, the GS1 EPC Tag Data Translation Standard [TDT] that makes such details more readily available to software as machine-readable data.

1.1 What is an EPC?

The Electronic Product Code (EPC) is a globally unique instance-granularity identifier that is designed to allow the automatic identification of objects anywhere. Two different physical objects should not share the same EPC identifier. Such instance-level identification enables each individual physical object to be tracked or traced individually as it moves through a supply chain or value network and the same EPC should not appear simultaneously in two vastly different locations over the same time period. EPC classes refer to collections of EPC instance identifiers that share common characteristics. Examples of EPC classes include classes for GTIN, GTIN+Lot (LGTIN). Note that although such EPC classes can be reported in EPCIS event data, an EPC class typically contains multiple members, so class-level traceability does not offer the high fidelity of instance-level identification.

The majority of EPC schemes are defined for GS1 identifiers at instance-level granularity, although a small number of EPC schemes are defined for non-GS1 identifiers. Figure 1-1 provides an overview of current EPC schemes and the correspondence to instance-level GS1 identifiers. An important feature in this Venn diagram is the grouping into older EPC schemes that were already defined before TDS v2.0 and newer EPC schemes that were introduced in TDS v2.0. The differences between these are discussed further in section 1.2 about TDS.

Figure 1-1 Overview of EPC schemes and correspondence to GS1 Application Identifiers

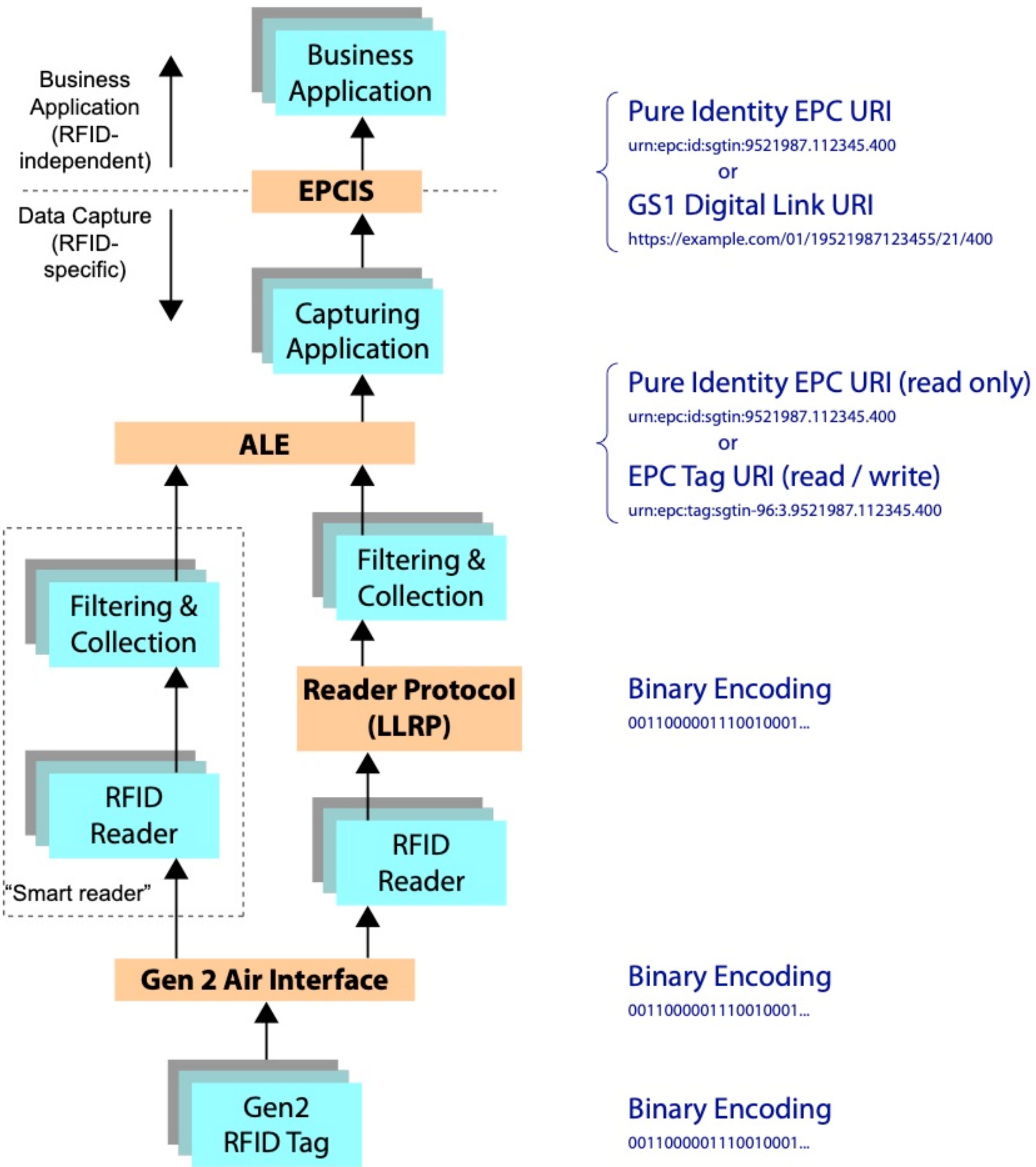


Note: GDTI-113 is deprecated since GS1 Tag Data Standard v1.9

Formally, an EPC is agnostic to the data carrier technology in which it is encoded. Although an EPC is often associated with low-cost passive RFID tags (in which it is encoded in a compact binary format), it can also be expressed as equivalent information in 2D bar codes as element strings or in information systems (such as EPCIS event data), typically in a URI format that is independent of the data carrier that was read. For example, a GS1 DataMatrix symbol that encodes a GTIN and Serial Number (corresponding to GS1 Application Identifiers (01) and (21) respectively) can be considered to be a data carrier expressing an SGTIN EPC identifier, even though it is encoded in a GS1 DataMatrix symbol as an element string rather than using the corresponding EPC binary format. Similarly, for extended packaging applications, the same SGTIN might instead be expressed as a GS1 Digital Link URI encoded natively within a QR Code®.

Figure 1-2 shows how different formats of EPC are used in different layers of the GS1 System Architecture [GS1Arch].

Figure 1-2 Different formats of EPC as used in different layers of the GS1 System Architecture



1.2 Where is an EPC defined? – in the GS1 EPC Tag Data Standard

The GS1 EPC Tag Data Standard [TDS] indicates how GS1 identification keys (GTIN, GLN, SSCC, GRAI, GIAI, GSRN, GSRNP, GDTI, GCN, ITIP, CPI) and a small number of other identifier constructs should be expressed as an Electronic Product Code (EPC).

For most EPC schemes, TDS defines a compact binary format suitable for encoding within the EPC/UII memory bank of an RFID tag that could be attached to tangible physical objects, such as individual instances of products, assets, components, coupons, loyalty cards etc. The binary format consists of an EPC header (typically the first 8 bits), which indicates the EPC scheme, a fast filter value (which can be used for distinguishing between different packaging levels or different kinds of object), as well as various other structural components or data fields within an EPC.

For EPC schemes defined before TDS 2.0, those fields typically indicate the company responsible for the object, the object class and a unique serial number. However, this approach required knowledge of the length of the GS1 Company Prefix component, as well as some rather complex rearrangement of the GS1 identifiers into a more structured format used in those EPC schemes, originally to enable lookup in the Object Name Service, which is no longer supported by GS1 on a global basis; lookup of identifiers is now primarily supported by resolver infrastructure for GS1 Digital Link URIs. The older EPC schemes based on GS1 identifiers use a partition table to handle variations in the length of the GS1 Company Prefix component, which in turn can limit the capacity of other components (such as the Item Reference) within those older EPC schemes; in most of the older EPC schemes (with the exception of GIAI and CPI), the GS1 Company Prefix component and the component that follows it are required to always sum to a fixed total number of digits for that EPC scheme.

For the new EPC schemes introduced in TDS 2.0, the GS1 identifier is encoded intact, without any rearrangement into separate fields to indicate GS1 Company Prefix and object class. These new EPC schemes neither require knowledge of the length of the GS1 Company Prefix component nor indicate the GS1 Company Prefix as a distinct structural component. These new binary encodings therefore do not make use of a partition table based on the length of the GS1 Company Prefix. Instead, any GS1 identification key that is all-numeric is encoded intact using 4 bits per digit and without any rearrangement of digits or removal of the check digit. This 4-bit encoding can be considered as an unsigned packed binary coded decimal encoding and although it is slightly less efficient than integer encoding, it ensures consistently predictable bit positions for the digits of a known GS1 Company Prefix, to support filtering over the air interface. This is particularly important for GS1 identifiers such as GTIN, ITIP and SSCC that use an indicator digit or extension digit before the GS1 Company Prefix; integer encoding of the values of GTIN, ITIP and SSCC would not result in predictable bit positions nor the possibility of using bitmask filters if the initial indicator digit or extension digit is unpredictable in the collection of tags being interrogated. For example, in the new EPC schemes, a GTIN is always treated as 14 digits and is encoded as 56 contiguous bits.

These new EPC schemes introduced in TDS 2.0 support variable-length encoding and multiple encoding options for each GS1 Application Identifier that can have an alphanumeric value, so the total number of bits for most of the new EPC schemes introduced in TDS 2.0 is also variable. For GS1 identification keys such as GIAI and CPI that begin with an initial numeric sequence followed by an alphanumeric sequence, the initial numeric sequence is encoded using 4 bits per digit, then a separator precedes the encoding of the alphanumeric sequence, itself beginning with an encoding indicator and length indicator, to indicate the encoding method used and the number of characters that follow, using that encoding method. This is intended to simplify the binary format and encoding/decoding rules, while maintaining efficient use of bits and still supporting selection of the primary GS1 identifier or the GS1 Company Prefix (if known) via the air interface. The new EPC schemes also introduced the option to encode additional AIDC data after the end of the EPC within the binary encoding of the EPC/UII memory bank.

TDS also defines URI formats for all EPC schemes – URN formats for the older EPC schemes defined before TDS v2.0 and GS1 Digital Link URI formats for each EPC scheme (old and new) that is based on GS1 identifiers. GS1 Digital Link URI formats are not defined for Tier 3 identifiers such as USDOD-96, ADI-var or GID-96.

For older EPC schemes introduced before TDS v2.0, the tag-encoding URN 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 the filter value (usually 3 bits). The tag-encoding URN is intended for low-level applications which need to write EPCs to tags or physically sort items based on packaging level.

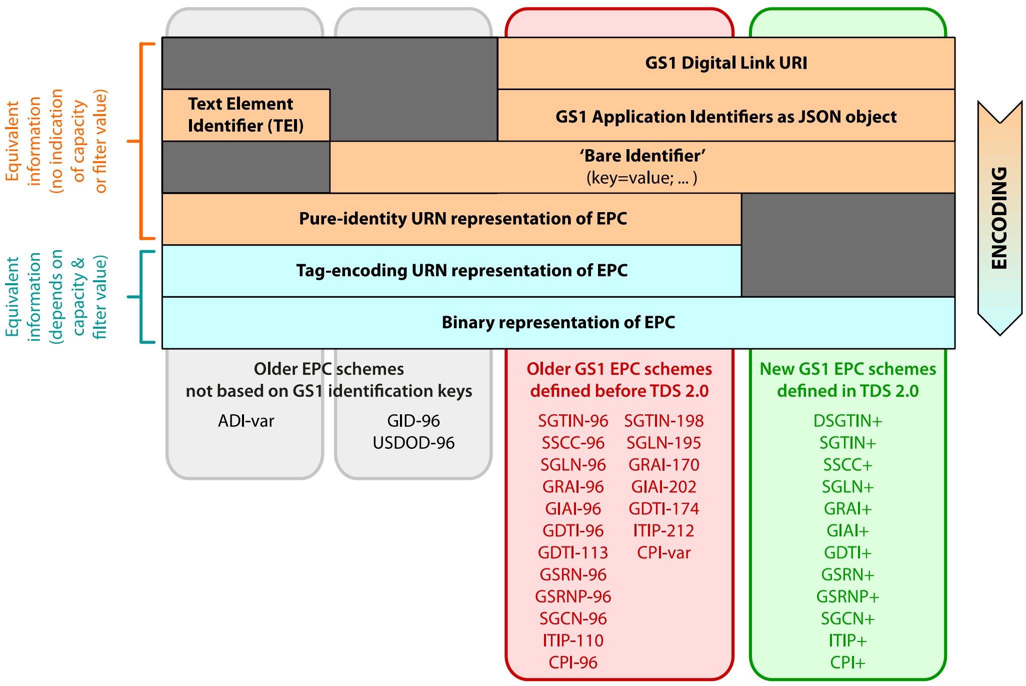
For older EPC schemes introduced before TDS v2.0, the pure-identity URN format isolates the application software from needing to know details about the bit-length of the tags or any fast filtering values, so that tags of different bit-lengths which code for the same unique object will result in the same pure-identity URN, even though their tag-encoding URNs and binary formats will be different. This means that if a manufacturer switches from using SGTIN-96 to SGTIN-198 for tagging a particular product instance, the pure-identity URN format of that SGTIN EPC will remain the same, even though the corresponding tag-encoding URN and binary format will be quite different.

For newer EPC schemes introduced in TDS v2.0, TDS does not define a tag-encoding URN format or pure-identity URN format – it only defines a binary format and the correspondence with element string or GS1 Digital Link URIs.

TDS normatively defines how to translate between these different formats of an EPC identifier (e.g. between binary format, URN formats, element strings, GS1 Digital Link URIs or other formats). Section E.3 of Appendix E of Tag Data Standard v2.0 provides examples of the pure-identity URN, tag-encoding URN and binary encoding for all EPC schemes introduced before TDS 2.0, together with examples of binary encoding and equivalent element strings or GS1 Digital Link URIs for the new EPC schemes introduced in TDS 2.0.

Figure 1-3 is a refinement of Figure 1-1 that shows which EPC formats are supported for each EPC scheme in TDS 2.0 and TDT 2.0. Element string and GS1 Digital Link URI are only supported for EPC schemes based on GS1 identifiers. Tag-encoding URN and Pure-identity URN are not defined for the new EPC schemes introduced in TDS 2.0. For EPC schemes that are not based on GS1 identifiers, instead of an element string or GS1 Digital Link URI format, TDT definition files provide a Text Element Identifier (TEI) format for ADI-var and a 'bare identifier' format for GID-96 and USDOD-96, also available for all EPC schemes.

Figure 1-3 EPC schemes and their various formats



Before the ratification of EPCIS / CBV 2.0 and TDS 2.0, the canonical format of an EPC was the pure-identity URN format, which was 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 or data carrier; the pure-identity URN can be just a pure identifier. However, pure-identity URNs have not been defined for the new EPC schemes introduced in TDS 2.0; for these new EPC schemes, TDS 2.0 defines a binary format as well as equivalent element strings and GS1 Digital Link URIs and the encoding/decoding rules to translate between these. Unlike pure-identity

URNs, GS1 Digital Link URIs can function like URLs and directly link or redirect to various kinds of information resources and services on the Web, via a simple Web request.

Now that TDS 2.0 and EPCIS / CBV 2.0 have been ratified, for all EPC schemes (old and new) that are based on GS1 identifiers, a constrained subset of GS1 Digital Link URIs may be used as an acceptable alternative to pure-identity URNs within EPCIS event data. If the data carrier content does not express a specific Web URI stem, domain name or hostname, then it is most advisable to use the URI stem for canonical GS1 Digital Link URIs, namely https://id.gs1.org/ . This approach promotes consistency when constructing a GS1 Digital Link URI from other formats that expressed no specific domain name, hostname or Web URI stem.

1.3 What is GS1 EPC Tag Data Translation?

The GS1 EPC Tag Data Standard [TDS] normatively defines EPC formats and encoding/decoding rules as several pages of human-readable instructions, diagrams, tables and worked examples.

This standard, the GS1 EPC Tag Data Translation Standard [TDT], complements TDS by providing such details in a machine-readable format, as a set of TDT definition files (one per EPC scheme) and a number of associated tables that are used in conjunction with these. TDT definition files may also make use of external tables, such as the table to lookup the length of a GS1 Company Prefix based on its initial digits (see https://www.gs1.org/standards/bc-epc-interop).

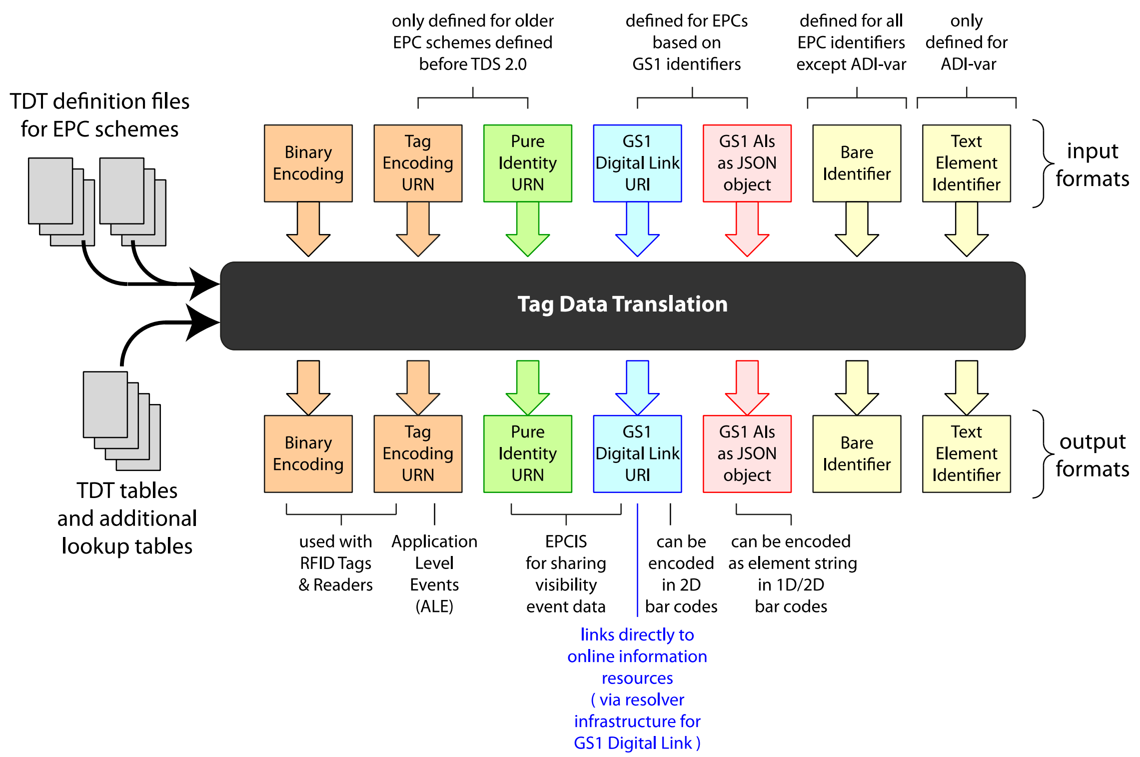
The three objectives in the original charter of the Tag Data Translation working group were:

- 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 [GS1Arch] to automatically translate between the binary and tag-encoding URN and pure-identity URN 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 [GS1Arch]. Open source implementations of software that demonstrate these capabilities may 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 EPC 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.

The aims of the original three objectives remain valid in TDT 2.0. However, the new EPC schemes introduced in TDS 2.0 do not define tag-encoding URN or pure-identity URN formats, although they do support translation to GS1 Digital Link URIs as well as the encoding of additional AIDC data after the end of the EPC in the EPC/UII memory bank, as explained in section 4 of this standard. The TDT definition files for the new EPC schemes are simpler but do rely on additional Tables F, K, E and B to support the flexible variable-length, variable-encoding nature of the new EPC schemes and the option of appending additional data after the EPC, based on GS1 Application Identifiers and their values.

A TDT implementation can translate one format of EPC into another format, within a particular EPC scheme. For example, it could translate from the binary format for a GTIN on a 96-bit tag to a pure-identity URN format of the same identifier, although it could not translate an SSCC into an SGTIN or vice versa. The TDT concept is illustrated in the figure below.

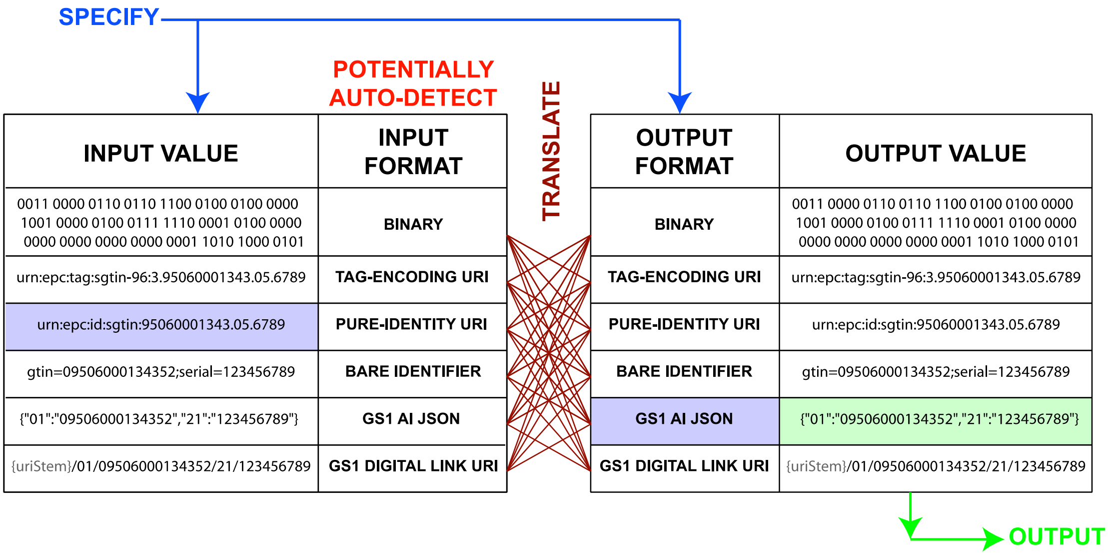
Figure 1-4 Translation between different formats using TDT definition files and tables



TDT aims to support the automatic detection of an EPC scheme and format (whether binary, tag-encoding URN, pure-identity URN) or an instance-level GS1 identifier expressed as an element string or GS1 Digital Link URI. However, when the input value is expressed as a GS1 element string, GS1 Digital Link URI, pure-identity URN or in 'bare identifier' notation, there may be multiple EPC schemes that match and it is necessary to make a choice about which EPC scheme to use. The choice of EPC scheme may depend on factors such as constraints on available memory in low-cost tags or a desire to encode additional AIDC data beyond the EPC binary string in the EPC/UII memory bank, a feature which is only supported in the new EPC schemes introduced in TDS 2.0.

TDT also aims to support validation of the input value and translation to an output value for a specified output format, as shown in the figure below, which provides examples for each format.

Figure 1-5 Tag Data Translation process with examples of different formats.



An implementation of Tag Data Translation may take an input value in one particular format (binary / tag-encoding URN / pure-identity URN, element string, GS1 Digital Link URI, bare identifier or text element identifier (ADI-var only)) and a specified output format, then return the result of translating the input value into the specified output format.

Tag Data Translation capabilities may be implemented at any level of [GS1Arch], from readers, through filtering middleware, as well as by applications, event repositories and networked databases that implement the EPCIS interface, as well as for translation to/from element strings or GS1 Digital Link URIs.

TDT definition files and tables can be used for validating EPC formats as well as for translating between the different formats in a consistent way. They may be helpful wherever there is a need to translate between these different EPC formats and their equivalent representations. This TDT standard describes how to interpret the machine-readable TDT definition files and associated tables. It contains details of their structure and elements and provides guidance on how they might be used in automatic translation or validation software, whether standalone or embedded in other systems.

By providing a machine-readable framework for validation and translation of EPC identifiers, TDT is designed to help to future-proof [GS1Arch] and in particular to reduce the pain or disruption if further EPC identifier schemes are introduced in the future, to support additional industry sectors and new applications and use cases.

Translation software may keep itself up to date by periodically checking for TDT definition files for each EPC scheme and downloading any new files. After these TDT definition files and auxiliary tables have been downloaded and stored locally, they can support offline translations or validations without the need for a reliable or continuous Internet connection. TDT 2.0 also introduces a manifest file that provides a list of all TDT definition files and tables that are considered current.

With TDT 2.0, the TDT definition files and associated tables are now all made available in XML and JSON format. Note that this does not impose a requirement for all levels of [GS1Arch] to implement XML or JSON 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. For example, a solution provider, such as the manufacturer of an RFID reader or RFID label printer, may periodically check for the latest TDT definition files and tables, then use data binding software to compile these into hierarchical software data objects, which could be saved more compactly as serialised objects accessible from the particular programming language in which their reader software/firmware is written. The solution provider could make these serialised objects available for download to owners of their products – or bundle them with firmware updates, thus eliminating the need for either embedding or real-time parsing of the original TDT definition files and tables in XML or JSON format within their solutions.

Individual TDT definition files are provided for each EPC scheme (i.e. separate files for SGTIN-96, SGTIN-198, SSCC-96, GID-96, SGTIN+, DSGTIN+ etc.) for older EPC schemes and for the new EPC schemes introduced in TDS 2.0, together with associated tables. The corresponding XML Schema Definition (XSD) files and JSON Schema files are also provided for validation purposes. These artefacts are available at https://ref.gs1.org/standards/tdt/artefacts

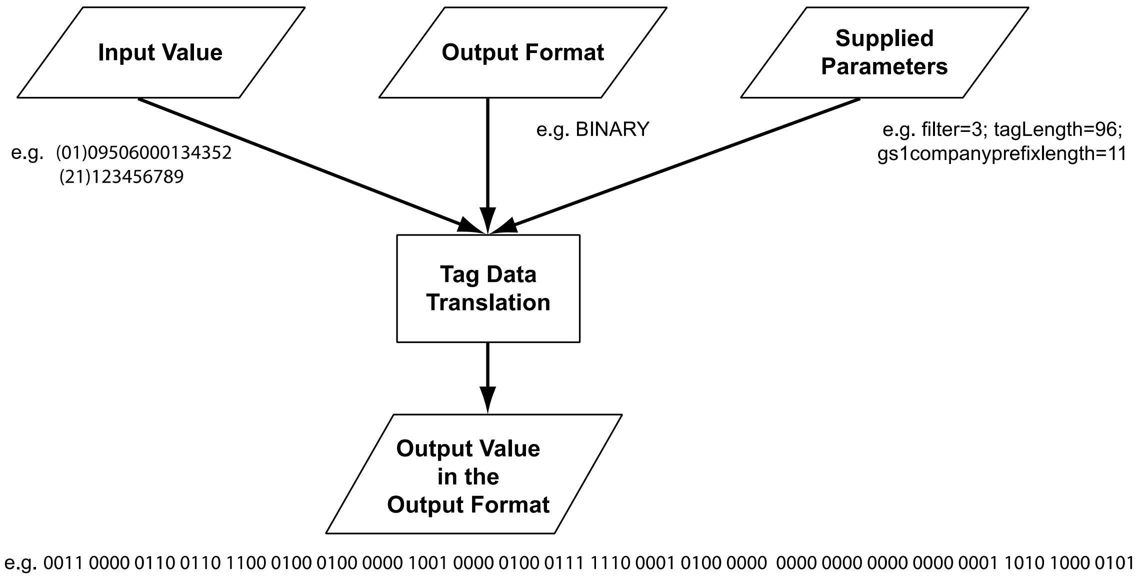
Version control is achieved within each TDT definition file via version numbers and timestamps of updates. A manifest file (in JSON and XML) is also provided, listing all current TDT definition files and tables and the date of last update for each of these. If any corrections or modifications are made to the current set of TDT definition files and tables, the manifest files SHALL also be updated accordingly and indicate the current set of files and tables. The purpose of the manifest file is to make it easier for translation software to check whether it has a complete set of files and to identify from the manifest file when the other files and tables have been added, updated or deprecated.

Because TDS 2.0 introduced new EPC schemes with simpler binary formats and encoding/decoding rules, as well as support for fields that are variable-length or variable-encoding, the TDT definition files for the new EPC schemes make use of Tables F, K, E and B to encode/decode those GS1 Application Identifiers correctly to/from the binary encoding, as well as to support encoding additional AIDC data after the binary encoding of the EPC. The TDT definition files for the new EPC schemes introduce a new field '`encodedAI`' that is used in conjunction with these tables. Section 3.17 of TDT 2.0 explains this in further detail.

2 Translation between various formats

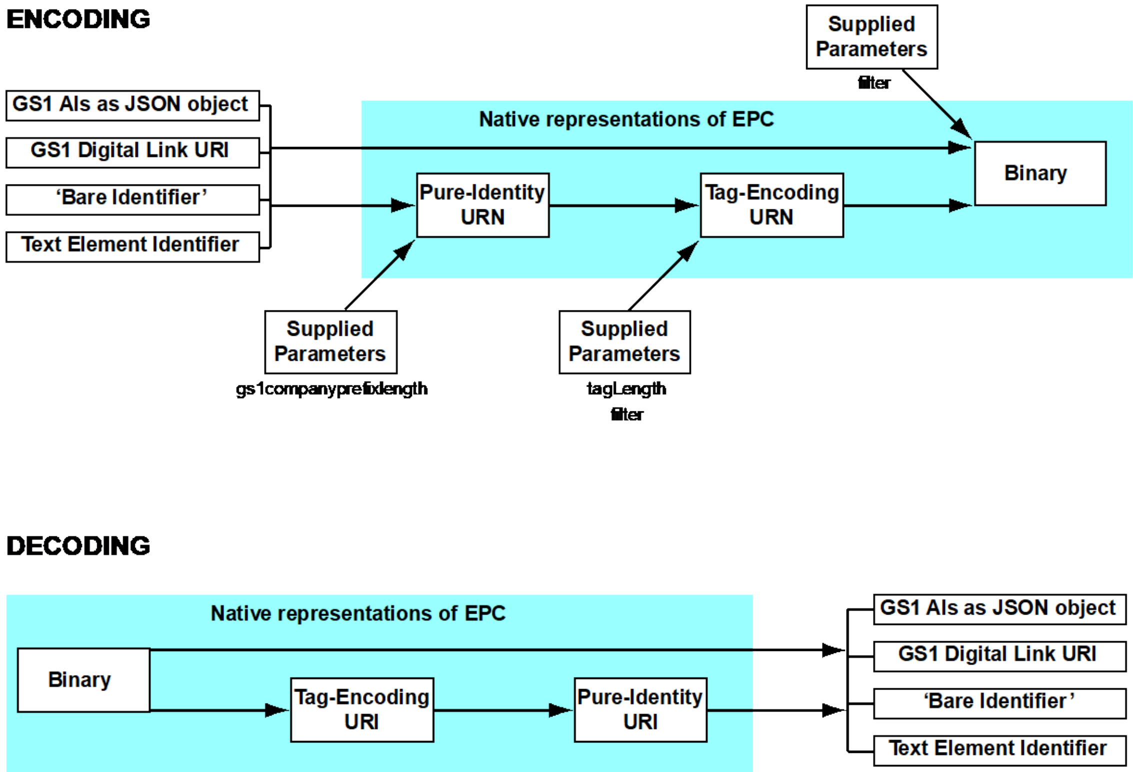
The figure below illustrates the provision of additional supplied parameters to supplement the details that can be extracted from the input value.

Figure 2-1 Flowchart showing input and output parameters to a Tag Data Translation process.



TDT refers to any translation of the format in the direction of the binary format as 'encoding', whereas any translation away from the binary format is 'decoding'. This is illustrated in the figure below.

Figure 2-2 Encoding and Decoding between different formats of an EPC.
Note that when encoding, additional parameters may need to be specified.



In the figure above, there are actually two distinct groups of supplied parameters – those such as `gs1companyprefixlength` which may be required for use in older EPC schemes if the input value is an element string or GS1 Digital Link URI – and others, such as `filter` and `dataToggle`, which are only required to format the output for specific formats, such as binary or tag-encoding URN; `dataToggle` is only available for use with the new EPC schemes introduced in TDS 2.0. Note that `tagLength` is not used for formatting the output value but may be used for selecting between older EPC schemes in situations where an input value such as an element string, bare identifier format or GS1 Digital Link URI could be encoded using more than one alternative EPC scheme; the value specified for `tagLength` indicates which EPC scheme is preferred when multiple schemes are possible, since the value of tagLength that is specified should match the scheme that specifies the same value of `tagLength` as a property of the `scheme` class (where specified). For example, in situations where the input is an element string or GS1 Digital Link URI that expresses values for GS1 Application Identifiers (01) = GTIN and (21) = Serial Number, it would be possible to encode the binary encoding of the EPC using either SGTIN-96, SGTIN-198 or SGTIN+. If `tagLength` is specified as "96" within the requiredFormattingParameters, then the SGTIN-96 scheme should be used in preference to the SGTIN-198 scheme.

In order to enable TDT implementations to check that all the required information has been supplied to perform a translation, the `level` component of the TDT definition 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 value in that output format level. Further details on these attributes appear in section 3, which describes the TDT definition files and their structure in further detail. For the binary or tag-encoding URN levels of many older EPC schemes introduced before TDS 2.0, `tagLength` is a required formatting parameter. This means that there can be situations where more than one TDT definition file has a pattern matching the input (e.g. if translating an SGTIN with an all-numeric serial number from pure-identity URN format to any format except binary or tag-encoding URN). In such situations, it should not matter which of the matching definition files is selected.

The newer EPC schemes introduced in TDS 2.0 support encoding of additional AIDC data after the binary encoding of the EPC. For such schemes, `dataToggle` is included within `requiredFormattingParameters`. Its value is set to 0 if no additional AIDC data is encoded or to 1 is additional AIDC data is encoded.

When encoding older EPC schemes based on GS1 identifiers, the length of the GS1 Company Prefix component can be specified via `gs1companyprefixlength`, which should be supplied when translating from element strings or GS1 Digital Link URIs to binary, tag-encoding URN or pure-identity URN formats for such older EPC schemes. As already mentioned in section 1.3, the GCP length lookup table at https://www.gs1.org/standards/bc-epc-interop may be useful, although it has incomplete global coverage.

The `filter` parameter can specify the filter value to use. For the appropriate choice of filter value to use with a particular identifier scheme, please refer to the filter tables defined in TDS.

The `tagLength` parameter is used to help an implementation of Tag Data Translation to select the appropriate TDT definition file among older EPC schemes that correspond to the same identifier but which differ in length, e.g. to choose between GRAI-96, GRAI-170 depending on whether the value of `tagLength` is set to 96 or 170. For the value of the `tagLength` parameter, it is necessary to consider the available size (in bits) for the EPC identifier memory in the RFID tag (e.g. 96 bits or higher) - and whether this is sufficient. [Non-normative example: for example, the GRAI-170 EPC scheme supports alphanumeric serial codes that 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 EPC scheme and the input format of the input value. This is particularly important when multiple tags are being read – when potentially several different EPC schemes could all be used together and read simultaneously.

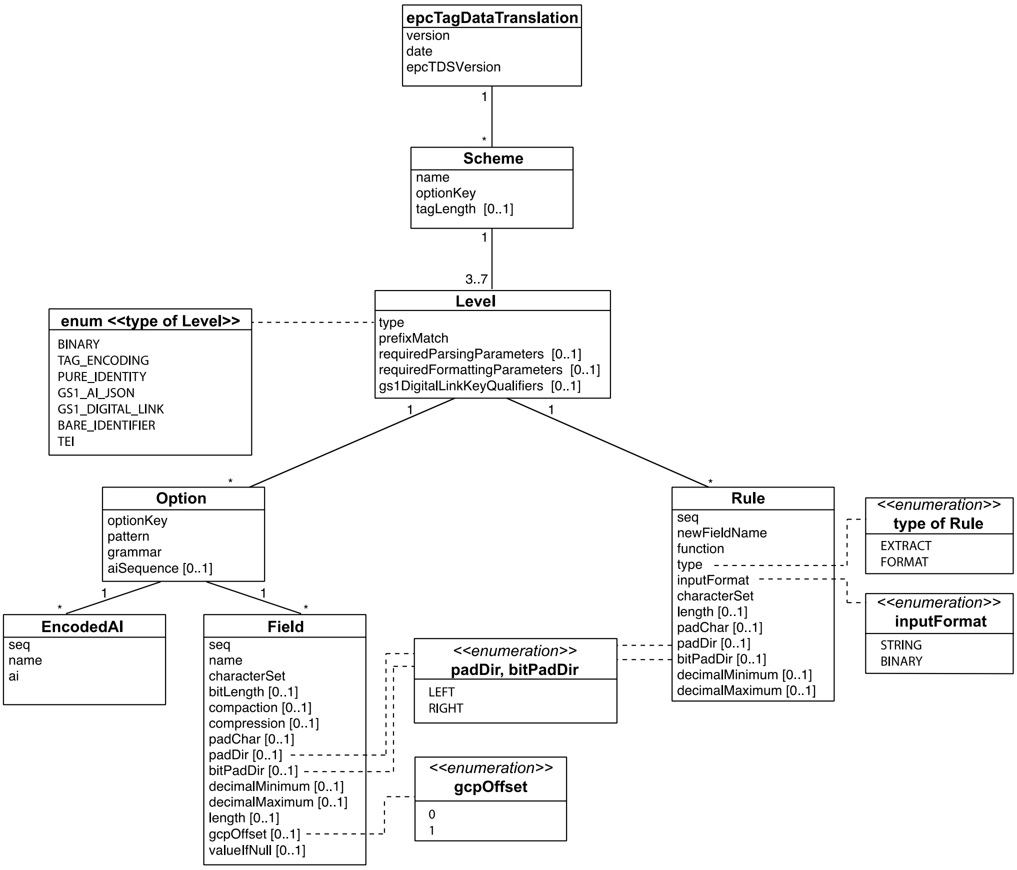
For example, a shipment arriving on a pallet may consist of a number of cases tagged with SGTIN identifiers and a returnable pallet identified by a GRAI identifier but also carrying an SSCC identifier to identify the shipment as a whole. If a portal reader at a dock door simply returns a number of binary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

3 Structure of TDT definition files

A TDT definition file is defined in TDS for each EPC scheme for which a binary format is defined. Machine-readable TDT definition files are normative artefacts of this standard and are provided in XML and JSON format.

Each TDT definition file is a hierarchical data structure with `epcTagDataTranslation` as its root element or main property and typically one `scheme` nested within that. The UML class diagram below defines the hierarchical structure of a TDT definition file.

Figure 3-1 UML class diagram for TDT definition files



Within each `scheme`, a separate `level` object is defined for each format of an EPC. Each `level` has a `type` property that is a value from a list of enumerated values that indicates correspondence to the binary format, an element string, GS1 Digital Link URI, tag-encoding URN, pure-identity URN or other format.

Within each `level` that is GS1_DIGITAL_LINK, the parameter `gs1DigitalLinkKeyQualifiers` (introduced in TDT 2.2) indicates the sequence of GS1 Application Identifiers that may appear within the URI path information after the primary identification key. For example, for schemes SGTIN-96, SGTIN-198, SGTIN+ and DSGTIN+, `gs1DigitalLinkKeyQualifiers` has the value `["22","10","21"]`, indicating the sequence in which these GS1 Application Identifiers (if present) should appear after the primary identification key in the URI path information of a GS1 Digital Link URI, following a post-processing step or before a pre-processing step. Section 3.4.1 provides further details about pre-processing of input and post-processing of output.

Within each `level` are one or more `option` objects. For older EPC schemes based on GS1 identifiers and defined before TDS 2.0, each `option` within a `level` corresponds to a row of the corresponding partition table for that EPC scheme, so each `level` typically contained seven `option` objects, corresponding to GS1 Company Prefix lengths in the range 6-12 digits.

For older EPC schemes based on GS1 identifiers, the appropriate `option` element is selected either by matching a hard-coded partition value from the input data (where this is supplied in binary format or URN format) – 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 definition files to specify the length and minimum and maximum values for each numeric field, which will often vary, depending on which `option` was selected – i.e. depending on the length of the GS1 Company Prefix used.

The TDT definition file for the ADI-var EPC scheme uses `option` elements differently, to support the permitted alternative variations within that EPC regarding how the unique identifier is constructed.

The TDT definition file for the new DSGTIN+ EPC scheme uses `option` elements in a further different way, to support different meanings of the prioritised date field (e.g. to distinguish between best before date, use by date, production date etc.)

Within each `option` element, the format of the EPC is expressed as both a regular expression `pattern` (for matching the input value), and as an Augmented Backus-Naur Form (ABNF) `grammar` for formatting the output value.

For older EPC schemes based on GS1 identifiers, the regular expression patterns and ABNF grammar are therefore subtly different for each `option` within a particular `level` – usually in the literal values of the bits that express the partition value and in the lengths of digits or bits for each of the subsequent `field` values (where delimiters such as a period '.' separate these fields within URN formats) – or in the case of the element strings, GS1 Digital Link URIs and binary format, the way in which groups of digits or bits are grouped within the regular expression pattern. This approach makes it easier to automatically detect 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. They appear within ADI-var and CPI-var schemes to indicate that a specific sub-pattern must not follow. For example, `pattern` values for CPI-var include sub-patterns such as ((?:(?!000000)[01]{6})+), which matches groups of 6 bits provided that not all six bits are set to zero (000000) because that set of bits acts as a delimiter within the binary encoding for CPI-var.

Within each `option`, the various fields matched using the regular expression capture groups are specified, together with any constraints that may apply to them (e.g. maximum and minimum values or constraints on length and character set), as well as information about how they should be properly formatted in both binary level and other levels (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).

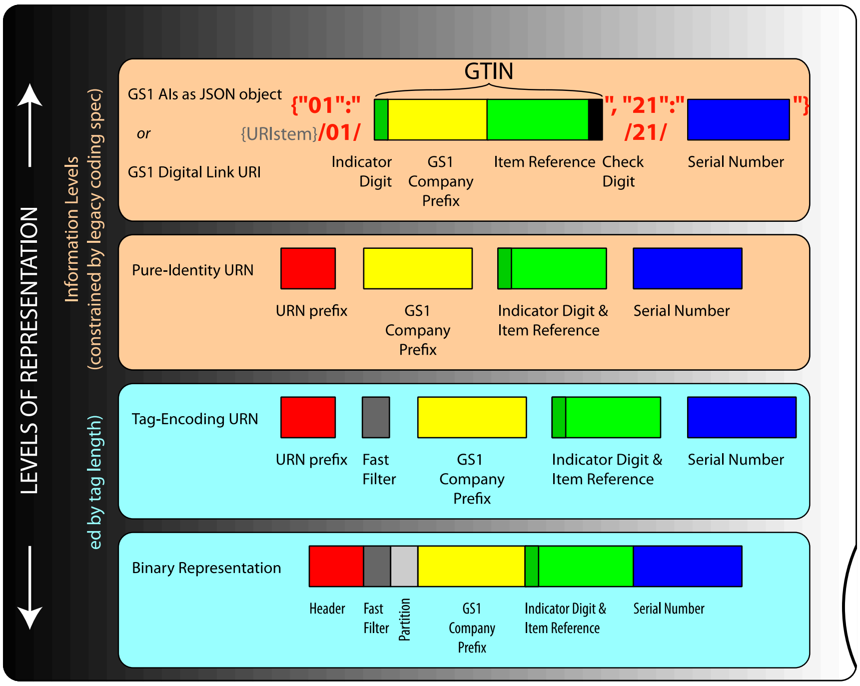
Within each `option`, the parameter `aiSequence` (introduced in TDT 2.2) indicates the sequence of GS1 Application Identifiers that are encoded within the EPC identifier when the `level` is either GS1_AI_JSON or GS1_DIGITAL_LINK.

Each `level` can also include zero ore more `rule` objects, which are explained in further detail later. These are used for computing additional field values derived from `field` values that are that have been extracted from the input value or are already known or previously computed using a preceding `rule`.

Within each `option`, one or more `field` objects are defined, to provide details about the structure and format of each structural component within an EPC format.

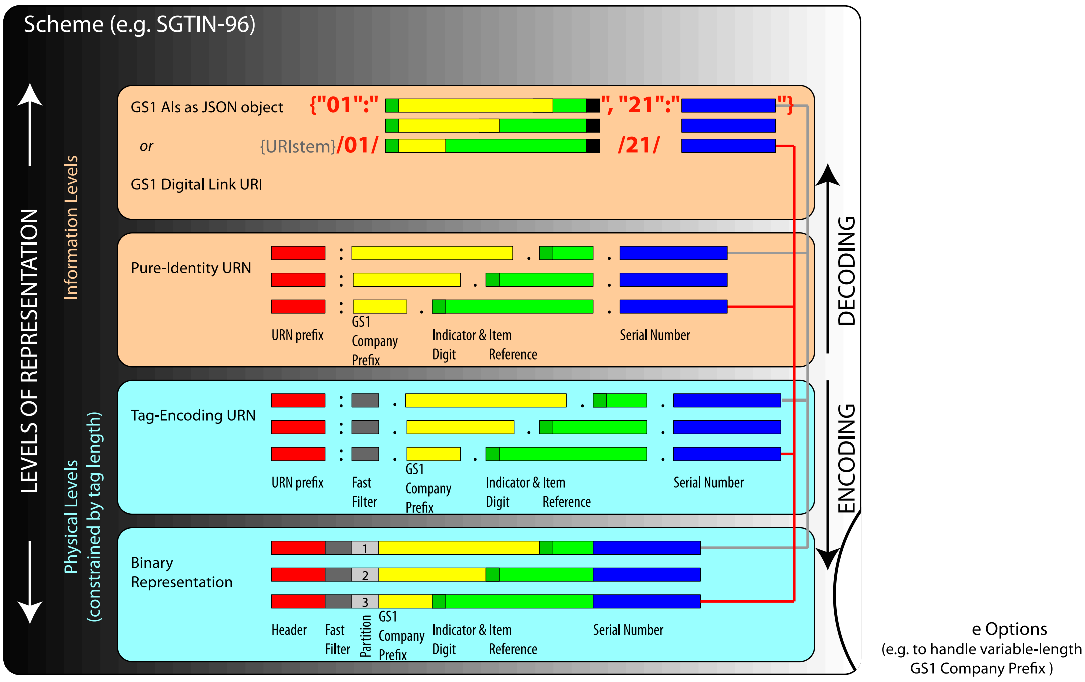
The figures below illustrate how this hierarchical structure of TDT definition files applies to the EPC schemes SGTIN-96 and SGTIN+, one TDT definition file per EPC `scheme`, each `scheme` containing one or more `level`, each `level` containing one or more `option` and, where appropriate, also containing one or more `rule`, each `option` containing one or more `field` structures.

592 **Figure 3-2** SGTIN-96 levels of representation



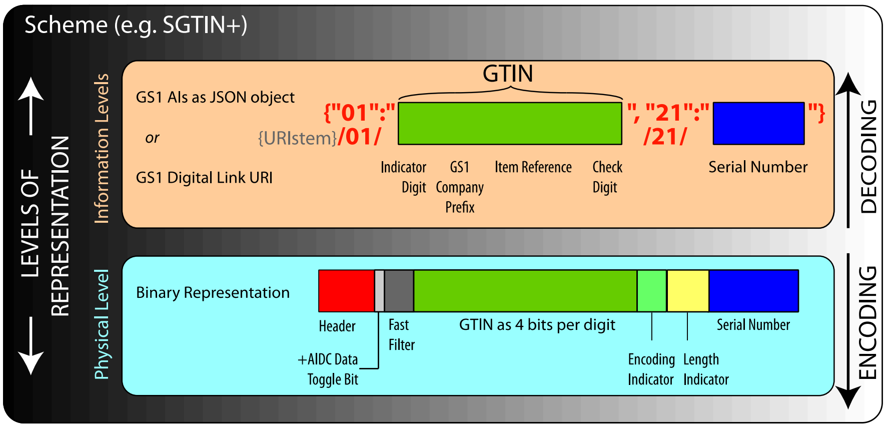
593

594 **Figure 3-3** SGTIN-96 levels with multiple encoding options



595

Figure 3-4 SGTIN+ levels of representation



3.1 Patterns (Regular Expressions)

Within each `option`, a regular expression `pattern` may be used to test for a match against an input value and extract groups of characters, digits or bits from the input value, so that their values may later be used for constructing the output value in the desired output format, after performing any additional processing that is required, such as translation between binary and base 10 (decimal), padding etc. The TDT standard refers to each of these variable parts as a `field`. A `field` is used to represent structural components within an EPC, such as the Serial Number, Filter value etc. For older EPC schemes defined before TDS 2.0, other examples of fields include the GS1 Company Prefix (which is typically related to the licensee of the GS1 identification key) and the Item Reference (or related fields such as Asset Reference, Location Reference etc., depending on the EPC scheme). For new EPC schemes introduced in TDS 2.0 and within the `level` elements that represent the element string and GS1 Digital Link URI formats for all EPC schemes based on GS1 identifiers, an intact GS1 identifier such as a GTIN or SSCC can also be a `field`. Further details about patterns are provided in section 3.5. For the binary `level` within the TDT definition files for new EPC schemes introduced in TDS 2.0, the regular expression `pattern` is not expected to match the whole of the binary encoding of the EPC identifier; typically it only matches the header, data toggle and filter value (and in the case of the DSGTIN+ scheme, also matches the prioritised data type indicator and prioritised date field); beyond these fields which are matched using the regular expression `pattern` in new EPC schemes, the remaining of the binary encoding of the EPC is handled using the information provided by `encodedAI` (explained in section 3.17) and if `dataToggle` matches a value of '1', then using section 4 to decode any additional AIDC data that was encoded after the binary encoding of the EPC in such new schemes introduced in TDS 2.0.

The values for `pattern` within TDT definition files within the binary level make no use of the 'match at end' anchor indicated by the $ character, since additional AIDC data may be encoded after the EPC binary encoding for new EPC schemes introduced in TDS 2.0 or trailing pad bits of '0' may be present up to the next 16-bit word boundary in all EPC schemes. Where additional AIDC data is encoded, this must immediately follow the end of the EPC binary string and there should be no intervening pad bits up to a 16-bit word boundary.

3.2 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 URI prefixes, strings and fixed binary headers with the variable components, i.e. the values of the various fields. For the `grammar` attributes of the TDT definition files, in accordance with the ABNF grammar conventions, fixed literal strings SHALL be single-quoted, whereas unquoted strings act as placeholders and SHALL indicate that the value of the field named by the unquoted string SHOULD BE inserted in place of the unquoted string. Further details about grammar are provided in section 3.5.

Square brackets denote that a sequence within the grammar that is optional or conditional. Square bracket notation is used within the TDT definition files for SGLN-96, SGLN-195 and SGLN+ in order to indicate that the grammar components corresponding to the GLN extension (254) and its value are conditional within the output formats for BARE_IDENTIFIER, ELEMENT_STRING or GS1_DIGITAL_LINK; if the value equals the value specified by the `valueIfNull` attribute of `field`, then the sequence within square brackets should not be included within the output string when the output is one of these output formats. Conversely, if the input format is BARE_IDENTIFIER, ELEMENT_STRING or GS1_DIGITAL_LINK and if the input string does not included information about the GLN extension (254) and its value, that component is considered to be null and the value given by the `valueIfNull` attribute ("0") SHALL be used in place of a null value when encoding to an output format that is BINARY, TAG_ENCODING or PURE_IDENTITY.

For the binary `level` within the TDT definition files for new EPC schemes introduced in TDS 2.0, the `grammar` also includes a field named `encodedAI`. This indicates the point at which the remainder of the EPC binary string is formatted or encoded as specified in section 3.17.

3.3 Rules for obtaining additional fields

Not all fields that are required for formatting the output value are obtained directly from pattern-matching of the input format. Sometimes additional fields are required to be computed. For example, when translating a SGTIN-96 from binary to element strings, it will be possible to extract a GS1 Company Prefix, Indicator Digit and Item Reference and Serial Number from pattern-matching on the binary input – but the output format needs other fields such as Check Digit, Indicator Digit, which SHOULD be computed from the fields that were extracted from the input value. For this reason, the TDT definition files may also include sequences of `rule` structures. Each `rule` expresses how an additional `field` may be computed via functions operating on one or more `field`(s) whose value(s) is/are already known. Further details about rules are provided in section 3.15.

Furthermore, there are some fields that cannot even be computed 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 format, it may be necessary to specify independently which length of tag to use (e.g. 96 bit or 198 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 output format. As illustrated in Figure 2-2, 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.

3.4 Using the information in TDT definition files within a translation process

The primary normative artefacts of the GS1 Tag Data Translation standard is the collection of TDT definition files and tables, which enables encoding and decoding between various formats for each particular EPC scheme. This generic design requires open and highly flexible format of rules for translation software to encode/decode based on the input value. A TDT definition file is a machine-readable file (in XML or JSON) that expresses the encoding/decoding and validation rules for each of the EPC schemes defined in the GS1 Tag Data Standard that has a binary encoding.

This chapter provides a descriptive explanation of how to interpret the TDT definition files in the context of a translation process. Chapter 4 provides a formal explanation of the elements and attributes within the TDT definition files.

There are seven fundamental steps to a translation:

- Use of a `prefixMatch` value and a regular expression `pattern` to automatically detect the input format and EPC scheme of the supplied input value
- If the detected input level is GS1_AI_JSON or GS1_DIGITAL_LINK, pre-processing of the input may be required – see section 3.4.1.1
- Using the capture groups within the regular expression `pattern` to extract values of each `field` from the input value. Capture groups are typically indicated using round brackets.
- Further processing of each `field` extracted from the input value, in order to translate from the input format to the desired output format. This includes splitting or joining of strings, translation between binary strings and numeric/alphanumeric strings, addition or removal of padding.

- Using the `rule` definitions to calculate any additional `field` values required for parsing the input or formatting the output. Such `rule` definitions are also used to indicate when to use percent-encoding to encode or decode specific symbol characters that need to be escaped within URN or URL / Web URI formats.
- Using the ABNF `grammar` to prepare the specified output format, substituting the actual value of each `field` where indicated in the `grammar`.
- If the output level is GS1_DIGITAL_LINK, additional post-processing may be necessary. This is described in section 3.4.1.2.

Note that the `prefixMatch` attribute in the TDT definition files is provided to allow TDT implementations to perform automatic detection of the input format more efficiently. For older EPC schemes introduced before TDS 2.0 and based on GS1 identifiers, multiple `option` elements are specified within a particular `level` element; each `option` will have a `pattern` attribute with a subtly different regular expression as its value. The `prefixMatch` attribute of the enclosing `level` element expresses a fragment of these patterns that is common to all of the nested `option` elements. If the value of the `prefixMatch` attribute fails to match the input value, a TDT implementation need not test each nested `option` for a pattern match, since they will not match if the `prefixMatch` does not already match the input value. Only for those levels where the `prefixMatch` attribute matches the input string value should the patterns of the nested `option` elements be considered for matching. Within the newer EPC schemes introduced in TDS 2.0, only the scheme DSGTIN+ makes use of multiple `option` elements, in order to distinguish between different meanings of the prioritised date value, e.g. one `option` element interprets the value as an expiration date, while other `option` elements interprets the value as a harvest date or production date.

Note that in the TDT definition files, the `prefixMatch` attribute SHALL be expressed as a substring to match the input value. The `prefixMatch` attribute SHOULD NOT be expressed in the TDT definition files as a regular expression value, since a simple string match should suffice. Software implementations MAY typically translate the `prefixMatch` attribute string value into a regular expression, if preferred, 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 '`\\.`'. However, for GS1 Digital Link URI format introduced in TDS 2.0 and TDT 2.0, `prefixMatch` cannot provide a highly specific match to the input value at the start of the input string because any domain name may be used and any arbitrary URI path information may also be present before the part of the URI path information that is characteristic of GS1 Digital Link URIs, such as the URI path information structure that begins /01/ for GS1 Digital Link URIs based on the GTIN identifier. Therefore, in TDT 2.0 `prefixMatch` is set to 'http' for each level that represents a GS1 Digital Link URI format and it is necessary to use the regular expression specified in each `pattern` in order to distinguish between the various EPC schemes and options when attempting auto-detection of the input format. Furthermore the regular expression `pattern` specified for GS1 Digital Link URIs is not expected to match at the start of the input string but instead matches the part that is specific to that EPC scheme, e.g. matching for /01/ and /21/ in all SGTIN EPC schemes including DSGTIN+. Accordingly, the regular expression `pattern` for GS1 Digital Link URIs does not have a leading caret (^) symbol (to require a match at the start of the string), whereas the `pattern` values for all other levels within in TDT 2.0 definition files do have such a caret as a 'match at start' anchor. The regular expression `pattern` for element string and GS1 Digital Link URI end with a word boundary anchor (\b) to effectively match to the end or to a non-word character such as the question mark character that precedes a URI query string.

3.4.1 Pre-processing of input and post-processing of output

The GS1 Tag Data Translation standard was originally developed to support translation between EPC binary strings, the EPC URN formats and the corresponding element strings of GS1 Application Identifiers. TDT v2.0 added support for GS1 Digital Link URI syntax, as well as providing machine-readable tables to support the encoding/decoding of additional AIDC data that may be encoded after the EPC binary string for the new EPC schemes introduced in version 2.0 of the GS1 Tag Data Standard.

A primary use of EPCs is as an open standard identifier with instance-granularity for use within EPCIS events. As a result, within GS1 Tag Data Translation, the pattern and grammar for the GS1_DIGITAL_LINK level corresponds to the constrained subset of GS1 Digital Link URIs that contain the bare minimum number of GS1 Application Identifiers needed to construct an instance-level identifier, such as GTIN (01) and Serial Number (21), even though GS1 Digital Link URI syntax supports some additional optional URI path elements and also supports expression of GS1 Application Identifiers in the URI query string to express various data attributes, such as expiration date or net weight of variable-measure trade items.

As a result of this, when a GS1 Digital Link URI is provided as the input value to an implementation of GS1 Tag Data Translation, an additional pre-processing step may be needed to transform it into the constrained format that is supported by the TDT definition files for each EPC scheme.

Conversely, when GS1 Digital Link URI is selected as the output format, a post-processing step may be needed to reinstate some specific GS1 Application Identifiers (e.g. consumer product variant (22) and batch/lot (10)) into the URI path information, since these would otherwise be excluded from the constrained GS1 Digital Link URI format prepared by using the grammar details provided by GS1 Tag Data Translation definition files.

3.4.1.1 Pre-processing of input

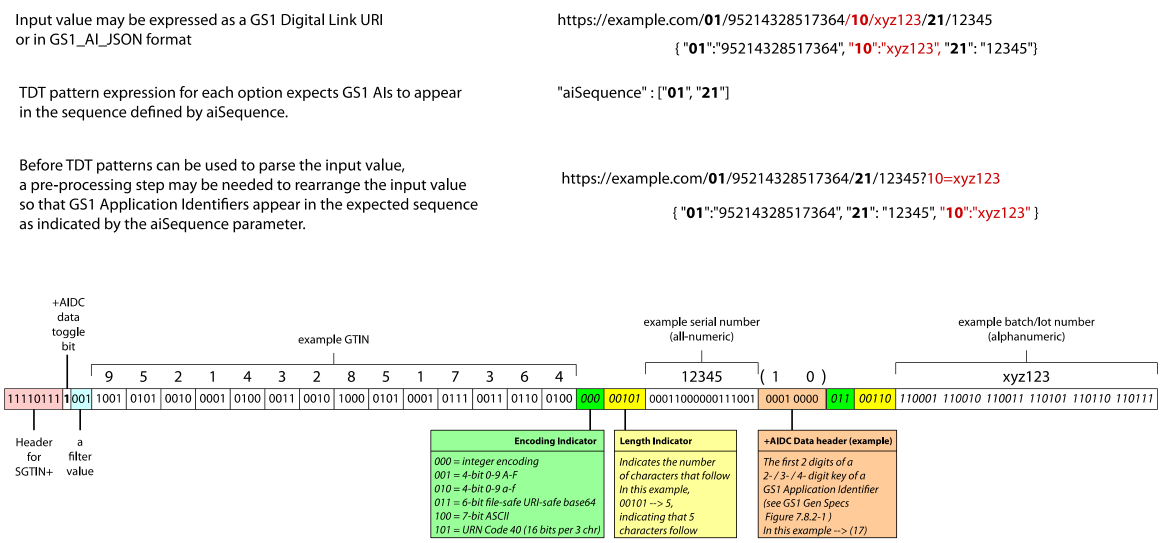
To assist with the pre-processing step, a new parameter, `aiSequence` appears within the `option` elements within the `level` element for GS1_DIGITAL_LINK and GS1_AI_JSON. This is an ordered list of the GS1 Application Identifiers handled by the pattern, corresponding to the GS1 Application Identifiers that will be encoded within the EPC binary string.

For SGTIN schemes, this corresponds to ["01","21"]. For DSGTIN, this corresponds to lists such as ["01","21","17"] etc., where the third element depends on which prioritised date GS1 AI is supported by that `option`.

If the input is provided using the GS1_AI_JSON notation, the regular expression patterns expect to match a JSON object in which the GS1 Application Identifiers appear strictly in the sequence specified by `aiSequence` otherwise the `pattern` provided within the TDT definition file cannot match the input. For any GS1 Application Identifiers not included within the `aiSequence` list, the ordering does not matter. For the new EPC schemes introduced in TDS 2.0, such additional GS1 Application Identifiers may be encoded after the EPC binary string, within the EPC/UII memory bank.

If the input is provided using GS1 Digital Link URI format and if the URI path information expresses any GS1 Application Identifiers that are not present within the `aiSequence` list, the pre-processing step must reformat the GS1 Digital Link URI input in order to remove those GS1 Application Identifiers and their values from the URI path information and express them via the URI query string instead, otherwise the `pattern` provided within the TDT definition file cannot match the input.

The figure below illustrates how a pre-processing step can make use of the details specified within the `aiSequence` parameter to rearrange the input value so that it can potentially match the `pattern` specified within the TDT definition file for that `option`.

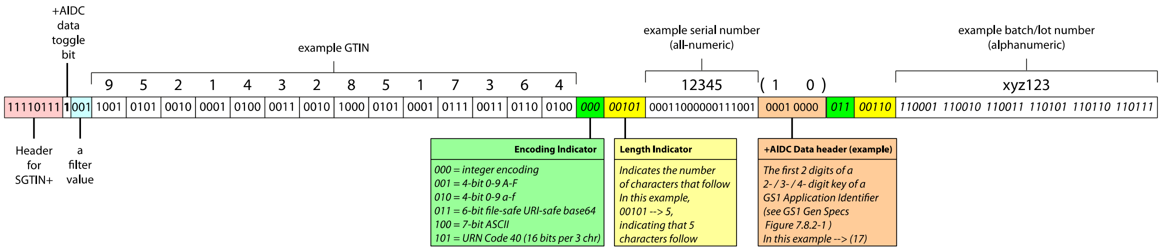


3.4.1.2 Post-processing of output

To assist with post-processing, a new parameter, `gs1DigitalLinkKeyQualifiers` appears within the `level` element for GS1_DIGITAL_LINK and provides an ordered list of GS1 Application Identifiers that may appear within the URI path information of a GS1 Digital Link URI **after** the primary identification key and its value. Note that the primary identification key (such as GTIN "01" for all SGTIN / DSGTIN schemes) is not included within the list of `gs1DigitalLinkKeyQualifiers` – it always precedes these within the URI path information.

If GS1_DIGITAL_LINK is selected as the output format, the set of decoded GS1 Application Identifiers and their values should be checked, in case any of them are listed within the ordered list specified by the `gs1DigitalLinkKeyQualifiers` parameter. For any such GS1 Application Identifiers, the post-processing step should reinstate those GS1 Application Identifiers and their values within the URI path information and in the specified sequence, instead of expressing those GS1 Application Identifiers and their values in the URI query string.

The figure below illustrates how a post-processing step can make use of the details specified within the `gs1DigitalLinkKeyQualifiers` parameter to rearrange the output value so that GS1 Application Identifiers that should appear within the URI path information of a syntactically valid GS1 Digital Link URI do actually appear within the URI path information (rather than the URI query string) and in the correct sequence, consistent with the formal grammar defined within the GS1 Digital Link URI Syntax standard.



An EPC binary string input may be decoded as a set of GS1 Application Identifiers and their values, e.g. { "**01**":"95214328517364", "**21**": "12345", "**10**":"xyz123" }

If the output format is selected to be GS1_DIGITAL_LINK, the TDT grammar will construct a constrained GS1 Digital Link URI such as:

https://id.gs1.org/**01**/95214328517364/**21**/12345?**10**=xyz123

in which any GS1 Application Identifiers not explicitly specified within the grammar parameter are considered to be expressed via the URI query string, as shown in red above.

However, it is possible that some of the decoded GS1 Application Identifiers, e.g. (10) Batch/Lot or (22) Consumer Product Variant should appear within the URI path information.

The parameter gs1DigitalLinkKeyQualifiers specifies an ordered list of GS1 Application Identifiers that can appear within the URI path information of a GS1 Digital Link URI after the primary key and its value.

For SGTIN+ and DSGTIN+, "gs1DigitalLinkKeyQualifiers" = ["**22**", "**10**", "**21**"]

A post-processing step can use the information specified by gs1DigitalLinkKeyQualifiers to check the decoded GS1 Application Identifiers to determine whether any of the GS1 Application Identifiers and their values that were decoded from the input value should actually appear in the URI path information instead.

In this example, AI (10) and its value should appear in the URI path information, before AI (21) and its value.

Here is the result, after the post-processing step: https://id.gs1.org/**01**/95214328517364/**10**/xyz123/**21**/12345

3.5 Definition of formats via Regular Expression Patterns and ABNF Grammar

The TDT standard uses regular expression patterns and Augmented Backus-Naur Form (ABNF) [ABNF] grammar expressions to express the structure of the EPC in various formats.

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 that indicate capturing groups in regular expressions.

The regular expression patterns provided in the TDT definition files SHALL be written according to the PERL-Compliant Regular Expressions [PCRE], 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 JavaScript, Perl, Java, C#, .NET all allow for negative lookahead. Note that the TDT definition files for ADI-var and CPI-var make use of the negative lookahead construct in the patterns at the BINARY level in order to make the patterns more restrictive and to avoid the situation where a valid binary string might match more than one option.

The ABNF grammar form allows the TDT definition files to express the output 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. The TDT definition files 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 `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 `field` elements with attributes for each of the fields enables 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 an all-numeric GS1 Company Prefix field.

3.6 Determination of the input format

A desirable feature of any Tag Data Translation software is the ability to automatically detect the format of the input string received, whether in binary, tag-encoding URN, pure-identity URN, element strings or GS1 Digital Link URIs, where required. Furthermore, the EPC scheme should also be detected. For older EPC schemes with a fixed bit count, the tag-length SHALL either be determined from the input value (i.e. given a binary string or tag-encoding URN), – or otherwise, where the input value does not indicate a particular tag-length (e.g. pure-identity URN, element strings or GS1 Digital Link URI format, 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 URN, an element string or GS1 identifier key expressed using Application Identifier (AI) format, 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 definition files specify a `prefixMatch` 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 `prefixMatch` to test for a different scheme or different format, without needing to try each `pattern` for all the `option` elements nested within each of these, since all of the nested regular expression patterns fall under the same value of `prefixMatch`.

3.7 Specification of the output format

The Tag Data Translation process only permits encoding or decoding between different formats 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 formats, such as binary, tag-encoding URN, pure-identity URN, element strings or GS1 Digital Link URIs, depending on which of these is supported by that scheme. Translation to/from Text Element Identifier strings is also possible for the Aerospace & Defence Identifier (ADI). Translation to/from a 'Bare Identifier' format is also supported for all current EPC schemes.

With this constraint, it should be possible for Tag Data Translation software to perform a translation if the input value and the output format level are specified.

3.8 Specifying supplied parameter values

Decoding from the binary level through the tag-encoding URN, pure-identity URN and finally to the element strings or GS1 Digital Link URIs only ever involves a potential loss of information. 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 URN, element string or GS1 Digital Link formats.

Encoding often requires additional information to be supplied independently of the input 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 values of parameters. These parameters are additional to those that are automatically extracted from parsing the input string using the matching groups of characters within the appropriate matching regular expression pattern.

Table 3-1 shows examples of how the input value should be formatted for serialized identifiers.

Table 3-1 – Example formats for supplying existing identifier formats as the input value.

EPC Scheme	Example format for input GS1 identifier keys, showing GS1 AIs in JSON format or 'bare identifier' format for EPC schemes where no GS1 element string format is defined.
SGTIN	`{"01":"00037000302414","21","10419703"}`
SSCC	`{"00":"000370003024147856"}`
SGLN	`{"414":"0003700030241","254":"1041970"}`
GRAI	`{"8003":"00037000302414274877906943"}`
GIAI	`{"8004":"00370003024149267890123"}`
GSRN	`{"8018":"061414123456789012"}`
GDTI	`{"253":"0073796100001"}`
GID	`generalmanager=5;objectclass=17;serial=23` [No corresponding GS1 element string format]
USDOD	`cageordodaac=AB123;serial=3789156` [No corresponding GS1 element string format]

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 format]

Note: TDT definition files support the following formats:

- 'TEI' for Text Element Identifier format of ADI-var only
- 'Bare identifier' for all EPC schemes
- 'Element string' and 'GS1 Digital Link URI' for all EPC schemes based on GS1 Tier 1 identifiers.
- 'Pure identity URN' and 'Tag encoding URN' for older EPC schemes introduced before TDS 2.0
- Binary format for all EPC schemes for which a binary format is defined in TDS. (*There are EPC schemes -- such as UPUI -- for which no binary encoding is currently defined in TDS, so TDT does not define a binary format or even provide a TDT definition file for such schemes.*)

Note that in Tag Data Translation implementations, the values extracted from the input format of the EPC SHALL always override the values extracted from the supplied parameters; i.e. the parameter string may specify '`filter=5`' – but if the input format of the EPC 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. Similarly, additional lookup mechanisms such as the tables at https://www.gs1.org/standards/bc-epc-interop can often be used to determine the length of a GS1 Company Prefix from its initial digits. In older EPC schemes where the value of `gs1companyprefixlength` needs to be known and can be determined from the input string, knowledge of the expected start position of the GS1 Company Prefix component (see details about `gcpOffset`) through the use of such lookup mechanisms, the length value obtained automatically by such a procedure SHALL override the corresponding value that may have been specified via the the supplied parameters in situations where there are conflicting values.

Nowadays, JavaScript Object Notation (JSON) is well supported as a portable and robust way of exchanging structured data such as lists and objects / associative arrays across many programming languages. However, JSON was still in its infancy when the GS1 Tag Data Translation standard was originally developed. 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 translated into an associative array in most modern programming languages, while remaining portable and independent of programming language. The equivalent JSON representation would enclose the associative array in curly brackets { } and use a comma instead of a semi-colon as the delimiter between multiple key : value pairs, using a colon rather than equals sign as the separator between each key and its corresponding value, i.e. an associative array of supplied parameters expressed in JSON as {key1 : value1, key2 : value2 } is expressed as a string formatted as "key1=value1;key2=value2".

3.9 Validation of values for fields and fields derived via rules

The `field` object and the `rule` object contain several properties (attributes) for validating and ensuring that the values for a particular `field` falls 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 definition files explicitly specify the format and constraints of each `field` in order to support future extensibility.

Within the TDT definition files for SGLN and within the level for BARE_IDENTIFIER, ELEMENT_STRING and GS1_DIGITAL_LINK, an additional attribute (`valueIfNull`) is present. If the input format is one of these levels and if the input string does not indicate a value for the GLN extension (254), the null value for `'serial'` or `'urlEscapedSerial'` SHALL be treated as if it were "0" when the output format is BINARY, TAG_ENCODING or PURE_IDENTITY.

If the input format is one of BINARY, TAG_ENCODING or PURE_IDENTITY and the input string expresses a value for the serial GLN extension (254) equal to the value of `valueIfNull` ("0"), then when translating to any of BARE_IDENTIFIER, ELEMENT_STRING or GS1_DIGITAL_LINK, the component that expresses the `valueIfNull` attribute SHALL NOT be included in the output string; this means that the component for GLN extension (254) and its value would be omitted.

3.10 Restricting and checking ranges for values of numeric fields in base 10

In some cases, the numeric range which can be expressed using the specified number of bits exceeds the maximum base 10 value permitted for that identifier in its formal specification.

For example, the serial number of an SSCC may be up to ten base 10 digits – permitting the base 10 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 TDS, the minimum and maximum limits of each numeric field in base 10 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` <= value of `field` <= `decimalMaximum`

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

Note: Many of the structural components within EPC schemes and TDT definition files correspond to 'big integers' that exceed the capacity of native integer representation in most programming languages. For this reason, translation software should consider the use of dedicated 'big integer' data types (where available) or additional software libraries/modules to support big integers correctly, in order to avoid unwanted rounding errors or loss of precision. It is for this reason that both `decimalMinimum` and `decimalMaxmimum` and other big integer values are expressed as numeric string values within the TDT definition files and tables, in order to avoid loss of precision or unwanted rounding errors when using native methods (such as `JSON.parse()` within JavaScript) for parsing JSON data, while such methods do not yet consistently provide adequate support for big integers across all programming languages.

3.11 Restricting and checking character ranges for values of fields

The `characterSet` property of the `field` object indicates the allowed range of characters which may be present in that field. The range is usually expressed using the same square-bracket notation as for character ranges within regular expressions, although for the URN formats and GS1 Digital Link URI formats, the pattern and characterSet now use non-capturing groups with explicit indication of percent-encoded sequences for symbol characters that must be 'escaped' in URN or URI format; this approach ensures that each valid symbol character is counted once even when it is percent-encoded as a 3-character sequence `%hh` where `h` is a placeholder for hexadecimal characters 0-9 and A-F. Further details about percent-encoding of symbol characters in URNs and Web URIs / URLs can be found in section 3.16 that explains the new `rule` functions `URNENCODE`, `URNDECODE`, `URLENCODE` and `URLDECODE`. The asterisk symbol (*) following the closing square bracket or end of the non-capturing group 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 ('$') to ensure that the characterSet matches the entire field. e.g. for [0-7]* in the TDT definition files, TDT implementations may use ^[0-7]*$ as the corresponding regular expression for matching if the character set was specified as [0-7]*.

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.*

(?:[A-Za-z0-9\"._-]|%21|%26|%27|%28|%29|%2A|%2B|%2C|%2F|%3A|%3B|%3C|%3D|%3E|%3F|%25)*

is an example of a non-capturing group that permits characters A-Z a-z 0-9 and all symbol characters within the 82-character GS1 invariant subset of ISO/IEC 646 when symbol characters are percent-encoded within a URL or GS1 Digital Link URI.

The `characterSet` attribute can be used to check that all of the characters fall within the permitted range. For example, the serial number for Component/Part Identifier (CPI) is required to be all-numeric, up to 12 digits, as defined for GS1 Application Identifier (8011). Accordingly, the characterSet for the field that corresponds to the CPI serial number is expressed as [0-9]*. If the input string specifies a serial number for CPI that contains any characters that are not wholly numeric, this should result in an error.

Many instance-granularity GS1 identifiers can be encoded using more than one EPC scheme – one only supporting numeric serial numbers (SGTIN-96), another for alphabetic serial numbers (SGTIN-198) as well as alternative new EPC schemes introduced in TDS 2.0, e.g. SGTIN+, DSGTIN+.

In EPC schemes introduced before TDS 2.0, the presence of the `compaction` attribute within a `field` or `rule` in the `BINARY level` 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 string, with leading pad characters if the `padChar` attribute is also present and the integer value has fewer digits than the `length` attribute specifies.

In the new EPC schemes introduced in TDS 2.0, the TDT definition files make no use of the `compaction` attribute; instead the `encodedAI` attribute indicates the sequence of GS1 Application Identifiers that are to be encoded next and translation software needs to make use of Table F to determine the available format for the value of each GS1 Application Identifier. Explicit 3-bit encoding indicators are used in the binary encoding of such new EPC schemes because they support variable encoding methods for alphanumeric character strings.

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 EPC 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 translated to a character string because the `compaction` attribute was present.

3.12 Padding of fields

For all older EPC schemes defined before TDS 2.0, TDT 2.0 makes no changes to the logic or rules for padding of fields that were already in place in TDT 1.6.

3.12.1 Changes since TDT v1.0

Certain fields within either the binary format, the URI formats and also the element string and GS1 Digital Link URI formats 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 [GS1GS]. 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.

Version 1.4 of TDT 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 TDT 1.4 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 field in levels other than the binary level.

When encoding to binary from any other level except for binary, the field itself may be padded (prior to any translation 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 translating 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` set to `"LEFT"` to indicate that the binary field should be prepended to the left with zero bits when encoding. In contrast, for the serial code of a GRAI-170 or GIAI-202 `bitPadDir` is set to `"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 level, there is sometimes a need to strip the leading/trailing bits from a particular direction prior to translation 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 translating to a character string.

After translation 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 output level or in the tag-encoding 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 in the output level or tag-encoding 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 formats 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 where attribute `type` is ”`BINARY`”.

3.12.2 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 EPC schemes MAY specify the space character (ASCII character 32 [20 hex]) or a different character to use.

The `padChar` attribute indicates the character to be used for padding in formats other than BINARY. 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 definition 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 levels except for binary may need to be padded with the padding character `padChar` within this format.

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 padding character `padChar` indicated in the output level or tag-encoding level in the direction `padDir` only immediately prior to translation to binary and that when decoding away from the binary level, such 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 levels except for binary should have `padChar=”0”` and `padDir=”LEFT”` because the leading zeros are significant and should appear in the URI formats, element strings, GS1 Digital Link URIs and 'bare identifier' format.

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 format.

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 other 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 other level, the TDT definition file should be considered invalid. Note that some fields that appear within the binary level do not appear in all other levels. For example, the filter value never appears in the pure-identity URN level. For this reason, in section 3.10.1, the flowchart advises checking of the tag-encoding URN format to see whether or not padChar and padDir are defined for each field corresponding to the fields defined within the binary level.

3.12.3 bitPadDir and bitLength

For `field` or `rule` elements contained within a `level` element where attribute `type` is `"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` is `'LEFT'`, whereas for the serial code field of GRAI-170, `bitPadDir` is `'RIGHT'`

3.12.4 Summary of padding rules

Figure 3-5 is a flowchart summary of the rules about whether or not to add or remove padding when encoding from a field in a level other than binary to the corresponding binary encoding.

Figure 3-6 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 an output level that is not binary (e.g. to be used in the URI formats, element strings, GS1 Digital Link URI format or 'bare identifier' format).

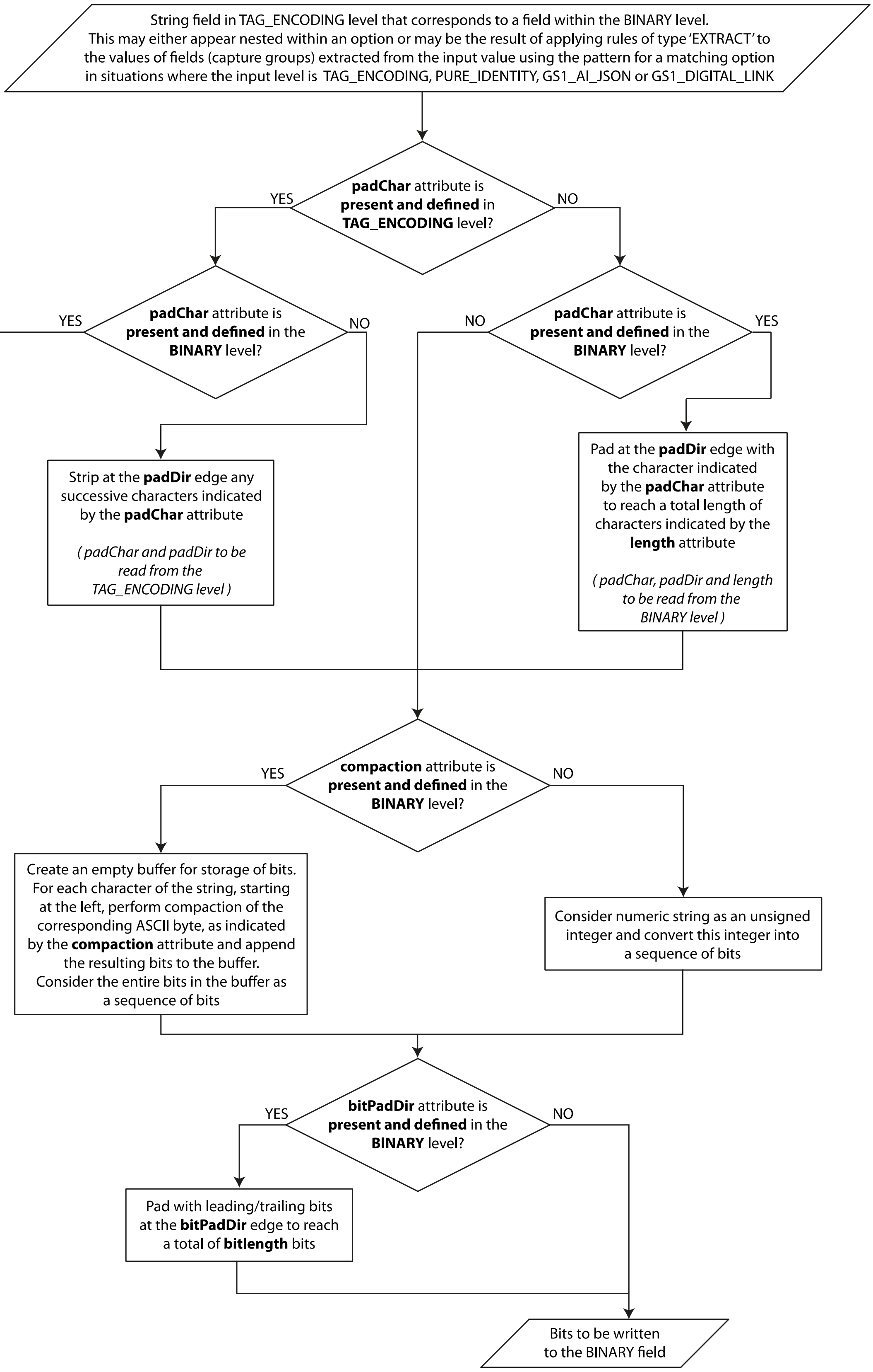
Note that in the tag-encoding URN format, pure-identity URN format and GS1 Digital Link URI format, some fields may support symbol characters and some of these may need to be escaped using percent-encoding when expressed within a URN format or Web URI / URL format.

In such situations, within the TDT definition file, a field that is present within the binary level may not be present with the same field name within the tag-encoding URN level. For example, SGTIN-198 supports serial numbers from the GS1 AI encodable character set 82, specified in Figure 7.11-1 of the GS1 General Specifications. The final field within the binary level of the TDT definition file for SGTIN-198 is named 'serial', whereas within the tag-encoding level, the final field is named 'urnEscapedSerial'. These are considered to be semantically equivalent fields and rules defined within the tag-encoding level (and also within the pure-identity level and GS1 Digital Link level) express the functions for converting between these semantically equivalent fields, by either applying or removing percent-encoding for those symbol characters that need to be escaped within URN or Web URI formats, as appropriate.

If the output format is binary and the input format is one of tag-encoding URN, pure-identity URN or GS1 Digital Link URI, any percent-encoded symbol characters that may be present in the capture groups extracted from matching the input value using the regular expression pattern must first be unescaped, by applying the rule(s) of type 'EXTRACT' in order to calculate the corresponding non-escaped field and value that can then be encoded into binary using the logic of Figure 3-5.

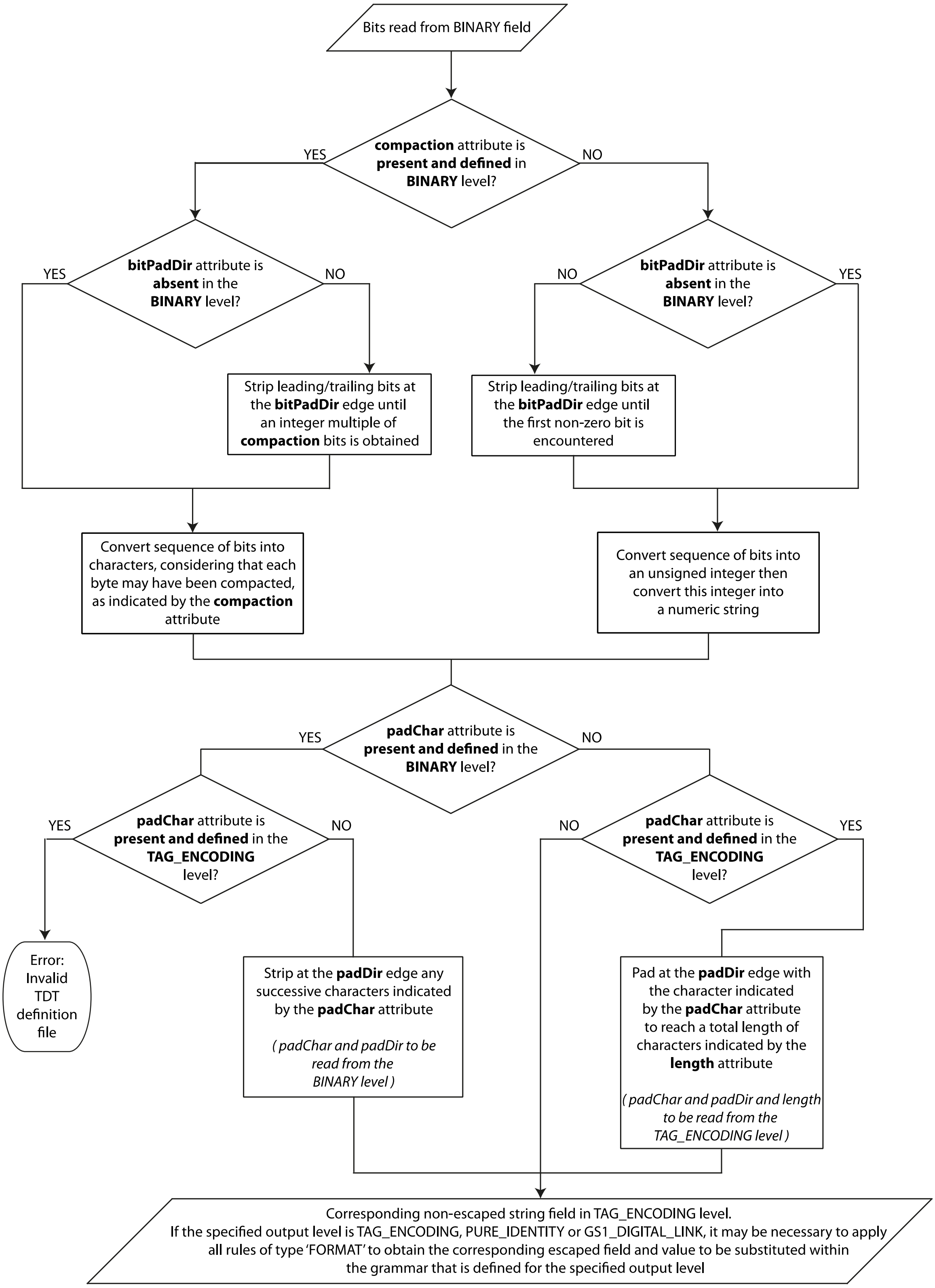
If the input format is binary and the specified output format is one of tag-encoding URN, pure-identity URN or GS1 Digital Link URI, after applying the logic of Figure 3-6 to obtain non-escaped output values for each field, it is necessary to apply any rules of type 'FORMAT' defined within the specified output level in order to calculate the corresponding escaped (percent-encoded) field and value to be substituted in the grammar that is defined for the specified output level.

1149 **Figure 3-5** Summary of rules about whether or not to add or remove padding to a field when encoding from
1150 other formats to binary encoding



1151

1152 **Figure 3-6** Summary of rules about whether or not to pad or strip a field when decoding from
1153 binary encoding to any format other than binary



1154

1155

For example, for a 96-bit SGTIN, for the `field` *whose* `name` *is "*`companyprefix`*", the other levels define a* `length` *attribute of 7, a* `padChar` *of '0' and the* `padDir` *as 'LEFT' for the option where* `optionKey` *is 7. For the corresponding binary level where* `optionKey` *is 7,* `bitLength` *is 24,* `bitPadDir` *is '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 of its leading zero bits at the LEFT edge, then translated to the integer 37000, then padded to the LEFT with the pad character '0' to reach a total of 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 dot delimiters, e.g. '..' in a URN 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 dot (.) delimiters because '0' is of length 1 character – not zero characters length as required by the `length` *attribute of the appropriate* `field` *object.*

3.13 Compaction and Compression of values of fields

In older EPC schemes defined before TDS 2.0, when strings other than purely numeric strings are to be encoded in the binary format, the `field` element contains an additional attribute, `compaction`. 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 reducing 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/IEC 15962 [ISO15962], 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.

3.14 Values of the name property of field objects within TDT definition files

The `name` property of `field` objects within the TDT definition files SHALL consist of lower case or lower camel case alphanumeric words with no spaces or hyphens. The use of a name within one EPC scheme does not imply any correlation with an identically named field within a different EPC scheme; each EPC scheme effectively uses its own private namespace for field names. The table below lists some field names that are used in the EPC schemes appearing in TDT definition files, although it is not exhaustive. However, the field names already defined in this table should be considered for re-use where appropriate when creating a new TDT definition file; a new TDT definition file should not redefine such field names to have a different meaning, nor should a different field name be introduced if one of the existing defined field names would suffice.

1195 **Table 3-2** Field names used within TDT definition files

Field name	EPC scheme(s) in which it appears	Explanation
assettype	GRAI-96 GRAI-170	Assigned by the managing entity to a particular class of asset
bestBeforeDate	DSGTIN+	End of the period under which the product will retain specific quality attributes or claims even though the product may continue to retain positive quality attributes after this date.
cage	ADI-var	A Commercial And Government Entity (CAGE) code (also including a NATO CAGE (NCAGE) code) - used within the ADI-var scheme)
cageordodaac	USDOD-96	Either a Commercial And Government Entity or a Department of Defense Activity Address Code (used with DOD-96 scheme) [USDOD]
comppartref	CPI-96 CPI-var	Assigned by the managing entity to a particular object class.
couponref	SGCN-96	Assigned by the managing entity for the coupon .
cpi	CPI-96 CPI-var CPI+	Component / Part Identifier
cpiserial	CPI-96 CPI-var	Assigned by the managing entity to an individual object.
dataToggle	CPI+ DSGTIN+ GDTI+ GIAI+ GRAI+ GSRN+ GSRNP+ ITIP+ SGCN+ SGLN+ SGTIN+ SSCC+	A single bit that appears immediately after the 8-bit header of the new EPC+ schemes and before the 3-bit filter value, indicating whether or not additional AIDC data is encoded after the EPC within the EPC/UII memory bank. When the single bit is set to 0, no additional AIDC data is encoded, whereas a value of 1 indicates that additional AIDC data is encoded.
documenttype		Identifies the Document Type within a company for a GDTI.
docType	GDTI-96 GDTI-113 GDTI-174	Identifies the Document Type within a company for a GDTI
dodaac	ADI-var	A Department of Defense Activity Address Code (used within the ADI-var scheme).

Field name	EPC scheme(s) in which it appears	Explanation
encodedAI	CPI+ DSGTIN+ GDTI+ GIAI+ GRAI+ GSRN+ GSRNP+ ITIP+ SGCN+ SGLN+ SGTIN+ SSCC+	Used in conjunction with **TDS tables F, K, E** and **B** to encode/decode GS1 Application Identifiers correctly to/from the binary encoding within the new EPC schemes introduced in TDS 2.0. Note that `encodedAI` does not behave exactly like other fieldnames in the sense of taking a single string or binary value. `encodedAI` appears within the `level` where `type` is `'BINARY'`, within the value of `grammar`, where it acts as a placeholder for the sequence of bits resulting from the binary encoding of the sequence GS1 Application Identifiers indicated by the list of values of the `encodedAI` property of `option`, ordered in ascending order of `seq`,using the internal values specified by name and formatted as defined in Table F for the specified ai. See also section 3.17 for further details about `encodedAI`. While `encodedAI` does not correspond to any capture group in the regular expression `pattern`, when decoding from binary, all remaining bits of the EPC identifier after the last matching capture group are considered to correspond to encodedAI in the new EPC schemes and should be decoded as values of the specified GS1 Application Identifiers and stored internally using the corresponding values of `name`, for use when constructing the output string.
expDate	DSGTIN+	Expiration date, which determines the limit of consumption or use of a product/coupon.

Field name	EPC scheme(s) in which it appears	Explanation
filter	ADI-var CPI-96 CPI-var GDTI-96 GDTI-113 GDTI-174 GIAI-96 GIAI-202 GIAI+ GRAI-96 GRAI-170 GRAI+ GSRN-96 GSRN+ GSRNP-96 GSRNP+ ITIP-110 ITIP-212 ITIP+ SGCN-96 SGCN+ SGLN-96 SGLN-195 SGLN+ SGTIN-96 SGTIN-198 SGTIN+ SSCC-96 SSCC+ USDOD-96	Fast filter value. For most EPC schemes, the filter value consists of 3 bits and supports an integer value in the range 0-7. ADI-var uses a 6-bit filter value, supporting integer values in the range 0-63. USDOD-96 uses a 4-bit filter value, supporting integer values in the range 0-15.
firstFreezeDate	DSGTIN+	The first freeze date is applicable to products that are frozen directly after slaughtering, harvesting, catching or after initial processing of the product.
gcn	SGCN+	Global Coupon Number
gdti	GDTI+	Global Document Type Identifier
gdtiprefix	GDTI-96 GDTI-113 GDTI-174	The initial 13 numeric digits of a GDTI before the alphanumeric serial component. This consists of the GS1 Company Prefix and Document Type (together totalling 12 digits), followed by the single digit GS1 check digit calculated over those 12 digits.
generalmanager	GID-96	Identifies an organisational entity that is responsible for maintaining the numbers in subsequent GID fields – Object Class and Serial Number.

Field name	EPC scheme(s) in which it appears	Explanation
giai	GIAI-96	Global Individual Asset Identifier
gln	SGLN-96 SGLN-195 SGLN+	Global Location Number
valueOf8003	GRAI+	The pad character of 0, followed by Global Returnable Asset Identifier
grai	GRAI-170	The entirety of the GRAI including its serial component, excluding the pad digit that immediately follows (8003) but which which is not part of the GRAI.
graiprefix	GRAI-96 GRAI-170	The initial 13 numeric digits of the GRAI, excluding the pad digit that immediately follows (8003), which is not part of the GRAI and also excluding the final serial component of the GRAI that appears after the check digit. These 13 digits consist of a GS1 Company Prefix and Asset Type, together totalling 12 digits, followed by a single digit GS1 check digit calculated over those 12 digits.
gs1companyprefix	CPI-96 CPI-var GDTI-96 GDTI-113 GDTI-174 GIAI-96 GIAI-202 GRAI-96 GRAI-170 GSRN-96 GSRNP-96 ITIP-110 ITIP-212 SGCN-96 SGLN-96 SGLN-195 SGTIN-96 SGTIN-198 SSCC-96	GS1 Company Prefix (GCP)
gs1companyprefixindex		An integer-based lookup key for accessing the real gs1Company Prefix – for use with 64-bit tags
gs1companyprefixlength		Length of a GS1 company prefix as a number of characters – base 10 integer e.g. for gs1company prefix = '0037000' → gs1companyprefixlength=7

Field name	EPC scheme(s) in which it appears	Explanation
gsrn	GSRN-96 GSRN+	Global Service Relation Number - Recipient
gsrnp	GSRNP-96 GSRNP+	Global Service Relation Number - Provider
gtin	DSGTIN+ SGTIN-96 SGTIN-198 SGTIN+	Global Trade Item Number
harvestDate	DSGTIN+	Date when an animal was slaughtered or killed, a fish has been caught, or a crop was harvested. This date is determined by the organisation conducting the harvesting.
indassetref	GIAI-96 GIAI-202	A serialised asset reference – for use with the GIAI
itemref	ITIP-110 ITIP-212 SGTIN-96 SGTIN-198	Identifies the Object Type or SKU within a particular company for a GTIN
itip	ITIP-110 ITIP-212 ITIP+	Identification of Trade Item Pieces
locationref	SGLN-96 SGLN-195	Identifies the Location within a company for a GLN
objectclass	GID-96	Identifies a class or "type" of thing within the GID scheme.
originalpartnumber	ADI-var	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)
packDate	DSGTIN+	The packaging date is the date when the goods were packed as determined by the packager.
piece	ITIP-110 ITIP-212	Within the ITIP, the piece number identifies an individual piece of the trade item.
prependedserial	SGCN-96	This corresponds to the string value of the field named `serial` but prefixed by the character "1", when using the "Numeric String" encoding / decoding methods defined in TDS 2.0 section 14.3.6 and 14.4.6.
prodDate	DSGTIN+	Production or assembly date, as determined by the manufacturer.
sellByDate	DSGTIN+	Indicates the date specified by the manufacturer as the last date the retailer is to offer the product for sale to the consumer.

Field name	EPC scheme(s) in which it appears	Explanation
serial	ADI-var CPI+ DSGTIN+ GDTI-96 GDTI-113 GDTI-174 GID-96 GRAI-96 GRAI-170 ITIP-110 ITIP-212 ITIP+ SGCN-96 SGLN-96 SGLN-195 SGLN+ SGTIN-96 SGTIN-198 SGTIN+ USDOD-96	Serial component – numeric or alphanumeric For schemes based on GTIN or ITIP (e.g. DSGTIN+, SGTIN+, SGTIN-96, SGTIN-198, ITIP+, ITIP-110, ITIP-212), this corresponds to the value of Application Identifier (21). For other schemes, the serial component corresponds to a serial component within the primary identifier or may correspond to an extension that may be used in conbination with the primary identifier in order to construct a compound identification key with globally unique instance-level granularity.
serialref	SSCC-96	A serialised reference – e.g. for use with the SSCC
serviceref	GSRN-96 GSRNP-96	Identifies the service relation within a particular company for a GSRN
sgcnprefix	SGCN-96	The initial 13 digits of the GCN, including the GS1 Company Prefix, Coupon Reference and Check Digit but excluding the serial component.
sscc	SSCC-96 SSCC+	Serial Shipping Container Code
tagLength	(all fixed-length schemes)	64/96/256 etc. – number of bits for the EPC identifier
total	ITIP-110 ITIP-212	Within the ITIP, the total count provides the total number of individual pieces of the trade item.
urlEncodedCPI	CPI+	This corresponds to the value of the field named cpi but using percent-encoding where appropriate to encode any literal symbol characters that must be escaped in URL or Web URI format
urlEscapedGdti	GDTI+	This corresponds to the value of the field named gdti but using percent-encoding where appropriate to encode any literal symbol characters that must be escaped in URL or Web URI format

Field name	EPC scheme(s) in which it appears	Explanation
urlEscapedGiai	GIAI+	This corresponds to the value of the field named giai but using percent-encoding where appropriate to encode any literal symbol characters that must be escaped in URL or Web URI format
urlEscapedGrai	GRAI+	This corresponds to the value of the field named grai but using percent-encoding where appropriate to encode any literal symbol characters that must be escaped in URL or Web URI format
urlEscapedIndAssetRef	GIAI-202	This corresponds to the value of the field named indassetref but using percent-encoding where appropriate to encode any literal symbol characters that must be escaped in URL or Web URI format
urlEscapedSerial	DSGTIN+ GDTI-174 GRAI-170 ITIP-212 ITIP+ SGLN-195 SGLN+ SGTIN-198 SGTIN+	This corresponds to the value of the field named serial but using percent-encoding where appropriate to encode any literal symbol characters that must be escaped in URL or Web URI format
urnEscapedIndAssetRef	GIAI-202	This corresponds to the value of the field named indassetref but using percent-encoding where appropriate to encode any literal symbol characters that must be escaped in URN format
urnEscapedSerial	GDTI-174 GRAI-170 ITIP-212 SGLN-195 SGTIN-198	This corresponds to the value of the field named serial but using percent-encoding where appropriate to encode any literal symbol characters that must be escaped in URN format
urnEncodedCompPartRef	CPI-var	This corresponds to the value of the field named comppartref but using percent-encoding where appropriate to encode any literal symbol characters that must be escaped in URN format

3.15 Rules and Derived Fields

Certain fields required for formatting the output format are not obtained simply from pattern matching of the input format. 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 rearrangement of the original identifier required in order to translate into the pure-identity URN format. Examples include string rearrangement 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. By working through an example for the GTIN, it is clear that although the processing steps are reversible between encoding into the pure-identity URN 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:

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

- `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.

3.15.2 Encoding the GTIN (i.e. translating from element string or Application Identifier format into pure-identity URN)

(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 older GS1 EPC schemes defined before TDS 2.0. 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.

TDT 2.0 introduces additional rules in all EPC schemes for GS1 Digital Link URI format, as well as for pure-identity URN and tag-encoding URN formats, to ensure that all symbol characters that need to be percent-encoded in URN or URL format (including GS1 Digital Link URI) are correctly encoded and decoded. This is explained in further detail in the next section – see details for new functions URNENCODE, URNDECODE, URLENCODE and URLDECODE introduced in TDT 2.0.

3.16 Core Functions

The core functions which SHALL be supported by Tag Data Translation software in order to encode/decode the EPC schemes are described in the table below.

Table 3-3 Basic built-in functions required to support encoding and deciding within the EPC schemes currently defined in TDS 2.0

Function and parameters	Result of function
`SUBSTR (string, offset)`	The substring starting at `offset` (`offset` =0 is the first character of string)

<table>
<tr><th>Function and parameters</th><th>Result of function</th></tr>
<tr><td><code>SUBSTR (string, offset, length)</code></td><td>The substring starting at <code>offset</code> (<code>offset</code> =0 is the first character of string) and of <code>length</code> characters</td></tr>
<tr><td><code>CONCAT (string1, string2, string3,…)</code></td><td>The sequential concatenation of the specified string parameters</td></tr>
<tr><td><code>LENGTH(string)</code></td><td>Returns the number of characters of a string</td></tr>
<tr><td><code>GS1CHECKSUM (string)</code></td><td>Computes the GS1 check digit given a string containing all the digits that precede (but do not include) the check digit</td></tr>
<tr><td><code>URNENCODE(string)</code></td><td>Returns a copy of the string in which each of the characters specified below is replaced with the corresponding percent-encoded sequence:
<table>
<tr><th>Symbol</th><th>"</th><th>&</th><th>/</th><th><</th><th>></th><th>?</th><th>#</th><th>%</th></tr>
<tr><td>Percent-encoded sequence</td><td>%22</td><td>%26</td><td>%2F</td><td>%3C</td><td>%3E</td><td>%3F</td><td>%23</td><td>%25</td></tr>
</table></td></tr>
<tr><td><code>URNDECODE(string)</code></td><td>Returns a copy of the string in which each of the percent-encoded sequences specified below is replaced with the corresponding symbol character:
<table>
<tr><th>Percent-encoded sequence</th><th>%22</th><th>%26</th><th>%2F</th><th>%3C</th><th>%3E</th><th>%3F</th><th>%23</th><th>%25</th></tr>
<tr><td>Symbol</td><td>"</td><td>&</td><td>/</td><td><</td><td>></td><td>?</td><td>#</td><td>%</td></tr>
</table></td></tr>
<tr><td><code>URLENCODE(string)</code></td><td>Returns a copy of the string in which each of the characters specified below is replaced with the corresponding percent-encoded sequence:
<table>
<tr><th>Symbol</th><th>!</th><th>&</th><th>'</th><th>(</th><th>)</th><th>*</th><th>+</th><th>,</th></tr>
<tr><td>Percent-encoded sequence</td><td>%21</td><td>%26</td><td>%27</td><td>%28</td><td>%29</td><td>%2A</td><td>%2B</td><td>%2C</td></tr>
</table>
<table>
<tr><th>Symbol</th><th>/</th><th>:</th><th>;</th><th><</th><th>=</th><th>></th><th>?</th><th>#</th><th>%</th></tr>
<tr><td>Percent-encoded sequence</td><td>%2F</td><td>%3A</td><td>%3B</td><td>%3C</td><td>%3D</td><td>%3E</td><td>%3F</td><td>%23</td><td>%25</td></tr>
</table></td></tr>
<tr><td><code>URLDECODE(string)</code></td><td>Returns a copy of the string in which each of the percent-encoded sequences specified below is replaced with the corresponding symbol character:
<table>
<tr><th>Percent-encoded sequence</th><th>%21</th><th>%26</th><th>%27</th><th>%28</th><th>%29</th><th>%2A</th><th>%2B</th><th>%2C</th></tr>
<tr><td>Symbol</td><td>!</td><td>&</td><td>'</td><td>(</td><td>)</td><td>*</td><td>+</td><td>,</td></tr>
</table>
<table>
<tr><th>Percent-encoded sequence</th><th>%2F</th><th>%3A</th><th>%3B</th><th>%3C</th><th>%3D</th><th>%3E</th><th>%3F</th><th>%23</th><th>%25</th></tr>
<tr><td>Symbol</td><td>/</td><td>:</td><td>;</td><td><</td><td>=</td><td>></td><td>?</td><td>#</td><td>%</td></tr>
</table></td></tr>
</table>

In order to make full use of the Tag Data Translation definition 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 older GS1 EPC schemes defined before TDS 2.0, when output in the GS1 element string or GS1 Digital Link URI format is required. Further details about calculation of the GS1 check digit can be found in section 7.9.1 of the GS1 General Specifications [GS1GS]; GS1 also maintains an online check digit calculator [GCheckD] at https://www.gs1.org/services/check-digit-calculator.

It is important to note that modern programming languages (including JavaScript, 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. The table below indicates a number of language-specific equivalents for the three-parameter SUBSTR function in Table 3-3.

Table 3-4 Comparison of how substring functions are defined in a number of modern programming languages. The parameters offset and length are of integer type

	`SUBSTR(string,offset,length)`	Notes
JavaScript	`String.substr(offset,length)` `String.substring(offset,endIndex)`	endIndex = offset+length the character at endIndex is excluded from the returned substring
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(offset, endIndex)`	endIndex = offset+length
Python	`String[offset:end]`	end = offset+length

3.17 Encoded GS1 Application Identifiers in new EPC schemes introduced in TDS 2.0

The new EPC schemes introduced in TDS 2.0 include variable-length structural components and some of these may also be alphanumeric. For alphanumeric structural components, TDS 2.0 makes use of encoding indicators to allow the most efficient encoding method to be selected, depending on the actual value, typically requiring fewer bits per character for more restrictive character sets.

Because of this flexibility in the new EPC schemes, it is not possible to declare in advance exactly how many bits will be required for the value of each GS1 Application Identifier that is encoded after the EPC header, AIDC data toggle and 3-bit filter value. Instead, a new `encodedAI` element appears nested within `option` and indicates (via the `ai` attribute) which GS1 Application Identifier is to have its value encoded next, as well as the `name` of an internal variable that should hold its value, similar to the use of the `name` attribute within `field` or `rule` elements. Also in common with `field` or `rule` elements, a `seq` attribute indicates the sequential order in which the value of the GS1 Application Identifier should be encoded. 'encodedAI' also appears in the ABNF grammar for new EPC schemes introduced in TDS 2.0 to indicate where the binary representation of the values of those encoded GS1 Application Identifiers appears in the binary string, namely after the bits that encode the filter value.

For each GS1 Application Identifier key specified via the `ai` attribute of an `encodedAI` element, the GS1 AI key (such as '01' or '21') should be found in column a of Table F. Columns b-h of Table F provide guidance about how the first component of its value should be formatted in binary. Columns i-o of Table F provide corresponding guidance about the formatting of the second component of its value, where a second component exists only for some GS1 Application Identifiers.

The remainder of this section provides a worked example for SGTIN+ assuming that the value of the GTIN (01) is 09506000134352 and the value of the Serial Number (21) is abc123. The flowcharts in chapter 12 explain each step of the process.

The BINARY `level` of the TDT definition file for SGTIN+ appears in XML as follows:

```
<level type="BINARY" prefixMatch="11110111" requiredFormattingParameters="filter,dataToggle">
  <option optionKey="1" pattern="^11110111([01])([01]{3})" grammar="'11110111' dataToggle filter
encodedAI">
    <field seq="1" decimalMinimum="0" decimalMaximum="1" characterSet="[01]*" bitPadDir="LEFT"
bitLength="1" name="dataToggle"/>
    <field seq="2" decimalMinimum="0" decimalMaximum="7" characterSet="[01]*" bitPadDir="LEFT"
bitLength="3" name="filter"/>
    <encodedAI ai="01" name="gtin" seq="3"/>
    <encodedAI ai="21" name="serial" seq="4"/>
  </option>
</level>
```

and equivalently in JSON as:

```
"level": [{
          "type": "BINARY",
          "prefixMatch": "11110111",
```

```
                    "requiredFormattingParameters": "filter,dataToggle",
                    "option": [{
                          "optionKey": 1,
                          "pattern": "^11110111([01])([01]{3})",
                          "grammar": "'11110111' dataToggle filter encodedAI",
                          "field": [{
                                      "seq": 1,
                                      "decimalMinimum": 0,
                                      "decimalMaximum": 1,
                                      "characterSet": "[01]*",
                                      "bitPadDir": "LEFT",
                                      "bitLength": 1,
                                      "name": "dataToggle"
                                },
                                {
                                      "seq": 2,
                                      "decimalMinimum": 0,
                                      "decimalMaximum": 7,
                                      "characterSet": "[01]*",
                                      "bitPadDir": "LEFT",
                                      "bitLength": 3,
                                      "name": "filter"
                                }
                          ],
                          "encodedAI": [{
                                "ai": "01",
                                "name": "gtin",
                                "seq": 3
                          }, {
                                "ai": "21",
                                "name": "serial",
                                "seq": 4
                          }]
                    }]
              },
          ...
         ]
```

This indicates that the first field (seq="1") is of bitLength=1 and encodes the value of variable 'dataToggle'.

The second field (seq="2") is of bitLength=3 and encodes the value of variable 'filter'.

The third (seq="3") piece of data encodes the value of GS1 Application Identifier (01), using the value in variable 'gtin'.

The fourth (seq="4") piece of data encodes the value of GS1 Application Identifier (21), using the value in variable 'serial'.

After encoding/decoding the 1-bit data toggle and 3-bit filter value, the value of the GTIN, AI (01) is encoded or decoded. Looking up AI (01) in Table F, a row can be found that looks like this in XML:

```
<row a="01" b="Fixed-length numeric" c="14.5.4" d="14" e="56"></row>
```

or equivalently, like this in JSON:

```
"rows": [
     ...
     {"a":"01", "b":"Fixed-length numeric", "c":"14.5.4", "d":"14", "e":"56"},
     ...
]
```

This indicates that the encoding method 'Fixed-length numeric' (as defined in section 14.5.4 of TDS 2.0) must be used, that the value should be 14 digits, encoded as 56 bits (using 4 bits per digit).

If encoding an SGTIN+, the 14-digit value of the GTIN must therefore be encoded as the next 56 bits following the EPC header, data toggle and filter value. If decoding an SGTIN+, the next 56 bits after the EPC header, data toggle and filter value must be read and decoded using the 'Fixed-length numeric' method and translated to a 14-digit value that is to be stored in the variable named 'gtin' (specified in the `encodedAI` element `<encodedAI ai="01" name="gtin" seq="3"/>`).

Moving on to the next encoded GS1 Application Identifier, a lookup of AI (21) in Table F finds a row that looks like this in XML:

```
<row a="21" b="Variable-length alphanumeric" c="14.5.6" f="3" g="5" h="20"></row>
```

or equivalently, like this in JSON:

```
"rows": [
     ...
     {"a":"21", "b":"Variable-length alphanumeric", "c":"14.5.6", "f":"3", "g":"5", "h":"20"},
     ...
]
```

This time, the encoding method is specified to be "Variable-length alphanumeric" as specified in section 14.5.6 of TDS 2.0. Also specified (via column f) is that a 3-bit encoding indicator shall be used, followed (via column g) by a 5-bit length indicator. Column h specifies that the maximum permitted length for the value is 20 characters for serial number (21). Section 14.5.6 of TDS 2.0 explains the encoding method 'Variable-length alphanumeric' and contains a decision tree and number of subsections that define encodings that depend on the actual character set used in the value. Table E lists the various encoding options and the corresponding character sets supported by each. For this example, if the serial number (21) is "abc123", it is most efficient to use lower-case hexadecimal encoding, corresponding to row 2 of Table E. Column f of Table E provides a regular expression for the supported character set. Column b of Table E provides the corresponding 3-bit value for the encoding indicator, in this case '010'.

So after encoding the 56 bits of GTIN, the next 3 bits should be the encoding indicator, set to '010' in this worked example for a serial number of 'abc123', followed by a 5-bit length indicator. If encoding the serial number (21) value of 'abc123', the length indicator should indicate 6 characters and should therefore appear as '00110', which is 6 in binary, left-padded to a total of 5 bits for the length indicator. Following this, the encoding method determines how many remaining bits express the value. In this example, variable-length lower case hexadecimal uses 4 bits per hexadecimal character, so 4 bits/character x 6 characters = 24 bits for encoding the value. In this example, those 24 bits would be '1010 1011 1100 0001 0010 0011', in order to encode 'abc123'. Note that Table B lists the number of bits needed to encode N characters for each value of the encoding indicator. Table B can also be used to calculate the number of bits, which avoids the need for floating-point calculations in constrained systems that might find a table lookup more efficient.

For decoding the serial number (21) from a binary string, the same row for AI (21) from Table F is also used, as shown earlier.

Column f indicates that a 3-bit encoding indicator must be read, followed by a 5-bit length indicator (indicated by column g). If the 3-bit encoding indicator is '010'. A lookup of '010' in column b of Table E reveals which encoding method had been used, in this case 'variable-length lower case hexadecimal' using 4 bits per character. After reading the 3-bit encoding indicator, column g of the Table F row for AI (21) indicates that a 5-bit length indicator should be read. If its value is '00110', then 6 characters have been encoded. Since the selected encoding method uses 4 bits per character, then it will be necessary to read 4 x 6 = 24 bits and to interpret these as 6 lower-case hexadecimal characters.

It should be clear from the above worked example that particularly for structural components that are variable-length or alphanumeric, it would not be possible to prescribe the bitLength value via a field element, so instead an `encodedAI` element is used to make use of lookup in Table F and Table E. Table B is also useful for looking up the number of bits to be used for each encoding method, for a specified number of characters.

Column a of Table B is as follows in XML:

```
<column id="a" name="Length" description="Number of digits or characters"></column>
```

or equivalently in JSON:

```
"columns": [
     {"id":"a","name":"Length","description":"Number of digits or characters"},
     ...
]
```

Searching the columns of Table B for a match where encodingIndicator="2" (the base 10 value corresponding to '010') yields the following column in XML:

```
<column id="d" name="Variable-length lower case hexadecimal" description="Bits required for numeric string
/ lower case hexadecimal encoding at 4 bits/digit" encodingIndicator="2" specSection="14.5.6.3"></column>
```

or equivalently, in JSON:

```
"columns": [
     ...
     {"id":"d","name":"Variable-length lower case hexadecimal","description":"Bits required for numeric
     string / lower case hexadecimal encoding at 4 bits/digit","encodingIndicator":2,
     "specSection":"14.5.6.3"},
     ...
]
```

In this example, if we know that the length is 6 characters, so searching Table B for a match where a="6" yields the following row in XML:

```
<row a="6" b="20" c="24" d="24" e="32" f="36" g="42" />
```

or equivalently, in JSON:

```
"rows": [
     ...
     {"a":"6","b":"20","c":"24","d":"24","e":"32","f":"36","g":"42"},
     ...
]
```

Reading the value of column d in this row reveals the number of bits to be read, in this case 24 bits.

Table B is particularly useful to avoid the need for any floating-point arithmetic calculations when the encoding method is either 'Variable-length numeric string' (encoding indicator '000') or ' Variable-length URN Code 40' (encoding indicator '101'), for which the number of bits is not simply an integer multiplied by the number of characters. * MERGEFORMAT* MERGEFORMAT

Figure 3-7 Encoding GS1 Application Identifiers

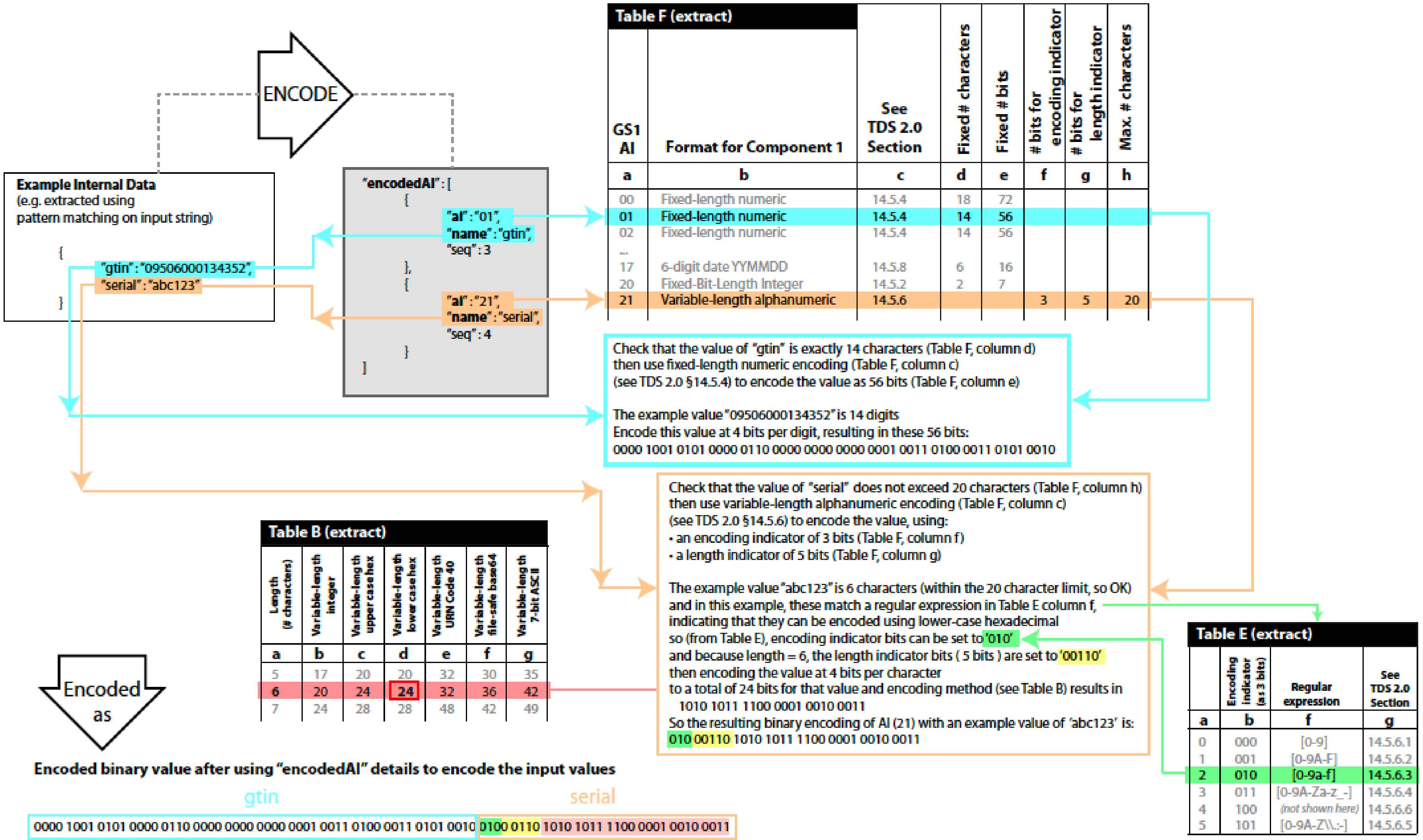
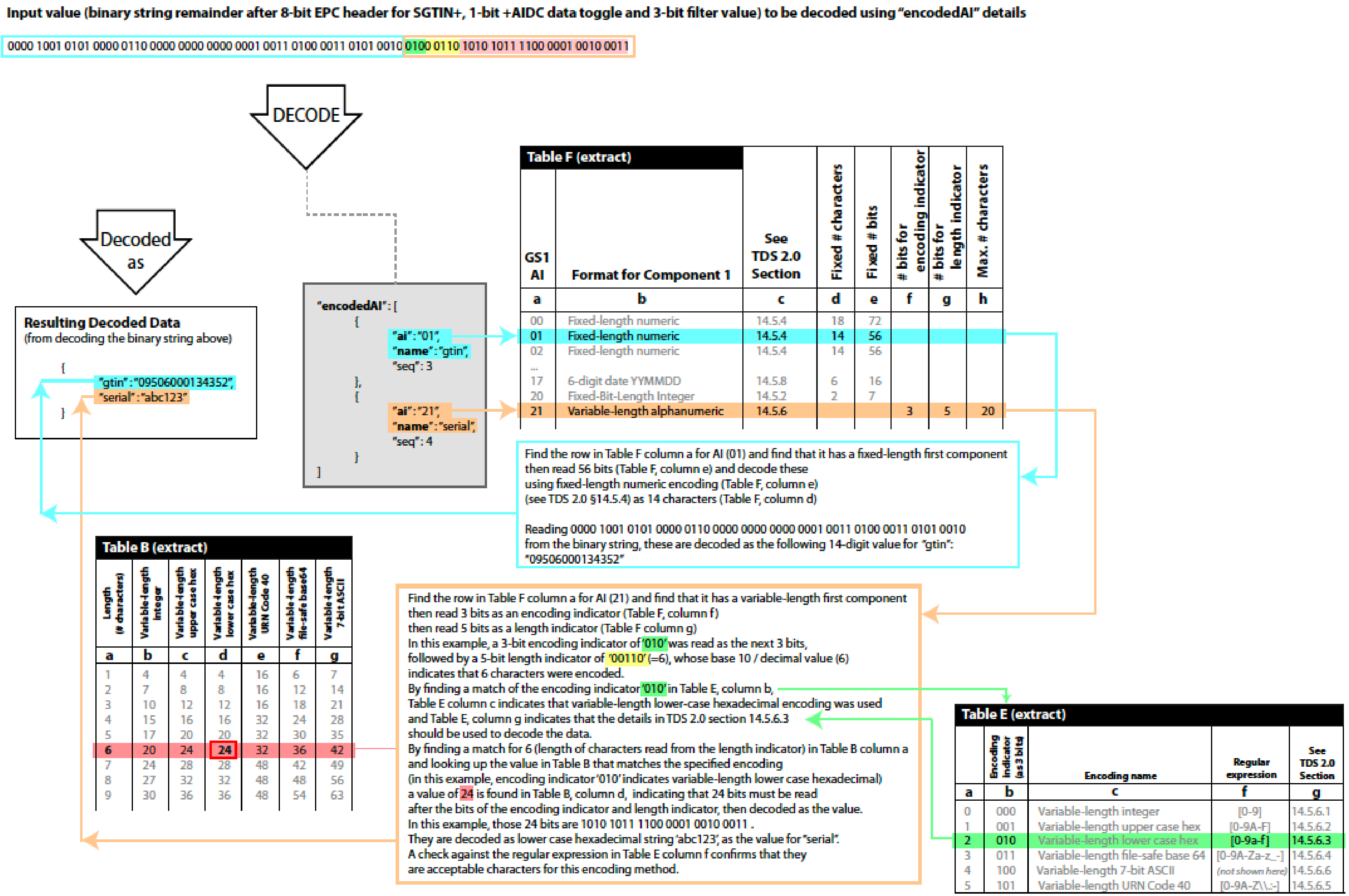


Figure 3-8 Decoding GS1 Application Identifiers



4 Encoding/Decoding of additional AIDC data after the EPC

The new EPC schemes introduced in TDS 2.0 all support the ability to encode additional AIDC data immediately after the EPC binary string within the EPC/UII memory bank, as explained in section 15.3 of TDS 2.0. The following subsections explain the encoding and decoding procedures in further detail, using worked examples. Chapter 12 provides flowcharts to describe each step in further detail.

4.1 Encoding additional AIDC data after the EPC

If encoding additional AIDC data, the dataToggle bit SHALL be set to 1. Consider the first piece of AIDC data to be encoded. Firstly, encode the corresponding GS1 Application Identifier key, using 4 bits per digit. For example, to encode expiration date (17), write '0001 0111'. Alternatively, to encode a net weight value using AI (3103), write '0011 0001 0000 0011'.

Next, lookup the GS1 Application Identifier key in column a of Table F. For the two examples above, Table F contains the following rows:

```
<row a="17" b="6-digit date YYMMDD" c="14.5.8" d="6" e="16"></row>
<row a="3103" b="Fixed-Bit-Length Numeric String" c="14.5.2" d="6" e="20"></row>
```

Then to encode the corresponding values, refer to the relevant sections of TDS 2.0 (as described in the sections indicated by column c of Table F).

For encoding methods that use fixed-length values, column d of table F indicates the number of characters for the value, while column e of Table F indicates the number of bits.

For encoding methods that support variable-length values, column f indicates the number of bits to encode for the encoding indicator (either column f is empty/absent or its value is 3), while column g indicates the number of bits to encode for the length indicator. The number of bits for the length indicator is always sufficient to be able to express the maximum permitted length for the value or component of the value (as expressed in column h).

The values of a small number of GS1 Application Identifiers are expressed within Table F as two components, rather than a single component. In this case, columns i-o may be populated with details for the second component of the value. For example, GS1 Application Identifier (7030) has the following row in Table F, with details for a first component (columns b-h) and a second component (i-o):

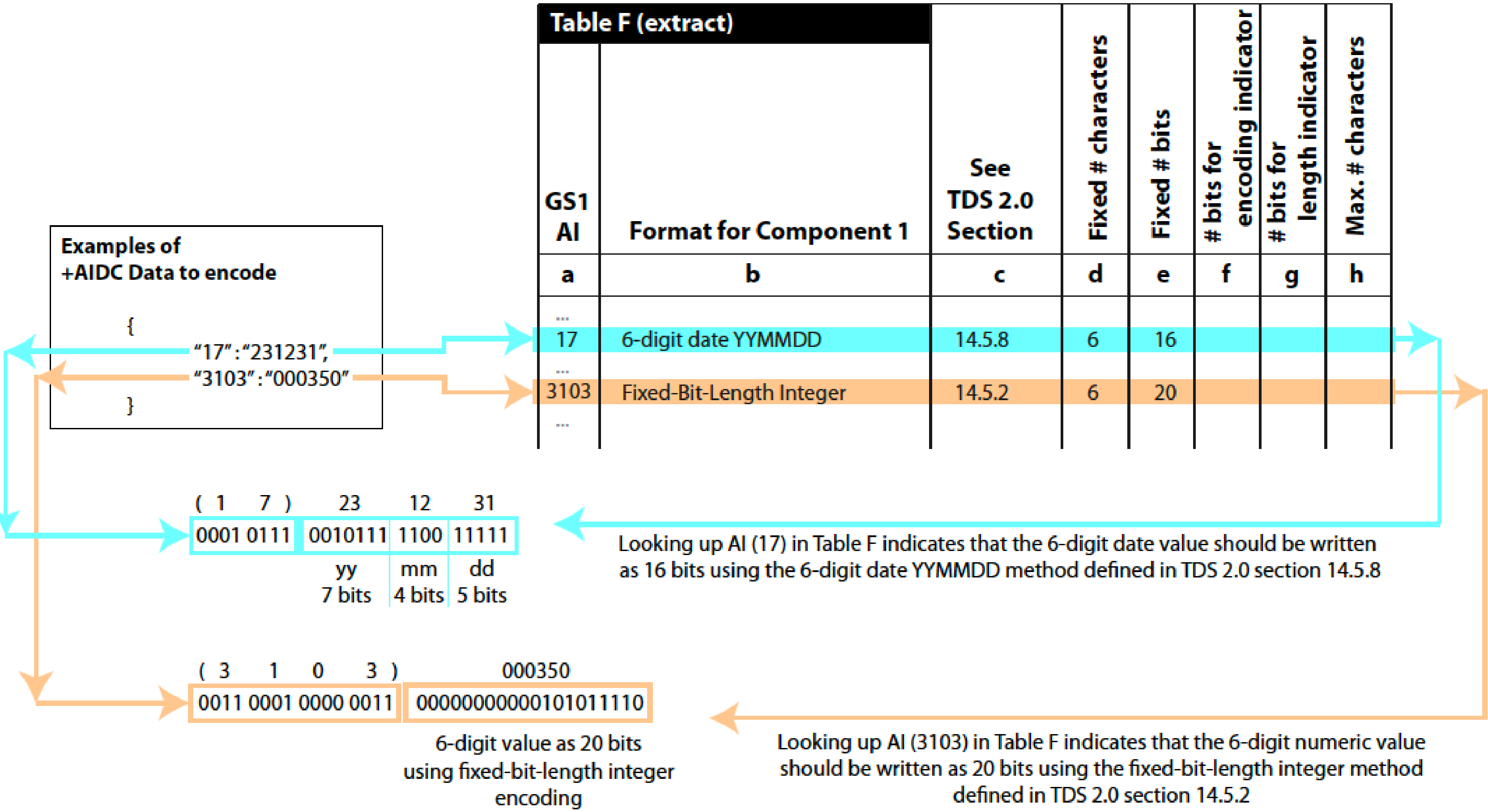
```
<row a="7030" b="Fixed-Bit-Length Numeric String" c="14.5.2" d="3" e="10"
     i="Variable-length alphanumeric" j="14.5.6" m="3" n="5" o="27"></row>
```

The value of each GS1 Application Identifier is then encoded using the methods specified in columns b / c (and column i / j) resulting in a further number of bits specified by the sum of column e and column o (if specified).

Any further AIDC data is encoded using the same procedure, writing the GS1 Application Identifier key first, as 8 bits for a 2-digit key such as (17) or (10), 12 bits for a 3-digit key such as (420) or 16 bits for a 4-digit key such as (3103), then using Table F to encode the corresponding value in binary.

Flowcharts in section 12.1 provide the logic for encoding additional AIDC data after the EPC.

Figure 4-1 Encoding AIDC data



4.2 Decoding additional AIDC data after the EPC

If the dataToggle bit was set to 1, then additional AIDC data has been encoded immediately after the binary EPC string. After reading the binary EPC string, using the procedure detailed in chapter 3.17, read a further 8 bits and decode these as two hexadecimal characters. If either character is in the range a-f/A-F, stop; alphanumeric data headers are not yet defined in TDS 2.0. If both characters are in the range

0-9, concatenate these and lookup the value in column a of Table K. Read the value of columns b and c. Column b indicates whether this corresponds to the first two digits of a 2-digit, 3-digit or 4-digit GS1 Application Identifier key. Column c indicates whether a further 0, 4 or 8 bits must be read (interpreting each set of 4 bits as a hexadecimal character). For example, if the first 8 bits correspond to hexadecimal characters '8' and '0', a lookup '80' in column a of Table K yields this row:

```
<row a="80" b="4" c="8"></row>
```

From this, column b indicates that '80' are the first two digits of a 4-digit GS1 Application Identifier key, (80xx) where xx is not yet known. Column c indicates that a further 8 bits must be read. If those further 8 bits correspond to hexadecimal characters '0' and '8', then the 4-bit GS1 Application Identifier is actually (8008), by concatenating the initial '80' (read from the initial 8-bit data header) with the additional digits '0' and '8'.

Next, lookup the GS1 Application Identifier key in column a of Table F. Doing so for (8008) yields the following row:

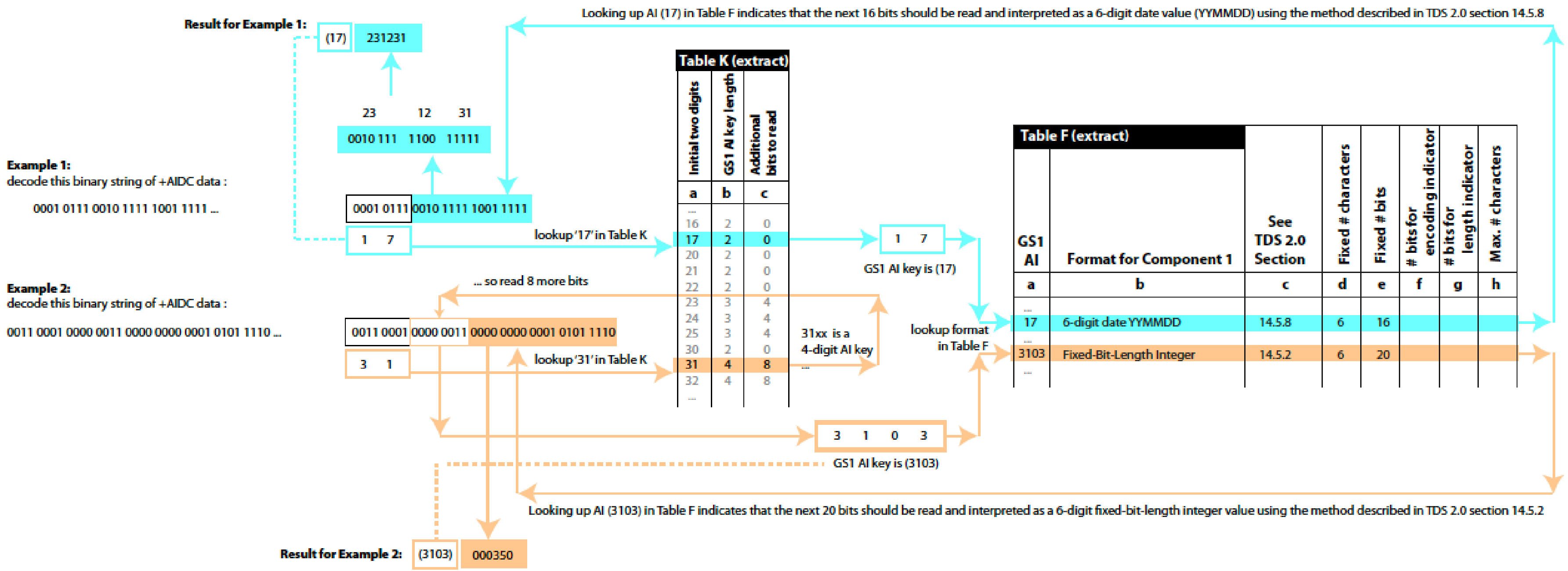
```
<row a="8008" b="Variable-precision date+time" c="14.5.11"></row>
```

Column b indicates that the next bits are encoded using the Variable-precision date+time method, described (see column c) in section 14.5.11 of TDS 2.0.

After decoding those bits using that method and setting those as the value of the corresponding GS1 Application Identifier key, (8008) in this example, repeat this procedure in case further GS1 Application Identifiers and their values are encoded, reading a further 8 bits for the next data header.

Flowcharts in section 12.2 provide the logic for decoding additional AIDC data after the EPC.

Figure 4-2 Decoding AIDC data



5 TDT Definition Files – formal definition

TDT definition files are currently provided in XML and JSON for the following EPC schemes:

SGTIN-96, SGTIN-198, SSCC-96, SGLN-96, SGLN-195, GRAI-96, GRAI-170, GIAI-96, GIAI-202, GDTI-96, GDTI-174, GSRN-96, GSRNP-96, SGCN-96, ITIP-110, ITIP-212, CPI-96, CPI-var, GID-96, USDOD-96, ADI-var, SGTIN+, DSGTIN+, SSCC+, SGLN+, GRAI+, GIAI+, GDTI+, GSRN+, GSRNP+, SGCN+, ITIP+, CPI+.

The remainder of this chapter is written in a way that attempts to use neutral language that can explain the structure of a TDT definition file, whether it is formatted in XML or JSON. An object corresponds to a data object or class within the UML class diagram (see Figure 3-1) and always corresponds to an XML element or JSON object/class/dictionary. In XML, simple datatype properties of an object may be expressed as inline XML attributes, while more complex object properties are expressed in XML as nested child elements because their values are structured. JSON has no concept of elements or inline attributes – only objects (also known as classes or dictionaries), lists (also known as arrays) and properties (also known as keys).

5.1 Root object

The `epcTagDataTranslation` object is the root object or root-level property of each TDT definition file. Within it, a number of metadata properties such as `version`, `date` and `epcTDSVersion` are defined, together with the `scheme` property (see section 5.2).

5.1.1 Datatype Properties (inline XML attributes)

Name	Description	Example Values
`version`	TDT Definition version number	2.0
`date`	Creation Date	2023-*** TBD
`epcTDSVersion`	TDS Specification version	2.0

5.1.2 Object Properties (nested XML elements)

Name	Description
`scheme`	Please see Section 5.2 for more details

5.2 Scheme object

For every EPC scheme defined in TDS, the `scheme` object provides details of encoding/decoding rules and formats for use by TDT implementations.

1533 ### 5.2.1 Datatype Properties (inline XML attributes)

Name	Description	Example Values
name	Name of the EPC scheme	SGTIN-96, SGTIN-198, SSCC-96, SGLN-96, SGLN-195, GRAI-96, GRAI-170, GIAI-96, GIAI-202, GDTI-96, GDTI-174, GSRN-96, GSRNP-96, SGCN-96, ITIP-110, ITIP-212, CPI-96, CPI-var, GID-96, USDOD-96, ADI-var, SGTIN+, DSGTIN+, SSCC+, SGLN+, GRAI+, GIAI+, GDTI+, GSRN+, GSRNP+, SGCN+, ITIP+, CPI+
optionKey	The name of a variable whose value determines which one of multiple options to select. Note that optionKey is no longer a required attribute within the scheme structure, although it is still specified for fixed-length EPC constructs. Even if the optionKey value is not specified within a scheme, nested option structures are nevertheless numbered with an optionKey attribute and translation is performed between option structures that have the same value of optionKey attribute present within the option structure.	companyprefixlength
tagLength	This refers to the length of the EPC identifier itself (e.g. the bits encoded from position 20_h 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. The tagLength attribute SHALL NOT be specified for a variable-length EPC identifier, including all the newer EPC schemes introduced in TDS 2.0.	96 or larger values.

1534 ### 5.2.2 Object Properties (nested XML Elements)

Name	Description
level	Contains option elements expressing a pattern, grammar and encoding/decoding rules for each format. See section 5.3 for further details.

5.3 Level object

For each format, `prefixMatch` and `type` is specified for each `level`. Nested within the `level` element are `option` objects (which provide the `pattern` regular expressions for parsing the input into fields and ABNF `grammar` for formatting the output), as well as `rule` objects used for computing additional `field` values from functional operations from `field` values that are already known.

Datatype Properties (inline XML Attributes)

Name	Description	Example Values
`type`	Indicates format	BINARY TAG_ENCODING PURE_IDENTITY BARE_IDENTIFIER ELEMENT_STRING GS1_DIGITAL_LINK TEI
`prefixMatch`	Prefix value required for each encoding/decoding level	00001010 uri:epc:tag:sscc-96 uri:epc:id:sscc sscc= (00)
`requiredParsingParameters`	Comma-delimited string listing names of fields whose values may need to be specified in the list of suppliedParameters in order to parse the fields of an input value at this level. Note that for `gs1companyprefixlength` it may be possible to determine this through use of the tables provided at https://www.gs1.org/standards/bc-epc-interop or through other means. See also the `gcpOffset` property of the Field object	gs1companyprefixlength
`requiredFormattingParameters`	Comma-delimited string listing names of fields whose values may need to be specified in the list of suppliedParameters in order to format the output value at this level. Note that if a value for `uriStem` is not specified via suppliedParameters, a value of https://id.gs1.org/ should be assumed as the default value, since this will result in a reference GS1 Digital Link URI.	filter,tagLength,uriStem, dataToggle

Name	Description	Example Values
`gs1DigitalLinkKeyQualifiers`	An ordered sequence of GS1 Application Identifiers that should appear in the specified order after the primary identification key and its value within the URI path information when the output is GS1 Digital Link. See section 3.4.1.2 for further details of post-processing of GS1 Digital Link URI output and the use of the `gs1DigitalLinkKeyQualifiers` parameter. Note that in the TDT definition files provided in JSON format, this is a JSON list using square brackets. For the TDT definition files provided in XML format, this parameter is provided as an inline XML attribute and expressed as a JSON string, in which every double quote character appears escaped as " Implementations of Tag Data Translation that rely on the XML definition files will need to unescape the double quote characters then use a JSON parsing function in order to create the corresponding ordered list of GS1 Application Identifiers for this parameter.	`["22","10","21"]` for all SGTIN / DSGTIN+ schemes.

Object Properties (nested XML Elements)

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

5.4 Option object

Each `option` object provides the `pattern` regular expressions for parsing the input into fields and ABNF `grammar` for formatting the output. For EPC schemes defined before TDS 2.0, multiple `option` elements are used as a way of implementing rows of partition tables and the corresponding variations in length of the GS1 Company Prefix component and often the next structural component (whose length typically decreases as the length of the GS1 Company Prefix component increases). The new EPC schemes introduced in TDS 2.0 do not make use of partition tables based on the length of the GS1 Company Prefix, which need not be known when using the new EPC schemes. The only new EPC scheme currently using multiple `option` elements is DSGTIN+, in which each `option` element corresponds to a different value of the 4-bit date type indicator fields (and interpretation of a different 6-digit date field for each `option`).

AIs SHALL be specified in a strict sequence when providing input for the DSGTIN+ scheme as ELEMENT_STRING, BARE_IDENTIFIER. The current TDT definition files for DSGTIN+ expect that, within ELEMENT_STRING and GS1_DIGITAL_LINK and BARE_IDENTIFIER, the GTIN (01) will be specified first, followed by the serial number (21) in second position, followed by the prioritised date field (e.g. (17) for expiration date) in third position. **This sequence is important**, because the patterns specified for DSGTIN+ will not match an alternative sequence (e.g. such as gtin, date, serial or date, gtin, serial). GS1 Digital Link already enforces/requires this sequence because the date field appears in the URI query string.

Datatype Properties (inline XML 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 output format is appropriate for the `<option>` element for the input format. 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' are used in GS1-based EPC schemes defined before TDS 2.0 and correspond to the length of the GS1 Company Prefix component. 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. In the case of DSGTIN+, the `optionKey` is used to distinguish between different meanings of the prioritised date field, e.g. best before data vs expiration date vs production date vs harvest date.
`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 of literal values and substituted variables (fields)	'00101111' filter cageordodaac serial N.B. single quoted strings indicate fixed literal strings, unquoted strings indicate substitution of the correspondingly named field values. Square brackets enclose grammar components that are optional or conditional.

Name	Description	Example Values
aiSequence	An ordered sequence of GS1 Application Identifiers that should appear in the specified order in order for the input value to be able to match the pattern provided in the TDT definition file. See section 3.4.1.1 for further details about pre-processing of the input and use of the aiSequence parameter. Note that in the TDT definition files provided in JSON format, this is a JSON list using square brackets. For the TDT definition files provided in XML format, this parameter is provided as an inline XML attribute and expressed as a JSON string, in which every double quote character appears escaped as " Implementations of Tag Data Translation that rely on the XML definition files will need to unescape the double quote characters then use a JSON parsing function in order to create the corresponding ordered list of GS1 Application Identifiers for this parameter.	["01","21"] for the GS1_AI_JSON and GS1_DIGITAL_LINK levels for all SGTIN EPC schemes. ["01","21","17"] and other variations for other prioritised date fields within the GS1_AI_JSON and GS1_DIGITAL_LINK levels for all DSGTIN+ EPC scheme.

1563 **Object Properties (nested XML Elements)**

Name	Description
field	Provides information about each of the variables, e.g. (min, max) values, allowed character set, length, padding etc.
encodedAI	For new EPC schemes defined in TDS 2.0, provides information about which GS1 Application Identifiers have their values appearing next within the binary string, according to the format rules defined in Table F for each of those GS1 Application Identifiers.

1564 ## 5.5 Field object

1565 **Datatype Properties (Inline XML 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	Minimum value allowed for this field in base 10	"0"
decimalMaximum	Maximum value allowed for this field in base 10	"9999999"
length	Required length of this field in string characters.	7

Name	Description	Example Values
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 regular expression character range notation or as a non-capturing group that expresses explicit percent-encoded sequences in place of the symbol characters that must be escaped in URN or URL formats.	[0-9]*,[01]*, [0-9A-HJ-NP-Z]*
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'
gcpOffset	For EPC schemes defined before TDS 2.0, the field that corresponds to a primary GS1 identification key constructed from a GS1 Company Prefix includes this property gcpOffset within all levels that are not BINARY, TAG_ENCODING or PURE_IDENTITY. The value (0 or 1) indicates the position of the GS1 Company Prefix relative to the start of the field value. A value of 0 for gcpOffset indicates that the GS1 Company Prefix starts at the start of the field value (with zero offset from the left). A value of 1 for gcpOffset indicates that the GS1 Company Prefix starts at the character of the field value (after an offset of 1 character from the left). SGTIN, ITIP and SSCC have a value of '1' for gcpOffset because of the presence of an indicator digit or extension digit preceding the GS1 Company Prefix. All other schemes have a value of '0' for gcpOffset. This enables comparison of the initial digits of the GS1 Company Prefix with the details provided at https://www.gs1.org/standards/bc-epc-interop (which may be helpful for automatically determining the length of the GS1 Company Prefix, where this needs to be known for many older EPC schemes defined before TDS 2.0)	'0', '1'

Name	Description	Example Values
`valueIfNull`	Specifies a value in one format (input or output) that matches a null or undefined value of the corresponding field within the other (output or input). If translating from any of BINARY, TAG_ENCODING or PURE_IDENTITY formats to any of BARE_IDENTIFIER, ELEMENT_STRING or GS1_DIGITAL_LINK, if the value of the field matches the value specified by valueIfNull, the field is considered null and SHALL NOT contribute to the output string. If translating from any of BARE_IDENTIFIER, ELEMENT_STRING or GS1_DIGITAL_LINK formats to any of BINARY, TAG_ENCODING or PURE_IDENTITY, if the value of the field is null, the value specified by `valueIfNull` ("0") SHALL be encoded within the output string to indicate that the input string contained a null value for this optional/conditional component.	`"0"`

5.5.1 Rule object

Datatype Properties (Inline XML 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 formatted as a string of characters	`'STRING', 'BINARY'`
`seq`	A sequence number to indicate the running order for rule functions sharing the same value of type. The rule functions 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 function.	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 minimum value allowed for this field in base 10	e.g. "0"
`decimalMaximum`	For numeric fields, the maximum value allowed for this field in base 10	e.g. "9999999"
`length`	Required length of this field in string characters.	7

Name	Description	Example Values
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. The range is usually expressed using the same square-bracket notation as for character ranges within regular expressions, although for the URN formats and GS1 Digital Link URI formats, the pattern and characterSet now use non-capturing groups with explicit indication of percent-encoded sequences for symbol characters that must be 'escaped' in URN or URI format; this approach ensures that each valid symbol character is counted once even when it is percent-encoded as a 3-character sequence %hh where h is a placeholder for hexadecimal characters 0-9 and A-F. Further details about percent-encoding of symbol characters in URNs and Web URIs / URLs can be found in section 3.16 that explains the new rule functions URNENCODE, URNDECODE, URLENCODE and URLDECODE.	[0-9],[01]

5.6 encodedAI object

Datatype Properties (Inline XML Attributes)

Name	Description	Example Values
seq	A sequence number to indicate the order in which GS1 Application Identifiers should be encoded in binary, using details provided in Table F. The binary values of the corresponding GS1 Application Identifiers should appear in order of ascending 'seq' value	1,2 …

Name	Description	Example Values
`name`	The name of the internal variable (or field). When encoding to a binary string, the named variable contains the value that is to be encoded in binary. When decoding a binary string, the sequence of bits read for the corresponding GS1 Application Identifier should be stored in the named variable, so that it can be used to prepare the output in the desired output format.	`gtin, serial, …`
`ai`	The 2/3/4-digit key of a GS1 Application Identifier (AI), also including GS1 AIs (such as (21)) that are used in the construction of instance-level compound keys, where appropriate.	`'00', '01', '21' etc.`

6 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 the table below:

Table 6-1 The two stages for processing rules in Tag Data Translation

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

The rules for each scheme are within the context of a particular format. The first sequence of rules, `'EXTRACT'` is tied to the input format level. The last sequence of rules, `'FORMAT'` is tied to the output format level. Each sequence may consist of zero or more `rule` elements. The rules within each sequence 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 output format level.

2. Determine the EPC scheme and input format 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 input format. 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 input format.

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.

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 capture groups 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 capture groups, from the start of the input value, e.g. the value from the first capture group should be considered as the value of the `field` element with `seq="1"`, the value of the second capture group 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 a big integer. If the `type` attribute of the input level was `"BINARY"`, treat the string of 0 and 1 characters matched by the regular expression capture group as a binary string and translate it to a base 10 big integer.

If the `decimalMinimum` attribute is specified, check that the value is not less than the base 10 minimum value specified.

If the `decimalMaximum` attribute is specified, check that the value is not greater than the base 10 maximum value specified.

If the input format was binary, perform any necessary stripping, translation of binary to big integer or string, padding, referring to the procedure described in the flowchart Figure 3-6.

5. Perform any rules of type EXTRACT within the input format 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 output format

To find the corresponding option in the output format within the same scheme, select the `level` element having the desired output format 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 output format 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 output format is binary, it is necessary to translate values from base 10 big integer or string to binary, performing any necessary stripping or padding, following the method described in the flowchart in Figure 3-5.

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.

7 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 GS1 System Architecture [GS1Arch] 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 translations.

8 Application Programming Interface

There are essentially two interfaces to consider for Tag Data Translation software, namely a client-side interface, which provides translation 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.

8.1 Client API

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

Translates `epcIdentifier` from one format into another within the same EPC scheme.

Parameters:

`epcIdentifier` – The epcIdentifier to be translated. This should be expressed as a string, in accordance with one of the grammars or patterns in the TDT definition 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 3-1.

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

filter=3;companyprefixlength=7;tagLength=96

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

1. `BINARY`
2. `TAG_ENCODING`
3. `PURE_IDENTITY`
4. `ELEMENT_STRING`
5. `GS1_DIGITAL_LINK`
6. `BARE_IDENTIFIER`
7. `TEI`

Returns:

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

Throws:

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

1. `TDTFileNotFound` – Reports if the software could not locate the configured TDT definition file or TDT table.
2. `TDTFieldBelowMinimum` – Reports a (numeric) Field that fell below the permitted `decimalMinimum` value specified by the TDT definition files.
3. `TDTFieldAboveMaximum` – Reports a (numeric) Field that exceeded the permitted `decimalMaximum` value specified by the TDT definition files
4. `TDTFieldOutsideCharacterSet` – Reports a Field containing characters outside the permitted `characterSet` range specified by the TDT definition files
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`.

8.2 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 format of the encoding/decoding rules based on the current remaining subscriptions.

9 TDT Schema, TDT Definition Files and TDT Tables

See https://ref.gs1.org/standards/tdt/artefacts for the **latest** version of the TDT tables and schema and TDT definition files for each EPC scheme. Older versions of TDT and its artefacts can be found at https://ref.gs1.org/standards/tdt/archive.

10 Glossary (non-normative)

This section provides a non-normative summary of terms used within this specification. Please refer to the www.gs1.org/glossary for the latest version. For normative definitions of these terms, please consult the relevant sections of the document.

Term	Defined / specified in	Meaning
(EPC) (Tag Data) Translation Software		A piece of software that performs translations between different formats of the EPC within any given EPC scheme. The translation software may be a library module or object which may be accessed by / embedded within any technology component in the GS1 System Architecture. 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 translations.
(Identification) Scheme		A well-defined method of assigning an identification code to an object / shipment / location / transaction
ABNF Grammar	[ABNF]	Augmented Backus-Naur Form. 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.
ADI	[TDS]	Aerospace and Defense Identifier. The ADI is designed for use by the aerospace and defense sector for the unique identification of parts or items.
Application Identifier (AI)	[GS1GS]	The field of two or more digits at the beginning of an element string that uniquely defines its format and meaning.
Binary		A sequence of binary digits or bits, consisting of only the digits '0' or '1'
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 3-3.

Term	Defined / specified in	Meaning
Checksum / Check Digit	[GS1GS] § 7.9.1 [GCheckD]	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 hash value of a data packet or software package – except that hash values 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 hash value than that it will fortuitously produce a valid check digit.
Decoding		A translation process away from the binary format, i.e in the direction: Binary → Tag-encoding URN → Pure-identity URN → GS1 Digital Link URI or Element String or Bare Identifier or TEI
Encoding		A translation process towards the binary format, i.e in the direction: GS1 Digital Link URI or Element String or Bare Identifier or TEI → Pure-identity URN → Tag-encoding URN → Binary
EPC Pure Identity URN or EPC Pure Identity URI	[TDS]	A concrete representation of an Electronic Product Code. The Pure Identity EPC URI is an Internet Uniform Resource Identifier that contains an Electronic Product Code and no other information.
EPC Tag Data Validation Software		Software which need not perform any translation but may nevertheless make use of the Tag Data Translation definition files in order to validate that an EPC in any of its formats conforms to a valid format.
EPC Tag URN or EPC Tag URI		A representation of the complete contents of the EPC Memory Bank of a Gen 2 RFID Tag, in the form of an Internet Uniform Resource Identifier that includes a decoded representation of EPC data fields, usable when the EPC Memory Bank contains a valid EPC Binary Encoding. In contrast to the Pure Identity EPC URI, the EPC Tag URI can represent the complete contents of the EPC Memory Bank, including control information in addition to the EPC.
Field		The variable elements of the EPC in any of its formats – 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 output format.
Filter Value		A 3-bit field of control information that is stored in the EPC Memory Bank of a Gen 2 RFID Tag when the tag contains certain types of EPCs. The filter value makes it easier to read desired RFID Tags in an environment where there may be other tags present, such as reading a pallet tag in the presence of a large number of item-level tags.
GID	[TDS]	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.
Global Coupon Number (GCN)	[GS1GS]	The GS1 identification key used to identify a coupon. The key comprises a GS1 Company Prefix, coupon reference, check digit and an optional serial number.
Global Document Type Identifier (GDTI)	[GS1GS]	The GS1 identification key used to identify a document type. The key comprises a GS1 Company Prefix, document type, check digit and optional serial number.
Global Individual Asset Identifier (GIAI)	[GS1GS]	The GS1 identification key used to identify an individual asset. The key comprises a GS1 Company Prefix and individual asset reference.

Term	Defined / specified in	Meaning
Global Location Number (GLN)	[GS1GS]	The GS1 identification key used to identify physical locations or parties. The key comprises a GS1 Company Prefix, location reference and check digit.
Global Returnable Asset Identifier (GRAI)	[GS1GS]	The GS1 identification key used to identify returnable assets. The key comprises a GS1 Company Prefix, asset type, check digit and optional serial number.
Global Service Relation Number (GSRN)	[GS1GS]	The GS1 identification key used to identify the relationship between an organisation offering services and the recipient or provider of services. The key comprises a GS1 Company Prefix, service reference, and check digit.
Global Trade Item Number (GTIN)	[GS1GS]	The GS1 identification key used to identify trade items. The key comprises a GS1 Company Prefix, an item reference and check digit.
GS1 Company Prefix (GCP)	[GS1GS]	A unique string of four to twelve digits used to issue GS1 identification keys. The first digits are a valid GS1 Prefix and the length must be at least one longer than the length of the GS1 Prefix. The GS1 Company Prefix is issued by a GS1 Member Organisation. As the GS1 Company Prefix varies in length, the issuance of a GS1 Company Prefix excludes all longer strings that start with the same digits from being issued as GS1 Company Prefixes.
GS1 Digital Link URI		A standardised Web URI format for GS1 identification keys, to enable linking/redirection to multiple types of information and services on the Web as well as use within Linked Data
GS1 identification key	[GS1GS]	A unique identifier for a class of objects (e.g., trade items) or an instance of an object (e.g., logistic unit). Examples include GTIN, SSCC, GLN, GRAI, GIAI, GDTI, GSRN. [TDS] defines EPC formats for GS1 identi.
GS1 System Architecture	[GS1Arch]	Defines and describes the architecture of the GS1 system of standards. The GS1 system is the collection of standards, guidelines, solutions, and services created by the GS1 community.
GSRNP	[GS1GS]	The Global Service Relation Number (GSRN) may be used to identify the provider of services in the context of a service relationship
Header	[TDS]	A binary EPC prefix which indicates the EPC scheme and may also indicate the tag length. 8 bit EPC headers are defined in the GS1 EPC Tag Data Standard for all EPC schemes for which a binary encoding is defined.
Identification of Trade Item Pieces (ITIP)	[GS1GS] [TDS]	The GTIN that is included in this element string is the GTIN for the complete trade item. The piece number identifies a piece of the trade item. The total count provides the total number of pieces of the trade item. The binary encoding of ITIP for RFID includes a mandatory Serial Number.
Identifier Format		The way in which the identifier is represented. Examples of different types of format include sequences of binary digits (bits), sequences of numeric or alphanumeric characters, as well as Uniform Resource Identifiers (URIs). Specifically, within the TDT definition files, `BINARY`, `TAG_ENCODING`, `PURE_IDENTITY`, `ELEMENT_STRING`, `BARE_IDENTIFIER` and `GS1_DIGITAL_LINK` formats.
Information Level[s]		Higher-level formats that say nothing about the physical tag length, nor include explicit information about the packaging/classification level. Information levels include `PURE_IDENTITY`, `ELEMENT_STRING`, `BARE_IDENTIFIER` and `GS1_DIGITAL_LINK` formats.
Input format		The format in which the identifier is supplied to the translation software. This may often be auto-detected from the input value.

Term	Defined / specified in	Meaning
Input value		The identifier to be translated. The format in which it is expressed is the input format.
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 input string. However, this approach fails when multiple options match the input 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 input and output levels.
Options		Variations to handle variable-length data partitions, such as those resulting from the variable-length GS1 Company Prefix in the GS1 family of EPC schemes. Where multiple options are specified, the same number of options should be specified for each format and translation should always translate from the matching option within the input format level to the corresponding option within the output format level.
Output format		The format in which the output from the translation software should be expressed. This must be specified by the client.
Physical Level[s]		Formats 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`, `TAG_ENCODING` formats.
prefixMatch		The prefixMatch is a substring which is used to determine the scheme of the input 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 prefixMatch matches at the start of the input value.
Regular Expression Pattern		Regular expression patterns are used for comparing string values with a defined pattern for the purposes of validation, as well as extraction of substrings that match patterns specified within capturing groups within the regular expression.
Rules		There are already a number of requirements to perform various string rearrangements 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
Serial Shipping Container Code (SSCC)	[GS1GS]	The GS1 identification key used to identify logistics units. The key comprises an extension digit, GS1 Company Prefix, serial reference and check digit.
Serialised		Provides a unique serial number for each unique object referenced using that EPC scheme
SGCN	[TDS]	Serialised Global Coupon Number (see GCN), including a mandatory serial number.

Term	Defined / specified in	Meaning
SGCN	[TDS]	Serialised Global Coupon Number (see GCN), supplemented by a mandatory serial number.
SGTIN	[TDS]	Serialised Global Trade Item Number. The SGTIN is used to assign a unique identity to a specific instance of a trade item.
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.
TDT Definition files	Provided in both XML and JSON formats at https://ref.gs1.org/standards/tdt/artefacts	A set of machine-readable data files that represent the patterns, grammar, rules, and field constraints for each identifier EPC scheme. Tag data translation software may periodically check for updated TDT definition files and TDT tables, which it can then use to update its own internal set of rules for performing the translations, whether this is done at run-time or compile-time.
URI	[RFC3986]	Uniform Resource Identifier, a compact sequence of characters that identifies an abstract or physical resource. URIs include both URNs, URLs and Web URIs. A Web URI is resolvable, whereas resolution of a URN is generally not well supported in a straightforward and uniform manner.
URN	[RFC8141]	Uniform Resource Name, a Uniform Resource Identifier (URI) that is assigned under the "urn" URI scheme and a particular URN namespace. Unlike a URL (Uniform Resource Locator) or Web URI, which may change when a web page moves from one website to another, a URN is intended to be a persistent reference, even if the underlying binding to a particular website address changes. A Web URI is resolvable, whereas resolution of a URN is generally not well supported in a straightforward and uniform manner.
USDOD	[TDS]	US Department of Defense identifier. The USDOD may be used to encode 96-bit Class 1 tags for shipping goods to the United States Department of Defense by a supplier who has already been assigned a CAGE (Commercial and Government Entity) code.

11 References

[ABNF] D. Crocker, "Augmented BNF for Syntax Specifications: ABNF", RFC5234, January 2008, https://www.rfc-editor.org/info/rfc5234

[GS1Arch] "The GS1 System Architecture", GS1 technical document, http://www.gs1.org/docs/gsmp/architecture/GS1_System_Architecture.pdf

[GCheckD] GS1 check digit calculator, GS1 online tool, https://www.gs1.org/services/check-digit-calculator

[GS1DL] GS1 Digital Link Standard: URI Syntax, https://www.gs1.org/standards/gs1-digital-link

[GS1GS] "GS1 General Specifications", GS1, https://www.gs1.org/standards/barcodes-epcrfid-id-keys/gs1-general-specifications

[ISO15962] ISO/IEC 15962, "Information technology – Radio frequency identification (RFID) for item management – Data protocol: data encoding rules and logical memory functions".

[PCRE] "PCRE – PERL-Compliant Regular Expressions", Philip Hazel, 2015, http://www.pcre.org

[RFC3986] T. Berners-Lee, "Uniform Resource Identifier (URI): Generic Syntax", RFC3986, January 2005, https://www.ietf.org/rfc/rfc3986

[RFC8141] P. Saint-Andre, "Uniform Resource Names (URNs)", RFC8141, April 2017, https://www.ietf.org/rfc/rfc8141

[TDS] GS1, "GS1 EPC Tag Data Standard" (TDS), GS1 standard, https://ref.gs1.org/standards/tds/

[UHFG2V2] EPCglobal, "EPC™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz, Release 2.1," GS1 Standard, July 2018, https://www.gs1.org/sites/default/files/docs/epc/gs1-epc-gen2v2-uhf-airinterface_i21_r_2018-09-04.pdf

[UML] "UML – Unified Modelling Language", Object Management Group, Inc., 2022, http://www.uml.org/

[USDOD] "United States Department of Defense Suppliers' Passive RFID Information Guide",

https://www.acq.osd.mil/log/LOG/.AIT.html/DoD_Suppliers_Passive_RFID_Info_Guide_v15update.pdf

12 Flowcharts to assist encoding and decoding GS1 Application Identifiers (within new EPC schemes and for additional AIDC data)

This chapter uses flowcharts to formally define the logic for encoding and decoding GS1 Application Identifiers, either within the new EPC schemes introduced in TDS 2.0 or for encoding/decoding additional AIDC data encoded after the EPC in such new EPC schemes.

Much of the logic is shared in both approaches. The main difference is that within the new EPC schemes introduced in TDS 2.0, the TDT definition file expresses (via `encodedAI`) which GS1 Application Identifiers should appear after the binary encoding of the filter value (and after the prioritised date field in the case of DSGTIN+), so in this situation,only the values of the GS1 Application Identifiers are encoded within the EPC and the corresponding GS1 Application Identifier keys are not encoded within the binary data. For example, in the SGTIN+ EPC scheme, the binary header value (11110111) effectively signals that the scheme is SGTIN+ and therefore the value of GTIN (01) and the value of Serial Number (21) will be encoded after the filter value, so there is no need to encode '01' or '21' within the binary string.

In contrast, for additional AIDC data encoded after the EPC identifier, the EPC scheme itself does not imply which GS1 Application Identifiers might be encoded afterwards, although care should be taken to respect the rules expressed within section 4.13.1 of the GS1 General Specifications regarding invalid pairings. For example, it would not be valid to encode (37) after an SGTIN+ EPC identifier because SGTIN+ expresses the values of (01) and (21) and section 4.13.1 of the GS1 General Specifications states that (01) and (37) is an invalid pairing, since (37) should be used in combination with SSCC (00) and contained GTIN (02) – so it would be acceptable for (37) and (02) to be encoded after the SSCC+ scheme, but not after SGTIN+, DSGTIN+ nor ITIP+.

TDS Table F defines the binary format for encoding GS1 Application Identifiers in the new EPC schemes and in AIDC data. It is provided in machine-readable format in TDT Table F. Most GS1 Application Identifiers are encoded in binary as a single component but there are a small number that are expressed using two components – typically a fixed-length first component (often numeric-only), followed by a second component that may be variable-length and possibly alphanumeric. This reflects how GS1 Application Identifiers are structured within the GS1 General Specifications, taking into account opportunities for more efficient encoding of fixed-length numeric components.

Section 12.1 provides flowcharts for the encoding process, while section 12.2 provides flowcharts for the decoding process.

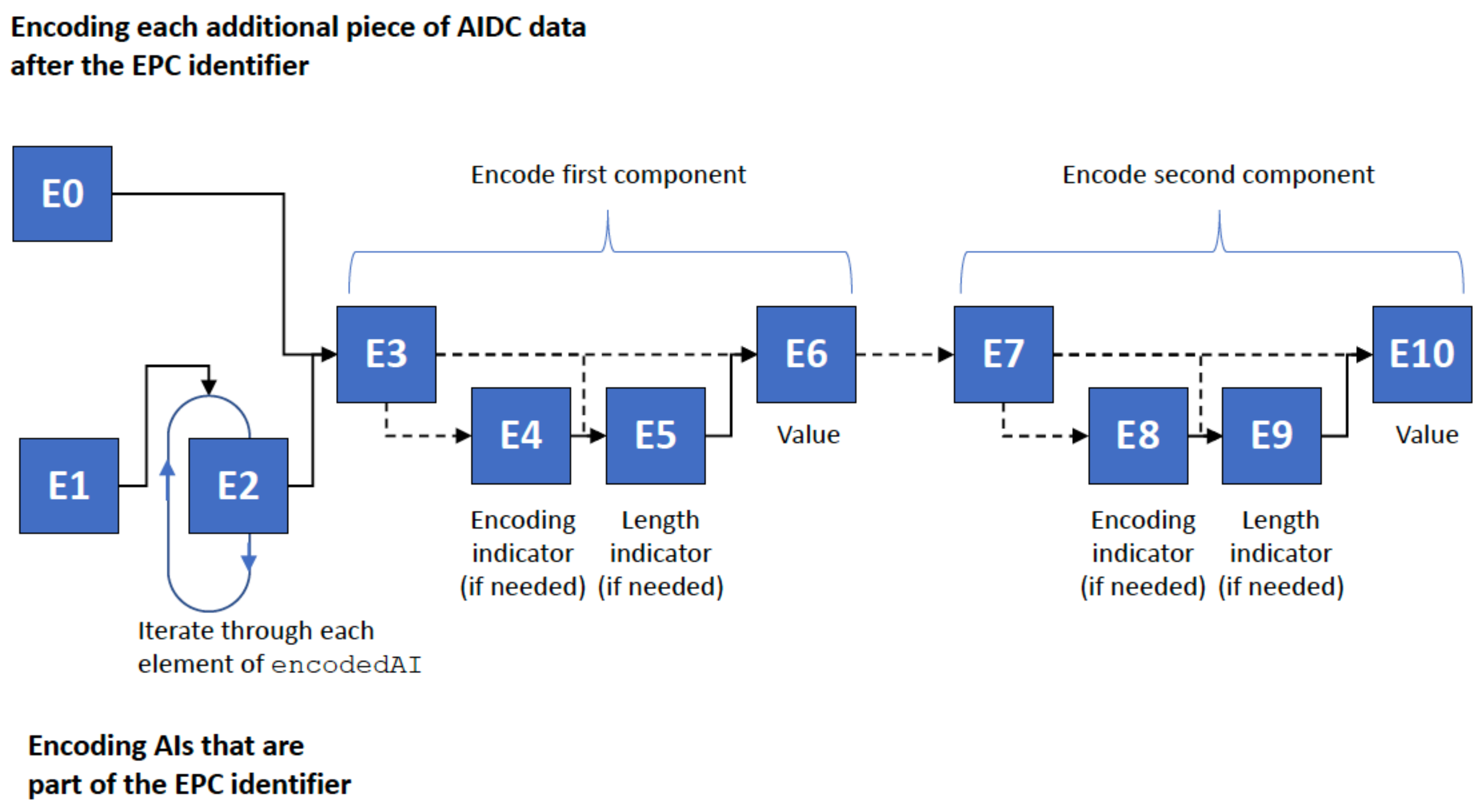
Both sections begin with a high-level flowchart showing the potential paths through the sequence of flowcharts, depending on whether the starting point was the encoding / decoding of additional AIDC data after the EPC identifier or whether the starting point was the encoding / decoding of GS1 Application Identifiers that are part of the EPC identifier.

Section 3.17 provides worked examples for encoding/decoding the values for GTIN (01) and for Serial Number (21) within the SGTIN+ EPC scheme. Chapter 4 provides worked examples for encoding/decodiing additional AIDC data after the binary encoding of any of the new EPC identifiers introduced in TDS 2.0.

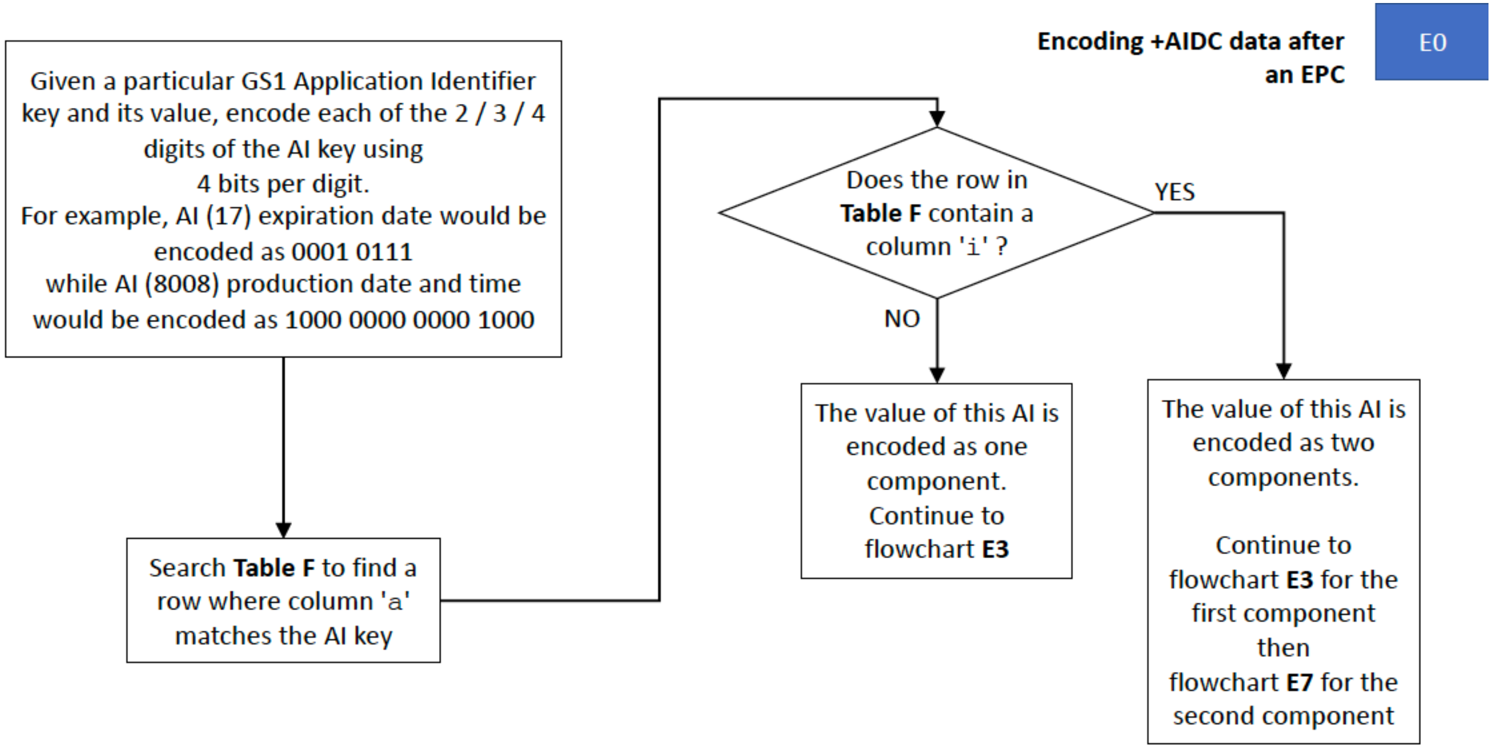
12.1 Encoding GS1 Application Identifiers

Figure 12-1 provides a high-level overview of the sequence of flowcharts for the encoding process.

Figure 12-1 Encoding each additional piece of AIDC data



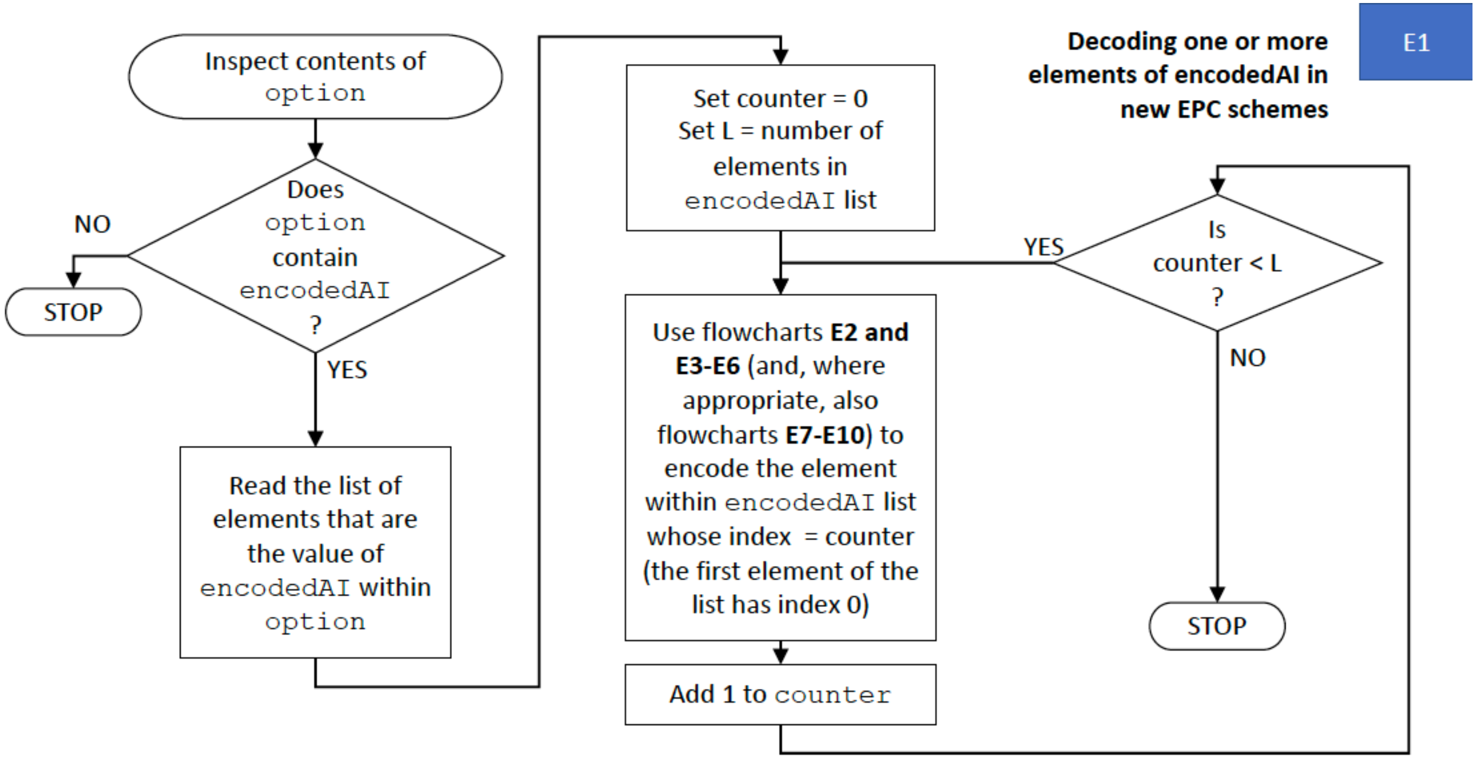
1808 **Figure 12-2** E0 - Encoding +AIDC data after an EPC



1809

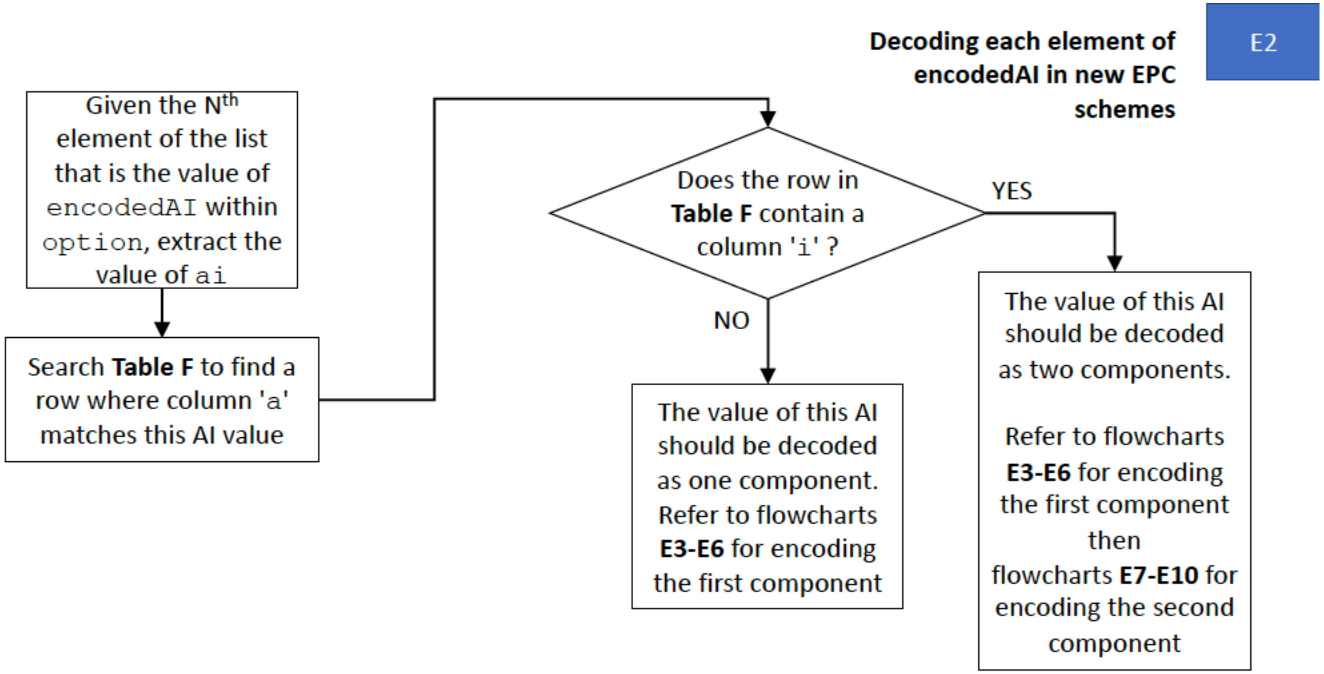
1810

1811 **Figure 12-3** E1 - Decoding one or more elements of encodedAI in new EPC schemes



1812

1813 **Figure 12-4** E2- Decoding each element of encodedAI in new EPC schemes



1814

Figure 12-5 E3 - Encoding the first component

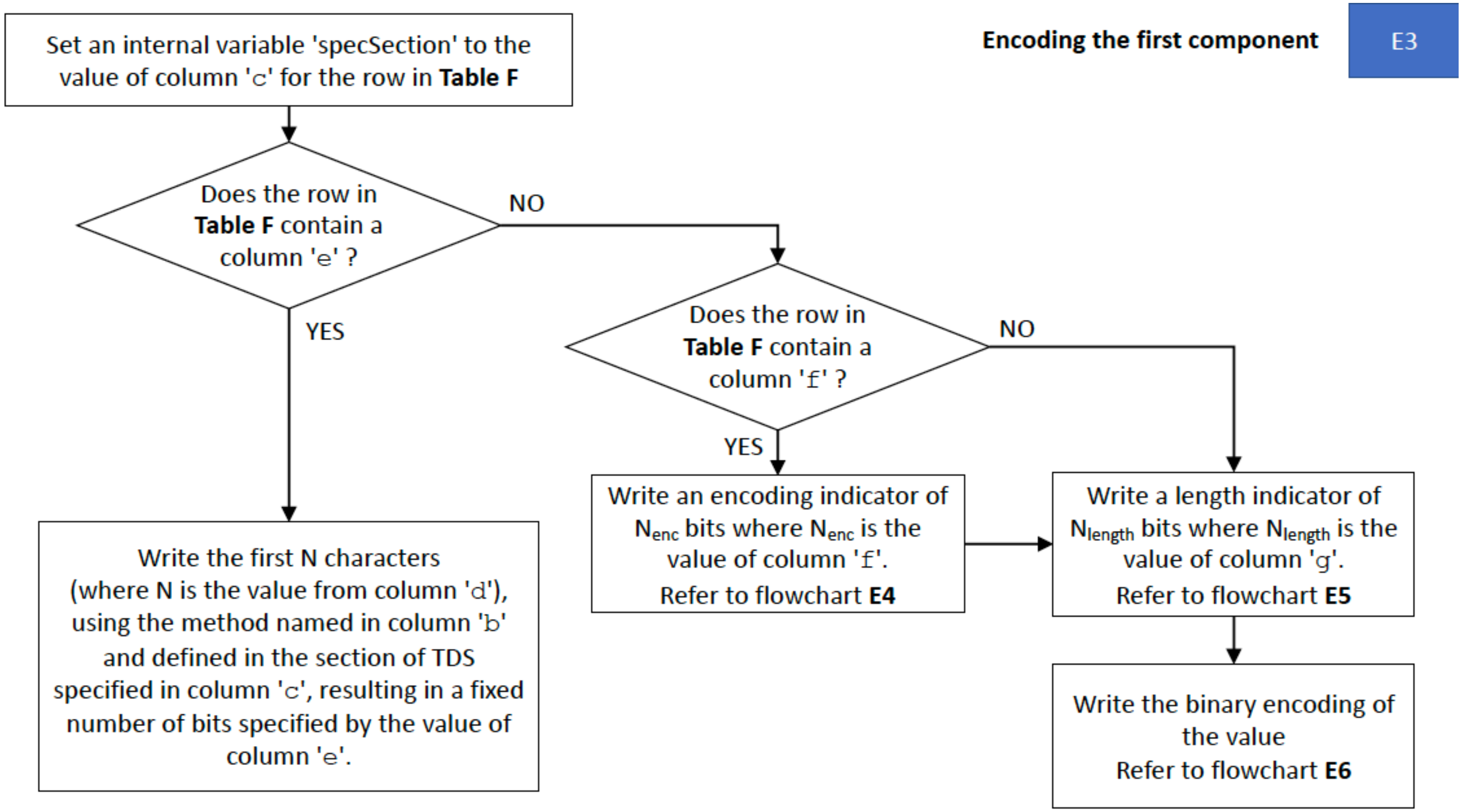


Figure 12-6 E4 - Encoding the encoding indicator for the first component

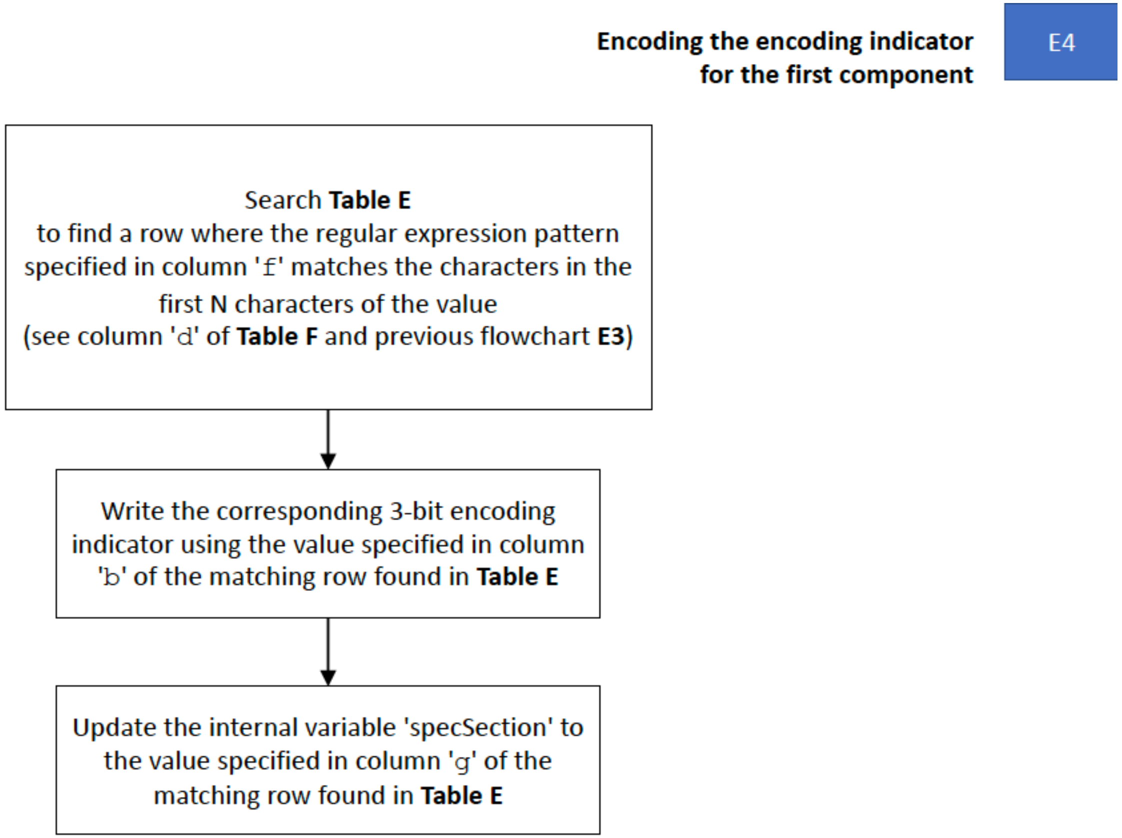


Figure 12-7 E5 - Encoding the length indicator for the first component

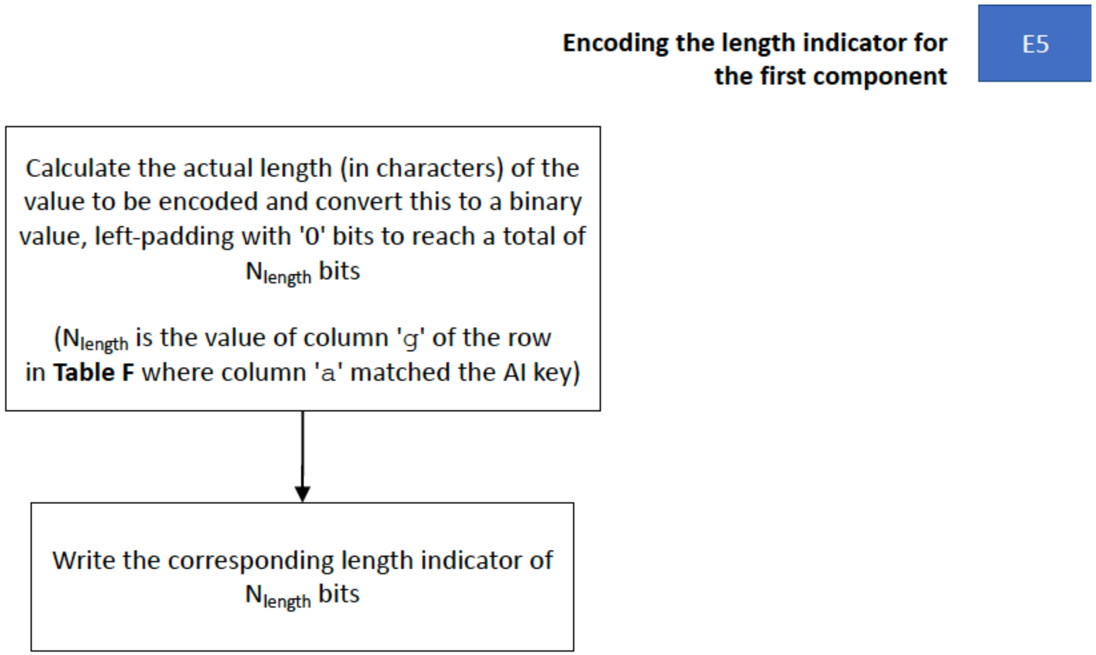
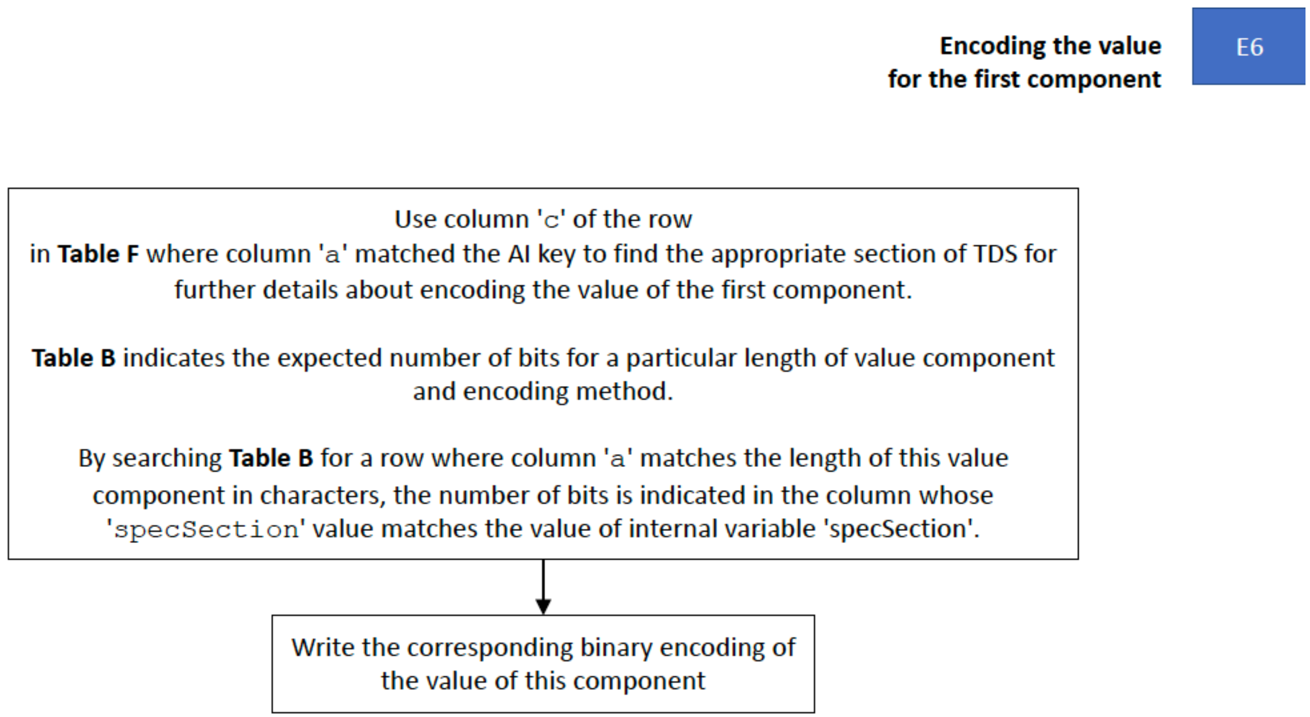
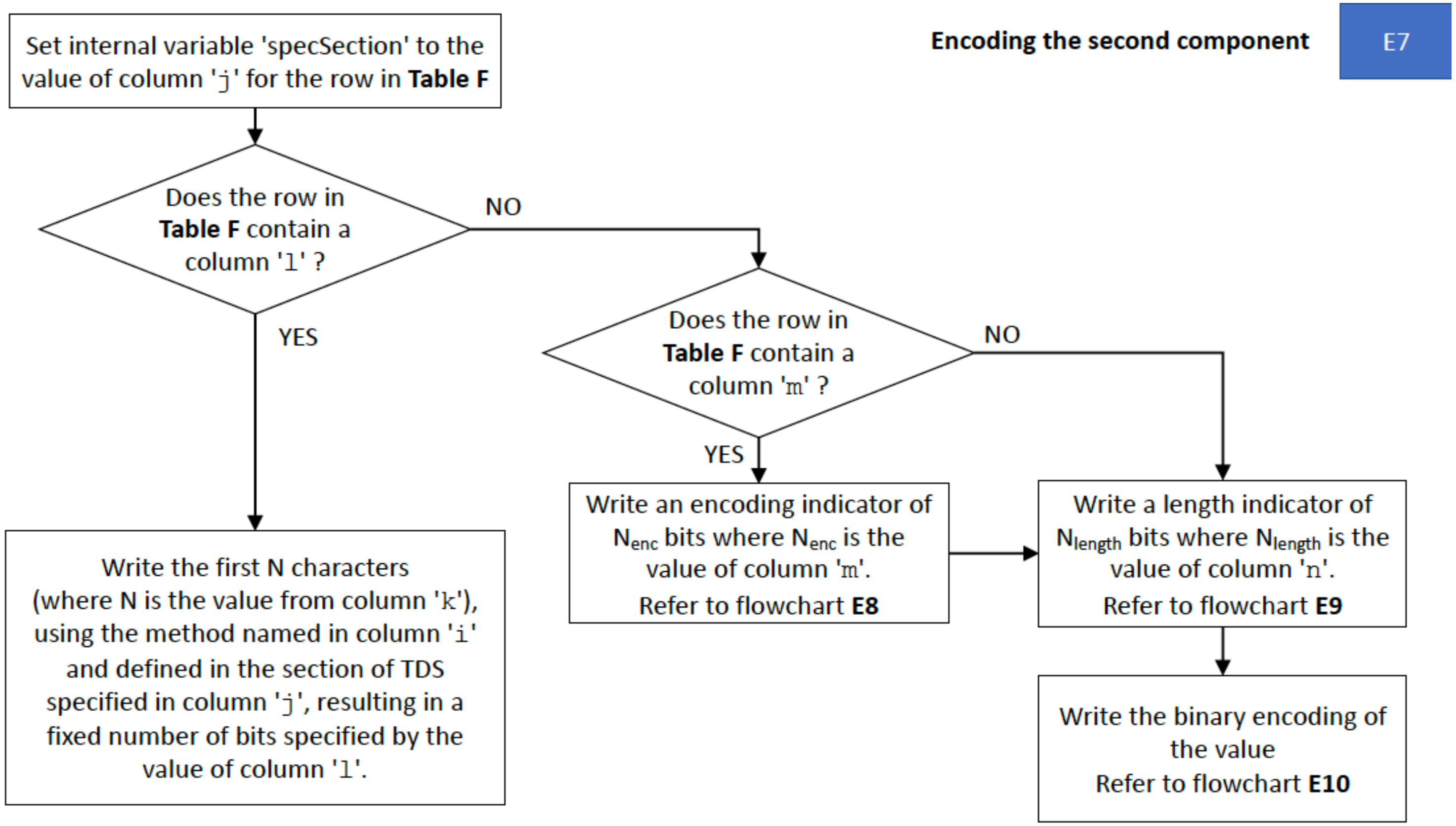


Figure 12-8 E6 - Encoding the value for the first component

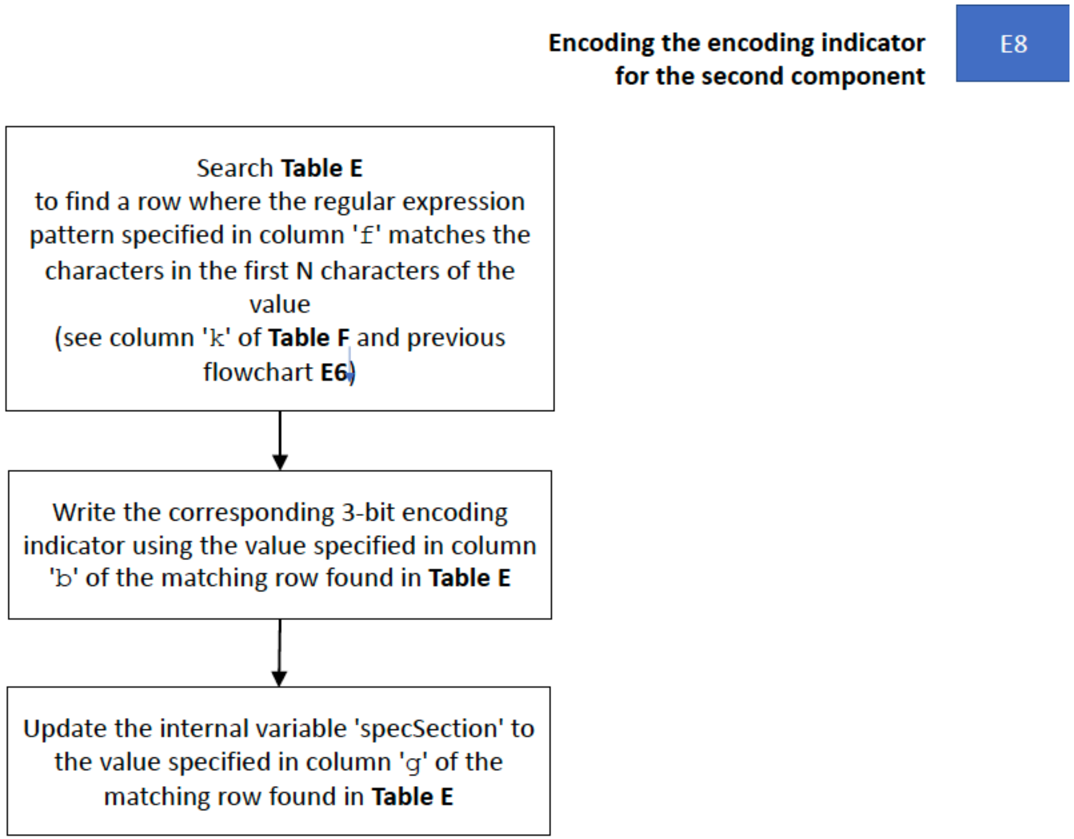


1823 **Figure 12-9** E7 - Encoding the second component

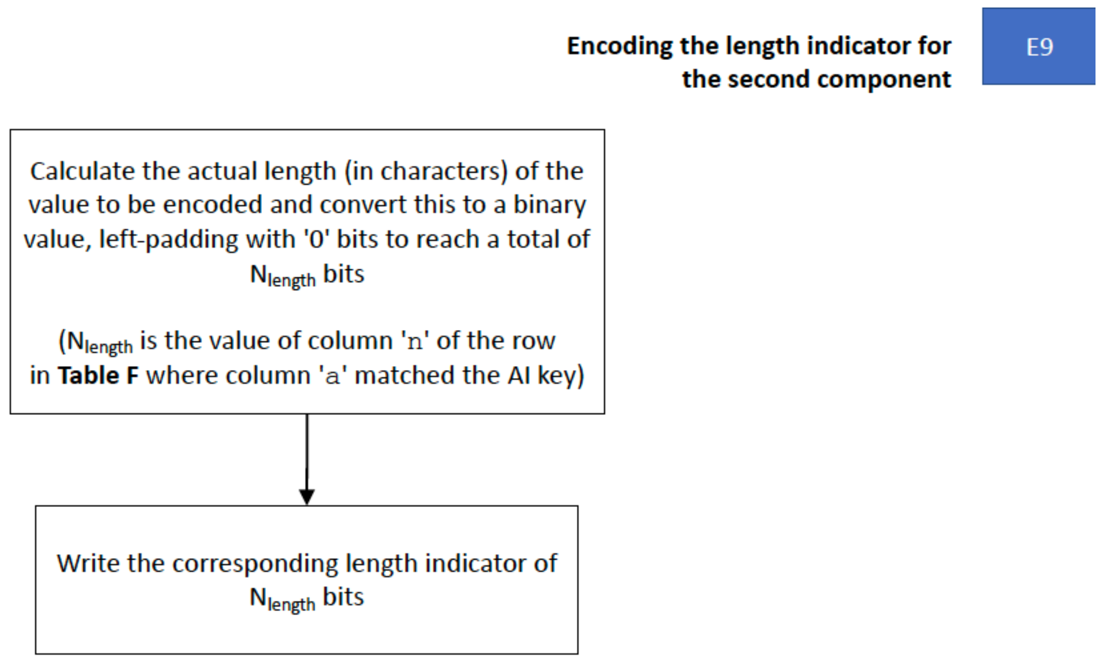


1824

Figure 12-10 E8 - Encoding the encoding indicator for the second component

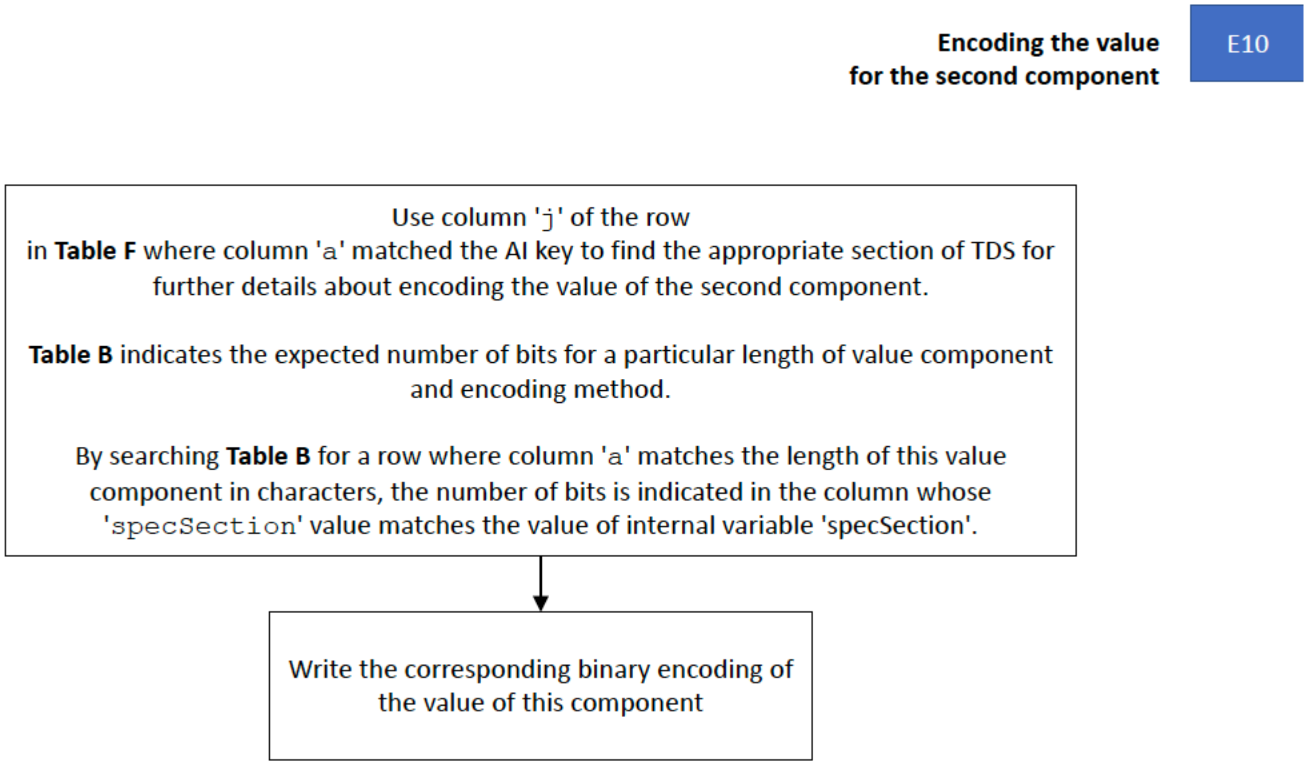


1827 **Figure 12-11** E9 - Encoding the length indicator for the second component



1828

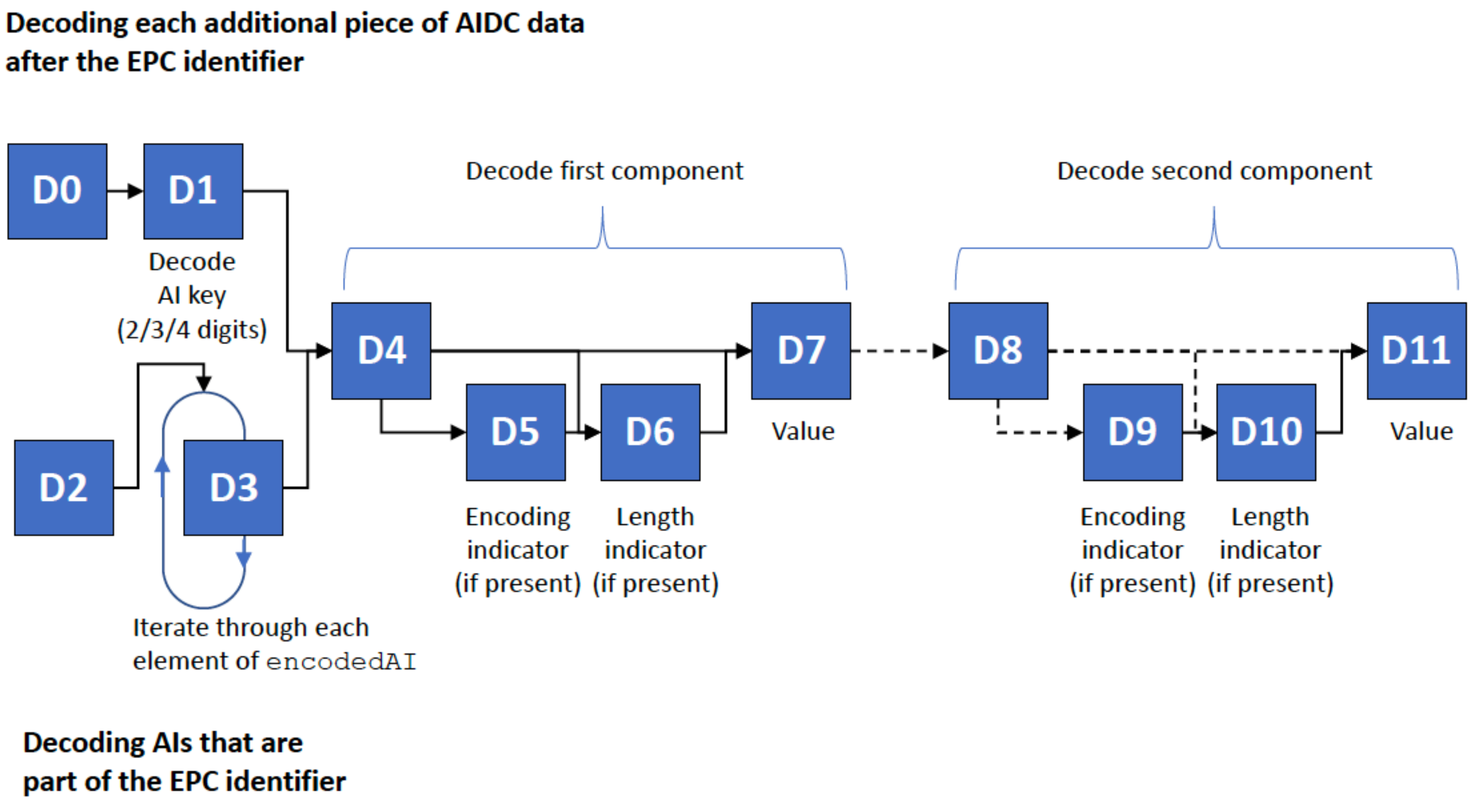
Figure 12-12 E10 - Encoding the value for the second component



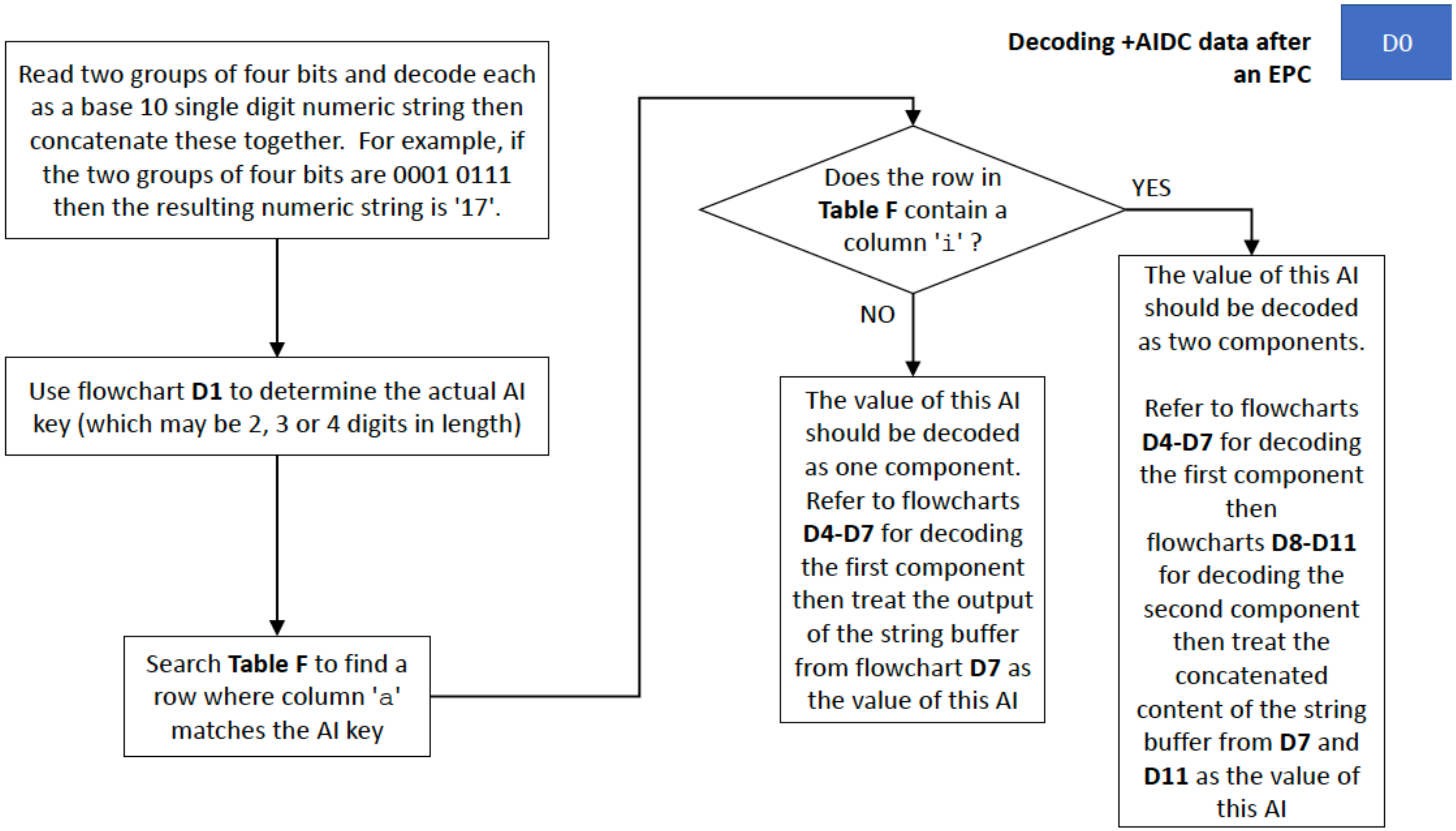
12.2 Decoding GS1 Application Identifiers

Figure 12-13 provides a high-level overview of the sequence of flowcharts for the decoding process.

Figure 12-13 Decoding each additional piece of AIDC data

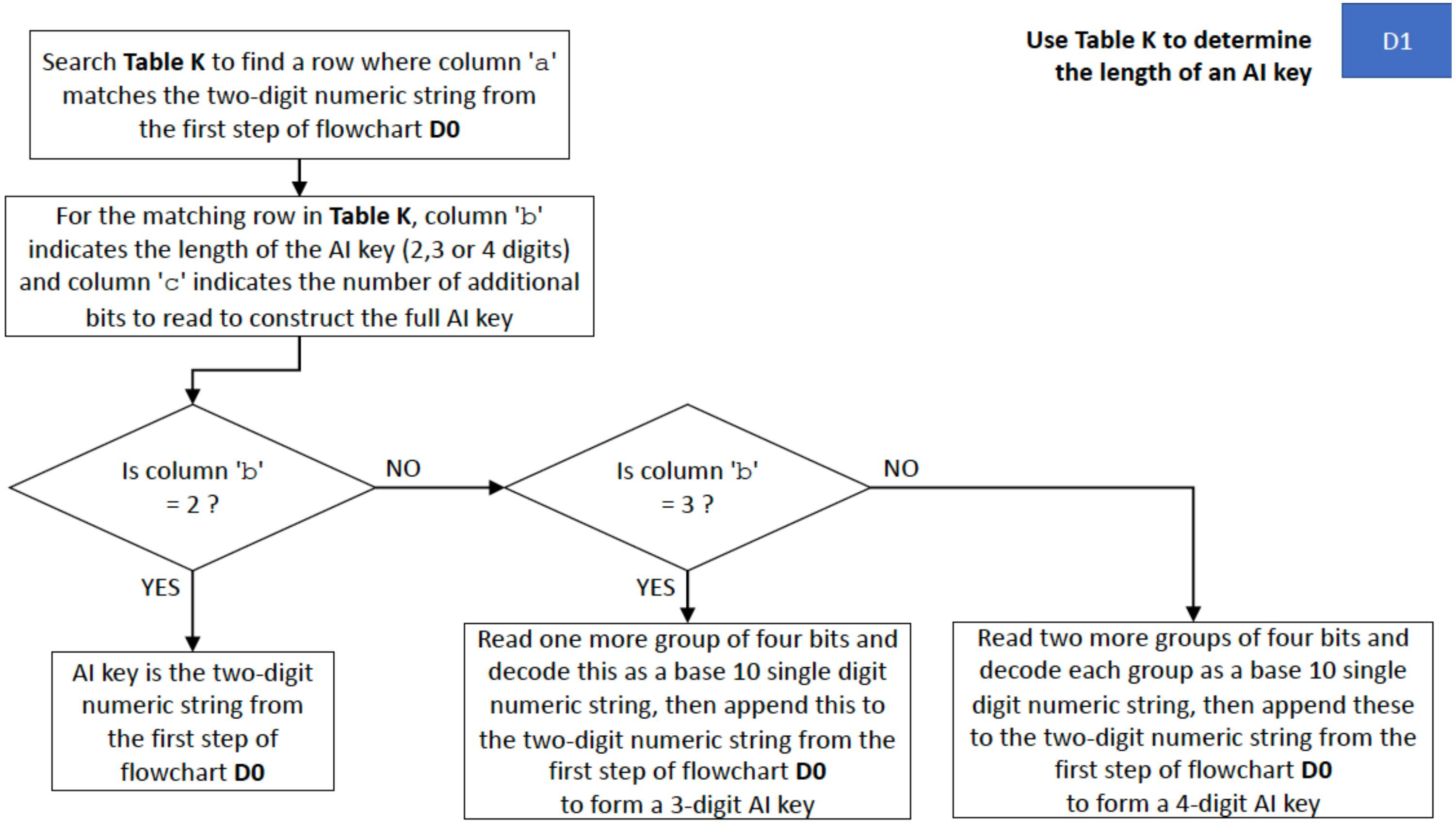


1837 **Figure 12-14** D0 - Decoding +AIDC data after an EPC



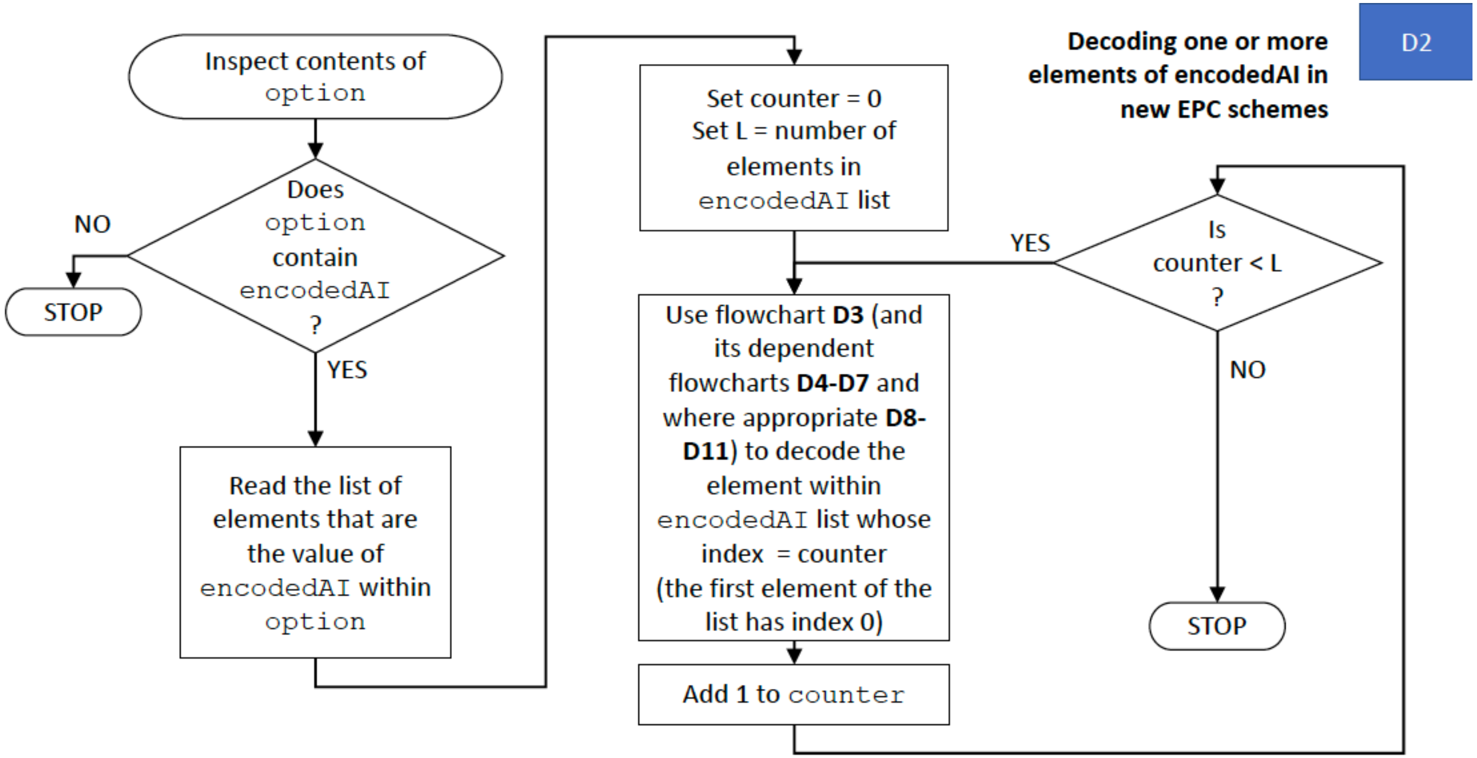
1838

1839 **Figure 12-15** D1 - Use Table K to determine the length of an AI key



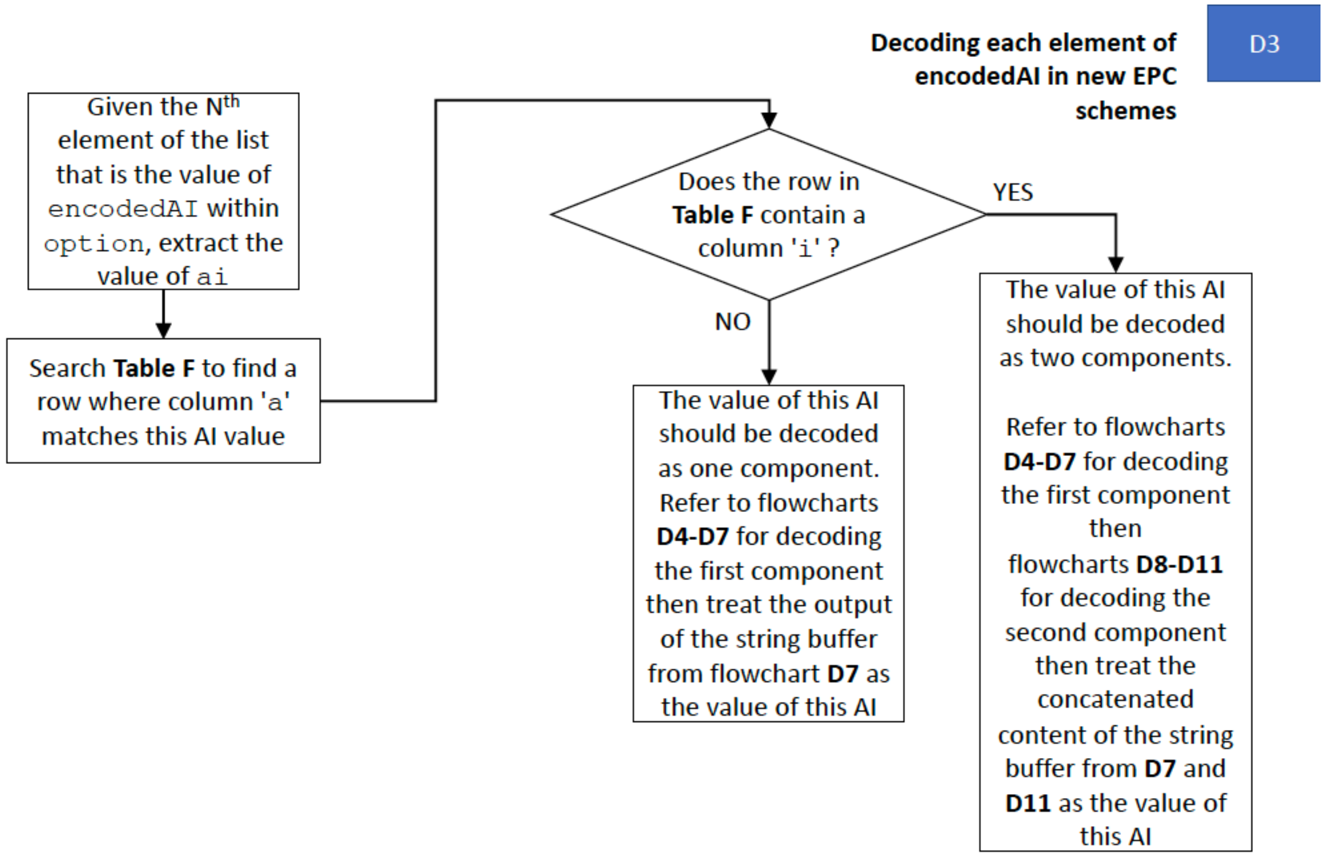
1840

1841 **Figure 12-16** D2 - Decoding one or more elements of encodedAI in new EPC schemes



1842

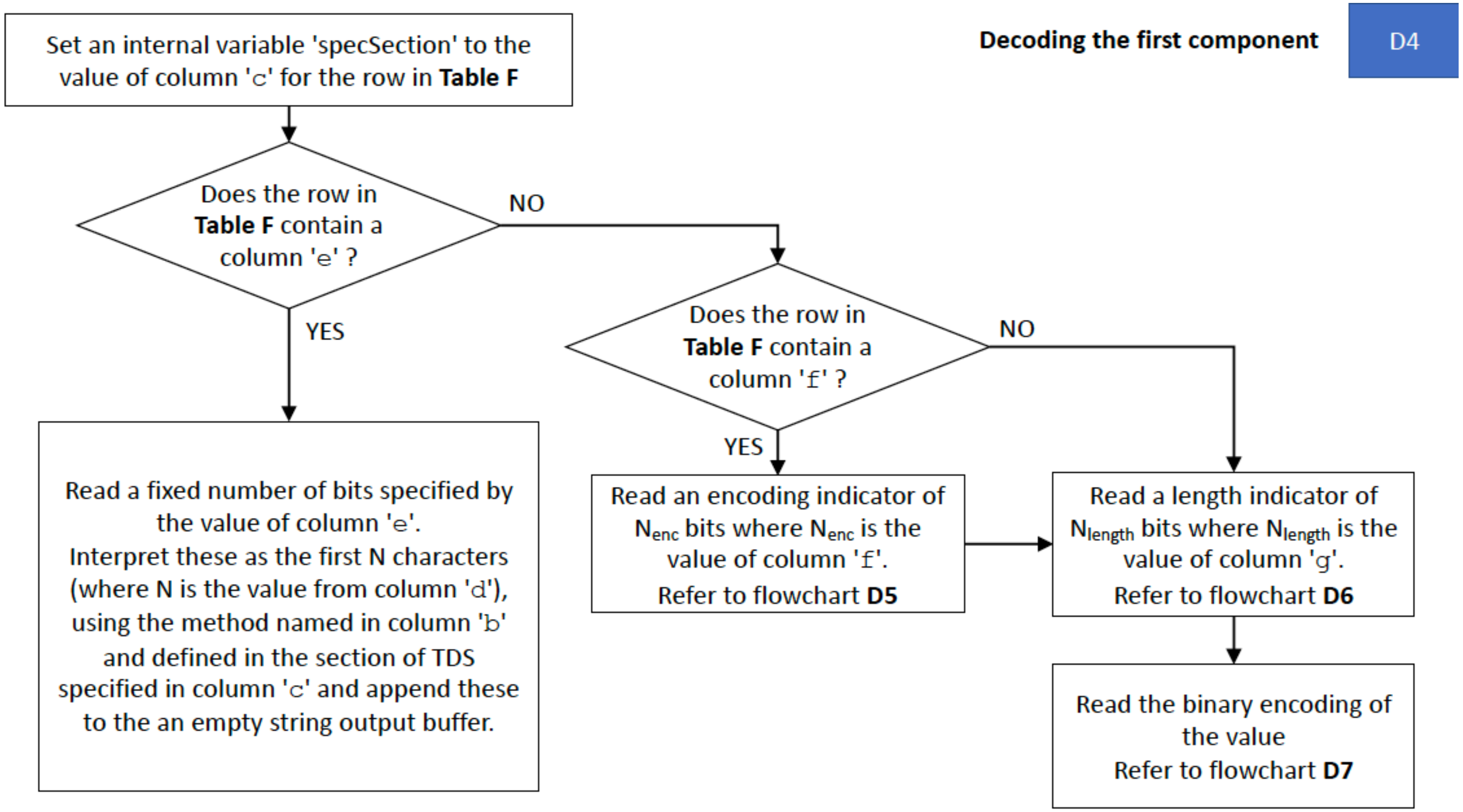
1843 **Figure 12-17** D3 - Decoding each element of encodedAI in new EPC schemes



1844

1845

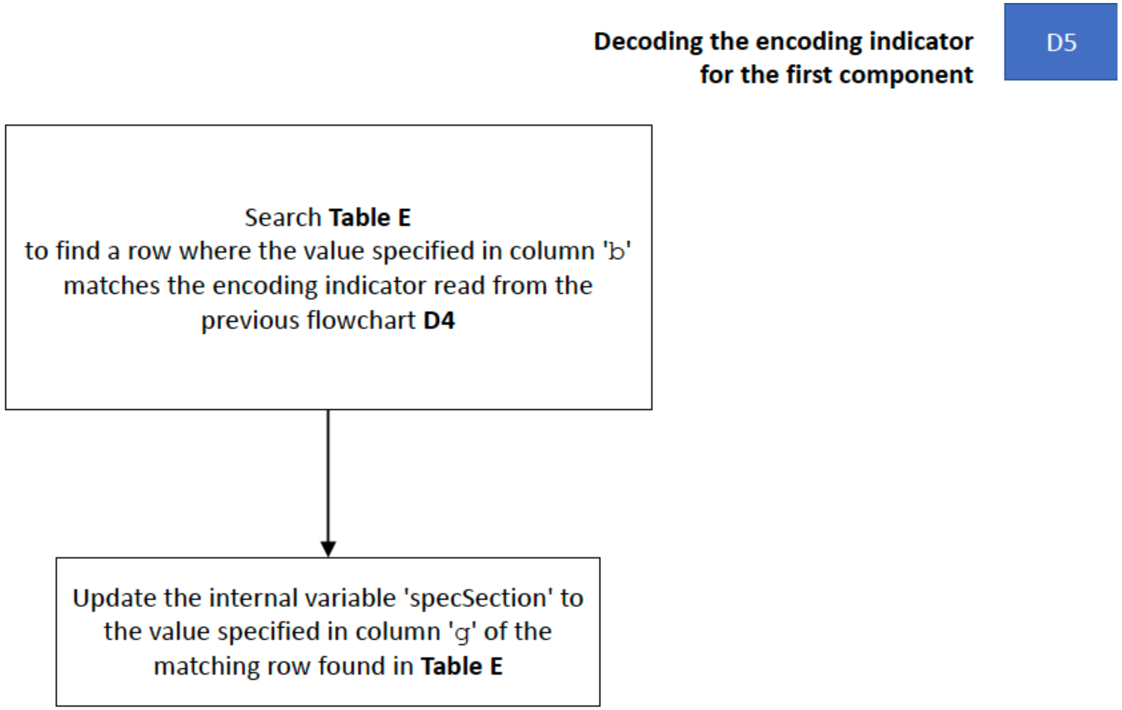
1846 **Figure 12-18** D4 - Decoding the first component



1847

1848

1849 **Figure 12-19** D5 - Decoding the encoding indicator for the first component



1850

Figure 12-20 D6 - Decoding the length indicator for the first component

Decoding the length indicator for the first component D6

Convert the length indicator read in flowchart **D4** to a base 10 integer value. This is the length of the first component in characters.

Table B indicates the number of bits that were used to encode a value component of a specific length using a specific encoding method.

By searching **Table B** for the '`id`' of a column whose '`specSection`' value matches the latest value of the internal variable 'specSection' from previous flowcharts **D4** or **D5**, the number of bits to be read n_b is specified in that column for the row whose value of column '`a`' matches the length of the first component in characters.

For example, if 'specSection' is ' 14.5.6.5' and the length of characters is 17, then a matching column of Table B is column '`e`' and the matching row is required to have a value of column '`a`' = 17. Reading the value of the matching column '`e`' for that row obtains a value of 96. i.e., 96 bits were used to encode a 17-character string using the basic URN Code 40 encoding method.

Figure 12-21 D7 - Decoding the value for the first component

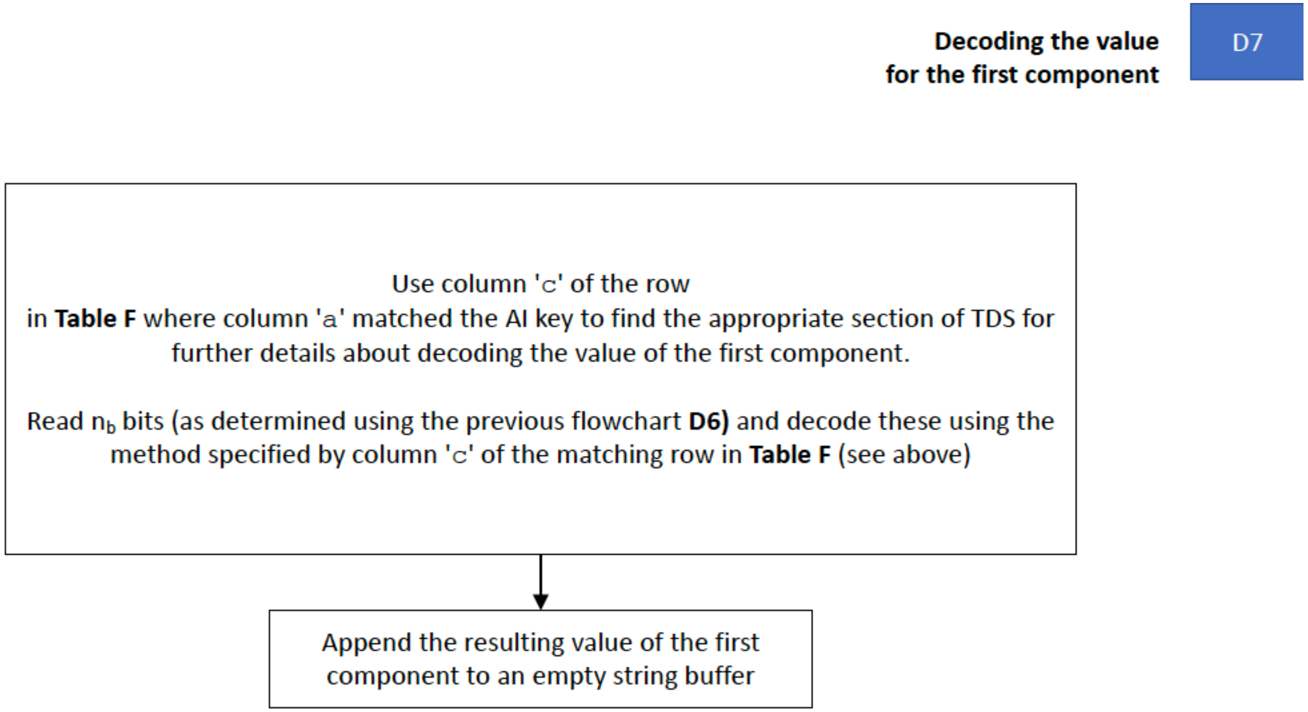


Figure 12-22 D8 - Decoding the second component

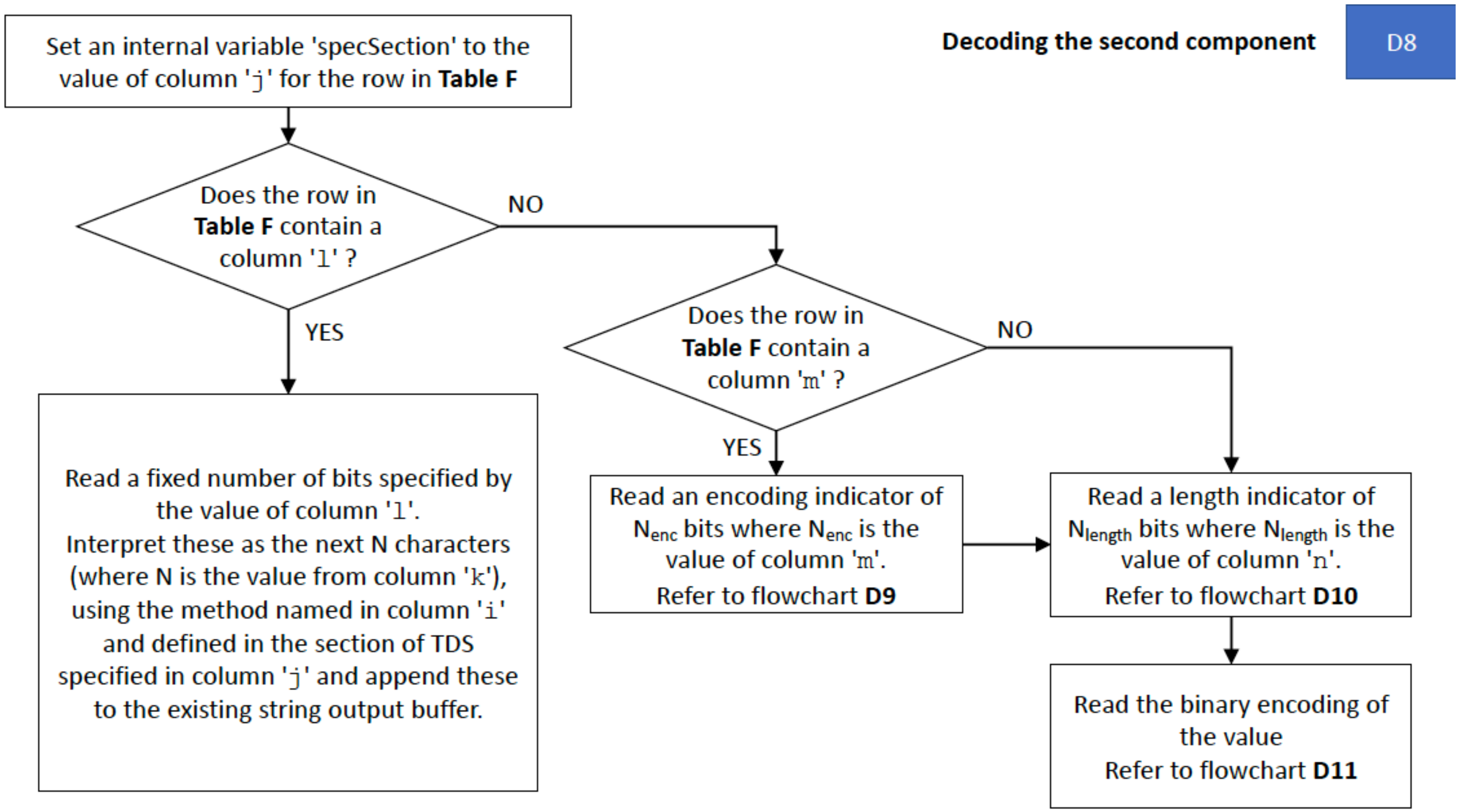


Figure 12-23 D9 - Decoding the encoding indicator for the second component

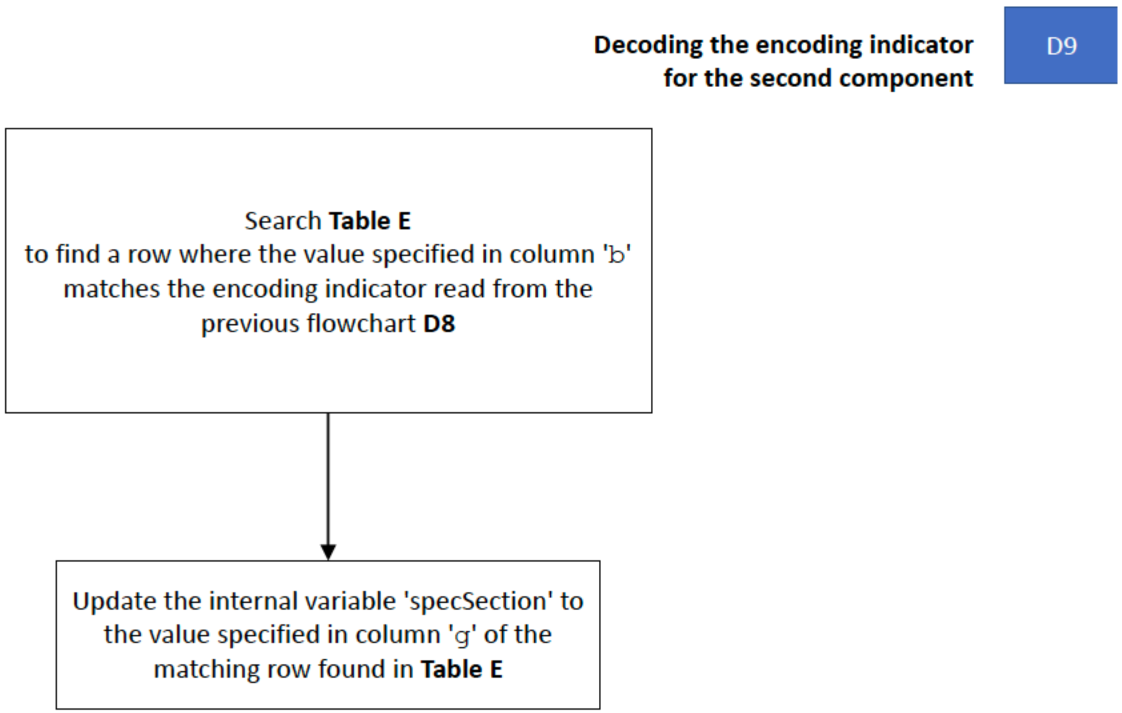


Figure 12-24 D10 - Decoding the length indicator for the second component

Decoding the length indicator for the second component

D10

Convert the length indicator read in flowchart **D8** to a base 10 integer value.
This is the length of the second component in characters.

Table B indicates the number of bits that were used to encode a value component of a specific length using a specific encoding method.

By searching **Table B** for the '`id`' of a column whose '`specSection`' value matches the latest value of the internal variable 'specSection' from previous flowcharts **D8** or **D9**, the number of bits to be read n_b is specified in that column for the row whose value of column '`a`' matches the length of the second component in characters.

For example, if 'specSection' is ' 14.5.6.5' and the length of characters is 17, then a matching column of Table B is column '`e`' and the matching row is required to have a value of column '`a`' = 17. Reading the value of the matching column '`e`' for that row obtains a value of 96. i.e., 96 bits were used to encode a 17-character string using the basic URN Code 40 encoding method.

Figure 12-25 D11 - Decoding the value for the second component