



EPC Tag Data Standard TDS

defines the Electronic Product Code™ and specifies the memory contents of Gen 2 RFID Tags

Release 2.1, Ratified, Feb 2024



4 Document Summary

| Document Item | Current Value |
|----------------------|---|
| Document Name | EPC Tag Data Standard TDS |
| Document Date | Feb 2024 |
| Document Version | 2.1 |
| Document Issue | |
| Document Status | Ratified |
| Document Description | defines the Electronic Product Code™ and specifies the memory contents of Gen 2 RFID Tags |

5 Contributors

| Name | Organisation |
|-------------------------|---|
| Jaewook Byun | Auto-ID Labs at KAIST |
| Jin Mitsugi | Auto-ID Labs at Keio University |
| HJ Cha | Avery Dennison RFID |
| Jeanne Duckett | 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 | Charming RFID |
| Josef Preishuber-Pflügl | CISC Semiconductor GmbH |
| François-Régis Dousset | DANONE SPA |
| Olivier Joyez | DECATHLON |
| Michael Isabell | CCL eAgile |
| Jim Springer | EM Microelectronic |
| Odarci Maia Junior | EMPRESA BRASILEIRA DE CORREIOS E TELEGRAFOS |
| Julie McGill | FoodLogiQ |
| Guilda Javaheri | Golden State Foods |
| 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 |
| Zhimin Li | GS1 China |
| Gao Peng | GS1 China |
| Yi Wang | GS1 China |

| Name | Organisation |
|----------------------|--------------------------|
| Ruoyun Yan | GS1 China |
| Marisa Lu | GS1 Chinese Taipei |
| Sandra Hohenecker | GS1 Germany |
| Roman Winter | GS1 Germany |
| 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 |
| Mohit Tomar | 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 |
| Andrew Meyer | GS1 US |
| Gena Morgan | GS1 US |
| Amber Walls | GS1 US |
| Megan Brewster | Impinj, Inc |
| Shinichi Ike | Johnson & Johnson |
| Blair Korman | Johnson & Johnson |
| Fabian Moritz Schenk | Lambda ID GmbH |
| Don Ferguson | Lyngsoe Systems Ltd. |
| Mark Harrison | Milecastle Media Limited |
| Danny Haak | Nedap |
| Chris Brown | Printronic Auto ID |
| Jeffrey Chen | Printronic Auto ID |

| Name | Organisation |
|--------------------|---------------------------------|
| Marisa Campos | PROAGRIND, Lda. |
| Akshay Koshti | Robert Bosch GmbH |
| Holly Mitchell | Seagull Scientific |
| Mo Ramzan | SML |
| Jerome Torro | SNCF Rolling Stock Department |
| Albertus Pretorius | Tonnjes ISI Patent Holding GmbH |
| Masatoshi Oka | TOPPAN |
| Taira Wakamiya | TOPPAN |
| Elizabeth Waldorf | TraceLink |

6 Log of Changes

| Release | Date of Change | Changed By | Summary of Change |
|---------|----------------|------------------------------------|--|
| 1.9.1 | 8 July 2015 | D. Buckley | New GS1 branding applied |
| 1.10 | Mar 2017 | Craig Alan Repec | Listed in full in the Abstract below |
| 1.11 | Sep 2017 | Craig Alan Repec | Listed in full in the Abstract below |
| 1.12 | April 2019 | Craig Alan Repec and Mark Harrison | <p>WR 19-076</p> <p>Added EPC URI for UPUI, to support EU 2018/574, as well as EPC URI for PGLN – GLN of Party AI (417) – in accordance with GS1 General Specifications 19.1;</p> <p>Added normative specifications around handling of GCP length for individually assigned GS1 Keys;</p> <p>Corrected ITIP pure identity pattern syntax;</p> <p>Introduced "Fixed Width Integer" encoding and decoding sections in support of ITIP binary encoding.</p> |
| 1.13 | September 2019 | Craig Alan Repec | <p>WR 19-262 Added IMOVN EPC for IMO Vessel Number;</p> <p>WR 19-264 corrected GSIN syntax erratum in section 6.3.12;</p> <p>corrected UPUI example erratum in section 7.16.</p> |
| 2.0 | Aug 2022 | Mark Harrison and Craig Alan Repec | <p>Major release; see comprehensive summary of changes in the "<i>Differences from EPC Tag Data Standard (TDS) Version 1.13</i>" section, immediately preceding section 1.</p> <p>Note that TDS will be updated as necessary to harmonise with GS1's Gen2 v3 Air Interface Protocol, once that standard has been published.</p> |

| Release | Date of Change | Changed By | Summary of Change |
|---------|----------------|------------------------------------|--|
| 2.1 | Feb 2024 | Mark Harrison and Craig Alan Repec | <p>Update to correct minor errors and errata in version 2.0.</p> <p>Updated URI grammar in sections 12 and 13.</p> <p>Clarified use of ISO/IEC 20248 DigSig, using GS1 AI (8030), in section 17.</p> <p>Updated section 9.2, including Figure 9-1 and Table 9-2, to reflect encoding of ISO/IEC 20248 DigSig in User Memory.</p> <p>Updated section 9.3, Figure 9-2 and Table 9-3 to reflect the Read User Memory (RUM) indicator specified in Gen2v3.</p> <p>Updated Table 9-4 to reflect Gen2v3 assignments to bits 214h-217h of XPC.</p> <p>Updated section 16 to reflect mandatory serialisation of TID specified in Gen2v3.</p> <p>Also added support for AIs (7241), (7242), (8030), (4330), (4331), (4332), (4333) and (7011).</p> <p>Additionally, the <i>Packed Objects ID Table for Data Format 9</i> in Section F.2 has been supplemented with an external, normative artefact in CSV format.</p> |

7 **Disclaimer**

8 GS1®, under its IP Policy, seeks to avoid uncertainty regarding intellectual property claims by requiring the participants in
 9 the Work Group that developed this **EPC Tag Data Standard TDS** to agree to grant to GS1 members a royalty-free licence
 10 or a RAND licence to Necessary Claims, as that term is defined in the GS1 IP Policy. Furthermore, attention is drawn to the
 11 possibility that an implementation of one or more features of this Specification may be the subject of a patent or other
 12 intellectual property right that does not involve a Necessary Claim. Any such patent or other intellectual property right is
 13 not subject to the licensing obligations of GS1. Moreover, the agreement to grant licences provided under the GS1 IP Policy
 14 does not include IP rights and any claims of third parties who were not participants in the Work Group.

15 Accordingly, GS1 recommends that any organisation developing an implementation designed to be in conformance with this
 16 Specification should determine whether there are any patents that may encompass a specific implementation that the
 17 organisation is developing in compliance with the Specification and whether a licence under a patent or other intellectual
 18 property right is needed. Such a determination of a need for licensing should be made in view of the details of the specific
 19 system designed by the organisation in consultation with their own patent counsel.

20 THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF
 21 MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR PARTICULAR PURPOSE, OR ANY WARRANTY OTHER WISE ARISING
 22 OUT OF THIS DOCUMENT. GS1 disclaims all liability for any damages arising from use or misuse of this document, whether
 23 special, indirect, consequential, or compensatory damages, and including liability for infringement of any intellectual
 24 property rights, relating to use of information in or reliance upon this document.

25 GS1 retains the right to make changes to this document at any time, without notice. GS1 makes no warranty for the use of
 26 this document and assumes no responsibility for any errors which may appear in the document, nor does it make a
 27 commitment to update the information contained herein.

28 GS1 and the GS1 logo are registered trademarks of GS1 AISBL.

Table of Contents

29

30 **Foreword..... 17**

31 **1 Introduction 24**

32 **2 Terminology and typographical conventions..... 25**

33 **3 Overview of TDS 25**

34 **4 The Electronic Product Code: A universal identifier for physical objects 28**

35 4.1 The need for a universal identifier: an example..... 29

36 4.2 Use of identifiers in a Business Data Context 29

37 4.3 Relationship between EPCs and GS1 keys..... 31

38 4.4 Use of the EPC in the GS1 System Architecture..... 34

39 **5 Common grammar elements 36**

40 **6 EPC URI 37**

41 6.1 Use of the EPC URI..... 38

42 6.2 Assignment of EPCs to physical objects..... 38

43 6.3 EPC URI syntax..... 39

44 6.3.1 Serialised Global Trade Item Number (SGTIN)..... 41

45 6.3.2 Serial Shipping Container Code (SSCC) 41

46 6.3.3 Global Location Number With or Without Extension (SGLN)..... 42

47 6.3.4 Global Returnable Asset Identifier (GRAI) 42

48 6.3.5 Global Individual Asset Identifier (GIAI) 43

49 6.3.6 Global Service Relation Number – Recipient (GSRN)..... 44

50 6.3.7 Global Service Relation Number – Provider (GSRNP) 44

51 6.3.8 Global Document Type Identifier (GDTI) 44

52 6.3.9 Component / Part Identifier (CPI) 45

53 6.3.10 Serialised Global Coupon Number (SGCN)..... 46

54 6.3.11 Global Identification Number for Consignment (GINC) 46

55 6.3.12 Global Shipment Identification Number (GSIN)..... 47

56 6.3.13 Individual Trade Item Piece (ITIP) 47

57 6.3.14 Unit Pack Identifier (UPUI) 48

58 6.3.15 Global Location Number of Party (PGLN)..... 49

59 6.3.16 General Identifier (GID)..... 49

60 6.3.17 US Department of Defense Identifier (DOD)..... 49

61 6.3.18 Aerospace and Defense Identifier (ADI) 50

62 6.3.19 BIC Container Code (BIC) 51

63 6.3.20 IMO Vessel Number (IMOVN) 52

64 6.4 EPC Class URI Syntax 53

65 6.4.1 GTIN + Batch/Lot (LGTIN) 53

66 **7 Correspondence between EPCs and GS1 Keys..... 54**

67 7.1 The GS1 Company Prefix (GCP) in EPC encodings 54

68 7.2 Determining length of the EPC CompanyPrefix component for individually assigned GS1 Keys 54

69 7.2.1 Individually assigned GTINs 54

| | | | |
|-----|-----------|---|-----------|
| 70 | 7.2.2 | Individually assigned GLNs..... | 55 |
| 71 | 7.2.3 | Other individually assigned GS1 Keys..... | 55 |
| 72 | 7.3 | Serialised Global Trade Item Number (SGTIN)..... | 56 |
| 73 | 7.3.1 | GTIN-12 and GTIN-13..... | 57 |
| 74 | 7.3.2 | GTIN-8..... | 57 |
| 75 | 7.3.3 | RCN-8..... | 58 |
| 76 | 7.3.4 | Company Internal Numbering (GS1 Prefixes 04 and 0001 – 0007)..... | 58 |
| 77 | 7.3.5 | Restricted Circulation (GS1 Prefixes 02 and 20 – 29)..... | 58 |
| 78 | 7.3.6 | Coupon Code Identification for Restricted Distribution (GS1 Prefixes 981-984 and 99)... | 58 |
| 79 | 7.3.7 | Refund Receipt (GS1 Prefix 980)..... | 58 |
| 80 | 7.3.8 | ISBN, ISMN, and ISSN (GS1 Prefixes 977, 978, or 979)..... | 58 |
| 81 | 7.4 | Serial Shipping Container Code (SSCC)..... | 59 |
| 82 | 7.5 | Global Location Number With or Without Extension (SGLN)..... | 60 |
| 83 | 7.6 | Global Returnable Asset Identifier (GRAI)..... | 62 |
| 84 | 7.7 | Global Individual Asset Identifier (GIAI)..... | 64 |
| 85 | 7.8 | Global Service Relation Number – Recipient (GSRN)..... | 65 |
| 86 | 7.9 | Global Service Relation Number – Provider (GSRNP)..... | 66 |
| 87 | 7.10 | Global Document Type Identifier (GDTI)..... | 67 |
| 88 | 7.11 | Component and Part Identifier (CPI)..... | 68 |
| 89 | 7.12 | Serialised Global Coupon Number (SGCN)..... | 69 |
| 90 | 7.13 | Global Identification Number for Consignment (GINC)..... | 70 |
| 91 | 7.14 | Global Shipment Identification Number (GSIN)..... | 71 |
| 92 | 7.15 | Individual Trade Item Piece (ITIP)..... | 72 |
| 93 | 7.16 | Unit Pack Identifier (UPUI)..... | 73 |
| 94 | 7.17 | Global Location Number of Party (PGLN)..... | 74 |
| 95 | 7.18 | GTIN + batch/lot (LGTIN)..... | 75 |
| 96 | 8 | URIs for EPC Pure identity patterns..... | 77 |
| 97 | 8.1 | Syntax..... | 77 |
| 98 | 8.2 | Semantics..... | 80 |
| 99 | 9 | Memory Organisation of Gen 2 RFID tags..... | 80 |
| 100 | 9.1 | Types of Tag Data..... | 80 |
| 101 | 9.2 | Gen 2 Tag Memory Map..... | 81 |
| 102 | 9.3 | PC bits..... | 86 |
| 103 | 9.4 | XPC bits..... | 88 |
| 104 | 10 | Filter Value..... | 89 |
| 105 | 10.1 | Use of "Reserved" and "All Others" Filter Values..... | 90 |
| 106 | 10.2 | Filter Values for SGTIN and DSGTIN+ EPC Tags..... | 90 |
| 107 | 10.3 | Filter Values for SSCC EPC Tags..... | 90 |
| 108 | 10.4 | Filter Values for SGLN EPC Tags..... | 91 |
| 109 | 10.5 | Filter Values for GRAI EPC Tags..... | 91 |
| 110 | 10.6 | Filter Values for GIAI EPC Tags..... | 91 |
| 111 | 10.7 | Filter Values for GSRN and GSRNP EPC Tags..... | 92 |
| 112 | 10.8 | Filter Values for GDTI EPC Tags..... | 92 |
| 113 | 10.9 | Filter Values for CPI EPC Tags..... | 92 |
| 114 | 10.10 | Filter Values for SGCN EPC Tags..... | 93 |
| 115 | 10.11 | Filter Values for ITIP EPC Tags..... | 93 |

| | | | |
|-----|-----------|---|------------|
| 116 | 10.12 | Filter Values for GID EPC Tags | 93 |
| 117 | 10.13 | Filter Values for DOD EPC Tags | 93 |
| 118 | 10.14 | Filter Values for ADI EPC Tags | 93 |
| 119 | 11 | Attribute bits (refer to 9.3 and 9.4)..... | 95 |
| 120 | 12 | EPC Tag URI and EPC Raw URI | 95 |
| 121 | 12.1 | Structure of the EPC Tag URI and EPC Raw URI | 95 |
| 122 | 12.2 | Control Information | 96 |
| 123 | 12.2.1 | Filter Values | 97 |
| 124 | 12.2.2 | Other control information fields | 97 |
| 125 | 12.3 | EPC Tag URI and EPC Pure Identity URI | 98 |
| 126 | 12.3.1 | EPC Binary Coding Schemes..... | 98 |
| 127 | 12.3.2 | EPC Pure Identity URI to EPC Tag URI | 101 |
| 128 | 12.3.3 | EPC Tag URI to EPC Pure Identity URI | 102 |
| 129 | 12.4 | Grammar | 102 |
| 130 | 13 | URIs for EPC Tag Encoding patterns..... | 103 |
| 131 | 13.1 | Syntax..... | 104 |
| 132 | 13.2 | Semantics | 106 |
| 133 | 14 | EPC Binary Encoding | 106 |
| 134 | 14.1 | Overview of Binary Encoding | 107 |
| 135 | 14.2 | EPC Binary Headers..... | 107 |
| 136 | 14.3 | Encoding procedure..... | 110 |
| 137 | 14.3.1 | "Integer" Encoding Method | 110 |
| 138 | 14.3.2 | "String" Encoding method | 111 |
| 139 | 14.3.3 | "Partition Table" Encoding method | 111 |
| 140 | 14.3.4 | "Unpadded Partition Table" Encoding method | 112 |
| 141 | 14.3.5 | "String Partition Table" Encoding method..... | 113 |
| 142 | 14.3.6 | "Numeric String" Encoding method | 114 |
| 143 | 14.3.7 | "6-bit CAGE/DODAAC" Encoding method..... | 115 |
| 144 | 14.3.8 | "6-Bit Variable String" Encoding method..... | 115 |
| 145 | 14.3.9 | "6-Bit Variable String Partition Table" Encoding method..... | 116 |
| 146 | 14.3.10 | "Fixed Width Integer" Encoding Method | 117 |
| 147 | 14.4 | Decoding procedure..... | 117 |
| 148 | 14.4.1 | "Integer" Decoding method..... | 118 |
| 149 | 14.4.2 | "String" Decoding method..... | 118 |
| 150 | 14.4.3 | "Partition Table" Decoding method..... | 119 |
| 151 | 14.4.4 | "Unpadded Partition Table" Decoding method..... | 120 |
| 152 | 14.4.5 | "String Partition Table" Decoding method | 120 |
| 153 | 14.4.6 | "Numeric String" Decoding method | 121 |
| 154 | 14.4.7 | "6-Bit CAGE/DoDAAC" Decoding method..... | 122 |
| 155 | 14.4.8 | "6-Bit Variable String" Decoding method..... | 122 |
| 156 | 14.4.9 | "6-Bit Variable String Partition Table" Decoding method | 123 |
| 157 | 14.4.10 | "Fixed Width Integer" Decoding method | 124 |
| 158 | 14.5 | Encoding/Decoding methods introduced in TDS 2.0 | 124 |
| 159 | 14.5.1 | " +AIDC Data Toggle Bit"..... | 126 |
| 160 | 14.5.2 | "Fixed-Bit-Length Integer" | 127 |

| | | | |
|-----|-----------|--|------------|
| 161 | 14.5.3 | "Prioritised Date" | 128 |
| 162 | 14.5.4 | "Fixed-Length Numeric" | 130 |
| 163 | 14.5.5 | "Delimited/Terminated Numeric" | 131 |
| 164 | 14.5.6 | "Variable-length alphanumeric" | 133 |
| 165 | 14.5.7 | "Single data bit" | 147 |
| 166 | 14.5.8 | "6-digit date YYMMDD" | 148 |
| 167 | 14.5.9 | "10-digit date+time YYMMDDhhmm" | 149 |
| 168 | 14.5.10 | "Variable-format date / date range" | 151 |
| 169 | 14.5.11 | "Variable-precision date+time" | 153 |
| 170 | 14.5.12 | "Country code (ISO 3166-1 alpha-2)" | 156 |
| 171 | 14.5.13 | "Variable-length integer without encoding indicator" | 158 |
| 172 | 14.5.14 | "Optional minus sign in 1 bit" | 159 |
| 173 | 14.6 | EPC Binary coding tables..... | 160 |
| 174 | 14.6.1 | Serialised Global Trade Item Number (SGTIN)..... | 160 |
| 175 | 14.6.2 | Serial Shipping Container Code (SSCC) | 164 |
| 176 | 14.6.3 | Global Location Number with or without Extension (SGLN)..... | 166 |
| 177 | 14.6.4 | Global Returnable Asset Identifier (GRAI) | 168 |
| 178 | 14.6.5 | Global Individual Asset Identifier (GIAI) | 171 |
| 179 | 14.6.6 | Global Service Relation Number - Recipient (GSRN) | 173 |
| 180 | 14.6.7 | Global Service Relation Number - Provider (GSRNP)..... | 175 |
| 181 | 14.6.8 | Global Document Type Identifier (GDTI) | 177 |
| 182 | 14.6.9 | CPI Identifier (CPI) | 180 |
| 183 | 14.6.10 | Global Coupon Number (SGCN) | 183 |
| 184 | 14.6.11 | Individual Trade Item Piece (ITIP) | 185 |
| 185 | 14.6.12 | General Identifier (GID)..... | 188 |
| 186 | 14.6.13 | DoD Identifier | 189 |
| 187 | 14.6.14 | ADI Identifier (ADI) | 189 |
| 188 | 15 | EPC Memory Bank contents | 190 |
| 189 | 15.1 | Encoding procedures | 190 |
| 190 | 15.1.1 | EPC Tag URI into Gen 2 EPC Memory Bank | 190 |
| 191 | 15.1.2 | EPC Raw URI into Gen 2 EPC Memory Bank..... | 190 |
| 192 | 15.2 | Decoding procedures | 192 |
| 193 | 15.2.1 | Gen 2 EPC Memory Bank into EPC Raw URI..... | 192 |
| 194 | 15.2.2 | Gen 2 EPC Memory Bank into EPC Tag URI | 192 |
| 195 | 15.2.3 | Gen 2 EPC Memory Bank into Pure Identity EPC URI | 193 |
| 196 | 15.2.4 | Decoding of control information | 193 |
| 197 | 15.3 | '+AIDC data' following new EPC schemes in the EPC/UII memory bank..... | 193 |
| 198 | 16 | Tag Identification (TID) Memory Bank Contents | 217 |
| 199 | 16.1 | Short Tag Identification (TID)..... | 217 |
| 200 | 16.2 | Extended Tag identification (XTID) | 218 |
| 201 | 16.2.1 | XTID Header | 219 |
| 202 | 16.2.2 | XTID Serialisation | 220 |
| 203 | 16.2.3 | Optional Command Support segment | 220 |
| 204 | 16.2.4 | BlockWrite and BlockErase segment..... | 221 |
| 205 | 16.2.5 | User Memory and BlockPermaLock segment..... | 222 |
| 206 | 16.2.6 | Optional Lock Bit segment | 223 |
| 207 | 16.3 | Serialised Tag Identification (STID) | 223 |

| | | | |
|-----|-----------|--|------------|
| 208 | 16.3.1 | STID URI grammar | 223 |
| 209 | 16.3.2 | Decoding procedure: TID Bank Contents to STID URI | 224 |
| 210 | 17 | User Memory Bank Contents | 224 |
| 211 | 18 | Conformance | 226 |
| 212 | 18.1 | Conformance of RFID Tag Data | 226 |
| 213 | 18.1.1 | Conformance of Reserved Memory Bank (Bank 00) | 226 |
| 214 | 18.1.2 | Conformance of EPC Memory Bank (Bank 01) | 226 |
| 215 | 18.1.3 | Conformance of TID Memory Bank (Bank 10) | 227 |
| 216 | 18.1.4 | Conformance of User Memory Bank (Bank 11) | 227 |
| 217 | 18.2 | Conformance of Hardware and Software Components | 227 |
| 218 | 18.2.1 | Conformance of hardware and software Components That Produce or Consume Gen 2 | |
| 219 | | Memory Bank Contents | 227 |
| 220 | 18.2.2 | Conformance of hardware and software Components that Produce or Consume URI Forms | |
| 221 | | of the EPC | 228 |
| 222 | 18.2.3 | Conformance of hardware and software components that translate between EPC Forms | |
| 223 | | | 229 |
| 224 | 18.3 | Conformance of Human Readable Forms of the EPC and of EPC Memory Bank contents | 230 |
| 224 | A | Character Set for Alphanumeric Serial Numbers | 231 |
| 225 | B | Glossary (non-normative) | 233 |
| 226 | C | References | 236 |
| 227 | D | Extensible Bit Vectors | 237 |
| 228 | E | (non-normative) Examples: EPC encoding and decoding | 238 |
| 229 | E.1 | Encoding a Serialised Global Trade Item Number (SGTIN) to SGTIN-96 | 238 |
| 230 | E.2 | Decoding an SGTIN-96 to a Serialised Global Trade Item Number (SGTIN) | 240 |
| 231 | E.3 | Summary Examples of All EPC schemes | 242 |
| 232 | F | Packed objects ID Table for Data Format 9 | 248 |
| 233 | F.1 | Tabular Format (non-normative) | 248 |
| 234 | F.2 | Comma-Separated-Value (CSV) format | 263 |
| 235 | G | 6-Bit Alphanumeric Character Set | 270 |
| 236 | H | (Intentionally Omitted) | 271 |
| 237 | I | Packed Objects structure | 272 |
| 238 | I.1 | Overview | 272 |
| 239 | I.2 | Overview of Packed Objects documentation | 272 |
| 240 | I.3 | High-Level Packed Objects format design | 272 |
| 241 | I.4 | Format Flags section | 274 |
| 242 | I.5 | Object Info section | 276 |
| 243 | I.6 | Secondary ID Bits section | 282 |
| 244 | I.7 | Aux Format section | 283 |
| 245 | I.8 | Data section | 284 |
| 246 | I.9 | ID Map and Directory encoding options | 287 |

| | | | |
|-----|----------|---|------------|
| 247 | J | Packed Objects ID tables | 292 |
| 248 | J.1 | Packed Objects data format registration file structure..... | 292 |
| 249 | J.2 | Mandatory and optional ID table columns..... | 294 |
| 250 | J.3 | Syntax of OIDs, IDstring, and FormatString Columns | 296 |
| 251 | J.4 | OID input/output representation | 299 |
| 252 | K | Packed Objects encoding tables | 301 |
| 253 | L | Encoding Packed Objects (non-normative) | 306 |
| 254 | M | Decoding Packed Objects (non-normative) | 310 |
| 255 | M.1 | Overview | 310 |
| 256 | M.2 | Decoding alphanumeric data..... | 311 |
| 257 | | | |

Index of figures

| | | |
|-----|--------------------|--|
| 258 | | |
| 259 | | |
| 260 | Figure 3-1 | Organisation of the EPC Tag Data Standard (TDS).....27 |
| 261 | Figure 4-1 | Example Visibility Data Stream29 |
| 262 | Figure 4-2 | Illustration of GRAI Identifier Namespace.....30 |
| 263 | Figure 4-3 | Illustration of EPC Identifier Namespace.....31 |
| 264 | Figure 4-4 | Illustration of Relationship of GS1 key and EPC Identifier Namespaces.....32 |
| 265 | Figure 4-5 | EPC Structures used within the GS1 System Architecture36 |
| 266 | Figure 6-1 | EPC Schemes and Where the Pure Identity Form is Defined.....39 |
| 267 | Figure 7-1 | Correspondence between SGTIN EPC URI and GS1 element string.....56 |
| 268 | Figure 7-2 | Correspondence between SSCC EPC URI and GS1 element string60 |
| 269 | Figure 7-3 | Correspondence between SGLN EPC URI without extension and GS1 element string61 |
| 270 | Figure 7-4 | Correspondence between SGLN EPC URI with extension and GS1 element string.....61 |
| 271 | Figure 7-5 | Correspondence between GRAI EPC URI and GS1 element string63 |
| 272 | Figure 7-6 | Correspondence between GIAI EPC URI and GS1 element string64 |
| 273 | Figure 7-7 | Correspondence between GSRN EPC URI and GS1 element string65 |
| 274 | Figure 7-8 | Correspondence between GSRNP EPC URI and GS1 element string.....66 |
| 275 | Figure 7-9 | Correspondence between GDTI EPC URI and GS1 element string67 |
| 276 | Figure 7-10 | Correspondence between CPI EPC URI and GS1 element string.....68 |
| 277 | Figure 7-11 | Correspondence between SGCN EPC URI and GS1 element string69 |
| 278 | Figure 7-12 | Correspondence between GINC EPC URI and GS1 element string70 |
| 279 | Figure 7-13 | Correspondence between GSIN EPC URI and GS1 element string71 |
| 280 | Figure 7-14 | Correspondence between ITIP EPC URI and GS1 element string.....72 |
| 281 | Figure 7-15 | Correspondence between UPUI EPC URI and GS1 element string.....73 |
| 282 | Figure 7-16 | Correspondence between PGLN EPC URI without extension and GS1 element string75 |
| 283 | Figure 7-17 | Correspondence between LGTIN EPC Class URI and GS1 element string.....76 |
| 284 | Figure 9-1 | Gen 2 Tag Memory Map83 |
| 285 | Figure 9-2 | Gen 2 Protocol Control (PC) Bits Memory Map.....85 |
| 286 | Figure 12-1 | Illustration of EPC Tag URI and EPC Raw URI.....96 |
| 287 | Figure 12-2 | Illustration of Filter Value within EPC Tag URI.....97 |
| 288 | Figure 14-1 | Example of the use of the +AIDC data toggle bit.....126 |
| 289 | Figure 14-2 | Prioritised date format support for 6-digit date values.....128 |
| 290 | Figure 14-3 | Example of numeric delimiter and terminator131 |
| 291 | Figure 14-4 | Examples of "Variable-length alphanumeric" encoding method.....133 |
| 292 | Figure 14-5 | Decision tree flowchart to select the most efficient encoding method based on the value being encoded.....134 |
| 294 | Figure 14-6 | Example value - alphanumeric, encoded as file-safe URI-safe base 64139 |
| 295 | Figure 14-7 | Use of the "Variable-length URN Code 40" method to encode 6 characters142 |
| 296 | Figure 14-8 | Example of alphanumeric encoded as 7-bit ASCII.....146 |
| 297 | Figure 14-9 | Efficient encoding of YYMMDD date value using 16 bits148 |

| | | |
|-----|--|-----|
| 298 | Figure 14-10 Encoding of YYMMDDhhmm date time value using 27 bits | 149 |
| 299 | Figure 14-11 Encoding of "Variable-format date / date range" | 151 |
| 300 | Figure 14-12 Encoding of "Variable-precision date+time" | 154 |
| 301 | Figure 14-13 ISO 3166-1 alpha-2 country code encoded as file-safe URI base 64..... | 157 |
| 302 | Figure 15-1 Example of '+AIDC data' in EPC/UII memory | 194 |
| 303 | Figure 15-2 Reading and interpreting additional bits after the 8-bit data header..... | 196 |
| 304 | Figure 15-3 Examples of encoding all-numeric and alphanumeric batch/lot number | 213 |
| 305 | Figure 15-4 Encoding more than one AIDC data value after the EPC | 214 |
| 306 | | |

Index of tables

| | | |
|-----|--------------------|--|
| 307 | | |
| 308 | | |
| 309 | Table 4-1 | EPC Schemes and Corresponding GS1 keys33 |
| 310 | Table 6-1 | EPC Class Schemes and Where the Pure Identity Form is Defined53 |
| 311 | Table 9-1 | Kinds of Data on a Gen 2 RFID Tag.....81 |
| 312 | Table 9-2 | Gen 2 Memory Map.....84 |
| 313 | Table 9-3 | Gen 2 Protocol Control (PC) Bits Memory Map86 |
| 314 | Table 10-1 | SGTIN Filter Values.....90 |
| 315 | Table 10-2 | SSCC Filter Values90 |
| 316 | Table 10-3 | SGLN Filter Values91 |
| 317 | Table 10-4 | GRAI Filter Values91 |
| 318 | Table 10-5 | GIAI Filter Values91 |
| 319 | Table 10-6 | GSRN and GSRNP Filter Values.....92 |
| 320 | Table 10-7 | GDTI Filter Values92 |
| 321 | Table 10-8 | CPI Filter Values.....92 |
| 322 | Table 10-9 | SGCN Filter Values.....93 |
| 323 | Table 10-10 | ITIP Filter Values.....93 |
| 324 | Table 10-11 | ADI Filter Values93 |
| 325 | Table 12-1 | Control information fields97 |
| 326 | Table 12-2 | EPC Binary Coding Schemes and their limitations99 |
| 327 | Table 14-1 | EPC Binary Header Values108 |
| 328 | Table 14-2 | Summary of Encoding/Decoding methods introduced in TDS 2.0124 |
| 329 | Table 14-3 | "Fixed-Length Numeric" encoding table130 |
| 330 | Table 14-4 | Encoding table for initial digits of "Delimited/Terminated Numeric" encoding method131 |
| 331 | Table 14-5 | Mapping table for "Variable-length upper case hexadecimal" encoding method.....137 |
| 332 | Table 14-6 | Mapping table for "Variable-length lower case hexadecimal" encoding method138 |
| 333 | Table 14-7 | Mapping table for "Variable-length 6-bit file-safe URI-safe base 64" encoding method.....140 |
| 334 | Table 14-8 | URN Code 40 character table142 |
| 335 | Table 14-9 | Character table for "Variable-length 7-bit ASCII" encoding method.....144 |
| 336 | Table 14-10 | Encoding table for "Country code (ISO 3166-1 alpha-2)"157 |
| 337 | Table 14-11 | SGTIN Partition Table.....161 |
| 338 | Table 14-12 | SGTIN-96 coding table161 |
| 339 | Table 14-13 | SGTIN-198 coding table162 |
| 340 | Table 14-14 | GRAI Partition Table.....168 |
| 341 | Table 14-15 | GRAI-170 coding table169 |
| 342 | Table 14-16 | GRAI+ coding table170 |
| 343 | Table 14-17 | GIAI-96 Partition Table.....171 |
| 344 | Table 14-18 | GIAI-96 coding table.....171 |
| 345 | Table 14-19 | CPI-96 coding table181 |
| 346 | Table 14-20 | CPI-var coding table182 |

| | | |
|-----|--|-----|
| 347 | Table 14-21 CPI+ coding table..... | 182 |
| 348 | Table 14-22 GID-96 coding table..... | 188 |
| 349 | Table 14-23 ADI-var coding table..... | 189 |
| 350 | Table 15-1 Recipe to Fill In Gen 2 EPC Memory Bank from EPC Tag URI | 190 |
| 351 | Table 15-2 Recipe to Fill In Gen 2 EPC Memory Bank from EPC Raw URI..... | 191 |
| 352 | Table 16-1 Short TID format..... | 218 |
| 353 | Table 16-2 Non-Normative example: Extended Tag Identification (XTID) format for the TID memory bank | |
| 354 | | 219 |
| 355 | Table 16-3 The XTID header | 219 |
| 356 | Table 16-4 Optional Command Support XTID Word..... | 220 |
| 357 | Table 16-5 XTID Block Write and Block Erase Information | 221 |
| 358 | Table 16-6 XTID Block PermaLock and User Memory Information | 223 |
| 359 | | |

360
361

Index of special encoding tables new to TDS 2.0

| Table | Description | TDS section |
|-------|--|------------------------|
| E | Table E lists the permitted values for encoding indicator together with the encoding methods and the character ranges supported by each method. | 14.5.6 |
| K | Table K is derived from GS1 Gen Specs Figure 7.8.1-2, adding an additional column to indicate how many additional bits need to be read beyond the initial eight bits of the data header. | 15.3 |
| F | After determining the GS1 Application Identifier key (whether 2,3 or 4 digits), a lookup in column a of Table F explains how the corresponding value is to be encoded. | |
| B | Table B calculates the number of bits required to encode the value of a string of length L depending on the encoding method selected . This table may be used to avoid the need for floating-point arithmetic calculations. | |

362

363 Foreword

364 Abstract

365 The EPC Tag Data Standard (TDS) defines the Electronic Product Code™, and also specifies the
366 memory contents of Gen 2 RFID Tags. In more detail, TDS covers two broad areas:

- 367 ■ The specification of the Electronic Product Code (EPC), including its representation at various
368 levels of the GS1 System Architecture and its correspondence to GS1 keys and other existing
369 codes.
- 370 ■ The specification of data that is carried on Gen 2 RFID tags, including the EPC, "user memory"
371 data, control information, and tag manufacture information.

372 Audience for this document

373 The target audience for this specification includes:

- 374 ■ EPC Middleware vendors
- 375 ■ RFID Tag users and encoders
- 376 ■ Reader vendors
- 377 ■ Application developers
- 378 ■ System integrators

379 Differences from EPC Tag Data Standard Version 1.6

380 The EPC Tag Data Standard Version 1.7 is fully backward-compatible with EPC Tag Data Standard
381 Version 1.6.

382 The EPC Tag Data Standard Version 1.7 includes these new or enhanced features:

- 383 ■ A new EPC Scheme, the Component and Part Identifier (CPI) scheme, has been added ;
- 384 ■ Various typographical errors have been corrected.

385 Differences from EPC Tag Data Standard Version 1.7

386 The EPC Tag Data Standard Version 1.8 is fully backward-compatible with EPC Tag Data Standard
387 Version 1.7.

388 The EPC Tag Data Standard Version 1.8 includes the following enhancements:

- 389 ■ The GIAI EPC Scheme has been allocated an additional Filter Value, "Rail Vehicle".

390 Differences from EPC Tag Data Standard Version 1.8

391 The EPC Tag Data Standard Version 1.9 is fully backward-compatible with EPC Tag Data Standard
392 Version 1.8.

393 The EPC Tag Data Standard Version 1.9 includes the following enhancements:

- 394 ■ A new EPC Class URI to represent the combination of a GTIN plus a Batch/Lot (LGTIN) has been
395 added.
- 396 ■ A new EPC Scheme the SerialisedGlobal Coupon Number (SGCN), has been added along with
397 the SGCN-96 binary encoding.

- 398 ■ A new EPC Scheme, the Global Service Relation Number – Provider" (GSRNP), has been added
- 399 along with the GSRNP-96 binary encoding. This corresponds to the addition of AI (8017) to
- 400 [GS1GS14.0];

- 401 ■ The existing GSRN EPC Scheme is retitled Global Service Relation Number – Recipient to
- 402 harmonise with [GS1GS14.0] update to AI (8018). The EPC Scheme name and URI is
- 403 unchanged, however, to preserve backward compatibility with TDS 1.8 and earlier.

- 404 ■ New AIs are added to the Packed Objects ID Table for EPC User Memory, to harmonise TDS with
- 405 [GS1GS14.0], thereby ensuring that all AIs can be encoded in both barcode and RFID data
- 406 carriers:
 - 407 □ Packaging Component Number: AI (243)
 - 408 □ Global Coupon Number: AI (255)
 - 409 □ Country Subdivision of Origin: AI (427)
 - 410 □ National Healthcare Reimbursement Number (NHRN) – Germany PZN: AI (710)
 - 411 □ National Healthcare Reimbursement Number (NHRN) – France CIP: AI (711)
 - 412 □ National Healthcare Reimbursement Number (NHRN) – Spain CN: AI (712)
 - 413 □ National Healthcare Reimbursement Number (NHRN) – Brazil DRN: AI (713)
 - 414 □ Component Part Identifier (8010)
 - 415 □ Component / Part Identifier Serial Number (8011)
 - 416 □ Global Service Relation Number – Provider: AI (8017)
 - 417 □ Service Relation Instance Number (SRIN): AI (8019)
 - 418 □ Extended Packaging URL: AI (8200)

- 419 ■ DEPRECATED "Secondary data for specific health industry products" AI (22) in the Packed
- 420 Objects ID Table for EPC User Memory, to harmonise TDS with the GS1 General Specifications;

- 421 ■ A new EPC binary encoding for the Global Document Type Identifier, GDTI-174, is to
- 422 accommodate all values of the GDTI serial number permitted by [GS1GS14.0] (1 – 17
- 423 alphanumeric characters, compared to 1 – 17 numeric characters permitted in earlier versions of
- 424 the GS1 General Specifications).

- 425 ■ DEPRECATED the GDTI-113 EPC Binary Encoding; the GDTI-174 Binary Encoding should be used
- 426 instead

- 427 ■ Updated all [GS1GS14.0] version and section references;
- 428 ■ Marked Attribute Bits information as pertaining only to Gen2 v 1.x tags;
- 429 ■ Changed "*ItemReference*" to "*ItemRefAndIndicator*" in SGTIN general syntax;
- 430 ■ Corrected provision on number of characters in "String" Encoding method's validity test from
- 431 "less than b/7" to "less than or equal to b/7";
- 432 ■ Corrected various errata.

433 **Differences from EPC Tag Data Standard Version 1.9**

434 The EPC Tag Data Standard Version 1.10 is fully backward-compatible with EPC Tag Data Standard

435 Version 1.9.

436 The EPC Tag Data Standard Version 1.10 includes the following enhancements:

- 437 ■ New EPC URIs have been added to represent the following identifiers:
 - 438 □ GINC
 - 439 □ GSIN
 - 440 □ BIC container code

- 441 ■ Clarification has been added regarding SGTIN Filter Values "Full Case for Transport" and "Unit
- 442 Load";
- 443 ■ GDTI EPC Scheme has been allocated an additional Filter Value, "Travel Document";
- 444 ■ ADI EPC Scheme has been allocated a number of additional Filter Values, to harmonise with the
- 445 2015 release of ATA's Spec 2000;
- 446 ■ New AIs have been added to the Packed Objects ID Table for EPC User Memory, to harmonise
- 447 TDS with [GS1GS17.0], thereby ensuring that all AIs can be encoded in both barcode and RFID
- 448 data carriers:
 - 449 □ Sell by date: AI (16)
 - 450 □ Percentage discount of a coupon: AI (394n)
 - 451 □ Catch area: AI (7005)
 - 452 □ First freeze date: AI (7006)
 - 453 □ Harvest date: AI (7007)
 - 454 □ Species for fishery purposes: AI (7008)
 - 455 □ Fishing gear type: AI (7009)
 - 456 □ Production method: AI (7010)
 - 457 □ Software version: AI (8012)
 - 458 □ Loyalty points of a coupon: AI (8111)
- 459 ■ "GS1-128 Coupon Extended Code - NSC" AI (8102) has been marked as DEPRECATED;
- 460 ■ Format string for "International Bank Account Number (IBAN)" AI (8007) has been corrected;
- 461 ■ SGCN coding table has been corrected to include the SGCN header;
- 462 ■ Short Tag Identification within the TID Memory Bank has been updated to align with
- 463 [UHFC1G2v2.0];
- 464 ■ Correspondence between EPCs and GS1 Keys has been updated to accommodate 4- and 5-digit
- 465 GCPs, to align with [GS1GS17.0];
- 466 ■ Abstract, Audience and overview of Differences have been moved to a new "Foreword" section
- 467 added after the Table of Contents.

468 Differences from EPC Tag Data Standard (TDS) Version 1.10

469 TDS v 1.11 is fully backward-compatible with TDS v 1.10.

470 TDS v 1.11 includes the following enhancements:

- 471 ■ A new EPC Scheme, the Individual Trade Item Piece (ITIP), has been added along with the ITIP-
- 472 110 and ITIP-212 binary encodings.
- 473 ■ The following new AIs have been added to the Packed Objects ID Table for EPC User Memory, to
- 474 harmonise TDS with [GS1GS17.1], thereby ensuring that all AIs can be encoded in both barcode
- 475 and RFID data carriers:
 - 476 □ GLN of the production or service location: AI (416)
 - 477 □ Refurbishment lot ID: AI (7020)
 - 478 □ Functional status: AI (7021)
 - 479 □ Revision status: AI (7022)
 - 480 □ Global Individual Asset Identifier (GIAI) of an Assembly: AI (7023)
- 481 ■ Format string for AIs 91-99 has been revised to allow for up to 90 characters (previously up to
- 482 30), in order to harmonise TDS with [GS1GS17.0];

483
484
485
486



Note: To harmonise with [GS1GS17.0], which have extended the length AIs 91-99 to 90 (previously 30) alphanumeric characters, TDS v 1.11 has extended the string format of AIs 91-99 (encoded by means of Packed Objects in User Memory) from 1*30an (alphanumeric, length 1 to 30) to 1*an (alphanumeric, no upper bound).

487
488
489
490
491
492
493
494
495
496
497
498

This revision to tables F.1 and Fs.2 of TDS is fully backward compatible, allowing a tag written per TDS 1.10 to decode properly per TDS 1.11. It is also mostly forward compatible, allowing a tag written per TDS 1.11 to decode properly per TDS 1.10, as long as the length of AI 91,...,99 is 30 or fewer. A tag written per TDS 1.10 with a longer value for one of these AIs may signal an error indicating that the value is too long, but other AIs will decode properly. Another minor issue is that the encoding algorithm will no longer enforce an upper limit on the length of an encoded value, so it will be possible to encode an AI 91-99 character value that is too long per [GS1GS] (e.g. 100 character). Therefore, **to ensure compliance with the GenSpecs and rest of the GS1 System, AI 91-99 character values encoded in User Memory should not exceed 90 characters in length.**

- Marked all EPC binary headers previously reserved for 64-bit encodings as now "Reserved for Future Use" (RFU), reflecting the July 2009 sunseting of the 64-bit encodings.

499

Differences from EPC Tag Data Standard (TDS) Version 1.11

500
501
502
503
504
505
506
507
508
509
510
511
512
513

TDS v 1.12 is fully backward-compatible with TDS v 1.11.

TDS v 1.12 includes the following enhancements:

- The following EPC Schemes have been added:
 - UPII
 - PGLN
- Guidance has been added (to section 7) to determine the length of the EPC CompanyPrefix component for individually assigned GS1 Keys
- "Fixed Width Integer" encoding and decoding methods have been added (to section 14) in support of ITIP,
- Coding method for the Piece and Total components of the ITIP has been corrected from "String" to "Fixed Width Integer"
- The following new AIs have been added to the Packed Objects ID Table for EPC User Memory, to harmonise TDS with [GS1GS19.1], thereby ensuring that all AIs can be encoded in both barcode and RFID data carriers:
 - Consumer product variant: AI (22)
 - Third party controlled, serialised extension of GTIN (TPX): AI (235)
 - Global Location Number of Party: AI (417)
 - National Healthcare Reimbursement Number (NHRN) – Portugal AIM: AI (714)
 - GS1 UIC with Extension 1 and Importer index (per EU 2018/574): AI (7040)
 - Global Model Number: AI (8013)
 - Identification of pieces of a trade item (ITIP) contained in a logistics unit: AI (8026)
 - Paperless coupon code identification for use in North America: AI (8112)

522

Differences from EPC Tag Data Standard (TDS) Version 1.12

523
524

TDS v 1.13 includes the following enhancement:

- Added IMOVN EPC URIO, to encode the IMO Vessel Number.

- 525
- 526
- Added Protocol ID: AI (7240) to the Packed Objects ID Table for EPC User Memory, to harmonise TDS with [GS1GS19.1], ensuring support for all GS1 AIs in User Memory.
- 527
- Corrected minor errata
- 528
- TDS v 1.13 is fully backward-compatible with TDS v 1.12.

529 Differences from EPC Tag Data Standard (TDS) Version 1.13

530 TDS version 2.0 introduces twelve new EPC schemes and simplified binary encoding to promote
531 greater interoperability with barcodes. Existing EPC schemes already defined in TDS 1.13 remain
532 valid and are not deprecated. The new EPC schemes do not use partition tables and the length of
533 the GS1 Company Prefix is neither significant nor does it need to be known for the new binary
534 encodings. Each of the new EPC schemes may also be appended with additional AIDC data after the
535 EPC. Where appropriate, the new schemes make use of encoding indicators and length indicators to
536 support efficient binary encodings when encoding fewer characters than the maximum permitted or
537 when using a more restricted character set (e.g. only using digits where alphanumeric characters
538 are allowed).

539 In order to continue support for filtering and selection over the air interface based on the GS1
540 Company Prefix or the primary GS1 identifier (such as GTIN, SSCC etc.) the primary identifier is
541 encoded using 4 bits per digit in most of the new EPC schemes; the exceptions to this statement are
542 the new GIAI+ and CPI+ schemes because the GIAI and CPI permit alphanumeric characters to
543 follow immediately after the GS1 Company Prefix, so for GIAI+ and CPI+, it is only the initial
544 numeric digits of the GIAI and CPI that are encoded using 4 bits per digit. This can include any
545 initial all-numeric digits of the Individual Asset Identifier or the Component/Part Reference. These
546 are aligned on nibble boundaries and ensure that in each of the new schemes the primary identifier
547 and GS1 Company Prefix component appears at well-defined bit positions relative to the start of the
548 EPC/UII memory bank irrespective of the value of any indicator digit or extension digit that may be
549 present. No URN syntax is defined for the new EPC schemes but mappings to element strings and
550 GS1 Digital Link URIs are indicated. Because EPCIS/CBV 2.0 accepts a constrained subset of GS1
551 Digital Link URIs (specifically at instance-level granularity and without additional data attributes) as
552 a valid alternative to pure identity EPC URNs, there is no major need to define URN syntax for the
553 new EPC schemes introduced in TDS 2.0.

554 The filter values already defined for EPC schemes prior to TDS 2.0 remain valid and unaltered and
555 are carried forward into the corresponding new EPC schemes. For example, the new schemes
556 SGTIN+ and DSGTIN+ share the same set of filter values already defined for SGTIN-96 and SGTIN-
557 198.

558 TDS 2.0 also introduces a new EPC binary encoding, DSGTIN+, a date-prioritised serialised GTIN in
559 which a critical date value appears before the GTIN within the binary encoding. This is expected to
560 be particularly useful for perishable goods, stock rotation and management of goods with limited
561 remaining shelf life. This enables an RFID reader to select products from any brand owner or
562 manufacturer where the critical date matches a specified value such as products whose use-by date
563 or sell-by date is today, so that they can be removed from the sales area or discounted for quick
564 sale.

565 TDS 2.0 now mentions GS1 Digital Link and recognises that a constrained subset of GS1 Digital Link
566 URIs may be used in EPCIS/CBV v2.0 event data, as a valid alternative to pure identity EPC URNs.

567 TDS v 2.0 includes the following enhancements and changes with respect to TDS v 1.13:

- 568
- Sensor data (as encoded in the XPC bits) is included in "Business Data" carried by tags (section [9.1](#)).
 - **Encodings new to TDS 2.0 are described counting bits from left to right.**
 - Clarification that the Length bits (10h-14h) in the PC Bits represent the number of 16-bit words comprising the EPC field (beginning with bit 20h), including any optional "AIDC data" appended to the EPC itself.
 - Description of the UMI bit (15h) has been aligned with § 6.3.2.1.2.2 of the Gen2v2 standard [UHFC1G2].
 - Description of the XPC W1 indicator (16h) has been aligned with § 6.3.2.1.2.5 of [UHFC1G2].
- 570
- 571
- 572
- 573
- 574
- 575
- 576

- 577 ■ Description of the Attribute bits moved from section 11 to sections [9.3](#) and [9.4](#).
- 578 ■ Description of XPC bits added as new section [9.4](#), aligned with § 6.3.2.1.2.5 of [UHFC1G2].
- 579 ■ Most EPC encoding examples have been updated to use sample GCP 9521141; the SGTIN
580 examples in section [E](#) use GTIN 09506000134352 to illustrate a resolvable GS1 Digital Link URI.
- 581 ■ Twelve (12) new EPC Binary Headers in the F0-FB range have been added to section [14.2](#) for
582 the new "EPC+" encoding schemes.
- 583 ■ EPC Binary Header FE has been reserved as an 'Unspecified' / 'Pad' Header for use with
584 optimised *Select* functionality tentatively planned for Gen2v3.
- 585 ■ The "Integer" Encoding Method (section [14.3.1](#)) now provides an explicit reminder that
586 "leading zeros are not permitted".
- 587 ■ Section [14.5](#) specifies new Encoding/Decoding methods introduced in TDS 2.0, specifically:
 - 588 □ "+AIDC Data Toggle Bit"
 - 589 □ "Fixed-Bit-Length Integer"
 - 590 □ "Prioritised Date"
 - 591 □ "Fixed-Length Numeric"
 - 592 □ "Delimited/Terminated Numeric"
 - 593 □ "Variable-length alphanumeric" (section [14.5.6](#)), including a decision tree to help
594 implementations determine the most efficient of the following encoding methods to use
595 (based on characters actually present in the value to be encoded):
 - 596 - Variable-length integer
 - 597 - Variable-length upper case hexadecimal
 - 598 - Variable-length lower case hexadecimal
 - 599 - Variable-length 6-bit file-safe URI-safe base 64
 - 600 - Variable-length URN Code 40
 - 601 - Variable-length 7-bit ASCII
 - 602 □ "Single data bit"
 - 603 □ "6-digit date YYMMDD"
 - 604 □ "10-digit date+time YYMMDDhhmm"
 - 605 □ "Variable-format date / date range"
 - 606 □ "Variable-precision date+time"
 - 607 □ "Country code (ISO 3166-1 alpha-2)"
- 608 ■ EPC Memory Bank Decoding procedures now specify (section [15.2.4](#)) one text string (rather
609 than two text strings in TDS 1.13) to include XPC_W1 and XPC_W2, when only the former or
610 both of these exist,
- 611 ■ Section [15.3](#) details encoding and decoding of the new "' +AIDC data' following new EPC
612 schemes in the EPC/UII memory bank"
- 613 ■ Within the XTID Header ([section 16.2.1](#)), an indicator (bit 9 in XTID) has been added to specify
614 that the XTID includes the Lock Bit Segment; for the Serialisation bits of the XTID Header,
615 clarification has been provided to state that bit 15 is MSB and bit 13 is LSB.
- 616 ■ The Optional Lock Bit Segment ([section 16.2.6](#)) has been added to XTID, to indicate the current
617 lock bit settings for the memory banks on the tag,
- 618 ■ The STID URI (section [16.3](#)) has been corrected to reflect the X, S and F indicators and 9-bit
619 MDID introduced by Gen2 v2.
- 620 ■ User Memory Bank Contents (section [17](#)) have been updated to reflect support for ISO/IEC
621 20248 Digital Signatures, and to refer to section [9.3](#) for an explanation of the UMI,
- 622 ■ Section [E](#) includes updated examples for all EPC (TDS 1.13) and EPC+ (TDS 2.0) schemes.

| | | |
|-----|---|--|
| 623 | ■ | Section F adds the following new GS1 Application Identifiers (AIs) for use in conjunction with |
| 624 | | Packed Objects: |
| 625 | □ | 395(***) |
| 626 | □ | 4300 |
| 627 | □ | 4301 |
| 628 | □ | 4302 |
| 629 | □ | 4303 |
| 630 | □ | 4304 |
| 631 | □ | 4305 |
| 632 | □ | 4306 |
| 633 | □ | 4307 |
| 634 | □ | 4308 |
| 635 | □ | 4309 |
| 636 | □ | 4310 |
| 637 | □ | 4311 |
| 638 | □ | 4312 |
| 639 | □ | 4313 |
| 640 | □ | 4314 |
| 641 | □ | 4315 |
| 642 | □ | 4316 |
| 643 | □ | 4317 |
| 644 | □ | 4318 |
| 645 | □ | 4319 |
| 646 | □ | 4320 |
| 647 | □ | 4321 |
| 648 | □ | 4322 |
| 649 | □ | 4323 |
| 650 | □ | 4324 |
| 651 | □ | 4325 |
| 652 | □ | 4326 |
| 653 | □ | 715 |
| 654 | □ | 723s |
| 655 | □ | 723s |
| 656 | □ | 723s |
| 657 | □ | 723s |
| 658 | □ | 723s |
| 659 | □ | 723s |
| 660 | □ | 723s |
| 661 | □ | 723s |
| 662 | □ | 723s |
| 663 | □ | 723s |

664 Differences from EPC Tag Data Standard Version 2.0

665 TDS v 2.1 is fully backward-compatible with TDS v 2.0.

666 TDS v 2.0 includes the following changes with respect to TDS v 1.13:

- 667 ■ Added index of figures
- 668 ■ Added index of tables
- 669 ■ Added text to Sections 6.3.16 and 14.6.12, General Identifier (GID), to indicate that **General**
- 670 **Manager Number issuance has been discontinued**, effective June 2023.
- 671 ■ Added index of encoding **Tables E, F, K and B**, introduced to TDS 2.0/2.1 in sections 14.5.6
- 672 and 15.3.
- 673 ■ Restored encoding Table B, which had been unintentionally omitted from the published version
- 674 of TDS 2.0, to section 15.3. Table B calculates the number of bits required to encode the value
- 675 of a string of length L depending on the encoding method selected. This may be used to avoid
- 676 the need for floating-point arithmetic calculations.
- 677 ■ Restored missing rows to Table K, which had been unintentionally shortened in the published
- 678 version of TDS 2.0. Table K now includes all rows, including those where the AI key is 2 digits,
- 679 so that those are explicit; this means that any 2-digit string not present in the full Table K is
- 680 currently also missing from the corresponding table in GenSpecs and does not correspond to a
- 681 currently defined AI key of 2, 3 or 4 digits.
- 682 ■ Corrected Table E to resolve contradiction between Table E and the encoding indicators
- 683 mentioned in sections 14.5.6.2 and 14.5.6.3.
- 684 ■ Section 17 (Packed Objects) now references new GS1 AI (8030) and clarifies the role of the
- 685 Party GLN (PGLN) as Domain Authority ID (DAID) when a [ISO20248] digital signature is
- 686 associated with a GS1 element string.
- 687 ■ Section E adds the following new GS1 Application Identifiers (AIs) for use in conjunction with
- 688 Packed Objects:
 - 689 □ AIDC media type: AI (7241)
 - 690 □ Version Control Number (VCN): AI (7242)
 - 691 □ Digital Signature (DigSig): AI (8030)
 - 692 □ Test by date: AI 7011
 - 693 □ Maximum temperature in Fahrenheit: AI (4330)
 - 694 □ Maximum temperature in Celsius: AI (4331)
 - 695 □ Minimum temperature in Fahrenheit: AI (4332)
 - 696 □ Minimum temperature in Celsius: AI (4333)
- 697 ■ Typographical errors have been corrected in the *Packed Objects ID Table for Data Format 9*, in
- 698 Sections F.1 (non-normative tabular format) and F.2 (normative CSV format).
- 699 ■ The *Packed Objects ID Table for Data Format 9* in Section F.2 has been **supplemented with an**
- 700 **external, normative artefact in CSV format**.

701 TDS v 2.1 also corrects minor errors in non-normative examples and other errata discovered after

702 the publication of TDS v 2.0.

703 1 Introduction

704 The EPC Tag Data Standard defines the Electronic Product Code™ (EPC), and specifies the memory

705 contents of Gen 2 RFID Tags. In more detail, TDS covers two broad areas:

- 706 ■ The specification of the Electronic Product Code, including its representation at various levels of
- 707 the GS1 Architecture and its correspondence to GS1 keys and other existing codes.

- 708
709
- The specification of data that is carried on Gen 2 RFID tags, including the EPC, "user memory" data, control information, and tag manufacture information.

710 The Electronic Product Code (EPC) is a universal identifier for any physical object. It is used in
711 information systems that need to track or otherwise refer to physical objects. A very large subset of
712 applications that use the EPC also rely upon RFID Tags as a data carrier. For this reason, a large
713 part of TDS is concerned with the encoding of EPCs onto RFID tags, along with defining the
714 standards for other data apart from the EPC that may be stored on a Gen 2 RFID tag.

715 Therefore, the two broad areas covered by TDS (the EPC and RFID) overlap in the parts where the
716 encoding of the EPC onto RFID tags is discussed. Nevertheless, it should always be remembered
717 that the EPC and RFID are not at all synonymous: EPC is an identifier, and RFID is a data carrier.
718 RFID tags contain other data besides EPC identifiers (and in some applications may not carry an EPC
719 identifier at all), and the EPC identifier exists in non-RFID contexts (those non-RFID contexts
720 including the URI form used within information systems, printed human-readable EPC URIs, and EPC
721 identifiers derived from barcode data following the procedures in this standard).

722 2 Terminology and typographical conventions

723 Within this specification, the terms SHALL, SHALL NOT, SHOULD, SHOULD NOT, MAY, NEED NOT,
724 CAN, and CANNOT are to be interpreted as specified in Annex G of the ISO/IEC Directives, Part 2,
725 2001, 4th edition [ISODir2]. When used in this way, these terms will always be shown in ALL CAPS;
726 when these words appear in ordinary typeface they are intended to have their ordinary English
727 meaning.

728 All sections of this document, with the exception of Section Introduction are normative, except
729 where explicitly noted as non-normative.

730 The following typographical conventions are used throughout the document:

- 731
- ALL CAPS type is used for the special terms from [ISODir2] enumerated above.
 - Monospace type is used for illustrations of identifiers and other character strings that exist within information systems.

734 The term "Gen 2 RFID Tag" (or just "Gen 2 Tag") as used in this specification refers to any RFID tag
735 that conforms to the EPCglobal UHF Class 1 Generation 2 Air Interface, Version 1.2.0 or later
736 [UHFC1G2], as well as any RFID tag that conforms to another air interface standard that shares the
737 same memory map. Bitwise addresses within Gen 2 Tag memory banks are indicated using
738 hexadecimal numerals ending with a subscript "h"; for example, 20_h denotes bit address
739 20 hexadecimal (32 decimal).

740 3 Overview of TDS

741 This section provides an overview of TDS and how the parts fit together.

742 TDS covers two broad areas:

- 743
- The specification of the EPC, including its representation at various levels of the GS1 System Architecture and its correspondence to GS1 keys and other existing codes.
 - The specification of data that is carried on Gen 2 RFID tags, including the EPC, "user memory" data, control information, and tag manufacture information.

747 The EPC is a universal identifier for any physical object, although EPC URI formats are also defined
748 for locations and organisations. It is used in information systems that need to track or otherwise
749 refer to physical objects. Within computer systems, including electronic documents, databases, and
750 electronic messages, the EPC takes the form of an Internet Uniform Resource Identifier (URI). This
751 is true regardless of whether the EPC was originally read from an RFID tag or some other kind of
752 data carrier. This URI is called the "Pure Identity EPC URI." The following is an example of a Pure
753 Identity EPC URI:

754 urn:epc:id:sgtin:9521141.012345.4711

755 This same identifier can also be encoded as a canonical **GS1 Digital Link URI** [GS1DL] as follows:

756 `https://id.gs1.org/01/09521141123454/21/4711`

757 or as a non-canonical GS1 Digital link URI such as:

758 `https://example.com/01/09521141123454/21/4711`

759 or even (with some additional URI path information):

760 `https://example.com/some/path/info/01/09521141123454/21/4711`

761 *Note that these example GS1 Digital Link URIs are not currently configured to redirect to a*
762 *demonstration Web page.*

763 A very large subset of applications that use EPCs also rely upon RFID tags as a data carrier. RFID is
764 often a very appropriate data carrier technology to use for applications involving visibility of physical
765 objects, because RFID permits data to be physically attached to an object such that reading the
766 data is minimally invasive to material handling processes. For this reason, a large part of TDS is
767 concerned with the encoding of EPCs onto RFID tags, along with defining the standards for other
768 data apart from the EPC that may be stored on a Gen 2 RFID tag. Owing to memory limitations of
769 RFID tags, the EPC is not stored in URI form on the tag, but is instead encoded into a compact
770 binary representation. This is called the "EPC Binary Encoding" and refers to on-tag encoding of the
771 EPC, regardless of the choice of which specific EPC scheme is used.

772 Therefore, the two broad areas covered by TDS (the EPC and RFID) overlap in the parts where the
773 encoding of the EPC onto RFID tags is discussed. Nevertheless, it should always be remembered
774 that the EPC and RFID are not at all synonymous: EPC is an identifier, and RFID is a data carrier.
775 RFID tags contain other data besides EPC identifiers (and in some applications may not carry an EPC
776 identifier at all), and the EPC identifier exists in non-RFID contexts (those non-RFID contexts
777 currently including the URI form used within information systems, printed human-readable EPC
778 URIs, and EPC identifiers derived from barcode data following the procedures in this standard).

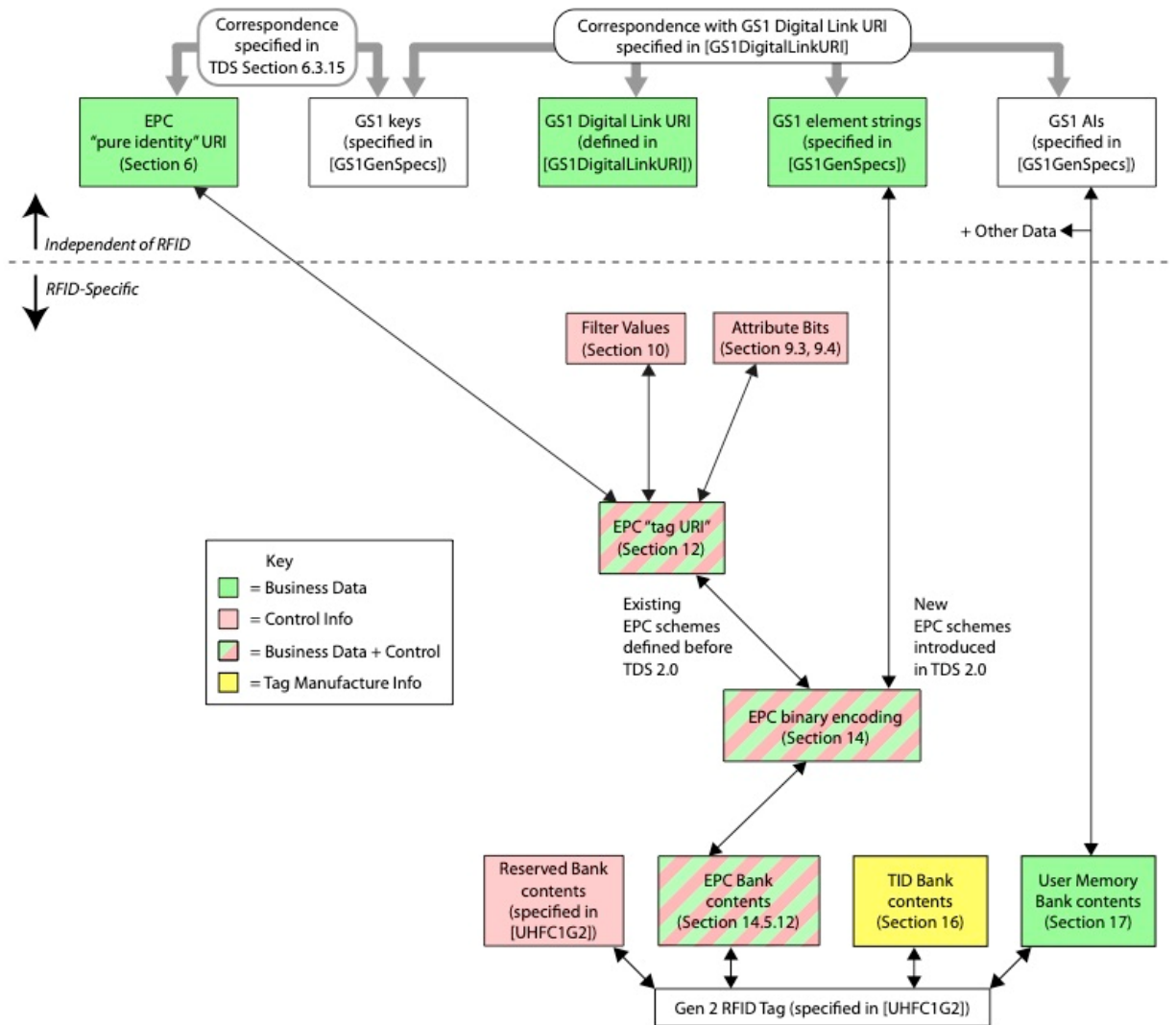
779 The term "Electronic Product Code" (or "EPC") is used when referring to the EPC regardless of the
780 concrete form used to represent it. The term "Pure Identity EPC URI" is used to refer specifically to
781 the text form the EPC takes within computer systems, including electronic documents, databases,
782 and electronic messages. The term "EPC Binary Encoding" is used specifically to refer to the form
783 the EPC takes within the memory of RFID tags.

784 The following figure illustrates the parts of TDS and how they fit together. (The colours in the figure
785 refer to the types of data that may be stored on RFID tags, explained further in Section [9.1](#)).

786 Note that filter values are included within the EPC Binary Encoding of many EPC schemes but are
787 specific to RFID tags and (with the exception of Application Level Events (ALE)), are not included at
788 any other layer of the GS1 System Architecture, nor are they present in element strings, pure
789 identity EPC URIs nor GS1 Digital Link URIs. They are intended primarily for low-level applications
790 rather than information exchange and do not reliably express logistic level (e.g. item, case, pallet),
791 nor should they be confused with the indicator digit of a GTIN-14 or the extension digit of an SSCC.
792 There are risks of relying on the filter value if this is not harmonised across the stakeholders who
793 use it.

794

Figure 3-1 Organisation of the EPC Tag Data Standard (TDS)



795

796
797

The first few sections define those aspects of the Electronic Product Code that are independent from RFID.

798
799

Section 4 provides an overview of the Electronic Product Code (EPC) and how it relates to other GS1 standards and the GS1 General Specifications.

800
801
802
803
804

Section 6 specifies the Pure Identity EPC URI form of the EPC. This is a textual form of the EPC, and is recommended for use in business applications and business documents as a universal identifier for any physical object for which visibility information is kept. In particular, this form is what is used as the "what" dimension of visibility data in the EPCIS specification, and is also available as an output from the Application Level Events (ALE) interface.

805
806

Section 7 specifies the correspondence between Pure Identity EPC URIs as defined in Section 6 and barcode element strings as defined in the GS1 General Specifications.

807
808

Section 7.11 specifies the Pure Identity Pattern URI, which is a syntax for representing sets of related EPCs, such as all EPCs for a given trade item regardless of serial number.

809
810

The remaining sections address topics that are specific to RFID, including RFID-specific forms of the EPC as well as other data apart from the EPC that may be stored on Gen 2 RFID tags.

811

Section 9 provides general information about the memory structure of Gen 2 RFID Tags.

812
813
814

Sections 10 and 11 specify "control" information that is stored in the EPC memory bank of Gen 2 tags along with a binary-encoded form of the EPC (EPC Binary Encoding). Control information is used by RFID data capture applications to guide the data capture process by providing hints about

815 what kind of object the tag is affixed to. Control information is not part of the EPC, and does not
816 comprise any part of the unique identity of a tagged object. There are two kinds of control
817 information specified: the "filter value" (Section 10) that makes it easier to read desired tags in an
818 environment where there may be other tags present, such as reading a pallet tag in the presence of
819 a large number of item-level tags, and "Attribute bits" (Sections 9.3 and 9.4) that provide additional
820 special attribute information such as alerting to the presence of hazardous material. The same
821 "Attribute bits" are available regardless of what kind of EPC is used, whereas the available "filter
822 values" are different depending on the type of EPC (and with certain types of EPCs, no filter value is
823 available at all).

824 Section 12 specifies the "tag" Uniform Resource Identifiers, which is a compact string representation
825 for the entire data content of the EPC memory bank of Gen 2 RFID Tags. This data content includes
826 the EPC together with "control" information as defined in Section 9.1. In the "tag" URI, the EPC
827 content of the EPC memory bank is represented in a form similar to the Pure Identity EPC URI.
828 Unlike the Pure Identity EPC URI, however, the "tag" URI also includes the control information
829 content of the EPC memory bank. The "tag" URI form is recommended for use in capture
830 applications that need to read control information in order to capture data correctly, or that need to
831 write the full contents of the EPC memory bank. "Tag" URIs are used in the Application Level Events
832 (ALE) interface, both as an input (when writing tags) and as an output (when reading tags).

833 Section 13 specifies the EPC Tag Pattern URI, which is a syntax for representing sets of related RFID
834 tags based on their EPC content, such as all tags containing EPCs for a given range of serial
835 numbers for a given trade item.

836 Sections 14 and 9.2 specify the contents of the EPC memory bank of a Gen 2 RFID tag at the bit
837 level. Section 14 specifies how to translate between the "tag" URI and the EPC Binary Encoding. The
838 binary encoding is a bit-level representation of what is actually stored on the tag, and is also what is
839 carried via the Low Level Reader Protocol (LLRP) interface. Section 9.2 specifies how this binary
840 encoding is combined with Attribute bits and other control information in the EPC memory bank.

841 Section 16 specifies the binary encoding of the TID memory bank of Gen 2 RFID Tags.

842 Section 17 specifies the binary encoding of the User memory bank of Gen 2 RFID Tags.

843 4 The Electronic Product Code: A universal identifier for 844 physical objects

845 The Electronic Product Code is designed to facilitate business processes and applications that need
846 to manipulate visibility data – data about observations of physical objects. The EPC is a universal
847 identifier that provides a unique identity for any physical object. The EPC is designed to be unique
848 across all physical objects in the world, over all time, and across all categories of physical objects. It
849 is expressly intended for use by business applications that need to track all categories of physical
850 objects, whatever they may be.

851 By contrast, GS1 identification keys defined in the GS1 General Specifications [GS1GS] can identify
852 categories of objects (GTIN), unique objects (SSCC, GLN, GIAI, GSRN, CPID), or a hybrid (GRAI,
853 GDTI, GCN) that may identify either categories or unique objects depending on the absence or
854 presence of a serial number. (Two other keys, GINC and GSIN, identify logical groupings, not
855 physical objects.) The GTIN, as the only category identification key, requires a separate serial
856 number to uniquely identify an object but that serial number is not considered part of the
857 identification key.

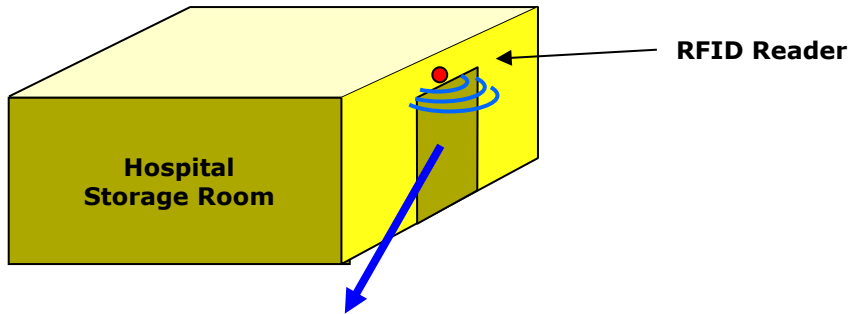
858 There is a well-defined correspondence between EPCs and GS1 keys. This allows any physical object
859 that is already identified by a GS1 key (or GS1 key + serial number combination) to be used in an
860 EPC context where any category of physical object may be observed. Likewise, it allows EPC data
861 captured in a broad visibility context to be correlated with other business data that is specific to the
862 category of object involved and which uses GS1 keys.

863 The remainder of this section elaborates on these points.

864 **4.1 The need for a universal identifier: an example**

865 The following example illustrates how visibility data arises, and the role the EPC plays as a unique
 866 identifier for any physical object. In this example, there is a storage room in a hospital that holds
 867 radioactive samples, among other things. The hospital safety officer needs to track what things have
 868 been in the storage room and for how long, in order to ensure that exposure is kept within
 869 acceptable limits. Each physical object that might enter the storage room is given a unique
 870 Electronic Product Code, which is encoded onto an RFID Tag affixed to the object. An RFID reader
 871 positioned at the storage room door generates visibility data as objects enter and exit the room, as
 872 illustrated below.

873 **Figure 4-1** Example Visibility Data Stream



| Visibility Data Stream at Storage Room Entrance | | | |
|---|----------|---------------------------------------|--|
| Time | In / Out | EPC | Comment |
| 8:23am | In | urn:epc:id:sgtin:9521141.012345.62852 | 10cc Syringe #62852 (trade item) |
| 8:52am | In | urn:epc:id:grai:9521141.54321.2528 | Pharma Tote #2528 (reusable transport) |
| 8:59am | In | urn:epc:id:sgtin:9521141.012345.1542 | 10cc Syringe #1542 (trade item) |
| 9:02am | Out | urn:epc:id:giai:9521141.17320508 | Infusion Pump #52 (fixed asset) |
| 9:32am | In | urn:epc:id:gsrc:9521141.0000010253 | Nurse Jones (service relation) |
| 9:42am | Out | urn:epc:id:gsrc:9521141.0000010253 | Nurse Jones (service relation) |
| 9:52am | In | urn:epc:id:gdti:9521141.00001.1618034 | Patient Smith's chart (document) |

874
 875 As the illustration shows, the data stream of interest to the safety officer is a series of events, each
 876 identifying a specific physical object and when it entered or exited the room. The unique EPC for
 877 each object is an identifier that may be used to drive the business process. In this example, the EPC
 878 (in Pure Identity EPC URI form) would be a primary key of a database that tracks the accumulated
 879 exposure for each physical object; each entry/exit event pair for a given object would be used to
 880 update the accumulated exposure database.

881 This example illustrates how the EPC is a single, *universal* identifier for any physical object. The
 882 items being tracked here include all kinds of things: trade items, reusable transports, fixed assets,
 883 service relations, documents, among others that might occur. By using the EPC, the application can
 884 use a single identifier to refer to any physical object, and it is not necessary to make a special case
 885 for each category of thing.

886 **4.2 Use of identifiers in a Business Data Context**

887 Generally speaking, an identifier is a member of set (or "namespace") of strings (names), such that
 888 each identifier is associated with a specific thing or concept in the real world. Identifiers are used
 889 within information systems to refer to the real world thing or concept in question. An identifier may
 890 occur in an electronic record or file, in a database, in an electronic message, or any other data
 891 context. In any given context, the producer and consumer must agree on which namespace of

892
893

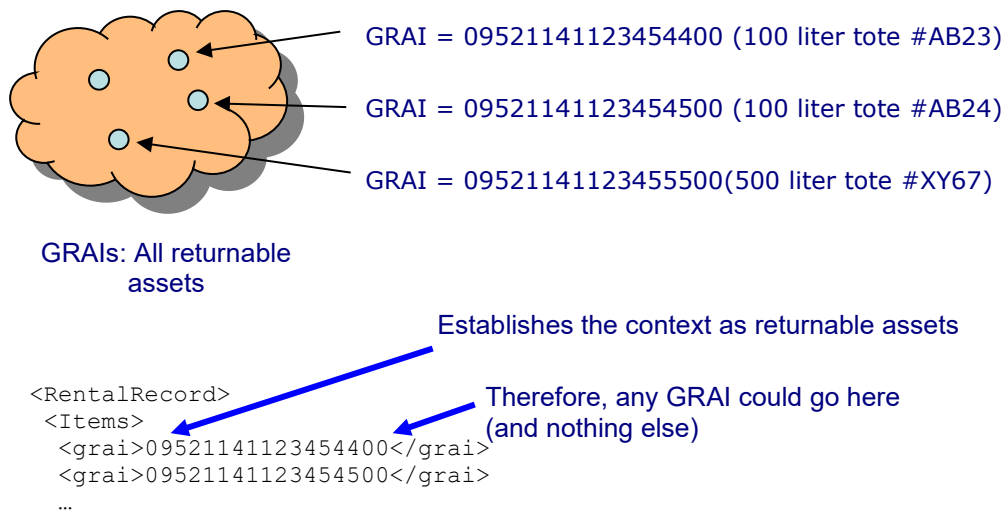
identifiers is to be used; within that context, any identifier belonging to that namespace may be used.

894
895
896
897
898
899
900

The keys defined in the GS1 General Specifications [GS1GS1] are each a namespace of identifiers for a particular category of real-world entity. For example, the Global Returnable Asset Identifier (GRAI) is a key that is used to identify returnable assets, such as plastic totes and pallet skids. The set of GRAI codes can be thought of as identifiers for the members of the set "all returnable assets." A GRAI code may be used in a context where only returnable assets are expected; e.g., in a rental agreement from a moving services company that rents returnable plastic crates to customers to pack during a move. This is illustrated below.

901

Figure 4-2 Illustration of GRAI Identifier Namespace

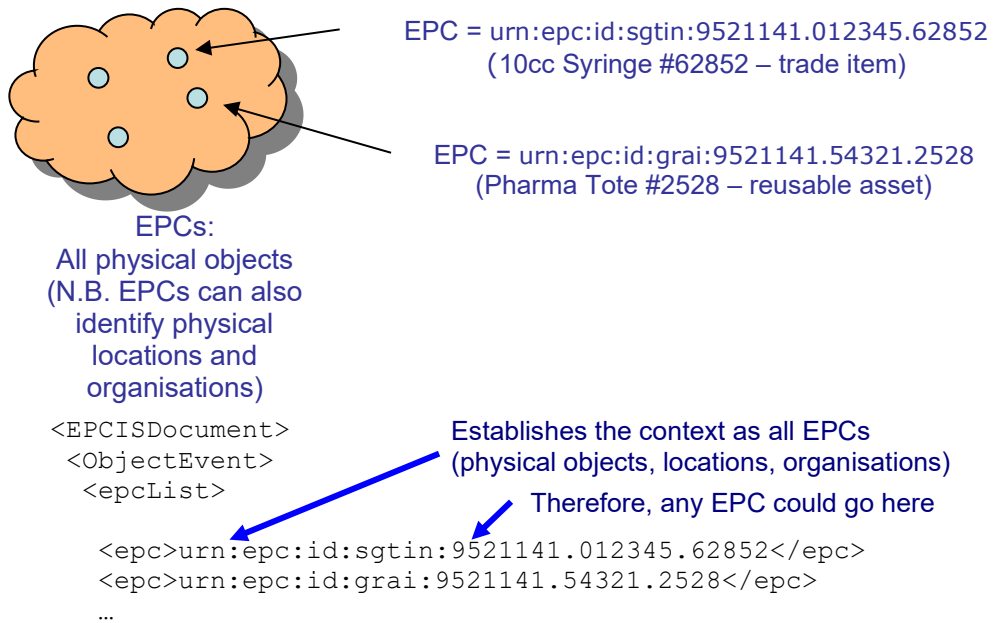


902
903
904
905

The upper part of the figure illustrates the GRAI identifier namespace. The lower part of the figure shows how a GRAI might be used in the context of a rental agreement, where only a GRAI is expected.

906

Figure 4-3 Illustration of EPC Identifier Namespace



907

908

909

910

911

912

913

914

915

916

In contrast, the EPC namespace is a space of identifiers for *any* physical object, physical location or organisation. The set of EPCs can be thought of as identifiers for the members of the set "all physical objects, physical locations or organisations." EPCs are used in contexts where any type of physical object may appear, such as in the set of observations arising in the hospital storage room example above. Note that the EPC URI as illustrated in [Figure 4-3](#) includes strings such as *sgtin*, *grai*, and so on as part of the EPC URI identifier. This is in contrast to GS1 Keys, where no such indication is part of the key itself; instead, this is indicated outside of the key, such as in the XML element name *<grai>* in the example in [Figure 4-2](#) in the Application Identifier (AI) that accompanies a GS1 key in a GS1 element string.

917

4.3 Relationship between EPCs and GS1 keys

918

919

920

921

922

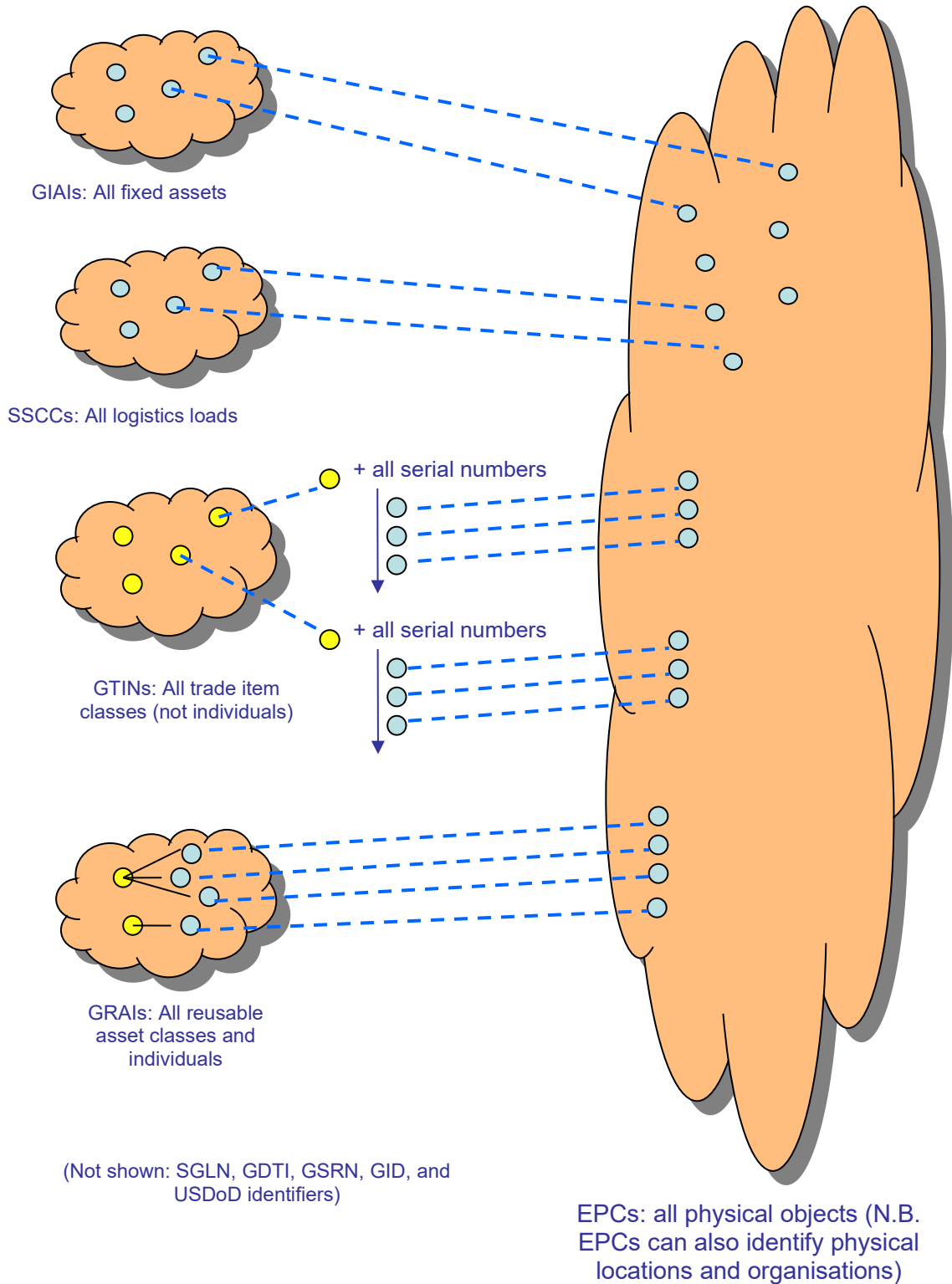
923

924

There is a well-defined relationship between EPCs and GS1 keys. For each GS1 key that denotes an individual physical object, there is a corresponding EPC, including both an EPC URI and a binary encoding for use in RFID tags. In addition, each GS1 key that denotes a class or grouping of physical objects has a corresponding URI form. These correspondences are formally defined by conversion rules specified in [Section 7](#), which define how to map a GS1 key to the corresponding EPC value and vice versa. The well-defined correspondence between GS1 keys and EPCs allows for seamless migration of data between GS1 key and EPC contexts as necessary.

925

Figure 4-4 Illustration of Relationship of GS1 key and EPC Identifier Namespaces



926

927

Not every GS1 key corresponds to an EPC, nor vice versa. Specifically:

928

929

930

931

932

- A Global Trade Item Number (GTIN) by itself does not correspond to an EPC, because a GTIN identifies a *class* of trade items, not an individual trade item. The combination of a GTIN and a unique serial number, however, *does* correspond to an EPC. This combination is called a Serialised Global Trade Item Number, or SGTIN. The GS1 General Specifications do not define the SGTIN as a GS1 key.

- 933 ■ In the GS1 General Specifications, the Global Returnable Asset Identifier (GRAI) can be used to
934 identify either a *class* of returnable assets, or an individual returnable asset, depending on
935 whether the optional serial number is included. Only the form that includes a serial number, and
936 thus identifies an individual, has a corresponding EPC. The same is true for the Global Document
937 Type Identifier (GDTI) and the Global Coupon Number (GCN) – hereafter, in this context,
938 "Serialised Global Coupon Number (SGCN)".
- 939 ■ There is an EPC corresponding to each Global Location Number (GLN), and there is also an EPC
940 corresponding to each combination of a GLN with an extension component. Collectively, these
941 EPCs are referred to as SGLNs.¹
- 942 ■ EPCs include identifiers for which there is no corresponding GS1 key. These include the General
943 Identifier and the US Department of Defense identifier and the Aerospace and Defense
944 Identifier.

945 The following table summarises the EPC schemes defined in this specification and their
946 correspondence to GS1 keys.

947 **Table 4-1** EPC Schemes and Corresponding GS1 keys

| EPC Scheme | Tag Encodings | Corresponding GS1 key | Typical use |
|------------|--|---|--|
| sgtin | sgtin-96 sgtin-198 sgtin+ dsgtin+ | GTIN key (plus added serial number) | Trade item |
| sscc | sscc-96 sscc+ | SSCC | Pallet load or other logistics unit load |
| sgln | sgln-96 sgln-195 sgln+ | GLN of physical location (with or without additional extension) | Location |
| grai | grai-96 grai-170 grai+ | GRAI (serial number mandatory) | Returnable/reusable asset |
| giai | giai-96 giai-202 giai+ | GIAI | Fixed asset |
| gsrn | gsrn-96 gsrn+ | GSRN – Recipient | Hospital admission or club membership |
| gsrnp | gsrnp-96 gsrnp+ | GSRN for service provider | Medical caregiver or loyalty club |
| gdti | gdti-96 gdti-113 (DEPRECATED) gdti-174 gdti+ | GDTI (serial number mandatory) | Document |
| cpi | cpi-96 cpi-var cpi+ | [none] | Technical industries (e.g. automotive) - components and parts |
| sgcn | sgcn-96 sgcn+ | GCN (serial number mandatory) | Coupon |

¹ Note that in this context, the letter "S" does not stand for "serialized" as it does in SGTIN. See Section [6.3.3](#) for an explanation.

| EPC Scheme | Tag Encodings | Corresponding GS1 key | Typical use |
|------------|-------------------------------|-----------------------|--|
| ginc | [none] | GINC | Logical grouping of goods intended for transport as a whole, assigned by a freight forwarder |
| gsin | [none] | GSIN | Logical grouping of logistic units travelling under one despatch advice and/or bill of lading |
| itip | itip-110 itip-212 itip+ | (8006) + (21) | One of multiple pieces comprising, and subordinate to, a whole (which is, in turn, identified by an SGTIN or the combination of AIs 01 + 21). |
| upui | [none] | GTIN + TPX | Pack identification to combat illicit trade |
| pqln | [none] | Party GLN | Identification of economic operator; identification of owning party or possessing party in the Chain of Custody (CoC) / Chain of Ownership (CoO) |
| gid | gid-96 | [none] | Unspecified |
| usdod | usdod-96 | [none] | US Dept of Defense supply chain |
| adi | adi-var | [none] | Aerospace and defense – aircraft and other parts and items |
| bic | [none] | [none] | Intermodal shipping containers |
| imovn | [none] | [none] | Vessel identificaton |

948 **4.4 Use of the EPC in the GS1 System Architecture**

949 The GS1 System Architecture [GS1Arch] is a collection of hardware, software, and data standards,
 950 together with shared network services, all in service of a common goal of enhancing business flows
 951 and computer applications. The GS1 System Architecture includes software standards at various
 952 levels of abstraction, from low-level interfaces to RFID reader devices all the way up to the business
 953 application level.

954 The EPC and related structures specified herein are intended for use at different levels within the
 955 GS1 System Architecture. Specifically:

- 956 ■ **Pure Identity EPC URI:** A representation of an EPC is as an Internet Uniform Resource
 957 Identifier (URI) called the Pure Identity EPC URI. Before TDS 2.0, the Pure Identity EPC URI was
 958 the preferred way to denote a specific physical object within business applications. The Pure
 959 Identity URI may also be used at the data capture level when the EPC is to be read from an
 960 RFID tag or other data carrier, in a situation where the additional "control" information present
 961 on an RFID tag is not needed.
- 962 ■ **GS1 Digital Link URI (as an alternative to Pure Identity EPC URIs):** Starting in TDS 2.0
 963 and EPCIS 2.0 / CBV 2.0, there is now recognition that a GS1 Digital Link URI (or a constrained
 964 subset of these, specifically at instance-level granularity and without additional data attributes)
 965 can provide an equivalent way to denote a specific physical object within business applications
 966 and traceability data. Furthermore, a GS1 Digital Link URI expresses GS1 Application Identifiers
 967 in a less convoluted syntax and can behave like a URL, linking to multiple kinds of online
 968 information and services, making use of resolver infrastructure for GS1 Digital Link and multiple
 969 link types defined in the GS1 Web vocabulary. GS1 Digital Link URIs can also be used as Linked
 970 Data identifiers to express factual claims (e.g. using terms defined in schema.org and the GS1
 971 Web Vocabulary).

- 972
973
974
975
976
977
978
979
- **EPC Tag URI:** The EPC memory bank of a Gen 2 RFID Tag contains the EPC plus additional "control information" that is used to guide the process of data capture from RFID tags. The EPC Tag URI is a URI string that denotes a specific EPC together with specific settings for the control information found in the EPC memory bank. In other words, the EPC Tag URI is a text equivalent of the entire EPC memory bank contents. The EPC Tag URI is typically used at the data capture level when reading from an RFID tag in a situation where the control information is of interest to the capturing application. It is also used when writing the EPC memory bank of an RFID tag, in order to fully specify the contents to be written.
 - **Binary Encoding:** The EPC memory bank of a Gen 2 RFID Tag actually contains a compressed encoding of the EPC and additional "control information" in a compact binary form. For the EPC schemes defined before TDS 2.0, there is a 1-to-1 translation between EPC Tag URIs and the binary contents of a Gen 2 RFID Tag. For the new EPC schemes and binary encodings introduced in TDS 2.0, no new EPC Tag URI syntax is defined and encoding/decoding is between the binary representation and the corresponding GS1 element strings or GS1 Digital Link URIs, as discussed in section [14.5](#). Normally, the binary encoding is only encountered at a very low level of software or hardware, and is translated to the EPC Tag URI or Pure Identity EPC URI form (for EPC schemes for which these are defined) before being presented to application logic. The binary encoding of the new EPC schemes introduced in TDS 2.0 would be more usually translated to GS1 element strings or GS1 Digital Link URIs. Starting in TDS 2.0 and EPCIS 2.0 / CBV 2.0, there is now recognition that a GS1 Digital Link URI (or a constrained subset of these, specifically at instance-level granularity and without additional data attributes) can provide an equivalent way to denote a specific physical object within business applications and traceability data.

980
981
982
983
984
985
986
987
988
989
990
991
992
993
994

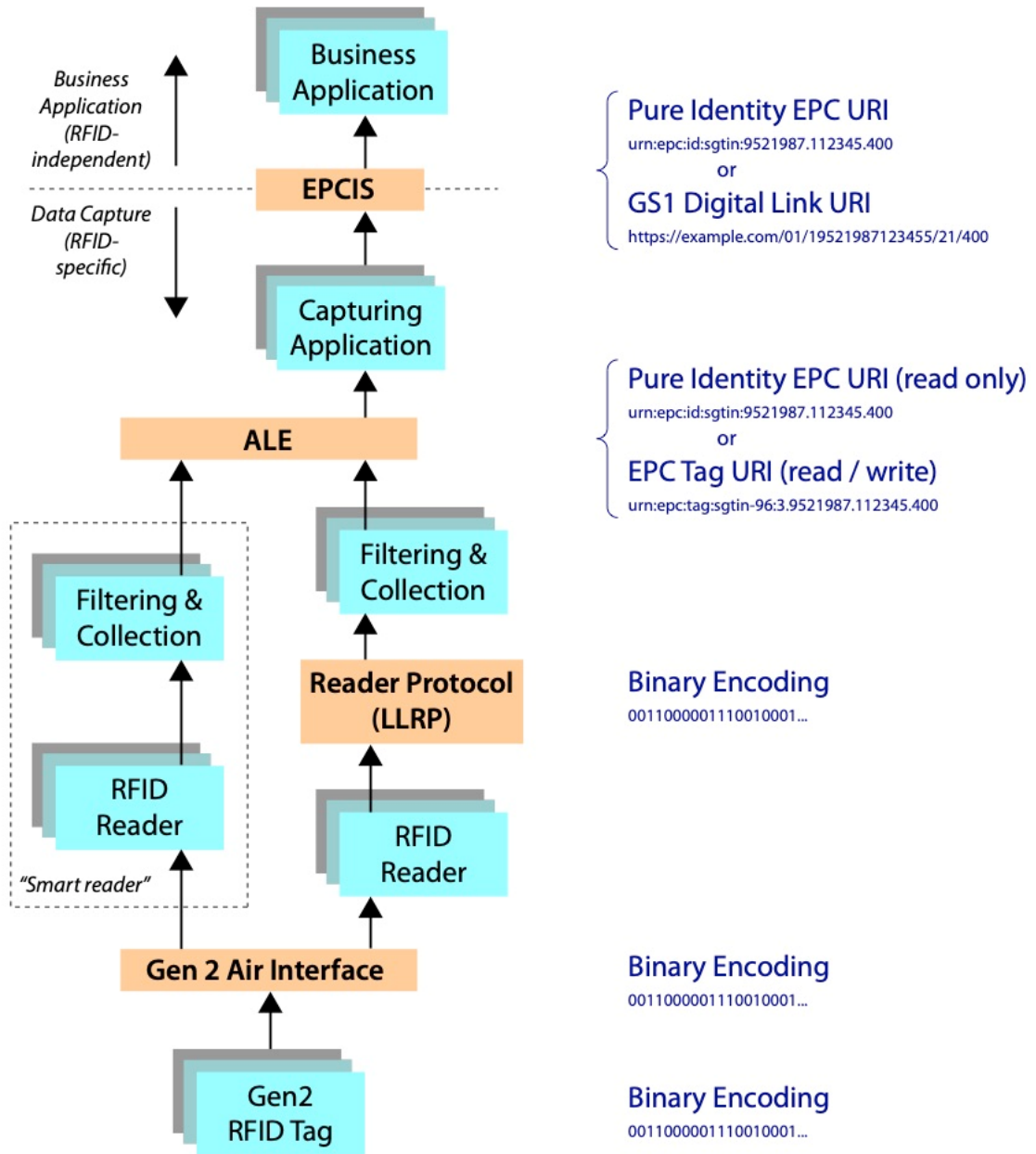
Note that both the Pure Identity EPC URI and the GS1 Digital Link URI are independent of choice of data carrier (e.g. EPC/RFID or barcodes), while the EPC Tag URI and the Binary Encoding are specific to Gen 2 RFID Tags because they include RFID-specific "control information" in addition to the unique EPC identifier.

995
996
997
998
999
1000

The figure below illustrates where these structures normally occur in relation to the layers of the GS1 System Architecture.

1001

Figure 4-5 EPC Structures used within the GS1 System Architecture



1002

5 Common grammar elements

The syntax of various URI forms defined herein is specified via ABNF grammar defined in [RFC5234] and [RFC7405]. The following grammar elements are used throughout this specification.

```

1006 ZeroComponent = "0"
1007 NonZeroDigit = "1" / "2" / "3" / "4" / "5" / "6" / "7" / "8" / "9"
1008 Digit = "0" / NonZeroDigit
1009 NonZeroComponent = NonZeroDigit 0*Digit
1010
1011 NumericComponent = ZeroComponent / NonZeroComponent
1012 PaddedNumericComponent = 1*Digit
  
```

```

1013 PaddedNumericComponentOrEmpty = 0*Digit
1014
1015 UpperAlpha = %x41-5A ; A-Z
1016 LowerAlpha = %x61-7A ; a-z
1017 OtherChar = "!" / "'" / "(" / ")" / "*" / "+" / "," / "-" / "." / ":" / ";" / "=" /
1018 " _
1019 UpperHexChar = Digit / "A" / "B" / "C" / "D" / "E" / "F"
1020 HexChar = UpperHexChar / "a" / "b" / "c" / "d" / "e" / "f"
1021 HexComponent = 1*UpperHexChar
1022 HexComponentOrEmpty = 0*UpperHexChar
1023 Escape = "%" HexChar HexChar
1024
1025 GS3A3Char = Digit / UpperAlpha / LowerAlpha / OtherChar / Escape
1026 GS3A3Component = 1*GS3A3Char
1027
1028 CPreChar = Digit / UpperAlpha / "-" / "%2F" / "%23"
1029 CPreComponent = 1*CPreChar
  
```

1030 The syntactic construct `GS3A3Component` is used to represent fields of GS1 codes that permit
 1031 alphanumeric and other characters as specified in Figure 7.12-1 of the GS1 General Specifications
 1032 (see Annex A.) Owing to restrictions on URN syntax as defined by [RFC2141], not all characters
 1033 permitted in the GS1 General Specifications may be represented directly in a URN. Specifically, the
 1034 characters " (double quote), % (percent), & (ampersand), / (forward slash), < (less than), >
 1035 (greater than), and ? (question mark) are permitted in the GS1 General Specifications but may not
 1036 be included directly in a URN. To represent one of these characters in a URN, escape notation must
 1037 be used in which the character is represented by a percent sign, followed by two hexadecimal digits
 1038 that give the ASCII character code for the character.

1039 The syntactic construct `CPreComponent` is used to represent fields that permit upper-case
 1040 alphanumeric and the characters hyphen, forward slash, and pound / number sign. Owing to
 1041 restrictions on URN syntax as defined by [RFC2141], not all of these characters may be represented
 1042 directly in a URN. Specifically, the characters # (pound / number sign) and / (forward slash) may
 1043 not be included directly in a URN. To represent one of these characters in a URN, escape notation
 1044 must be used in which the character is represented by a percent sign, followed by two hexadecimal
 1045 digits that give the ASCII character code for the character.

1046 6 EPC URI

1047 This section specifies the "pure identity URI" form of the EPC, or simply the "EPC URI." Before TDS
 1048 2.0, the EPC URI was the preferred way within an information system to denote a specific physical
 1049 object. Starting in TDS 2.0 and EPCIS 2.0 / CBV 2.0, there is now recognition that a GS1 Digital
 1050 Link URI (or a constrained subset of these, specifically at instance-level granularity and without
 1051 additional data attributes) is an equivalent way to denote a specific physical object within business
 1052 applications and traceability data, as discussed in further detail in section 4.4.

1053 The EPC URI is a string having the following form:

```

1054 urn:epc:id:scheme:component1.component2....
  
```

1055 where `scheme` names an EPC scheme, and `component1`, `component2`, and following parts are the
 1056 remainder of the EPC whose precise form depends on which EPC scheme is used. The available EPC
 1057 schemes are specified below in [Figure 6-1](#) in Section 6.3.

1058 An example of a specific EPC URI is the following, where the scheme is `sgtin`:

```

1059 urn:epc:id:sgtin:95211141.012345.4711
  
```

1060 Each EPC scheme provides a namespace of identifiers that can be used to identify physical objects
1061 of a particular type. Collectively, the EPC URIs from all schemes are unique identifiers for any type
1062 of physical object.

1063 **6.1 Use of the EPC URI**

1064 The structure of the EPC URI guarantees worldwide uniqueness of the EPC across all types of
1065 physical objects and applications. In order to preserve worldwide uniqueness, each EPC URI must be
1066 used in its entirety when a unique identifier is called for, and not broken into constituent parts nor
1067 the `urn:epc:id:` prefix abbreviated or dropped.

1068 When asking the question "do these two data structures refer to the same physical object?", where
1069 each data structure uses an EPC URI to refer to a physical object, the question may be answered
1070 simply by comparing the full EPC URI strings as specified in [RFC3986], Section 6.2. In most cases,
1071 the "simple string comparison" method suffices, though if a URI contains percent-encoding triplets
1072 the hexadecimal digits may require case normalisation as described in [RFC3986], Section 6.2.2.1.
1073 The construction of the EPC URI guarantees uniqueness across all categories of objects, provided
1074 that the URI is used in its entirety.

1075 In other situations, applications may wish to exploit the internal structure of an EPC URI for
1076 purposes of filtering, selection, or distribution. For example, an application may wish to query a
1077 database for all records pertaining to instances of a specific product identified by a GTIN. This
1078 amounts to querying for all EPCs whose GS1 Company Prefix and item reference components match
1079 a given value, disregarding the serial number component. Another example is found in the Object
1080 Name Service (ONS) [ONS], which uses the first component of an EPC to delegate a query to a
1081 "local ONS" operated by an individual company. This allows the ONS system to scale in a way that
1082 would be quite difficult if all ONS records were stored in a flat database maintained by a single
1083 organisation. Note that although GS1's ONS standard has not yet been deprecated or withdrawn, it
1084 is no longer maintained and the infrastructure for ONS is no longer supported by GS1 Global Office.
1085 The GS1 Digital Link standard [GS1DL] specifies not only a Web URI syntax for GS1 identifiers but
1086 also a resolver / resolution capability for linking a GS1 Digital Link URI to one or more sources of
1087 relevant information and services, as a modern successor to ONS.

1088 While the internal structure of the EPC may be exploited for filtering, selection, and distribution as
1089 illustrated above, it is essential that the EPC URI be used in its entirety when used as a unique
1090 identifier.

1091 **6.2 Assignment of EPCs to physical objects**

1092 The act of allocating a new EPC and associating it with a specific physical object is called
1093 "commissioning." It is the responsibility of applications and business processes that commission
1094 EPCs to ensure that the same EPC is never assigned to two different physical objects; that is, to
1095 ensure that commissioned EPCs are unique. Typically, commissioning applications will make use of
1096 databases that record which EPCs have already been commissioned and which are still available. For
1097 example, in an application that commissions SGTINs by assigning serial numbers sequentially, such
1098 a database might record the last serial number used for each base GTIN.

1099 Because visibility data and other business data that refers to EPCs may continue to exist long after a
1100 physical object ceases to exist, an EPC is ideally never reused to refer to a different physical object,
1101 even if the reuse takes place after the original object ceases to exist. There are certain situations,
1102 however, in which this is not possible; some of these are noted below. Therefore, applications that
1103 process historical data using EPCs should be prepared for the possibility that an EPC may be reused
1104 over time to refer to different physical objects, unless the application is known to operate in an
1105 environment where such reuse is prevented.

1106 Seven of the EPC schemes specified herein correspond to GS1 keys, and so EPCs from those
1107 schemes are used to identify physical objects that have a corresponding GS1 key. When assigning
1108 these types of EPCs to physical objects, all relevant GS1 rules must be followed in addition to the
1109 rules specified herein. This includes the GS1 General Specifications [GS1GS], the GTIN Management
1110 Standard, and so on. In particular, an EPC of this kind may only be commissioned by the licensee of
1111 the GS1 Company Prefix that is part of the EPC, or has been delegated the authority to do so by the
1112 GS1 Company Prefix licensee.

6.3 EPC URI syntax

This section specifies the syntax of an EPC URI.

The formal grammar for the EPC URI is as follows:

```
EPC-URI =
    SGTIN-URI /
    SSCC-URI /
    SGLN-URI /
    GRAI-URI /
    GIAI-URI /
    GSRN-URI /
    GDTI-URI /
    CPI-URI /
    SGCN-URI /
    GINC-URI /
    GSIN-URI /
    ITIP-URI /
    UPUI-URI /
    PGLN-URI /
    GID-URI /
    DOD-URI /
    ADI-URI /
    BIC-URI /
    IMOVN-URI
```

where the various alternatives on the right hand side are specified in the sections that follow.

Each EPC URI scheme is specified in one of the following subsections, as follows:

Figure 6-1 EPC Schemes and Where the Pure Identity Form is Defined

| EPC Scheme | Specified In | Corresponding GS1 key | Typical use |
|------------|-------------------------------|--|---------------------------------------|
| sgtin | Section 6.3.1 | GTIN (with added serial number) | Trade item |
| sscc | Section 6.3.2 | SSCC | Logistics unit |
| sgln | Section 6.3.3 | GLN (with or without additional extension) | Location ² |
| grai | Section 6.3.4 | GRAI (serial number mandatory) | Returnable asset |
| giai | Section 6.3.5 | GIAI | Fixed asset |
| gsrn | Section 6.3.6 | GSRN – Recipient | Hospital admission or club membership |

² While GLNs may be used to identify both locations and parties, the SGLN corresponds only to AI 414, which [GS1GS] specifies is to be used to identify locations, and not parties.

| EPC Scheme | Specified In | Corresponding GS1 key | Typical use |
|------------|--------------------------------|---------------------------------|--|
| gsrnp | Section 6.3.7 | GSRN – Provider | Medical caregiver or loyalty club |
| gdti | Section 6.3.8 | GDTI (serial number mandatory) | Document |
| cpi | Section 6.3.9 | [none] | Technical industries (e.g. automotive sector) for unique identification of parts and components |
| sgcn | Section 6.3.10 | GCN (serial number mandatory) | Coupon |
| ginc | Section 6.3.11 | GINC | Logical grouping of goods intended for transport as a whole, assigned by a freight forwarder |
| gsin | Section 6.3.12 | GSIN | Logical grouping of logistic units travelling under one despatch advice and/or bill of lading |
| itip | Section 6.3.13 | AI (8006) combined with AI (21) | One of multiple pieces comprising, and subordinate to, a whole (which is, in turn, identified by an SGTIN or the combination of AIs 01 + 21). |
| upui | Section 6.3.14 | GTIN and TPX | Pack identification to combat illicit trade |
| pglN | Section 6.3.15 | Party GLN – AI (417) | Identification of economic operator; identification of owning party or possessing party in the Chain of Custody (CoC) / Chain of Ownership (CoO) |
| gid | Section 6.3.16 | [none] | Unspecified |
| usdod | Section 6.3.17 | [none] | US Dept of Defense supply chain |
| adi | Section 6.3.18 | [none] | Aerospace and Defense sector for unique identification of aircraft and other parts and items |
| bic | Section 6.3.19 | [none] | Intermodal shipping containers |
| imovN | Section 6.3.20 | [none] | Vessel identificaton |

Note that no new Pure Identity EPC URI formats are defined for the new EPC schemes and binary encodings introduced in TDS 2.0.

1139
1140

6.3.1 Serialised Global Trade Item Number (SGTIN)

The Serialised Global Trade Item Number EPC scheme is used to assign a unique identity to an instance of a trade item, such as a specific instance of a product or SKU.

General syntax:

```
urn:epc:id:sgtin:CompanyPrefix.ItemRefAndIndicator.SerialNumber
```

Example:

```
urn:epc:id:sgtin:9521141.012345.4711
```

Grammar:

```
SGTIN-URI = %s"urn:epc:id:sgtin:" SGTINURIBody
```

```
SGTINURIBody = 2 (PaddedNumericComponent ".") GS3A3Component
```

The number of characters in the two `PaddedNumericComponent` fields must total 13 (not including any of the dot characters).

The Serial Number field of the SGTIN-URI is expressed as a `GS3A3Component`, which permits the representation of all characters permitted in the Application Identifier 21 Serial Number according to the GS1 General Specifications. SGTIN-URIs that are derived from 96-bit tag encodings, however, will have Serial Numbers that consist only of digits and which have no leading zeros (unless the entire serial number consists of a single zero digit). These limitations are described in the encoding procedures, and in Section [12.3.1](#).

The SGTIN consists of the following elements:

- The **GS1 Company Prefix**, assigned by GS1 to a managing entity or its delegates. This is the same as the GS1 Company Prefix digits within a GS1 GTIN key. See Section [7.3.2](#) for the case of a GTIN-8.
- The **Item Reference**, assigned by the managing entity to a particular object class. The Item Reference as it appears in the EPC URI is derived from the GTIN by concatenating the Indicator Digit of the GTIN (or a zero pad character, if the EPC URI is derived from a GTIN-8, GTIN-12, or GTIN-13) and the Item Reference digits, and treating the result as a single numeric string. See Section [7.3.2](#) for the case of a GTIN-8.
- The **Serial Number**, assigned by the managing entity to an individual object. The serial number is not part of the GTIN, but is formally a part of the SGTIN.

6.3.2 Serial Shipping Container Code (SSCC)

The Serial Shipping Container Code EPC scheme is used to assign a unique identity to a logistics handling unit, such as the aggregate contents of a shipping container or a pallet load.

General syntax:

```
urn:epc:id:sscc:CompanyPrefix.SerialReference
```

Example:

```
urn:epc:id:sscc:9521141.1234567890
```

Grammar:

```
SSCC-URI = %s"urn:epc:id:sscc:" SSCCURIbody
```

```
SSCCURIbody = PaddedNumericComponent "." PaddedNumericComponent
```

The number of characters in the two `PaddedNumericComponent` fields must total 17 (not including any of the dot characters).

1182 The SSCC consists of the following elements:

- 1183 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity. This is the same as the GS1
1184 Company Prefix digits within a GS1 SSCC key.
- 1185 ■ The **Serial Reference**, assigned by the managing entity to a particular logistics handling unit.
1186 The Serial Reference as it appears in the EPC URI is derived from the SSCC by concatenating
1187 the Extension Digit of the SSCC and the Serial Reference digits, and treating the result as a
1188 single numeric string.

1189 **6.3.3 Global Location Number With or Without Extension (SGLN)**

1190 The SGLN EPC scheme is used to assign a unique identity to a physical location, such as a specific
1191 building or a specific unit of shelving within a warehouse.

1192 **General syntax:**

1193 `urn:epc:id:sgln:CompanyPrefix.LocationReference.Extension`

1194 **Example:**

1195 `urn:epc:id:sgln:9521141.12345.400`

1196 **Grammar:**

1197 `SGLN-URI = %s"urn:epc:id:sgln:" SGLNURIBody`

1198 `SGLNURIBody = PaddedNumericComponent "." PaddedNumericComponentOrEmpty "."`
1199 `GS3A3Component`

1200 The number of characters in the two `PaddedNumericComponent` fields must total 12 (not including
1201 any of the dot characters).

1202 The Extension field of the SGLN-URI is expressed as a `GS3A3Component`, which permits the
1203 representation of all characters permitted in the Application Identifier 254 Extension according to
1204 the GS1 General Specifications. SGLN-URIs that are derived from 96-bit tag encodings, however,
1205 will have Extensions that consist only of digits and which have no leading zeros (unless the entire
1206 extension consists of a single zero digit). These limitations are described in the encoding
1207 procedures, and in Section [12.3.1](#).

1208 The SGLN consists of the following elements:

- 1209 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity. This is the same as the GS1
1210 Company Prefix digits within a GS1 GLN key.
- 1211 ■ The **Location Reference**, assigned uniquely by the managing entity to a specific physical
1212 location.
- 1213 ■ The **GLN Extension**, assigned by the managing entity to an individual unique location. If the
1214 entire GLN Extension is just a single zero digit, it indicates that the SGLN stands for a GLN,
1215 without an extension.

1216 **!** **Non-Normative:** Explanation (non-normative): Note that the letter "S" in the term "SGLN"
1217 does not stand for "serialised" as it does in SGTIN. This is because a GLN without an
1218 extension also identifies a unique location, as opposed to a class of locations, and so both
1219 GLN and GLN with extension may be considered as "serialised" identifiers. The term SGLN
1220 merely distinguishes the EPC form, which can be used either for a GLN by itself or GLN with
1221 extension, from the term GLN which always refers to the unextended GLN identifier. The
1222 letter "S" does not stand for anything.

1223 **6.3.4 Global Returnable Asset Identifier (GRAI)**

1224 The Global Returnable Asset Identifier EPC scheme is used to assign a unique identity to a specific
1225 returnable asset, such as a reusable shipping container or a pallet skid.

1226 **General syntax:**
1227 `urn:epc:id:grai:CompanyPrefix.AssetType.SerialNumber`

1228 **Example:**
1229 `urn:epc:id:grai:9521141.12345.400`

1230 **Grammar:**
1231 `GRAI-URI = %s"urn:epc:id:grai:" GRAIURIBody`
1232 `GRAIURIBody = PaddedNumericComponent "." PaddedNumericComponentOrEmpty "."`
1233 `GS3A3Component`

1234 The number of characters in the two `PaddedNumericComponent` fields must total 12 (not including
1235 any of the dot characters).

1236 The Serial Number field of the GRAI-URI is expressed as a `GS3A3Component`, which permits the
1237 representation of all characters permitted in the Serial Number according to the GS1 General
1238 Specifications. GRAI-URIs that are derived from 96-bit tag encodings, however, will have Serial
1239 Numbers that consist only of digits and which have no leading zeros (unless the entire serial number
1240 consists of a single zero digit). These limitations are described in the encoding procedures, and in
1241 Section [12.3.1](#).

1242 The GRAI consists of the following elements:

- 1243 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity. This is the same as the GS1
1244 Company Prefix digits within a GS1 GRAI key.
- 1245 ■ The **Asset Type**, assigned by the managing entity to a particular class of asset.
- 1246 ■ The **Serial Number**, assigned by the managing entity to an individual object. Because an EPC
1247 always refers to a specific physical object rather than an asset class, the serial number is
1248 mandatory in the GRAI-EPC.

1249 **6.3.5 Global Individual Asset Identifier (GIAI)**

1250 The Global Individual Asset Identifier EPC scheme is used to assign a unique identity to a specific
1251 asset, such as a forklift or a computer.

1252 **General syntax:**
1253 `urn:epc:id:giai:CompanyPrefix.IndividualAssetReference`

1254 **Example:**
1255 `urn:epc:id:giai:9521141.12345400`

1256 **Grammar:**
1257 `GIAI-URI = %s"urn:epc:id:giai:" GIAIURIBody`
1258 `GIAIURIBody = PaddedNumericComponent "." GS3A3Component`

1259 The Individual Asset Reference field of the GIAI-URI is expressed as a `GS3A3Component`, which
1260 permits the representation of all characters permitted in the Serial Number according to the GS1
1261 General Specifications. GIAI-URIs that are derived from 96-bit tag encodings, however, will have
1262 Serial Numbers that consist only of digits and which have no leading zeros (unless the entire serial
1263 number consists of a single zero digit). These limitations are described in the encoding procedures,
1264 and in Section [12.3.1](#).

1265 The GIAI consists of the following elements:

- 1266 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity. The Company Prefix is the
1267 same as the GS1 Company Prefix digits within a GS1 GIAI key.
- 1268 ■ The **Individual Asset Reference**, assigned uniquely by the managing entity to a specific asset.

6.3.6 Global Service Relation Number – Recipient (GSRN)

The Global Service Relation Number EPC scheme is used to assign a unique identity to a service recipient.

General syntax:

`urn:epc:id:gsrcn:CompanyPrefix.ServiceReference`

Example:

`urn:epc:id:gsrcn:9521141.1234567890`

Grammar:

GSRN-URI = %s"urn:epc:id:gsrcn:" GSRNURIBody

GSRNURIBody = PaddedNumericComponent "." PaddedNumericComponent

The number of characters in the two `PaddedNumericComponent` fields must total 17 (not including any of the dot characters).

The GSRN consists of the following elements:

- The **GS1 Company Prefix**, assigned by GS1 to a managing entity. This is the same as the GS1 Company Prefix digits within a GS1 GSRN key.
- The **Service Reference**, assigned by the managing entity to a particular service recipient.

6.3.7 Global Service Relation Number – Provider (GSRNP)

The Global Service Relation Number – Provider (GSRNP) EPC scheme is used to assign a unique identity to a service provider.

General syntax:

`urn:epc:id:gsrcnp:CompanyPrefix.ServiceReference`

Example:

`urn:epc:id:gsrcnp:9521141.1234567890`

Grammar:

GSRNP-URI = %s"urn:epc:id:gsrcnp:" GSRNURIBody

GSRNPURIBody = PaddedNumericComponent "." PaddedNumericComponent

The number of characters in the two `PaddedNumericComponent` fields must total 17 (not including any of the dot characters).

The GSRNP consists of the following elements:

- The **GS1 Company Prefix**, assigned by GS1 to a managing entity. This is the same as the GS1 Company Prefix digits within a GS1 GSRN key.
- The **Service Reference**, assigned by the managing entity to a particular service provider.

6.3.8 Global Document Type Identifier (GDTI)

The Global Document Type Identifier EPC scheme is used to assign a unique identity to a specific document, such as land registration papers, an insurance policy, and others.

General syntax:

`urn:epc:id:gdti:CompanyPrefix.DocumentType.SerialNumber`

1306

Example:

1307

```
urn:epc:id:gdti:9521141.12345.400
```

1308

Grammar:

1309

```
GDTI-URI = %s"urn:epc:id:gdti:" GDTIURIBody
```

1310

```
GDTIURIBody = PaddedNumericComponent "." PaddedNumericComponentOrEmpty "."
```

1311

```
GS3A3Component
```

1312

The number of characters in the first `PaddedNumericComponent` field and the

1313

`PaddedNumericComponentOrEmpty` field must total 12 (not including any of the dot characters).

1314

The Serial Number field of the GDTI-URI is expressed as a `GS3A3Component`, which permits the

1315

representation of all characters permitted in the Serial Number according to the GS1 General

1316

Specifications. GDTI-URIs that are derived from 96-bit tag encodings, however, will have Serial

1317

Numbers that have no leading zeros (unless the entire serial number consists of a single zero digit).

1318

These limitations are described in the encoding procedures, and in Section [12.3.1](#).

1319

The GDTI consists of the following elements:

1320

- The **GS1 Company Prefix**, assigned by GS1 to a managing entity. This is the same as the GS1 Company Prefix digits within a GS1 GDTI key.

1321

1322

- The **Document Type**, assigned by the managing entity to a particular class of document.

1323

- The **Serial Number**, assigned by the managing entity to an individual document. Because an

1324

EPC always refers to a specific document rather than a document class, the serial number is

1325

mandatory in the GDTI-EPC.

1326

6.3.9 Component / Part Identifier (CPI)

1327

The Component / Part EPC identifier is designed for use by the technical industries (including the automotive sector) for the unique identification of parts or components.

1328

1329

The CPI EPC construct provides a mechanism to directly encode unique identifiers in RFID tags and

1330

to use the URI representations at other layers of the GS1 System Architecture.

1331

General syntax:

1332

```
urn:epc:id:cpi:CompanyPrefix.ComponentPartReference.Serial
```

1333

Example:

1334

```
urn:epc:id:cpi:9521141.123ABC.123456789
```

1335

```
urn:epc:id:cpi:9521141.123456.123456789
```

1336

Grammar:

1337

```
CPI-URI = %s"urn:epc:id:cpi:" CPIURIBody
```

1338

```
CPIURIBody = PaddedNumericComponent "." CPreComponent "." NumericComponent
```

1339

The Component / Part Reference field of the CPI-URI is expressed as a `CPreComponent`, which

1340

permits the representation of all characters permitted in the Component / Part Reference according

1341

to the GS1 General Specifications. CPI-URIs that are derived from 96-bit tag encodings, however,

1342

will have Component / Part References that consist only of digits, with no leading zeros, and whose

1343

length is less than or equal to 15 minus the length of the GS1 Company Prefix. These limitations are

1344

described in the encoding procedures, and in Section [12.3.1](#).

1345

The CPI consists of the following elements:

1346

- The **GS1 Company Prefix**, assigned by GS1 to a managing entity or its delegates.

1347

- The **Component/Part Reference**, assigned by the managing entity to a particular object class.

1348

- The **Serial Number**, assigned by the managing entity to an individual object.

1349 The managing entity or its delegates ensure that each CPI is issued to no more than one physical
1350 component or part. Typically this is achieved by assigning a component/part reference to designate
1351 a collection of instances of a part that share the same form, fit or function and then issuing serial
1352 number values uniquely within each value of component/part reference in order to distinguish
1353 between such instances.

1354 **6.3.10 Serialised Global Coupon Number (SGCN)**

1355 The Global Coupon Number EPC scheme is used to assign a unique identity to a coupon.

1356 **General syntax:**

1357 `urn:epc:id:sgcn:CompanyPrefix.CouponReference.SerialComponent`

1358 **Example:**

1359 `urn:epc:id:sgcn:4012345.67890.04711`

1360 **Grammar:**

1361 `SGCN-URI = %s"urn:epc:id:sgcn:" SGCNURIBody`

1362 `SGCNURIBody = PaddedNumericComponent "." PaddedNumericComponentOrEmpty "."`
1363 `PaddedNumericComponent`

1364 The number of characters in the first `PaddedNumericComponent` field and the
1365 `PaddedNumericComponentOrEmpty` field must total 12 (not including any of the dot characters).

1366 The Serial Component field of the SGCN-URI is expressed as a `PaddedNumericComponent`, which
1367 may contain up to 12 digits, including leading zeros, as per the GS1 General Specifications. The
1368 SGCN consists of the following elements:

- 1369 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity. This is the same as the GS1
1370 Company Prefix digits within a GS1 GCN key.
- 1371 ■ The **Coupon Reference**, assigned by the managing entity for the coupon.
- 1372 ■ The **Serial Component**, assigned by the managing entity to a unique instance of the coupon.
1373 Because an EPC always refers to a specific coupon rather than a coupon class, the serial number
1374 is mandatory in the SGCN-EPC.

1375 **6.3.11 Global Identification Number for Consignment (GINC)**

1376 The Global Identification Number for Consignment EPC scheme is used to assign a unique identity to
1377 a logical grouping of goods (one or more physical entities) that has been consigned to a freight
1378 forwarder and is intended to be transported as a whole.

1379 **General syntax:**

1380 `urn:epc:id:ginc:CompanyPrefix.ConsignmentReference`

1381 **Example:**

1382 `urn:epc:id:ginc:9521141.xyz3311cba`

1383 **Grammar:**

1384 `GINC-URI = %s"urn:epc:id:ginc:" GINCURIBody`

1385 `GINCURIBody = PaddedNumericComponent "." GS3A3Component`

1386 The Consignment Reference field of the GINC-URI is expressed as a `GS3A3Component`, which
1387 permits the representation of all characters permitted in the Serial Number according to the GS1
1388 General Specifications.

1389 The GINC consists of the following elements:

- 1390 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity. The Company Prefix is the
- 1391 same as the GS1 Company Prefix digits within a GS1 GINC key.
- 1392 ■ The **Consignment Reference**, assigned uniquely by the freight forwarder.

1393 6.3.12 Global Shipment Identification Number (GSIN)

1394 The Global Shipment Identification Number EPC scheme is used to assign a unique identity to a
 1395 logical grouping of logistic units for the purpose of a transport shipment from that consignor (seller)
 1396 to the consignee (buyer).

1397 **General syntax:**

1398 `urn:epc:id:gsin:CompanyPrefix.ShipperReference`

1399 **Example:**

1400 `urn:epc:id:gsin:9521141.123456789`

1401 **Grammar:**

1402 `GSIN-URI = %s"urn:epc:id:gsin:" GSINURIBody`

1403 `GSINURIBody = PaddedNumericComponent "." PaddedNumericComponent`

1404 The number of characters in the two `PaddedNumericComponent` fields must total 16 (not including
 1405 the dot character).

1406 The GSIN consists of the following elements:

- 1407 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity. This is the same as the GS1
- 1408 Company Prefix digits within a GS1 GSIN key.
- 1409 ■ The **Shipper Reference**, assigned by the consignor (seller) of goods.

1410 6.3.13 Individual Trade Item Piece (ITIP)

1411 The Individual Trade Item Piece EPC scheme is used to assign a unique identity to a subordinate
 1412 element of a trade item (e.g., left and right shoes, suit trousers and jacket, DIY trade item consisting
 1413 of several physical units), the latter of which comprises multiple pieces.

1414 **General syntax:**

1415 `urn:epc:id:itip:CompanyPrefix.ItemRefAndIndicator.Piece.Total.SerialNumber`

1416 **Example:**

1417 `urn:epc:id:itip:9521141.012345.01.02.987`

1418 **Grammar:**

1419 `ITIP-URI = %s"urn:epc:id:itip:" ITIPURIBody`

1420 `ITIPURIBody = 4(PaddedNumericComponent ".") GS3A3Component`

1421 The number of characters in the first two `PaddedNumericComponent` fields must total 13 (not
 1422 including any of the dot characters).

1423 The number of characters in each of the last two `PaddedNumericComponent` fields must be exactly
 1424 2 (not including any of the dot characters).

1425 The combined number of characters in the four `PaddedNumericComponent` fields must total 17
 1426 (not including any of the dot characters).

1427 The Serial Number field of the ITIP-URI is expressed as a `GS3A3Component`, which permits the
 1428 representation of all characters permitted in the Application Identifier 21 Serial Number according to

1429 the GS1 General Specifications. ITIP-URIs that are derived from 110-bit tag encodings, however,
 1430 will have Serial Numbers that consist only of digits and which have no leading zeros (unless the
 1431 entire serial number consists of a single zero digit). These limitations are described in the encoding
 1432 procedures, and in Section [7.3.1](#).

1433 The ITIP consists of the following elements:

- 1434 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity or its delegates. This is the
 1435 same as the GS1 Company Prefix digits within a GS1 GTIN key. See Section [2.3.2](#) for the case
 1436 of a GTIN-8.
- 1437 ■ The **Item Reference**, assigned by the managing entity to a particular object class. The Item
 1438 Reference as it appears in the EPC URI is derived from the GTIN by concatenating the Indicator
 1439 Digit of the GTIN (or a zero pad character, if the EPC URI is derived from a GTIN-8, GTIN-12, or
 1440 GTIN-13) and the Item Reference digits, and treating the result as a single numeric string. See
 1441 Section [2.3.2](#) for the case of a GTIN-8.
- 1442 ■ The **Piece Number**
- 1443 ■ The **Total** Quantity of Pieces subordinate to the GTIN
- 1444 ■ The **Serial Number**, assigned by the managing entity to an individual object. The serial number
 1445 is not part of the GTIN, but is formally a part of both the SGTIN and the ITIP.

1446 **6.3.14 Unit Pack Identifier (UPUI)**

1447 The Unit Pack Identifier EPC scheme is used to uniquely identify an individual item for tobacco
 1448 traceability in accordance with EU 2018/574.

1449 **General syntax:**

1450 `urn:epc:id:upui:CompanyPrefix.ItemRefAndIndicator.TPX`

1451 **Example:**

1452 `urn:epc:id:upui:9521141.089456.51qIgY)%3C%26Jp3*j7'SDB`

1453 **Grammar:**

1454 `UPUI-URI = %s"urn:epc:id:upui:" UPUI-URIBody`

1455 `UPUI-URIBody = 2(PaddedNumericComponent ".") GS3A3Component`

1456 The number of characters in the first two `PaddedNumericComponent` fields must total 13 (not
 1457 including any of the dot characters).

1458 The `TPX` field of the UPUI-URI is expressed as a `GS3A3Component`, which permits the
 1459 representation of all characters permitted in Application Identifier (235), Third Party Controlled,
 1460 Serialised Extension of GTIN, according to the GS1 General Specifications.

1461 The UPUI consists of the following elements:

- 1462 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity or its delegates. This is the
 1463 same as the GS1 Company Prefix digits within a GS1 GTIN key. See Section [7.3.2](#) for the case
 1464 of a GTIN-8.
- 1465 ■ The **Item Reference**, assigned by the managing entity to a particular object class. The Item
 1466 Reference as it appears in the EPC URI is derived from the GTIN by concatenating the Indicator
 1467 Digit of the GTIN (or a zero pad character, if the EPC URI is derived from a GTIN-8, GTIN-12, or
 1468 GTIN-13) and the Item Reference digits, and treating the result as a single numeric string. See
 1469 Section [7.3.2](#) for the case of a GTIN-8.
- 1470 ■ The **Third Party Controlled, Serialised Extension of GTIN**, assigned by a third party
 1471 managing entity to an individual object to uniquely identify an individual item for tobacco
 1472 traceability in accordance with EU 2018/574.

1473 6.3.15 Global Location Number of Party (PGLN)

1474 The PGLN EPC scheme is used to assign a unique identity to a party, such as a an economic
1475 operator or a cost center.

1476 **General syntax:**

1477 `urn:epc:id:pglن:CompanyPrefix.PartyReference`

1478 **Example:**

1479 `urn:epc:id:pglن:9521141.89012`

1480 **Grammar:**

1481 `PGLN-URI = %s"urn:epc:id:pglن:" PGLNURIBody`

1482 `PGLNURIBody = PaddedNumericComponent "." PaddedNumericComponentOrEmpty`

1483 The number of characters in the first `PaddedNumericComponent` field and the
1484 `PaddedNumericComponentOrEmpty` field must total 12 (not including any of the dot characters).

1485 The PGLN consists of the following elements:

- 1486 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity. This is the same as the GS1
1487 Company Prefix digits within a GS1 GLN key.
- 1488 ■ The **Party Reference**, assigned uniquely by the managing entity to a specific party.

1489 6.3.16 General Identifier (GID)

1490 The General Identifier EPC scheme is independent of any specifications or identity scheme outside
1491 TDS.

1492 **General syntax:**

1493 `urn:epc:id:gid:ManagerNumber.ObjectClass.SerialNumber`

1494 **Example:**

1495 `urn:epc:id:gid:95100000.12345.400`

1496 **Grammar:**

1497 `GID-URI = %s"urn:epc:id:gid:" GIDURIBody`

1498 `GIDURIBody = 2(NumericComponent ".") NumericComponent`

1499 The GID consists of the following elements:

- 1500 ■ The **General Manager Number** identifies an organisational entity (essentially a company,
1501 manager or other organisation) that is responsible for maintaining the numbers in subsequent
1502 fields – Object Class and Serial Number. Note that a General Manager Number is *not* a GS1
1503 Company Prefix. A General Manager Number may only be used in GID EPCs. **NOTE that**
1504 **General Manager Number issuance has been discontinued**, effective June 2023.
- 1505 ■ The **Object Class** is used by an EPC managing entity to identify a class or "type" of thing. These
1506 object class numbers, of course, must be unique within each General Manager Number domain.
- 1507 ■ Finally, the **Serial Number** code, or serial number, is unique within each object class. In other
1508 words, the managing entity is responsible for assigning unique, non-repeating serial numbers
1509 for every instance within each object class.

1510 6.3.17 US Department of Defense Identifier (DOD)

1511 The US Department of Defense identifier is defined by the United States Department of Defense.
1512 This tag data construct may be used to encode 96-bit Class 1 tags for shipping goods to the United

1513 States Department of Defense by a supplier who has already been assigned a CAGE (Commercial
1514 and Government Entity) code.

1515 At the time of this writing, the details of what information to encode into these fields is explained in
1516 a document titled "United States Department of Defense Suppliers' Passive RFID Information Guide"
1517 [USDOD].

1518 Note that the DoD Guide explicitly recognises the value of cross-branch, globally applicable
1519 standards, advising that "suppliers that are EPCglobal subscribers and possess a unique [GS1]
1520 Company Prefix may use any of the identity types and encoding instructions described in the EPC™
1521 Tag Data Standards document to encode tags."

1522 **General syntax:**

1523 `urn:epc:id:usdod:CAGECodeOrDODAAC.SerialNumber`

1524 **Example:**

1525 `urn:epc:id:usdod:2S194.12345678901`

1526 **Grammar:**

1527 `DOD-URI = %s"urn:epc:id:usdod:" DODURIBody`

1528 `DODURIBody = CAGECodeOrDODAAC "." DoDSerialNumber`

1529 `CAGECodeOrDODAAC = CAGECode / DODAAC`

1530 `CAGECode = 5 (CAGECodeOrDODAACChar)`

1531 `DODAAC = 6 (CAGECodeOrDODAACChar)`

1532 `DoDSerialNumber = NumericComponent`

1533 `CAGECodeOrDODAACChar = Digit / %x41-48 / %x4A-4E / %x50-5A ; 0-9 A-H J-N P-Z`

1534 **6.3.18 Aerospace and Defense Identifier (ADI)**

1535 The variable-length Aerospace and Defense EPC identifier is designed for use by the aerospace and
1536 defense sector for the unique identification of parts or items. The existing unique identifier
1537 constructs are defined in the Air Transport Association (ATA) Spec 2000 standard [SPEC2000], and
1538 the US Department of Defense Guide to Uniquely Identifying items [UID]. The ADI EPC construct
1539 provides a mechanism to directly encode such unique identifiers in RFID tags and to use the URI
1540 representations in EPCIS and ALE.

1541 Within the Aerospace & Defense sector identification constructs supported by the ADI EPC,
1542 companies are uniquely identified by their Commercial And Government Entity (CAGE) code or by
1543 their Department of Defense Activity Address Code (DODAAC). The NATO CAGE (NCAGE) code is
1544 issued by NATO / Allied Committee 135 and is structurally equivalent to a CAGE code (five character
1545 uppercase alphanumeric excluding capital letters I and O) and is non-colliding with CAGE codes
1546 issued by the US Defense Logistics Information Service (DLIS). Note that in the remainder of this
1547 section, all references to CAGE apply equally to NCAGE.

1548 ATA Spec 2000 defines that a unique identifier may be constructed through the combination of the
1549 CAGE code or DODAAC together with either:

- 1550 ■ A serial number (SER) that is assigned uniquely within the CAGE code or DODAAC; or
- 1551 ■ An original part number (PNO) that is unique within the CAGE code or DODAAC and a sequential
1552 serial number (SEQ) that is uniquely assigned within that original part number.

1553 The US DoD Guide to Uniquely Identifying Items defines a number of acceptable methods for
1554 constructing unique item identifiers (UIIs). The UIIs that can be represented using the Aerospace
1555 and Defense EPC identifier are those that are constructed through the combination of a CAGE code
1556 or DODAAC together with either:

- 1557 ■ a serial number that is unique within the enterprise identifier. (UII Construct #1)

- 1558 ■ an original part number and a serial number that is unique within the original part number (a
1559 subset of UII Construct #2)

1560 Note that the US DoD UID guidelines recognise a number of unique identifiers based on GS1
1561 identifier keys as being valid UIDs. In particular, the SGTIN (GTIN + Serial Number), GIAI, and
1562 GRAI with full serialisation are recognised as valid UIDs. These may be represented in EPC form
1563 using the SGTIN, GIAI, and GRAI EPC schemes as specified in Sections [6.3.1](#), [6.3.5](#), and [6.3.4](#),
1564 respectively; the ADI EPC scheme is *not* used for this purpose. Conversely, the US DoD UID
1565 guidelines also recognise a wide range of enterprise identifiers issued by various issuing agencies
1566 other than those described above; such UIDs do not have a corresponding EPC representation.

1567 For purposes of identification via RFID of those aircraft parts that are traditionally not serialised or
1568 not required to be serialised for other purposes, the ADI EPC scheme may be used for assigning a
1569 unique identifier to a part. In this situation, the first character of the serial number component of
1570 the ADI EPC SHALL be a single '#' character. This is used to indicate that the serial number does not
1571 correspond to the serial number of a traditionally serialised part because the '#' character is not
1572 permitted to appear within the values associated with either the SER or SEQ text element identifiers
1573 in ATA Spec 2000 standard.

1574 For parts that are traditionally serialised / required to be serialised for purposes other than having a
1575 unique RFID identifier, and for all usage within US DoD UID guidelines, the '#' character SHALL NOT
1576 appear within the serial number element.

1577 The ATA Spec 2000 standard recommends that companies serialise uniquely within their CAGE code.
1578 For companies who do serialise uniquely within their CAGE code or DODAAC, a zero-length string
1579 SHALL be used in place of the Original Part Number element when constructing an EPC.

1580 **General syntax:**

1581 `urn:epc:id:adi:CAGECodeOrDODAAC.OriginalPartNumber.Serial`

1582 **Examples:**

1583 `urn:epc:id:adi:2S194..12345678901`

1584 `urn:epc:id:adi:W81X9C.3KL984PX1.2WMA52`

1585 **Grammar:**

1586 `ADI-URI = %s"urn:epc:id:adi:" ADIURIBody`

1587 `ADIURIBody = CAGECodeOrDODAAC "." ADIComponent "." ADIExtendedComponent`

1588 `ADIComponent = 0*ADICChar`

1589 `ADIExtendedComponent = 0*1"%23" 1*ADICChar`

1590 `ADICChar = UpperAlpha / Digit / OtherADICChar`

1591 `OtherADICChar = "-" / "%2F"`

1592 `CAGECodeOrDODAAC` is defined in Section [6.3.17](#).

1593 **6.3.19 BIC Container Code (BIC)**

1594 *ISO 6346 is an [international standard](#) covering the coding, identification and marking of [intermodal](#)*
1595 *([shipping](#)) [containers](#) used within [containerized intermodal freight transport](#). The standard*
1596 *establishes a visual identification system for every container that includes a unique serial number*
1597 *(with [check digit](#)), the owner, a country code, a size, type and equipment category as well as any*
1598 *operational marks. The standard is managed by the [International Container Bureau](#) (BIC).*

1599 (source: https://en.wikipedia.org/wiki/ISO_6346#Identification_System)

1600 The BIC consists of the following elements:

- 1601 ■ The **owner code** consists of three capital letters of the Latin alphabet to indicate the owner or
1602 principal operator of the container. Such code needs to be registered at the [Bureau International](#)
1603 [des Conteneurs](#) in Paris to ensure uniqueness worldwide.

- 1604
- 1605
- 1606
- 1607
- 1608
- 1609
- 1610
- 1611
- 1612
- The **equipment category identifier** consists of one of the following capital letters of the Latin alphabet:
 - U for all freight containers
 - J for detachable freight container-related equipment
 - Z for trailers and chassis
 - The **serial number** consists of 6 numeric digits, assigned by the owner or operator, uniquely identifying the container within that owner/operator's fleet.
 - The **check digit** consists of one numeric digit providing a means of validating the recording and transmission accuracies of the owner code and serial number.

1613 The individual elements of the BIC are not separated by dots (".") in the EPC URI syntax.

1614 **General syntax:**

1615 urn:epc:id:bic:*BICContainerCode*

1616 **Example:**

1617 urn:epc:id:bic:CSQU3054383

1618 **Grammar:**

1619 BIC-URI = %s"urn:epc:id:bic:" BICURIBody

1620 BICURIBody = OwnerCode EquipCatId SerialNumber CheckDigit

1621 OwnerCode = 3(OwnerCodeChar)

1622 EquipCatId = CatIdChar

1623 SerialNumber = 6(Digit)

1624 CheckDigit = Digit

1625 OwnerCodeChar = %x41-48 / %x4A-4E / %x50-5A ; A-H J-N P-Z

1626 CatIdChar = "J" / "U" / "Z"

1627 **6.3.20 IMO Vessel Number (IMOVN)**

1628 *The IMO (International Maritime Organization) ship identification number scheme was introduced in*
 1629 *1987 through adoption of resolution A.600(15), as a measure aimed at enhancing "maritime safety,*
 1630 *and pollution prevention and to facilitate the prevention of maritime fraud". It aimed at assigning a*
 1631 *permanent number to each ship for identification purposes. That number would remain unchanged*
 1632 *upon transfer of the ship to other flag(s) and would be inserted in the ship's certificates. When*
 1633 *made mandatory, through SOLAS regulation XI/3 (adopted in 1994), specific criteria of passenger*
 1634 *ships of 100 gross tonnage and upwards and all cargo ships of 300 gross tonnage and upwards were*
 1635 *agreed.*

1636
 1637 *SOLAS regulation XI-1/3 requires ships' identification numbers to be permanently marked in a*
 1638 *visible place either on the ship's hull or superstructure. Passenger ships should carry the marking on*
 1639 *a horizontal surface visible from the air. Ships should also be marked with their ID numbers*
 1640 *internally.*

1641 *This number is assigned to the total portion of the hull enclosing the machinery space and is the*
 1642 *determining factor, should additional sections be added.*

1643 *The IMO number is never reassigned to another ship and is shown on the ship's certificates.*

1644 (source: <http://www.imo.org/en/OurWork/MSAS/Pages/IMO-identification-number-scheme.aspx>)

1645 The IMOVN consists of the following element:

- 1646
- a unique, **seven-digit vessel number**.

1647 **General syntax:**
 1648 `urn:epc:id:imovn:IMOVesselNumber`

1649 **Example:**
 1650 `urn:epc:id:imovn:9176187`

1651 **Grammar:**
 1652 `IMOVN-URI = %s"urn:epc:id:imovn:" IMOVNURIBody`
 1653 `IMOVNURIBody = VesselNumber`
 1654 `VesselNumber = 7(Digit)`

1655 6.4 EPC Class URI Syntax

1656 This section specifies the syntax of an EPC Class URI.

1657 The formal grammar for the EPC class URI is as follows:

1658 `EPCClass-URI = LGTIN-URI`

1659 where the various alternatives on the right hand side are specified in the sections that follow.

1660 Each EPC Class URI scheme is specified in one of the following subsections, as follows:

1661 **Table 6-1** EPC Class Schemes and Where the Pure Identity Form is Defined

| EPC Class Scheme | Specified In | Corresponding GS1 key | Typical use |
|------------------|---------------|----------------------------|--|
| lgtin | Section 6.4.1 | GTIN + Batch or Lot Number | Class of objects belonging to a given batch or lot |

1662 6.4.1 GTIN + Batch/Lot (LGTIN)

1663 The GTIN+ Batch/Lot scheme is used to denote a class of objects belonging to a given batch or lot
 1664 of a given GTIN.

1665 **General syntax:**
 1666 `urn:epc:class:lgtin:CompanyPrefix.ItemRefAndIndicator.Lot`

1667 **Example:**
 1668 `urn:epc:class:lgtin:4012345.012345.998877`

1669 **Grammar:**
 1670 `LGTIN-URI = %s"urn:epc:class:lgtin:" LGTINURIBody`
 1671 `LGTINURIBody = 2(PaddedNumericComponent ".") GS3A3Component`

1672 The number of characters in the two `PaddedNumericComponent` fields must total 13 (not
 1673 including any of the dot characters).

1674 The Lot field of the LGTIN-URI is expressed as a `GS3A3Component`, which permits the
 1675 representation of all characters permitted in the Application Identifier (10) Batch or Lot Number
 1676 according to the GS1 General Specifications.

1677 The LGTIN consists of the following elements:

- 1678 ■ The **GS1 Company Prefix**, assigned by GS1 to a managing entity or its delegates. This is the
 1679 same as the GS1 Company Prefix digits within a GS1 GTIN key. See Section [7.3.2](#) for the case
 1680 of a GTIN-8.

- 1681
1682
1683
1684
1685
1686
1687
1688
- The **Item Reference and Indicator**, assigned by the managing entity to a particular object class. The Item Reference and Indicator as it appears in the EPC URI is derived from the GTIN by concatenating the Indicator Digit of the GTIN (or a zero pad character, if the EPC URI is derived from a GTIN-8, GTIN-12, or GTIN-13) and the Item Reference digits, and treating the result as a single numeric string. See Section [7.3.2](#) for the case of a GTIN-8.
 - The **Batch or Lot Number**, assigned by the managing entity to an distinct batch or lot of a class of objects. The batch or lot number is not part of the GTIN, but is used to distinguish individual groupings of the same class of objects from each other.

1689 **7 Correspondence between EPCs and GS1 Keys**

1690 As discussed in Section [4.3](#), there is a well-defined relationship between Electronic Product Codes
1691 (EPCs) and seven keys (plus the component / part identifier) defined in the GS1 General
1692 Specifications [GS1GS]. This section specifies the correspondence between EPCs and GS1 keys.

1693 **7.1 The GS1 Company Prefix (GCP) in EPC encodings**

1694 The correspondence between EPCs and GS1 keys relies on identifying the portion of a GS1 key that
1695 is the GS1 Company Prefix. The GS1 Company Prefix (GCP) is a 4- to 12-digit number assigned by a
1696 GS1 Member Organisation to a managing entity, and the managing entity is free to create GS1 keys
1697 using that GCP. For purposes of the EPC Tag Data Standard, a 4- or 5-digit GCP is treated as a block
1698 of 100 6-digit GCPs or a block of 10 6-digit GCPs, respectively. In the EPC URI, the GCP is encoded
1699 in the *CompanyPrefix* component, which SHALL include the 4- or 5-digit GCP and the following 2 or
1700 1 digits of the GS1 key, as though it were a 6-digit GCP. This value is then encoded into the EPC
1701 binary encodings using Partition Value 6 (binary: 110).

1702 **7.2 Determining length of the EPC CompanyPrefix component for individually
1703 assigned GS1 Keys**

1704 In some instances, a GS1 Member Organisation assigns an individually assigned (AKA "single issue"
1705 or "one off") GS1 key, such as a complete GTIN, GLN, or other key, to a subscribing organisation. In
1706 such cases, a subscribing organisation SHALL NOT use the digits comprising a particular individually
1707 assigned key to construct any other kind of GS1 key. For example, if a subscribing organisation is
1708 issued an individually assigned GLN, it SHALL NOT create SSCCs using the 12 digits of the
1709 individually assigned GLN as though it were a 12-digit GS1 Company Prefix.

1710 Note that an individually assigned key will generally resolve (e.g., via GEPIR) back to the issuing
1711 MO—as the GCP in question has been assigned by the MO to itself for the purpose of generating
1712 individually assigned keys—rather than to the organisation to which the key was issued. The
1713 allocation of individually assigned keys, based on a common GCP, to disparate subscribing
1714 organisations who have no particular relationship to each other, effectively prevents use of the
1715 *CompanyPrefix* component of EPC encodings for purposes of filtering/correlation/querying to the
1716 level of an individual organisation.

1717 **7.2.1 Individually assigned GTINs**

1718 When encoding an individually assigned GTIN as an EPC, the GTIN-12, GTIN-13 or GTIN-8 issued by
1719 the MO must first be converted to a 14-digit number by prepending two, one or six leading zeroes,
1720 respectively, to the individually assigned GTIN, as specified in sections and [7.3.1](#) and [7.3.2](#).

1721 The individually assigned GTIN, after any necessary padding to increase its length to 14 digits, is
1722 stripped of its check digit (which is omitted from all EPC encodings) and indicator digit or leading
1723 zero, and SHALL be contained in the *CompanyPrefix* component of the EPC, whose length SHALL be
1724 fixed at 12 digits for an individually assigned GTIN. For a GTIN-12, GTIN-13 or GTIN-8, the
1725 *ItemRefAndIndicator* component of the resulting SGTIN EPC is a single zero digit. For a GTIN-
1726 14, the *ItemRefAndIndicator* component of the resulting SGTIN EPC consists of the GTIN-14's
1727 leading zero or indicator digit.

1728 Note that these rules also apply to individually assigned GTINs assigned by third parties with the
1729 permission of GS1.

1730 **Syntax:**
1731 `urn:epc:id:sgtin:CompanyPrefix.ItemRefAndIndicator.SerialNumber`

1732 **Example:**
1733 GS1 element string: (01)09526567890126(21)4711
1734 EPC URI: `urn:epc:id:sgtin:952656789012.0.4711`

1735 The corresponding EPC Binary encoding (SGTIN-96 and SGTIN-198) uses Partition Value 0, per
1736 Table 14-2 (*SGTIN Partition Table*).

1737 7.2.2 Individually assigned GLNs

1738 When encoding an individually assigned GLN as an EPC, the entire individually assigned GLN
1739 (stripped of its check digit, which is omitted from EPC encodings) occupies the *CompanyPrefix*
1740 component of the EPC, whose length is fixed at 12 digits.

1741 For the resulting SGLN EPC, the *LocationReference* component is a zero-length string. The *Extension*
1742 component of the SGLN EPC reflects the value of the GLN extension component, AI (254); if the
1743 input GS1 element string did not include a GLN extension component (AI 254), the *Extension*
1744 component of the SGLN EPC comprises a single zero digit ('0').

1745 Note that these rules also apply to individually assigned GLNs (e.g., national business numbers)
1746 assigned by third parties with the permission of GS1.

1747 **Syntax:**
1748 `urn:epc:id:sgln:CompanyPrefix..Extension`

1749 **Example (without extension):**
1750 GS1 element string: (414)9526567890126
1751 EPC URI: `urn:epc:id:sgln:952656789012..0`

1752 **Example (with extension):**
1753 GS1 element string: (414)9526567890126(254)4711
1754 EPC URI: `urn:epc:id:sgln:952656789012..4711`

1755 The corresponding EPC Binary encoding (SGLN-96 and SGLN-195) uses Partition Value 0, per Table
1756 14-7 (*SGLN Partition Table*).

1757 7.2.3 Other individually assigned GS1 Keys

1758 Other individually assigned GS1 Keys (e.g., SSCC, GIAI) should be encoded as EPCs with
1759 *CompanyPrefix* components that are 12 digits in length.

1760 In such cases, a subscribing organisation SHALL NOT use the digits comprising a particular
1761 individually assigned key to construct any other GS1 key. For example, if a subscribing organisation
1762 is issued an individually assigned SSCC, it SHALL NOT create additional SSCCs using the 12 digits of
1763 the individually assigned SSCC as though it were a 12-digit GCP.

1764 **Example (SSCC):**
1765 GS1 element string: (00)095265678901234568
1766 EPC URI: `urn:epc:id:sscc:952656789012.03456`

1767 **Example (GIAI):**
1768 GS1 element string: (8004)952656789012345678901234567890
1769 EPC URI: `urn:epc:id:giai:952656789012.345678901234567890`

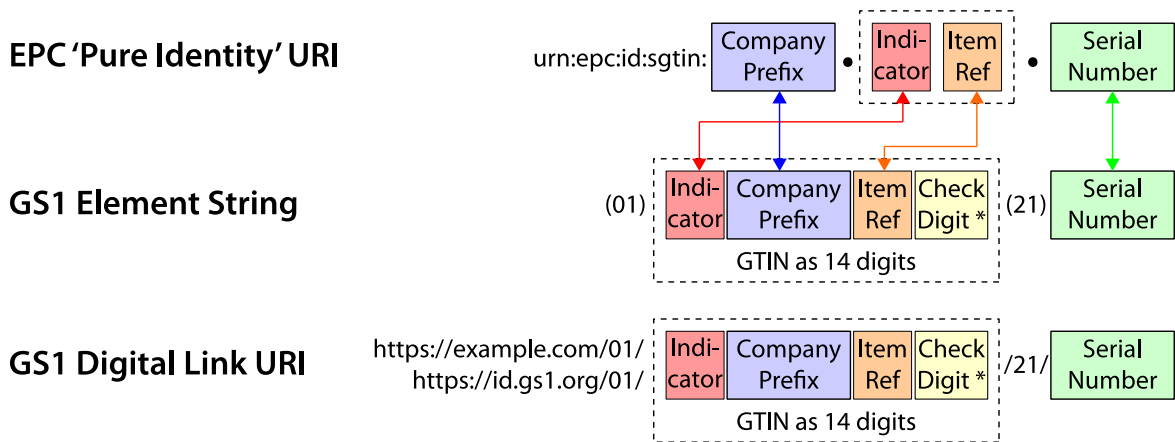
1770 The corresponding EPC Binary encoding uses Partition Value 0, per the respective Partition Table in
 1771 section 14.

1772 **7.3 Serialised Global Trade Item Number (SGTIN)**

1773 The SGTIN EPC (Section 6.3.1) does not correspond directly to any GS1 key, but instead
 1774 corresponds to a combination of a GTIN key plus a serial number. The serial number in the SGTIN is
 1775 defined to be equivalent to AI 21 in the GS1 General Specifications.

1776 The correspondence between the SGTIN EPC URI and a GS1 element string consisting of a GTIN key
 1777 (AI 01) and a serial number (AI 21) is depicted graphically below:

1778 **Figure 7-1** Correspondence between SGTIN EPC URI and GS1 element string



* the GS1 Check Digit is calculated over the preceding digits

1779
 1780 (Note that in the case of a GTIN-12 or GTIN-13, a zero pad character takes the place of the
 1781 Indicator Digit in the figure above.)

1782 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be
 1783 written as follows:

1784 EPC URI: $urn:epc:id:sgtin:d_2...d_{(L+1)} \cdot d_1 d_{(L+2)} d_{(L+3)}...d_{13} \cdot s_1 s_2...s_K$

1785 GS1 element string: $(01) d_1 d_2...d_{14} (21) s_1 s_2...s_K$

1786 where $1 \leq K \leq 20$.

1787 **To find the GS1 element string corresponding to an SGTIN EPC URI:**

- 1788 1. Number the digits of the first two components of the EPC as shown above. Note that there will
 1789 always be a total of 13 digits.
- 1790 2. Number the characters of the serial number (third) component of the EPC as shown above. Each
 1791 s_i corresponds to either a single character or to a percent-escape triplet consisting of a %
 1792 character followed by two hexadecimal digit characters.
- 1793 3. Calculate the check digit $d_{14} = (10 - ((3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13}) + (d_2 + d_4 + d_6 +$
 1794 $d_8 + d_{10} + d_{12})) \bmod 10)) \bmod 10$.
- 1795 4. Arrange the resulting digits and characters as shown for the GS1 element string. If any s_i in the
 1796 EPC URI is a percent-escape triplet %xx, in the GS1 element string replace the triplet with the
 1797 corresponding character according to Table I.3.1-1 (For a given percent-escape triplet %xx, find
 1798 the row of Table I.3.1-1 that contains xx in the "Hex Value" column; the "Graphic symbol"
 1799 column then gives the corresponding character to use in the GS1 element string.)

1800
1801
1802
1803
1804
1805
1806
1807
1808
1809

To find the EPC URI corresponding to a GS1 element string that includes both a GTIN (AI 01) and a serial number (AI 21):

1. Number the digits and characters of the GS1 element string as shown above.
2. Except for a GTIN-8, determine the number of digits L in the GS1 Company Prefix. This may be done, for example, by reference to an external table of company prefixes. See Section [7.3.2](#) for the case of a GTIN-8.
3. Arrange the digits as shown for the EPC URI. Note that the GTIN check digit d_{14} is not included in the EPC URI. For each serial number character s_i , replace it with the corresponding value in the "URI Form" column of [Table I.3.1-1](#) – either the character itself or a percent-escape triplet if s_i is not a legal URI character.

Example:

EPC URI: `urn:epc:id:sgtin:9521141.012345.32a%2Fb`

GS1 element string: `(01)09521141123454(21)32a/b`

In this example, the slash (/) character in the serial number must be represented as an escape triplet in the EPC URI.

1810
1811
1812
1813
1814

7.3.1 GTIN-12 and GTIN-13

To find the EPC URI corresponding to the combination of a GTIN-12 or GTIN-13 and a serial number, first convert the GTIN-12 or GTIN-13 to a 14-digit number by adding two or one leading zero characters, respectively, as shown in [GS1GS] Section 3.3.2.

Example:

GTIN-12: `614141123452`

Corresponding 14-digit number: `00614141123452`

Corresponding SGTIN-EPC: `urn:epc:id:sgtin:0614141.012345.Serial`

Example:

GTIN-13: `9521141890127`

Corresponding 14-digit number: `09521141890127`

Corresponding SGTIN-EPC: `urn:epc:id:sgtin:9521141.089012.Serial`

1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826

7.3.2 GTIN-8

A GTIN-8 is a special case of the GTIN that is used to identify small trade items.

The GTIN-8 code consists of eight digits $N_1, N_2...N_8$, where the first digits N_1 to N_L are the GS1-8 Prefix (where $L = 1, 2, \text{ or } 3$), the next digits N_{L+1} to N_7 are the Item Reference, and the last digit N_8 is the check digit. The GS1-8 Prefix is a one-, two-, or three-digit index number, administered by the GS1 Global Office. It does not identify the origin of the item. The Item Reference is assigned by the GS1 Member Organisation. The GS1 Member Organisations provide procedures for obtaining GTIN-8s.

To find the EPC URI corresponding to the combination of a GTIN-8 and a serial number, the following procedure SHALL be used. For the purpose of the procedure defined above in Section [7.2.3](#), the GS1 Company Prefix portion of the EPC shall be constructed by prepending five zeros to the first three digits of the GTIN-8; that is, the GS1 Company Prefix portion of the EPC is eight digits and shall be `00000N1N2N3`. The Item Reference for the procedure shall be the remaining GTIN-8 digits apart from the check digit, that is, N_4 to N_7 . The Indicator Digit for the procedure shall be zero.

Example:

GTIN-8: `95010939`

1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843

1844 Corresponding SGTIN-EPC: `urn:epc:id:sgtin:00000950.01093.Serial`

1845 **7.3.3 RCN-8**

1846 An RCN-8 is an 8-digit code beginning with GS1-8 Prefixes 0 or 2, as defined in [GS1GS]
1847 Section 2.1.11.1. These are reserved for company internal numbering, and are not GTIN-8 codes.
1848 RCN-8 codes SHALL NOT be used to construct SGTIN EPCs, and the procedure for GTN-8 codes does
1849 not apply.

1850 **7.3.4 Company Internal Numbering (GS1 Prefixes 04 and 0001 – 0007)**

1851 The GS1 General Specifications reserve codes beginning with either 04 or 0001 through 0007 for
1852 company internal numbering. (See [GS1GS], Sections 2.1.11.2 and 2.1.11.3.)

1853 These numbers SHALL NOT be used to construct SGTIN EPCs. A future version of TDS may specify
1854 normative rules for using Company Internal Numbering codes in EPCs.

1855 **7.3.5 Restricted Circulation (GS1 Prefixes 02 and 20 – 29)**

1856 The GS1 General Specifications reserve codes beginning with either 02 or 20 through 29 for
1857 restricted circulation for geopolitical areas defined by GS1 member organisations and for variable
1858 measure trade items. (See [GS1GS], Sections 2.1.11.1 and 2.1.11.1.4)

1859 These numbers SHALL NOT be used to construct SGTIN EPCs. A future version of TDS may specify
1860 normative rules for using Restricted Circulation codes in EPCs.

1861 **7.3.6 Coupon Code Identification for Restricted Distribution (GS1 Prefixes 981-984 1862 and 99)**

1863 Coupons may be identified by constructing codes according to Sections 2.6.1-2.6.3 of the GS1
1864 General Specifications. The resulting numbers begin with GS1 Prefixes 981-984 and 99. Strictly
1865 speaking, however, a coupon is not a trade item, and these coupon codes are not actually trade
1866 item identification numbers.

1867 Therefore, coupon codes for restricted distribution SHALL NOT be used to construct SGTIN EPCs.

1868 **7.3.7 Refund Receipt (GS1 Prefix 980)**

1869 Section 2.6.4 of the GS1 General Specification specifies the construction of codes to represent
1870 refund receipts, such as those created by bottle recycling machines for redemption at point-of-sale.
1871 The resulting number begins with GS1 Prefix 980. Strictly speaking, however, a refund receipt is not
1872 a trade item, and these refund receipt codes are not actually trade item identification numbers.

1873 Therefore, refund receipt codes SHALL NOT be used to construct SGTIN EPCs.

1874 **7.3.8 ISBN, ISMN, and ISSN (GS1 Prefixes 977, 978, or 979)**

1875 The GS1 General Specifications provide for the use of a 13-digit identifier to represent International
1876 Standard Book Number, International Standard Music Number, and International Standard Serial
1877 Number codes. The resulting code is a GTIN whose GS1 Prefix is 977, 978, or 979.

1878 **7.3.8.1 ISBN and ISMN**

1879 ISBN and ISMN codes are used for books and printed music, respectively. The codes are defined by
1880 ISO (ISO 2108 for ISBN and ISO 10957 for ISMN) and administered by the International ISBN
1881 Agency (<http://www.isbn-international.org/>) and affiliated national registration agencies. ISMN is a
1882 separate organisation (<http://www.ismn-international.org/>) but its management and coding
1883 structure are similar to the ones of ISBN.

1884 While these codes are not assigned by GS1, they have a very similar internal structure that readily
1885 lends itself to similar treatment when creating EPCs. An ISBN code consists of the following parts,
1886 shown below with the corresponding concept from the GS1 system:

| | | | |
|------|---|---|--|
| 1887 | Prefix Element + Registrant Group Element | = | GS1 Prefix (978 or 979 plus more digits) |
| 1888 | Registrant Element | = | Remainder of GS1 Company Prefix |
| 1889 | Publication Element | = | Item Reference |
| 1890 | Check Digit | = | Check Digit |

1891 The Registrant Group Elements are assigned to ISBN registration agencies, who in turn assign
1892 Registrant Elements to publishers, who in turn assign Publication Elements to individual publication
1893 editions. This exactly parallels the construction of GTIN codes. As in GTIN, the various components
1894 are of variable length, and as in GTIN, each publisher knows the combined length of the Registrant
1895 Group Element and Registrant Element, as the combination is assigned to the publisher. The total
1896 length of the "978" or "979" Prefix Element, the Registrant Group Element, and the Registrant
1897 Element is in the range of 6 to 12 digits, which is exactly the range of GS1 Company Prefix lengths
1898 permitted in the SGTIN EPC. The ISBN and ISMN can thus be used to construct SGTINs as specified
1899 in this standard.

1900 To find the EPC URI corresponding to the combination of an ISBN or ISMN and a serial number, the
1901 following procedure SHALL be used. For the purpose of the procedure defined above in
1902 Section [7.2.3](#), the GS1 Company Prefix portion of the EPC shall be constructed by concatenating the
1903 ISBN/ISMN Prefix Element (978 or 979), the Registrant Group Element, and the Registrant Element.
1904 The Item Reference for the procedure shall be the digits of the ISBN/ISMN Publication Element. The
1905 Indicator Digit for the procedure shall be zero.

1906 **Example:**

1907 ISBN: 978-81-7525-766-5

1908 Corresponding SGTIN-EPC: `urn:epc:id:sgtin:978817525.0766.Serial`

1909 **7.3.8.2 ISSN**

1910 The ISSN is the standardised international code which allows the identification of any serial
1911 publication, including electronic serials, independently of its country of publication, of its language or
1912 alphabet, of its frequency, medium, etc. The code is defined by ISO (ISO 3297) and administered by
1913 the International ISSN Agency (<http://www.issn.org/>).

1914 The ISSN is a GTIN starting with the GS1 prefix 977. The ISSN structure does not allow it to be
1915 expressed in an SGTIN format. Therefore, pending formal requirements emerging from the serial
1916 publication sector, it is not currently possible to create an SGTIN on the basis of an ISSN.

1917 **7.4 Serial Shipping Container Code (SSCC)**

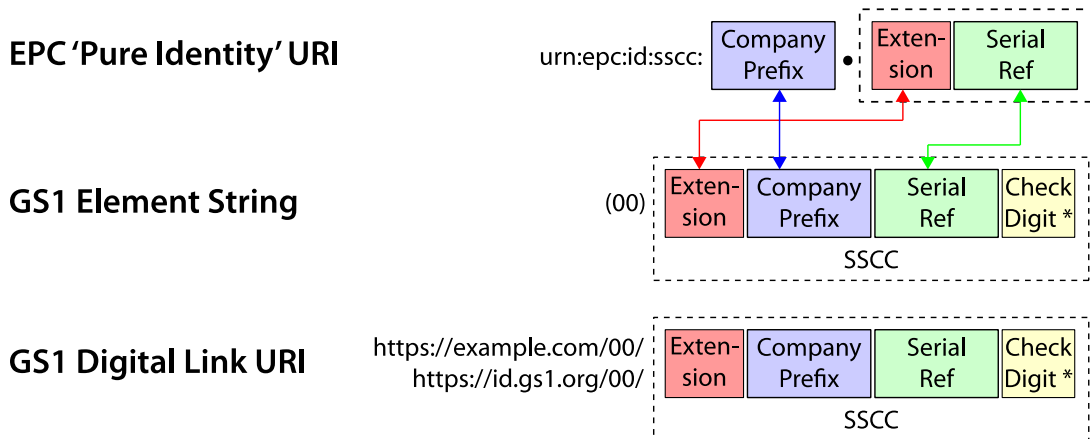
1918 The SSCC EPC (Section [6.3.2](#)) corresponds directly to the SSCC key defined in Sections 2.2.1 and
1919 3.3.1 of the GS1 General Specifications [GS1GS].

1920
1921

The correspondence between the SSCC EPC URI and a GS1 element string consisting of an SSCC key (AI 00) is depicted graphically below:

1922

Figure 7-2 Correspondence between SSCC EPC URI and GS1 element string



1923

* the GS1 Check Digit is calculated over the preceding digits

1924
1925

Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be written as follows:

1926

EPC URI: `urn:epc:id:sscc:d2d3...d(L+1) . d1d(L+2)d(L+3)...d17`

1927

GS1 element string: `(00) d1d2...d18`

1928

To find the GS1 element string corresponding to an SSCC EPC URI:

1929
1930

1. Number the digits of the two components of the EPC as shown above. Note that there will always be a total of 17 digits.
2. Calculate the check digit $d_{18} = (10 - ((3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13} + d_{15} + d_{17}) + (d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12} + d_{14} + d_{16}))) \bmod 10) \bmod 10$.
3. Arrange the resulting digits and characters as shown for the GS1 element string.

1934

To find the EPC URI corresponding to a GS1 element string that includes an SSCC (AI 00):

1935
1936

1. Number the digits and characters of the GS1 element string as shown above.
2. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example, by reference to an external table of company prefixes.
3. Arrange the digits as shown for the EPC URI. Note that the SSCC check digit d₁₈ is not included in the EPC URI.

1940

Example:

1941

EPC URI: `urn:epc:id:sscc:9521141.1234567890`

1942

GS1 element string: `(00)195211412345678900`

1943

7.5 Global Location Number With or Without Extension (SGLN)

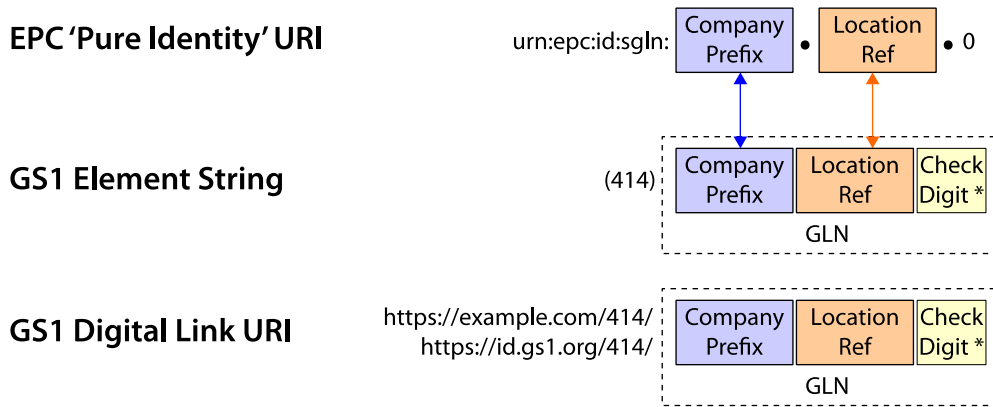
1944
1945

The SGLN EPC (Section 6.3.3) corresponds either directly to a Global Location Number key (GLN) as specified in Sections 2.4.4 and 3.7.9 of the GS1 General Specifications [GS1GS], or to the combination of a GLN key plus an extension number as specified in Section 3.5.11 of [GS1GS]. An extension number of zero is reserved to indicate that an SGLN EPC denotes an unextended GLN, rather than a GLN plus extension. (See Section 6.3.3 for an explanation of the letter "S" in "SGLN.")

1946
1947
1948

1949 The correspondence between the SGLN EPC URI and a GS1 element string consisting of a GLN key
 1950 (AI 414) *without* an extension is depicted graphically below:

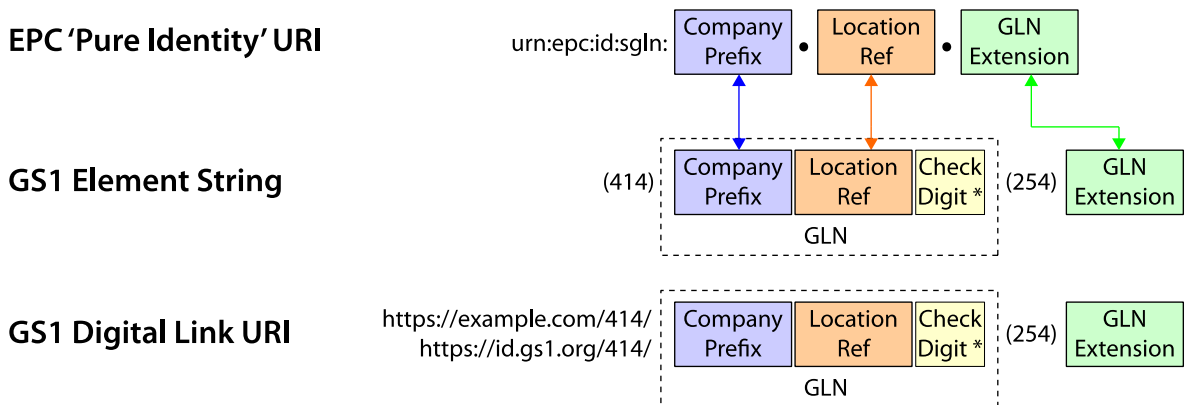
1951 **Figure 7-3** Correspondence between SGLN EPC URI without extension and GS1 element string



* the GS1 Check Digit is calculated over the preceding digits

1952
 1953 The correspondence between the SGLN EPC URI and a GS1 element string consisting of a GLN key
 1954 (AI 414) together with an extension (AI 254) is depicted graphically below:

1955 **Figure 7-4** Correspondence between SGLN EPC URI with extension and GS1 element string



* the GS1 Check Digit is calculated over the preceding digits

1956
 1957 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be
 1958 written as follows:

1959 EPC URI: urn:epc:id:sgln: $d_1d_2...d_L \cdot d_{(L+1)}d_{(L+2)}...d_{12} \cdot s_1s_2...s_K$

1960 GS1 element string: (414) $d_1d_2...d_{13}$ (254) $s_1s_2...s_K$

1961 **To find the GS1 element string corresponding to an SGLN EPC URI:**

- 1962
 1963
 1964
 1965
 1966
 1967
 1968
1. Number the digits of the first two components of the EPC as shown above. Note that there will always be a total of 12 digits.
 2. Number the characters of the *Extension* (third) component of the EPC as shown above. Each s_i corresponds to either a single character or to a percent-escape triplet consisting of a % character followed by two hexadecimal digit characters.
 3. Calculate the check digit $d_{13} = (10 - ((3(d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12}) + (d_1 + d_3 + d_5 + d_7 + d_9 + d_{11}))) \bmod 10) \bmod 10$.

1969
1970
1971
1972
1973
1974
1975

1976
1977

1978

1979
1980

1981
1982
1983
1984
1985

1986

1987

1988

1989

1990

1991

1992
1993

1994

1995
1996
1997
1998
1999
2000

4. Arrange the resulting digits and characters as shown for the GS1 element string. If any s_i in the EPC URI is a percent-escape triplet $\%xx$, in the GS1 element string replace the triplet with the corresponding character according to [Table I.3.1-1](#) (For a given percent-escape triplet $\%xx$, find the row of [Table I.3.1-1](#) that contains xx in the "Hex Value" column; the "Graphic symbol" column then gives the corresponding character to use in the GS1 element string.). If the serial number consists of a single character s_i and that character is the digit zero ('0'), omit the extension from the GS1 element string.

To find the EPC URI corresponding to a GS1 element string that includes a GLN (AI 414), with or without an accompanying extension (AI 254):

1. Number the digits and characters of the GS1 element string as shown above.
2. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example, by reference to an external table of company prefixes.
3. Arrange the digits as shown for the EPC URI. Note that the GLN check digit d_{13} is not included in the EPC URI. For each serial number character s_i , replace it with the corresponding value in the "URI Form" column of [Table I.3.1-1](#) – either the character itself or a percent-escape triplet if s_i is not a legal URI character. If the input GS1 element string did not include an extension (AI 254), use a single zero digit ('0') as the entire serial number $s_1s_2...s_K$ in the EPC URI.

Example (without extension):

EPC URI: `urn:epc:id:sgln:9521141.12345.0`

GS1 element string: `(414)9521141123454`

Example (with extension):

EPC URI: `urn:epc:id:sgln:9521141.12345.32a%2Fb`

GS1 element string: `(414)9521141123454(254)32a/b`

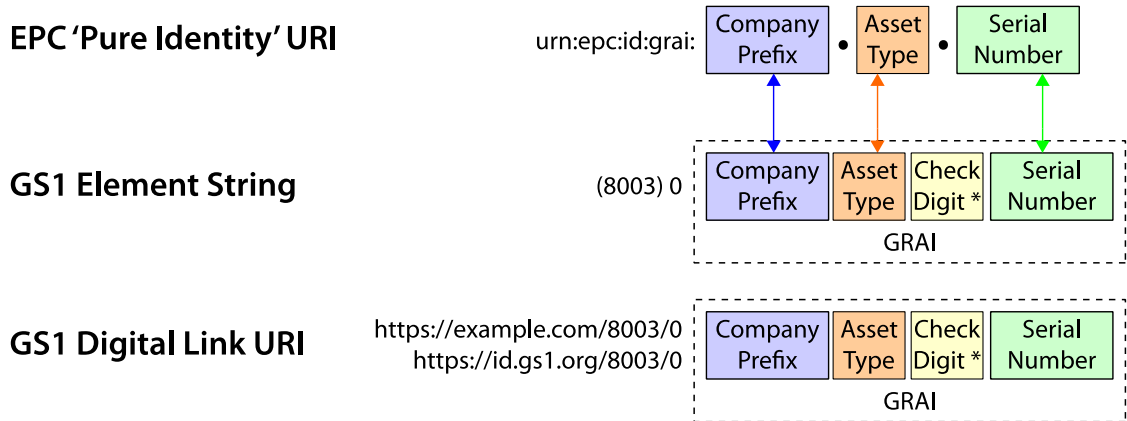
In this example, the slash (/) character in the serial number must be represented as an escape triplet in the EPC URI.

7.6 Global Returnable Asset Identifier (GRAI)

The GRAI EPC (Section [6.3.4](#)) corresponds directly to a serialised GRAI key defined in Sections 2.3.1 and 3.9.3 of the GS1 General Specifications [GS1GS]. Because an EPC always identifies a specific physical object, only GRAI keys that include the optional serial number have a corresponding GRAI EPC. GRAI keys that lack a serial number refer to asset classes rather than specific assets, and therefore do not have a corresponding EPC (just as a GTIN key without a serial number does not have a corresponding EPC).

2001

Figure 7-5 Correspondence between GRAI EPC URI and GS1 element string



2002

* the GS1 Check Digit is calculated over the preceding digits

2003

Note that the GS1 element string includes an extra zero ('0') digit following the Application Identifier (8003). This zero digit is extra padding in the element string, and is *not* part of the GRAI key itself.

2004

Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be written as follows:

2005

EPC URI: urn:epc:id:grai: $d_1d_2\dots d_L \cdot d_{(L+1)}d_{(L+2)}\dots d_{12} \cdot s_1s_2\dots s_K$

2006

GS1 element string: (8003)0 $d_1d_2\dots d_{13}s_1s_2\dots s_K$

2007

To find the GS1 element string corresponding to a GRAI EPC URI:

2008

1. Number the digits of the first two components of the EPC as shown above. Note that there will always be a total of 12 digits.
2. Number the characters of the serial number (third) component of the EPC as shown above. Each s_i corresponds to either a single character or to a percent-escape triplet consisting of a % character followed by two hexadecimal digit characters.
3. Calculate the check digit $d_{13} = (10 - ((3(d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12}) + (d_1 + d_3 + d_5 + d_7 + d_9 + d_{11})) \bmod 10)) \bmod 10$.
4. Arrange the resulting digits and characters as shown for the GS1 element string. If any s_i in the EPC URI is a percent-escape triplet %xx, in the GS1 element string replace the triplet with the corresponding character according to [Table I.3.1-1](#) (For a given percent-escape triplet %xx, find the row of [Table I.3.1-1](#) that contains xx in the "Hex Value" column; the "Graphic symbol" column then gives the corresponding character to use in the GS1 element string.)

2012

To find the EPC URI corresponding to a GS1 element string that includes a GRAI (AI 8003):

2022

1. If the number of characters following the (8003) application identifier is less than or equal to 14, stop: this element string does not have a corresponding EPC because it does not include the optional serial number.
2. Number the digits and characters of the GS1 element string as shown above.
3. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example, by reference to an external table of company prefixes.
4. Arrange the digits as shown for the EPC URI. Note that the GRAI check digit d_{13} is not included in the EPC URI. For each serial number character s_i , replace it with the corresponding value in the "URI Form" column of [Table I.3.1-1](#) – either the character itself or a percent-escape triplet if s_i is not a legal URI character.

2023

2024

2025

2026

2027

2028

2029

2030

2031

2032

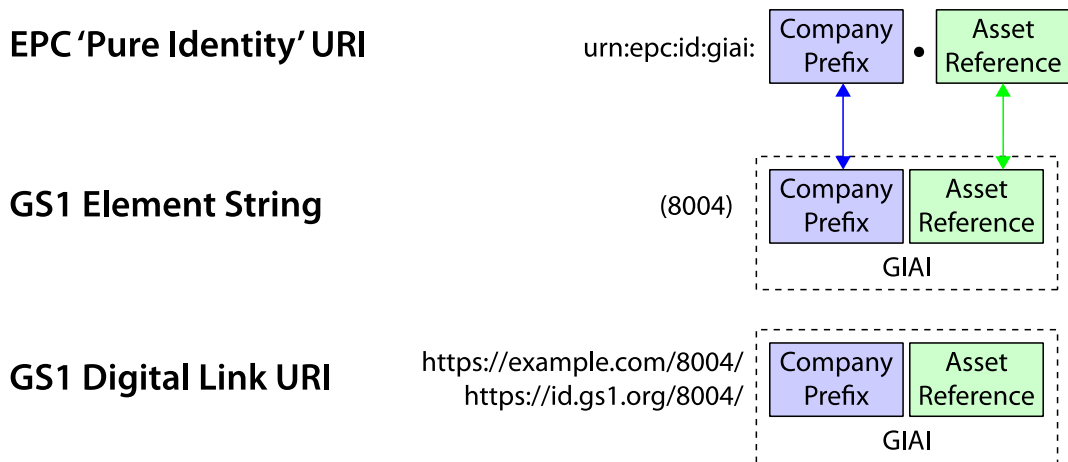
2033

2034 **Example:**
 2035 EPC URI: urn:epc:id:grai:9521141.12345.32a%2Fb
 2036 GS1 element string: (8003)0952114112345432a/b
 2037 In this example, the slash (/) character in the serial number must be represented as an escape
 2038 triplet in the EPC URI.

7.7 Global Individual Asset Identifier (GIAI)

2040 The GIAI EPC (Section 6.3.5) corresponds directly to the GIAI key defined in Sections 2.3.2 and
 2041 3.9.4 of the GS1 General Specifications [GS1GS].
 2042 The correspondence between the GIAI EPC URI and a GS1 element string consisting of a GIAI key
 2043 (AI 8004) is depicted graphically below:

2044 **Figure 7-6** Correspondence between GIAI EPC URI and GS1 element string



2045 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be
 2046 written as follows:
 2047

2048 EPC URI: urn:epc:id:grai:d₁d₂...d_L.s₁s₂...s_K

2049 GS1 element string: (8004) d₁d₂...d_Ls₁s₂...s_K

To find the GS1 element string corresponding to a GIAI EPC URI:

- 2051 1. Number the characters of the two components of the EPC as shown above. Each s_i corresponds
 2052 to either a single character or to a percent-escape triplet consisting of a % character followed by
 2053 two hexadecimal digit characters.
- 2054 2. Arrange the resulting digits and characters as shown for the GS1 element string. If any s_i in the
 2055 EPC URI is a percent-escape triplet %xx, in the GS1 element string replace the triplet with the
 2056 corresponding character according to [Table I.3.1-1](#) (For a given percent-escape triplet %xx, find
 2057 the row of [Table I.3.1-1](#) that contains xx in the "Hex Value" column; the "Graphic symbol"
 2058 column then gives the corresponding character to use in the GS1 element string.)

To find the EPC URI corresponding to a GS1 element string that includes a GIAI (AI 8004):

- 2061 1. Number the digits and characters of the GS1 element string as shown above.
- 2062 2. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example,
 2063 by reference to an external table of company prefixes.

- 2064 3. Arrange the digits as shown for the EPC URI. For each serial number character s_i , replace it
 2065 with the corresponding value in the "URI Form" column of [Table I.3.1-1](#) – either the character
 2066 itself or a percent-escape triplet if s_i is not a legal URI character.

2067 EPC URI: urn:epc:id:giai:9521141.32a%2Fb

2068 GS1 element string: (8004) 952114132a/b

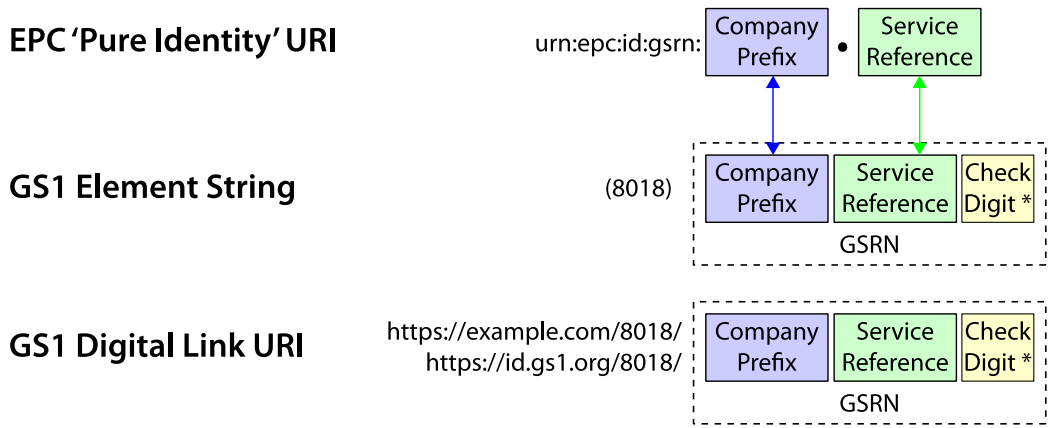
2069 In this example, the slash (/) character in the serial number must be represented as an escape
 2070 triplet in the EPC URI.

2071 **7.8 Global Service Relation Number – Recipient (GSRN)**

2072 The GSRN EPC (Section 6.3.6) corresponds directly to the GSRN – Recipient key defined in Sections
 2073 2.5.2 and 3.9.14 of the GS1 General Specifications [GS1GS].

2074 The correspondence between the GSRN EPC URI and a GS1 element string consisting of a GSRN key
 2075 (AI 8018) is depicted graphically below:

2076 **Figure 7-7** Correspondence between GSRN EPC URI and GS1 element string



* the GS1 Check Digit is calculated over the preceding digits

2077 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be
 2078 written as follows:

2080 EPC URI: urn:epc:id:gsmn: $d_1d_2...d_L.d_{(L+1)}d_{(L+2)}...d_{17}$

2081 GS1 element string: (8018) $d_1d_2...d_{18}$

2082 **To find the GS1 element string corresponding to a GSRN EPC URI:**

- 2083 1. Number the digits of the two components of the EPC as shown above. Note that there will
 2084 always be a total of 17 digits.
- 2085 2. Calculate the check digit $d_{18} = (10 - ((3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13} + d_{15} + d_{17}) + (d_2 +$
 2086 $d_4 + d_6 + d_8 + d_{10} + d_{12} + d_{14} + d_{16})) \bmod 10)) \bmod 10$.
- 2087 3. Arrange the resulting digits and characters as shown for the GS1 element string.

2088 **To find the EPC URI corresponding to a GS1 element string that includes a GSRN –**
 2089 **Recipient (AI 8018):**

- 2090 1. Number the digits and characters of the GS1 element string as shown above.
- 2091 2. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example,
 2092 by reference to an external table of company prefixes.
- 2093 3. Arrange the digits as shown for the EPC URI. Note that the GSRN check digit d_{18} is not included
 2094 in the EPC URI.

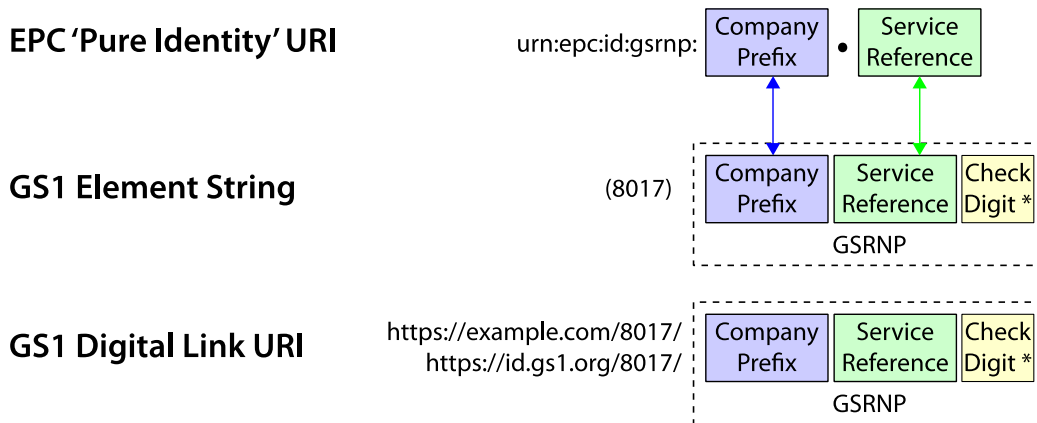
2095 **Example:**
 2096 EPC URI: urn:epc:id:gsrcn:9521141.1234567890
 2097 GS1 element string: (8018) 952114112345678906

7.9 Global Service Relation Number – Provider (GSRNP)

2099 The GSRNP EPC (Section 6.3.6) corresponds directly to the GSRN – Provider key defined in Sections
 2100 2.5.1 and 3.9.14 of the GS1 General Specifications [GS1GS].

2101 The correspondence between the GSRNP EPC URI and a GS1 element string consisting of a GSRN –
 2102 Provider key (AI 8017) is depicted graphically below:

Figure 7-8 Correspondence between GSRNP EPC URI and GS1 element string



* the GS1 Check Digit is calculated over the preceding digits

2104
 2105 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be
 2106 written as follows:

2107 EPC URI: urn:epc:id:gsrcn: $d_1d_2...d_L \cdot d_{(L+1)}d_{(L+2)}...d_{17}$

2108 GS1 element string: (8017) $d_1d_2...d_{18}$

2109 To find the GS1 element string corresponding to a GSRNP EPC URI:

- 2110 1. Number the digits of the two components of the EPC as shown above. Note that there will
 2111 always be a total of 17 digits.
- 2112 2. Calculate the check digit $d_{18} = (10 - ((3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13} + d_{15} + d_{17}) + (d_2 +$
 2113 $d_4 + d_6 + d_8 + d_{10} + d_{12} + d_{14} + d_{16}))) \bmod 10) \bmod 10$.
- 2114 3. Arrange the resulting digits and characters as shown for the GS1 element string.

2115 To find the EPC URI corresponding to a GS1 element string that includes a GSRN – 2116 Provider (AI 8017):

- 2117 1. Number the digits and characters of the GS1 element string as shown above.
- 2118 2. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example,
 2119 by reference to an external table of company prefixes.
- 2120 3. Arrange the digits as shown for the EPC URI. Note that the GSRN check digit d_{18} is not included
 2121 in the EPC URI.

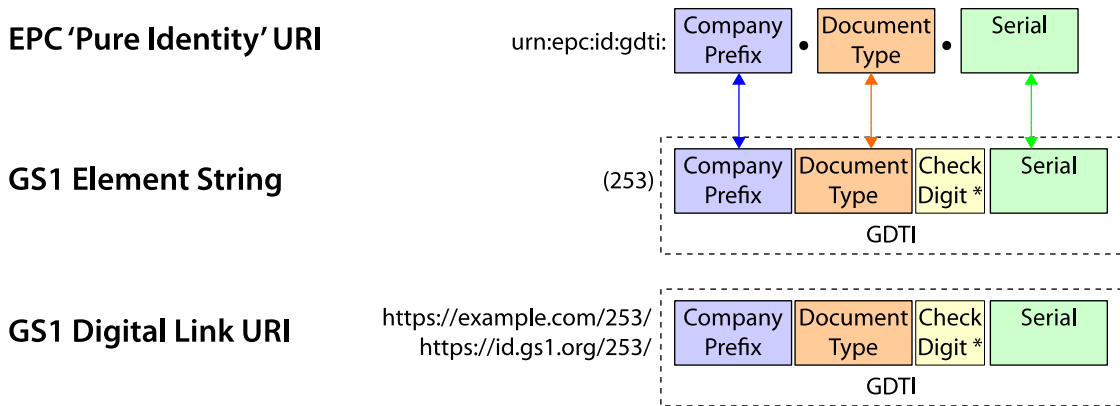
2122 **Example:**
 2123 EPC URI: urn:epc:id:gsrcn:9521141.1234567890

2124 GS1 element string: (8017) 952114112345678906

2125 **7.10 Global Document Type Identifier (GDTI)**

2126 The GDTI EPC (Section 6.3.7) corresponds directly to a serialised GDTI key defined in Sections 2.6.9
 2127 and 3.5.10 of the GS1 General Specifications [GS1GS]. Because an EPC always identifies a specific
 2128 physical object, only GDTI keys that include the optional serial number have a corresponding GDTI
 2129 EPC. GDTI keys that lack a serial number refer to document classes rather than specific documents,
 2130 and therefore do not have a corresponding EPC (just as a GTIN key without a serial number does
 2131 not have a corresponding EPC).

2132 **Figure 7-9** Correspondence between GDTI EPC URI and GS1 element string



* the GS1 Check Digit is calculated over the preceding digits

2133 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be
 2134 written as follows:

2136 EPC URI: $urn:epc:id:gdti:d_1d_2...d_L \cdot d_{(L+1)}d_{(L+2)}...d_{12} \cdot s_1s_2...s_K$

2137 GS1 element string: $(253) d_1d_2...d_{13}s_1s_2...s_K$

2138 **To find the GS1 element string corresponding to a GDTI EPC URI:**

- 2139 1. Number the digits of the first two components of the EPC as shown above. Note that there will
 2140 always be a total of 12 digits.
- 2141 2. Number the characters of the serial number (third) component of the EPC as shown above.
 2142 Each s_i corresponds to either a single character or to a percent-escape triplet consisting of a %
 2143 character followed by two hexadecimal digit characters.
- 2144 3. Calculate the check digit $d_{13} = (10 - ((3(d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12}) + (d_1 + d_3 + d_5 + d_7 + d_9$
 2145 $+ d_{11})) \bmod 10)) \bmod 10$.
- 2146 4. Arrange the resulting digits and characters as shown for the GS1 element string. If any s_i in the
 2147 EPC URI is a percent-escape triplet %xx, in the GS1 element string replace the triplet with the
 2148 corresponding character according to [Table I.3.1-1](#) (For a given percent-escape triplet %xx, find
 2149 the row of [Table I.3.1-1](#) that contains xx in the "Hex Value" column; the "Graphic symbol"
 2150 column then gives the corresponding character to use in the GS1 element string.)

2151 **To find the EPC URI corresponding to a GS1 element string that includes a GDTI (AI 253):**

- 2152 1. If the number of characters following the (253) application identifier is less than or equal to 13,
 2153 stop: this element string does not have a corresponding EPC because it does not include the
 2154 optional serial number.
- 2155 2. Number the digits and characters of the GS1 element string as shown above.

2156
2157

3. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example, by reference to an external table of company prefixes.

2158
2159
2160
2161

4. Arrange the digits as shown for the EPC URI. Note that the GDTI check digit d_{13} is not included in the EPC URI. For each serial number character s_i , replace it with the corresponding value in the "URI Form" column of [Table I.3.1-1](#) – either the character itself or a percent-escape triplet if s_i is not a legal URI character.

2162

Example:

2163

EPC URI: urn:epc:id:gdti:9521141.12345.006847

2164

GS1 element string: (253)9521141123454006847

2165

7.11 Component and Part Identifier (CPI)

2166
2167
2168

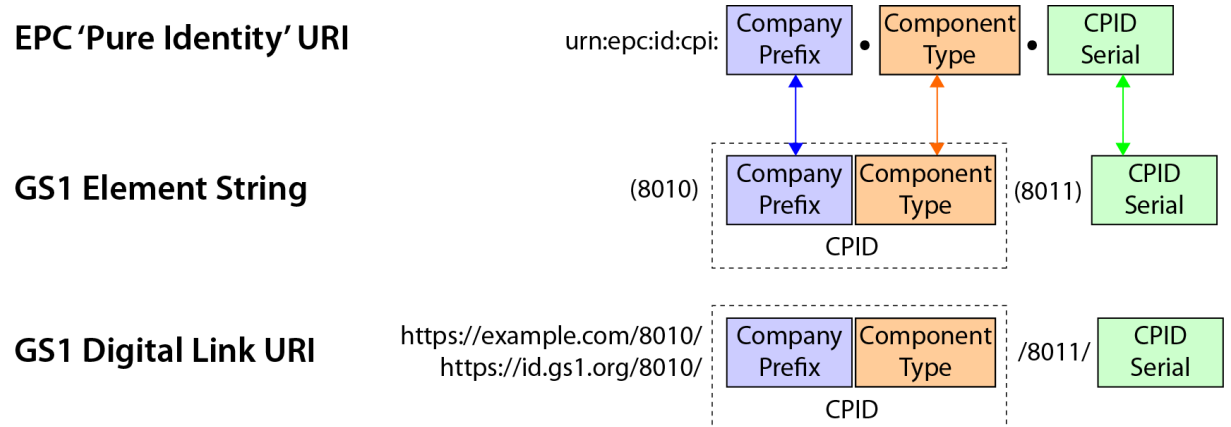
The CPI EPC (Section 6.3.9) does not correspond directly to any GS1 key, but instead corresponds to a combination of two data elements defined in sections 3.9.10 and 3.9.11 of the GS1 General Specifications [GS1GS].

2169
2170
2171

The correspondence between the CPI EPC URI and a GS1 element string consisting of a Component / Part Identifier (AI 8010) and a Component / Part serial number (AI 8011) is depicted graphically below:

2172

Figure 7-10 Correspondence between CPI EPC URI and GS1 element string



2173

Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be written as follows:

2174

EPC URI: urn:epc:id:cpi: $d_1d_2...d_L \cdot d_{(L+1)}d_{(L+2)}...d_N \cdot s_1s_2...s_K$

2175

GS1 element string: (8010) $d_1d_2...d_N$ (8011) $s_1s_2...s_K$

2176

where $1 \leq N \leq 30$ and $1 \leq K \leq 12$.

2177

To find the GS1 element string corresponding to a CPI EPC URI:

2178

1. Number the digits of the three components of the EPC as shown above. Each d_i in the second component corresponds to either a single character or to a percent-escape triplet consisting of a % character followed by two hexadecimal digit characters.
2. Arrange the resulting digits and characters as shown for the GS1 element string. If any d_i in the EPC URI is a percent-escape triplet %xx, in the GS1 element string replace the triplet with the corresponding character according to [Table I.3.1-1 \(G\)](#). (For a given percent-escape triplet %xx, find the row of [Table I.3.1-1](#) that contains xx in the "Hex Value" column; the "Graphic symbol" column then gives the corresponding character to use in the GS1 element string.)

2183

2184

2185

2186

2187

2188
2189
2190
2191
2192
2193
2194
2195

To find the EPC URI corresponding to a GS1 element string that includes both a Component / Part Identifier (AI 8010) and a Component / Part Serial Number (AI 8011):

1. Number the digits and characters of the GS1 element string as shown above.
2. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example, by reference to an external table of company prefixes.
3. Arrange the characters as shown for the EPC URI. For each component/part character d_i , replace it with the corresponding value in the "URI Form" column of [Table I.3.1-1 \(G\)](#) – either the character itself or a percent-escape triplet if d_i is not a legal URI character.

Example:

EPC URI: urn:epc:id:cpi:9521141.5PQ7%2FZ43.12345

GS1 element string: (8010) 95211415PQ7/Z43 (8011) 12345

Spaces have been added to the GS1 element string for clarity, but they are not normally present. In this example, the slash (/) character in the component/part reference must be represented as an escape triplet in the EPC URI.

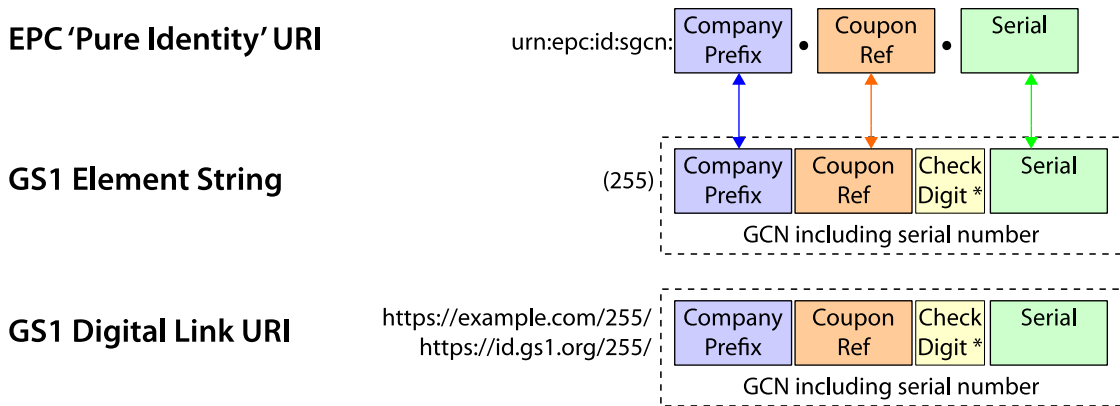
2196
2197
2198
2199
2200
2201

7.12 Serialised Global Coupon Number (SGCN)

The SGCN EPC (Section 6.3.10) corresponds directly to a serialised GCN key defined in Sections 2.6.1 and 3.5.12 of the GS1 General Specifications [GS1GS]. Because an EPC always identifies a specific physical or digital object, only SGCN keys that include the serial number have a corresponding SGCN EPC. GCN keys that lack a serial number refer to coupon classes rather than specific coupons, and therefore do not have a corresponding EPC.

2202
2203
2204
2205
2206
2207
2208

Figure 7-11 Correspondence between SGCN EPC URI and GS1 element string



* the GS1 Check Digit is calculated over the preceding digits

Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be written as follows:

EPC URI: urn:epc:id:sgcn: $d_1d_2...d_L.d_{(L+1)}d_{(L+2)}...d_{12}.s_1s_2...s_K$

GS1 element string: (255) $d_1d_2...d_{13}s_1s_2...s_K$

To find the GS1 element string corresponding to a SGCN EPC URI:

1. Number the digits of the first two components of the EPC as shown above. Note that there will always be a total of 12 digits.
2. Number the characters of the serial number (third) component of the EPC as shown above. Each s_i is a digit character.

2209
2210
2211
2212
2213
2214
2215
2216
2217
2218

- 2219 3. Calculate the check digit $d_{13} = (10 - ((3(d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12}) + (d_1 + d_3 + d_5 + d_7 + d_9 + d_{11})) \bmod 10)) \bmod 10$.
- 2220
- 2221 4. Arrange the resulting digits as shown for the GS1 element string.

2222 **To find the EPC URI corresponding to a GS1 element string that includes a GCN (AI 255):**

- 2223 1. If the number of characters following the (255) application identifier is less than or equal to 13, stop: this element string does not have a corresponding EPC because it does not include the optional serial number.
- 2224
- 2225
- 2226 2. Number the digits and characters of the GS1 element string as shown above.
- 2227 3. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example, by reference to an external table of company prefixes.
- 2228
- 2229 4. Arrange the digits as shown for the EPC URI. Note that the GCN check digit d_{13} is not included in the EPC URI.
- 2230

2231 **Example:**

2232 EPC URI: urn:epc:id:sgcn:9521141.67890.04711

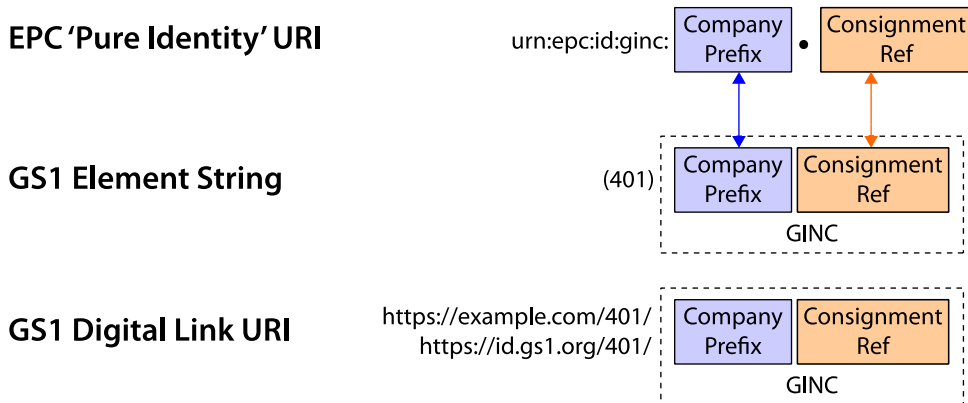
2233 GS1 element string: (255)952114167890904711

2234 **7.13 Global Identification Number for Consignment (GINC)**

2235 The GINC EPC (Section 6.5.1) corresponds directly to the GINC key defined in Sections 2.2.2 and 3.7.2 of the GS1 General Specifications [GS1GS].

2237 The correspondence between the GINC EPC URI and a GS1 element string consisting of a GINC key (AI 401) is depicted graphically below:

2239 **Figure 7-12** Correspondence between GINC EPC URI and GS1 element string



2240 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be written as follows:

2243 EPC URI: urn:epc:id:ginc: $d_1d_2\dots d_L.s_1s_2\dots s_K$

2244 GS1 element string: (401) $d_1d_2\dots d_Ls_1s_2\dots s_K$

2245 **To find the GS1 element string corresponding to a GINC EPC URI:**

- 2246 1. Number the characters of the two components of the EPC as shown above. Each s_i corresponds to either a single character or to a percent-escape triplet consisting of a % character followed by two hexadecimal digit characters.
- 2247
- 2248
- 2249 2. Arrange the resulting digits and characters as shown for the GS1 element string. If any s_i in the EPC URI is a percent-escape triplet %xx, in the GS1 element string replace the triplet with the
- 2250

2251 corresponding character according to [Table I.3.1-1](#) (For a given percent-escape triplet %*xxx*, find
 2252 the row of [Table I.3.1-1](#) that contains *xxx* in the "Hex Value" column; the "Graphic symbol"
 2253 column then gives the corresponding character to use in the GS1 element string.)

2254 **To find the EPC URI corresponding to a GS1 element string that includes a GINC (AI 401):**

- 2255 1. Number the digits and characters of the GS1 element string as shown above.
 2256 2. Determine the number of digits *L* in the GS1 Company Prefix. This may be done, for example,
 2257 by reference to an external table of company prefixes.
 2258 3. Arrange the digits as shown for the EPC URI. For each serial number character *s_i*, replace it
 2259 with the corresponding value in the "URI Form" column of [Table I.3.1-1](#) – either the character
 2260 itself or a percent-escape triplet if *s_i* is not a legal URI character.

2261 **Example:**

2262 EPC URI: urn:epc:id:ginc:9521141.xyz47%2F11

2263 GS1 element string: (401)9521141xyz47/11

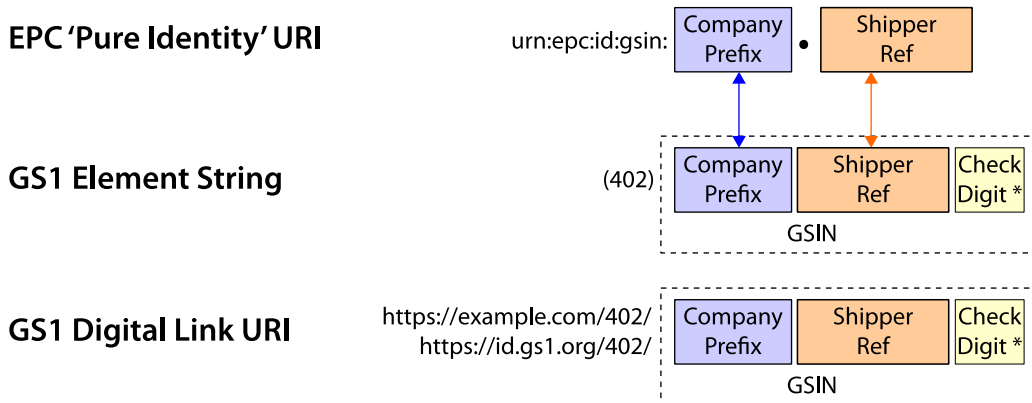
2264 In this example, the slash (/) character in the serial number must be represented as an escape
 2265 triplet in the EPC URI.

2266 **7.14 Global Shipment Identification Number (GSIN)**

2267 The GSIN EPC (Section 6.5.2) corresponds directly to the GSIN key defined in Sections 2.2.3 and
 2268 3.7.3 of the GS1 General Specifications [GS1GS].

2269 The correspondence between the GSIN EPC URI and a GS1 element string consisting of an GSIN key
 2270 (AI 402) is depicted graphically below:

2271 **Figure 7-13** Correspondence between GSIN EPC URI and GS1 element string



* the GS1 Check Digit is calculated over the preceding digits

2272
 2273 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be
 2274 written as follows:

2275 EPC URI: urn:epc:id:gsin:*d*₁*d*₂...*d*_{*L*} • *d*_(*L*+1) *d*_(*L*+2) *d*_(*L*+3) ... *d*₁₆

2276 GS1 element string: (402) *d*₁ *d*₂ ... *d*₁₇

2277 **To find the GS1 element string corresponding to an GSIN EPC URI:**

- 2278 1. Number the digits of the two components of the EPC as shown above. Note that there will
 2279 always be a total of 16 digits.
 2280 2. Calculate the check digit $d_{17} = (10 - (((d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13} + d_{15}) + 3(d_2 + d_4 +$
 2281 $d_6 + d_8 + d_{10} + d_{12} + d_{14} + d_{16})) \bmod 10)) \bmod 10$.

- 2282 Arrange the resulting digits and characters as shown for the GS1 element string.
- 2283 1. To find the EPC URI corresponding to a GS1 element string that includes a GSIN (AI 402):
- 2284 2. Number the digits and characters of the GS1 element string as shown above.
- 2285 3. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example,
- 2286 by reference to an external table of company prefixes.
- 2287 4. Arrange the digits as shown for the EPC URI. Note that the GSIN check digit d_{17} is not included
- 2288 in the EPC URI.

2289 **Example:**

2290 EPC URI: `urn:epc:id:gsin:9521141.123456789`

2291 GS1 element string: `(402) 95211411234567892`

7.15 Individual Trade Item Piece (ITIP)

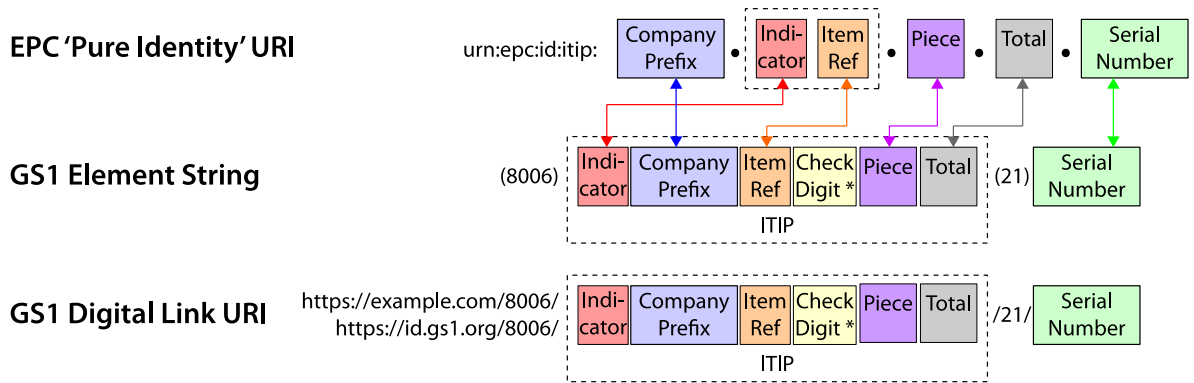
2293 The ITIP EPC (Section 6.3.13) does not correspond directly to any GS1 key, but instead

2294 corresponds to a combination of AIs (8006) and (21).

2295 The correspondence between the ITIP EPC URI and a GS1 element string consisting of AI (8006)

2296 and AI (21) is depicted graphically below:

2297 **Figure 7-14** Correspondence between ITIP EPC URI and GS1 element string



2298 * the GS1 Check Digit is calculated over the preceding digits

2299 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be

2300 written as follows:

2301 EPC URI: `urn:epc:id:itip: d2...d(L+1) . d1d(L+2) d(L+3)...d13 .) . d1d2 . d1d2 . s1s2...sK`

2302 GS1 element string: `(8006) d1d2...d18 (21) s1s2...sK`

2303 where $1 \leq K \leq 20$.

2304 To find the GS1 element string corresponding to an ITIP EPC URI:

- 2305 1. Number the digits of the first four components of the EPC as shown above. Note that there will
- 2306 always be a total of 17 digits.
- 2307 2. Number the characters of the serial number (seventh) component of the EPC as shown above.
- 2308 Each s_i corresponds to either a single character or to a percent-escape triplet consisting of a %
- 2309 character followed by two hexadecimal digit characters.
- 2310 3. Calculate the check digit $d_{14} = (10 - ((3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13}) + (d_2 + d_4 + d_6 +$
- 2311 $d_8 + d_{10} + d_{12})) \bmod 10) \bmod 10$.
- 2312 4. Arrange the resulting digits and characters as shown for the GS1 element string. If any s_i in the
- 2313 EPC URI is a percent-escape triplet %xx, in the GS1 element string replace the triplet with the

2314 corresponding character according to [Table I.3.1-1](#) (For a given percent-escape triplet %*xxx*, find
 2315 the row of [Table I.3.1-1](#) that contains *xxx* in the "Hex Value" column; the "Graphic symbol"
 2316 column then gives the corresponding character to use in the GS1 element string.)

2317 **To find the EPC URI corresponding to a GS1 element string that includes both AI (8006)**
 2318 **and AI (21):**

- 2319 1. Number the digits and characters of the GS1 element string as shown above.
- 2320 Except for a GTIN-8, determine the number of digits *L* in the GS1 Company Prefix. This may be
 2321 done, for example, by reference to an external table of company prefixes. See Section [7.3.2](#) for the
 2322 case of a GTIN-8.
- 2323 2. Arrange the digits as shown for the EPC URI. Note that the GTIN check digit *d*₁₄ is not included
 2324 in the EPC URI. For each serial number character *s*_{*i*}, replace it with the corresponding value in
 2325 the "URI Form" column of [Table I.3.1-1](#) – either the character itself or a percent-escape triplet if
 2326 *s*_{*i*} is not a legal URI character.

2327 **Example:**

2328 EPC URI: urn:epc:id:itip:9521141.012345.04.04.32a%2Fb

2329 GS1 element string: (8006)095211411234540404(21)32a/b

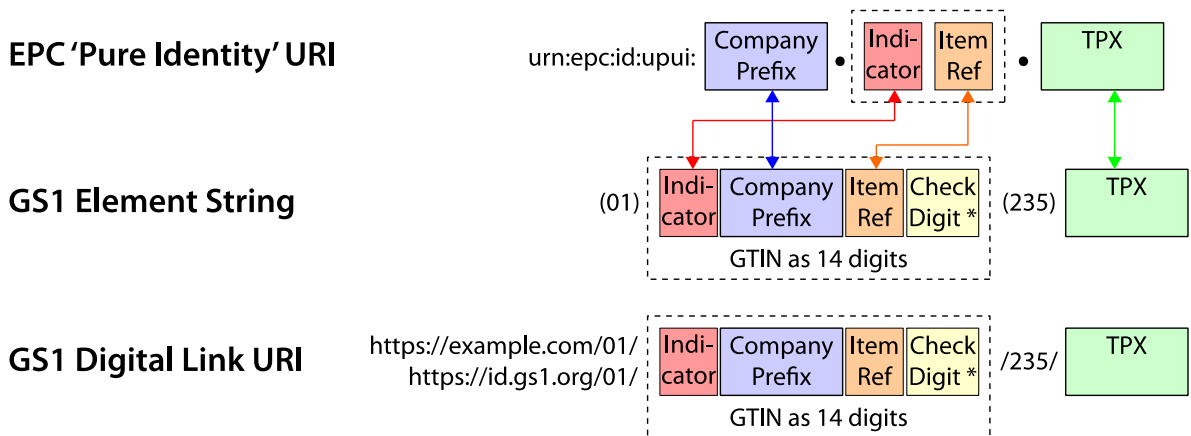
2330 In this example, the slash (/) character in the serial number must be represented as an escape
 2331 triplet in the EPC URI.

2332 **7.16 Unit Pack Identifier (UPUI)**

2333 The UPUI EPC (Section 6.3.14) does not correspond directly to any GS1 key, but instead
 2334 corresponds to a combination of a GTIN key plus a *Third Party Controlled, Serialised Extension of*
 2335 *GTIN* (TPX), as specified in the GS1 General Specifications [GS1GS].

2336 The correspondence between the UPUI EPC URI and a GS1 element string consisting of a GTIN key
 2337 (AI 01) and a *Third Party Controlled, Serialised Extension of GTIN* (AI 235) is depicted graphically
 2338 below:

2339 **Figure 7-15** Correspondence between UPUI EPC URI and GS1 element string



* the GS1 Check Digit is calculated over the preceding digits

2340
 2341 (Note that in the case of a GTIN-12 or GTIN-13, a zero pad character takes the place of the
 2342 Indicator Digit in the figure above.)

2343 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be
 2344 written as follows:

2345 EPC URI: urn:epc:id:upui: *d*₂...*d*_(*L*+1) . *d*₁*d*_(*L*+2) *d*_(*L*+3)...*d*₁₃ . *s*₁*s*₂...*s*_{*K*}

2346 GS1 element string: $(01) d_1 d_2 \dots d_{14} (235) s_1 s_2 \dots s_K$

2347 where $1 \leq K \leq 28$.

2348 **To find the GS1 element string corresponding to a UPI EPC URI:**

- 2349 1. Number the digits of the first two components of the EPC as shown above. Note that there will
2350 always be a total of 13 digits.
- 2351 2. Number the characters of the third component (TPX) of the EPC as shown above. Each s_i
2352 corresponds to either a single character or to a percent-escape triplet consisting of a % character
2353 followed by two hexadecimal digit characters.
- 2354 3. Calculate the check digit $d_{14} = (10 - ((3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13}) + (d_2 + d_4 + d_6 +$
2355 $d_8 + d_{10} + d_{12})) \bmod 10)) \bmod 10$.
- 2356 4. Arrange the resulting digits and characters as shown for the GS1 element string. If any s_i in the
2357 EPC URI is a percent-escape triplet %xx, in the GS1 element string replace the triplet with the
2358 corresponding character according to [Table I.3.1-1](#) (For a given percent-escape triplet %xx, find
2359 the row of [Table I.3.1-1](#) that contains xx in the "Hex Value" column; the "Graphic symbol"
2360 column then gives the corresponding character to use in the GS1 element string.)

2361 **To find the EPC URI corresponding to a GS1 element string that includes both a GTIN (AI**
2362 **01) and a Third Party Controlled, Serialised Extension of GTIN (AI 235):**

- 2363 1. Number the digits and characters of the GS1 element string as shown above.
- 2364 2. Except for a GTIN-8, determine the number of digits L in the GS1 Company Prefix. This may be
2365 done, for example, by reference to an external table of company prefixes. See Section [7.3.2](#) for
2366 the case of a GTIN-8.
- 2367 3. Arrange the digits as shown for the EPC URI. Note that the GTIN check digit d_{14} is not included
2368 in the EPC URI. For each serial number character s_i , replace it with the corresponding value in
2369 the "URI Form" column of [Table I.3.1-1](#) – either the character itself or a percent-escape triplet if
2370 s_i is not a legal URI character.

2371 **Example:**

2372 EPC URI: urn:epc:id:upui:9521141.089456.51qIqY)%3C%26Jp3*j7'SDB

2373 GS1 element string: (01)09521141894569(235)51qIqY)<&Jp3*j7'SDB

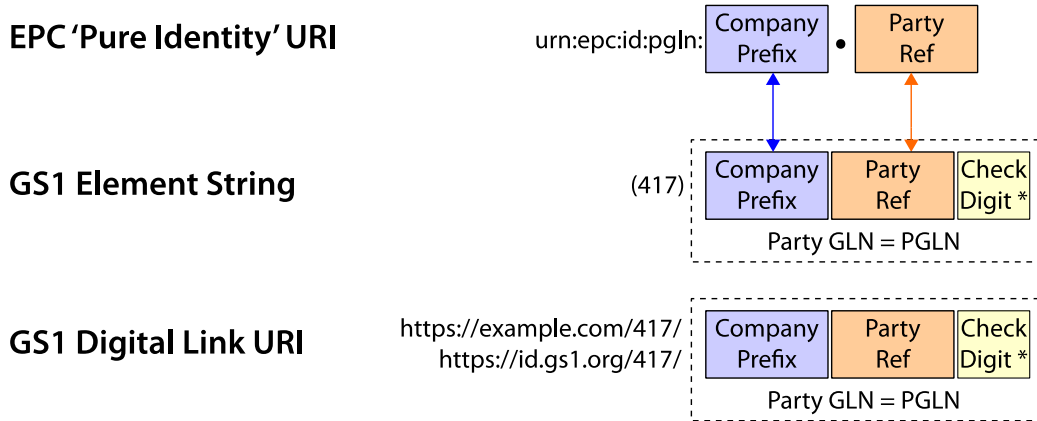
2374 In this example, the 'less than' (<) and ampersand (&) characters in the serial number must be
2375 represented as an escape triplet in the EPC URI.

2376 **7.17 Global Location Number of Party (PGLN)**

2377 The PGLN EPC (Section 6.3.15) corresponds directly to the Global Location Number of a Party
2378 (PARTY) as specified in the GS1 General Specifications [GS1GS].

2379 The correspondence between the PGLN EPC URI and a GS1 element string consisting of a GLN Party
 2380 key (AI 417) is depicted graphically below:

2381 **Figure 7-16** Correspondence between PGLN EPC URI without extension and GS1 element string



* the GS1 Check Digit is calculated over the preceding digits

2382
 2383 Formally, the correspondence is defined as follows. Let the EPC URI and the GS1 element string be
 2384 written as follows:

2385 EPC URI: `urn:epc:id:pglN:d1d2...dL.d(L+1)d(L+2)...d12.s1s2...sK`

2386 GS1 element string: `(417) d1d2...d13`

2387 **To find the GS1 element string corresponding to an PGLN EPC URI:**

- 2388 1. Number the digits of the first two components of the EPC as shown above. Note that there will
 2389 always be a total of 12 digits.
- 2390 2. Calculate the check digit $d_{13} = (10 - ((3(d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12}) + (d_1 + d_3 + d_5 + d_7 + d_9$
 2391 $+ d_{11})) \bmod 10)) \bmod 10$.
- 2392 3. Arrange the resulting digits as shown for the GS1 element string.

2393 **To find the EPC URI corresponding to a GS1 element string that includes a GLN (AI 417):**

- 2394 1. Number the digits and characters of the GS1 element string as shown above.
- 2395 2. Determine the number of digits L in the GS1 Company Prefix. This may be done, for example,
 2396 by reference to an external table of company prefixes.
- 2397 3. Arrange the digits as shown for the EPC URI. Note that the GLN check digit d_{13} is not included in
 2398 the EPC URI.

2399 **Example:**

2400 EPC URI: `urn:epc:id:pglN:9521141.89012`

2401 GS1 element string: `(417) 9521141890127`

2402 **7.18 GTIN + batch/lot (LGTIN)**

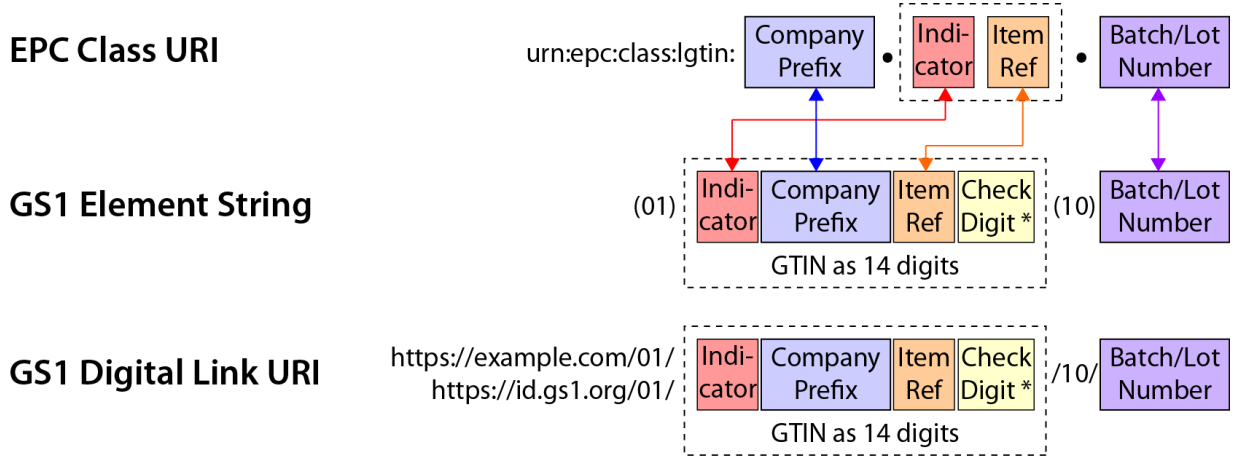
2403 The LGTIN EPC Class (Section 6.3.1) does not correspond directly to any GS1 key, but instead
 2404 corresponds to a combination of a GTIN key plus a Batch/Lot Number. The Batch/Lot Number in the
 2405 LGTIN is defined to be equivalent to AI 10 in the GS1 General Specifications.

2406
2407

The correspondence between the LGTIN EPC Class URI and a GS1 element string consisting of a GTIN key (AI 01) and a Batch/Lot Number (AI 10) is depicted graphically below:

2408

Figure 7-17 Correspondence between LGTIN EPC Class URI and GS1 element string



2409

* the GS1 Check Digit is calculated over the preceding digits

2410
2411

(Note that in the case of a GTIN-12 or GTIN-13, a zero pad character takes the place of the Indicator Digit in the figure above.)

2412
2413

Formally, the correspondence is defined as follows. Let the EPC Class URI and the GS1 element string be written as follows:

2414

EPC Class URI: `urn:epc:class:lgtn:d2d3...d(L+1).d1d(L+2)d(L+3)...d13.s1s2...sK`

2415

GS1 element string: `(01) d1d2...d14 (10) s1s2...sK`

2416

where $1 \leq K \leq 20$.

2417

To find the GS1 element string corresponding to an LGTIN EPC Class URI:

2418
2419

1. Number the digits of the first two components of the URI as shown above. Note that there will always be a total of 13 digits.
2. Number the characters of the Batch/Lot Number (third) component of the URI as shown above. Each s_i corresponds to either a single character or to a percent-escape triplet consisting of a % character followed by two hexadecimal digit characters.

2420
2421
2422

3. Calculate the check digit $d_{14} = (10 - ((3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13}) + (d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12}))) \bmod 10) \bmod 10$.

2425
2426
2427
2428
2429

4. Arrange the resulting digits and characters as shown for the GS1 element string. If any s_i in the URI is a percent-escape triplet `%xx`, in the GS1 element string replace the triplet with the corresponding character according to [Table I.3.1-1](#) (For a given percent-escape triplet `%xx`, find the row of [Table I.3.1-1](#) that contains `xx` in the "Hex Value" column; the "Graphic symbol" column then gives the corresponding character to use in the GS1 element string.)

2430
2431

To find the EPC Class URI corresponding to a GS1 element string that includes both a GTIN (AI 01) and a Batch/Lot Number (AI 10):

2432

1. Number the digits and characters of the GS1 element string as shown above.
2. Except for a GTIN-8, determine the number of digits L in the GS1 Company Prefix. This may be done, for example, by reference to an external table of company prefixes. See [Section 7.3.2](#) for the case of a GTIN-8.
3. Arrange the digits as shown for the EPC Class URI. Note that the GTIN check digit d_{14} is not included in the EPC Class URI. For each serial number character s_i , replace it with the corresponding value in the "URI Form" column of [Table I.3.1-1](#) – either the character itself or a percent-escape triplet if s_i is not a legal URI character.

2433
2434
2435

2436
2437
2438
2439

2440

Example:

2441

EPC Class URI: urn:epc:class:lgTin:9521141.712345.32a%2Fb

2442

GS1 element string: (01)79521141123453(10) 32a/b

2443

In this example, the slash (/) character in the serial number must be represented as an escape triplet in the EPC Class URI.

2444

2445

For GTIN-12, GTIN-13, GTIN-8 and other forms of the GTIN, see the subsections of Section 7.1. The considerations in those sections apply in an analogous manner to LGTIN.

2446

2447

8 URIs for EPC Pure identity patterns

2448

Certain software applications need to specify rules for filtering lists of EPC pure identities according to various criteria. This specification provides a Pure Identity Pattern URI form for this purpose. A Pure Identity Pattern URI does not represent a single EPC, but rather refers to a set of EPCs. A typical Pure Identity Pattern URI looks like this:

2449

2450

2451

urn:epc:idpat:sgtin:0652642.*.*

2452

2453

This pattern refers to any EPC SGTIN, whose GS1 Company Prefix is 0652642, and whose Item Reference and Serial Number may be anything at all. The tag length and filter bits are not considered at all in matching the pattern to EPCs.

2454

2455

2456

The new EPC schemes defined in TDS v2.0 have not defined an equivalent EPC Pure Identity URI syntax nor a corresponding EPC Pure Identity Pattern URI syntax; instead the encoding/decoding is between the binary string and the corresponding GS1 element string, GS1 Digital Link URI or equivalently, the set of GS1 Application Identifiers and their values, as shown in [Figure 3-1](#).

2457

2458

2459

2460

In general, there is a Pure Identity Pattern URI scheme corresponding to each Pure Identity EPC URI scheme (Section [6.3](#)), whose syntax is essentially identical except that any number of fields starting at the right may be a star (*). This is more restrictive than EPC Tag Pattern URIs (Section [13](#)), in that the star characters must occupy adjacent rightmost fields and the range syntax is not allowed at all.

2461

2462

2463

2464

2465

The pure identity pattern URI for the DoD Construct is as follows:

2466

urn:epc:idpat:usdod:CAGECodeOrDODAACPat.serialNumberPat

2467

with similar restrictions on the use of star (*).

2468

8.1 Syntax

2469

The grammar for Pure Identity Pattern URIs is given below.

2470

IDPatURI = %s"urn:epc:idpat:" IDPatBody

2471

IDPatBody =

2472

GIDIDPatURIBody /

2473

SGTINIDPatURIBody /

2474

SGLNIDPatURIBody /

2475

GIAIIDPatURIBody /

2476

SSCCIDPatURIBody /

2477

GRAIIDPatURIBody /

2478

GSRNIDPatURIBody /

2479

GSRNPIDPatURIBody /

2480

GDTIIDPatURIBody /

2481

SGCNIDPatURIBody /

```

2482         GINCIDPatURIBody /
2483         GSINIDPatURIBody /
2484         DODIDPatURIBody /
2485         ADIIDPatURIBody /
2486         CPIIDPatURIBody /
2487         ITIPIDPartURIBody /
2488         UPUIIDPatURIBody/
2489         PGLNIDPatURIBody
2490     GIDIDPatURIBody = %s"gid:" GIDIDPatURIMain
2491     GIDIDPatURIMain =
2492         2(NumericComponent ".") NumericComponent
2493         / 2(NumericComponent ".") "*"
2494         / NumericComponent ".*.*"
2495         / ".*.*.*"
2496     SGTINIDPatURIBody = %s"sgtin:" SGTINPatURIMain
2497     SGTINPatURIMain =
2498         2(PaddedNumericComponent ".") GS3A3Component
2499         / 2(PaddedNumericComponent ".") "*"
2500         / PaddedNumericComponent ".*.*"
2501         / ".*.*.*"
2502     GRAIIDPatURIBody = %s"grai:" SGLNGRAIIDPatURIMain
2503     SGLNIDPatURIBody = %s"sgln:" SGLNGRAIIDPatURIMain
2504     SGLNGRAIIDPatURIMain =
2505         PaddedNumericComponent "." PaddedNumericComponentOrEmpty "."
2506     GS3A3Component
2507         / PaddedNumericComponent "." PaddedNumericComponentOrEmpty ".*"
2508         / PaddedNumericComponent ".*.*"
2509         / ".*.*.*"
2510     SSCCIDPatURIBody = %s"sscc:" SSCCIDPatURIMain
2511     SSCCIDPatURIMain =
2512         PaddedNumericComponent "." PaddedNumericComponent
2513         / PaddedNumericComponent ".*"
2514         / ".*.*"
2515     GIAIIDPatURIBody = %s"giai:" GIAIIDPatURIMain
2516     GIAIIDPatURIMain =
2517         PaddedNumericComponent "." GS3A3Component
2518         / PaddedNumericComponent ".*"
2519         / ".*.*"
2520     GSRNIDPatURIBody = %s"gsrn:" GSRNIDPatURIMain
2521     GSRNPIDPatURIBody = %s"gsrnp:" GSRNIDPatURIMain
2522     GSRNIDPatURIMain =
2523         PaddedNumericComponent "." PaddedNumericComponent
2524         / PaddedNumericComponent ".*"
2525         / ".*.*"
2526     GDTIIDPatURIBody = %s"gditi:" GDTIIDPatURIMain
2527     GDTIIDPatURIMain =
2528         PaddedNumericComponent "." PaddedNumericComponentOrEmpty "."
2529     GS3A3Component

```



```
2530 / PaddedNumericComponent "." PaddedNumericComponentOrEmpty "*"
2531 / PaddedNumericComponent ".*.*"
2532 / ".*.*"
2533 CPIIDPatURIBody = %s"spi:" CPIIDPatMain
2534 CPIIDPatMain =
2535     PaddedNumericComponent "." CPreComponent "." NumericComponent
2536 / PaddedNumericComponent "." CPreComponent "*"
2537 / PaddedNumericComponent ".*.*"
2538 / ".*.*"
2539 SGCNIDPatURIBody = %s"sgcn:" SGCNIDPatURIMain
2540 SGCNIDPatURIMain =
2541     PaddedNumericComponent "." PaddedNumericComponentOrEmpty "."
2542 PaddedNumericComponent
2543 / PaddedNumericComponent "." PaddedNumericComponentOrEmpty "*"
2544 / PaddedNumericComponent ".*.*"
2545 / ".*.*"
2546 GINCIDPatURIBody = %s"ginc:" GINCIDPatURIMain
2547 GINCIDPatURIMain =
2548     PaddedNumericComponent "." GS3A3Component
2549 / PaddedNumericComponent "*"
2550 / ".*"
2551 GSINIDPatURIBody = %s"gsin:" GSINIDPatURIMain
2552 GSINIDPatURIMain =
2553     PaddedNumericComponent "." PaddedNumericComponent
2554 / PaddedNumericComponent "*"
2555 / ".*"
2556 ITIPIDPatURIBody = %s"itip:" ITIPPatURIMain
2557 ITIPPatURIMain =
2558     4(PaddedNumericComponent ".") GS3A3Component
2559 / 4(PaddedNumericComponent ".") "*"
2560 / 2(PaddedNumericComponent ".") ".*.*"
2561 / PaddedNumericComponent ".*.*.*"
2562 / ".*.*.*.*"
2563 UPUIIDPatURIBody = %s"upui:" UPUIPatURIMain
2564 UPUIPatURIMain =
2565     2(PaddedNumericComponent ".") GS3A3Component
2566 / 2(PaddedNumericComponent ".") "*"
2567 / PaddedNumericComponent ".*.*"
2568 / ".*.*"
2569 PGLNIDPatURIBody = %s"pgl:" PGLNPatURIMain
2570 PGLNPatURIMain =
2571     2(PaddedNumericComponent ".")
2572 / PaddedNumericComponent "*"
2573 / ".*"
2574 DODIDPatURIBody = %s"usdod:" DODIDPatMain
2575 DODIDPatMain =
2576     CAGECodeOrDODAAC "." DoDSerialNumber
2577 / CAGECodeOrDODAAC "*"
2578 / ".*"
2579 ADIIDPatURIBody = %s"adi:" ADIIDPatMain
```

```

2580 ADIIDPatMain =
2581     CAGCodeOrDODAAC "." ADIComponent "." ADIExtendedComponent
2582     / CAGCodeOrDODAAC "." ADIComponent ".*"
2583     / CAGCodeOrDODAAC ".*.*"
2584     / ".*.*"
2585
2585 BICIDPatURIBody = %s"bic:" BICIDPatMain
2586
2586 BICIDPatMain = BICURIBody / "*"
2587
2587 IMOVNIDPatURIBody = %s"imovn:" IMOVNPatMain
2588
2588 IMOVNPatMain = VesselNumber / "*"
2589
2590
  
```

2591 8.2 Semantics

2592 The meaning of a Pure Identity Pattern URI (`urn:epc:idpat:`) is formally defined as denoting a
 2593 set of a set of pure identity EPCs, respectively.

2594 The set of EPCs denoted by a specific Pure Identity Pattern URI is defined by the following decision
 2595 procedure, which says whether a given Pure Identity EPC URI belongs to the set denoted by the
 2596 Pure Identity Pattern URI.

2597 Let `urn:epc:idpat:Scheme:P1.P2...Pn` be a Pure Identity Pattern URI. Let
 2598 `urn:epc:id:Scheme:C1.C2...Cn` be a Pure Identity EPC URI, where the `Scheme` field of both
 2599 URIs is the same. The number of components (n) depends on the value of `Scheme`.

2600 First, any Pure Identity EPC URI component C_i is said to *match* the corresponding Pure Identity
 2601 Pattern URI component P_i if:

- 2602 ■ P_i is a `NumericComponent`, and C_i is equal to P_i ; or
- 2603 ■ P_i is a `PaddedNumericComponent`, and C_i is equal to P_i both in numeric value as well as in
 2604 length; or
- 2605 ■ P_i is a `GS3A3Component`, `ADIExtendedComponent`, `ADIComponent`, or `CPreComponent`
 2606 and C_i is equal to P_i , character for character; or
- 2607 ■ P_i is a `CAGCodeOrDODAAC`, and C_i is equal to P_i ; or
- 2608 ■ P_i is a `StarComponent` (and C_i is anything at all)

2609 Then the Pure Identity EPC URI is a member of the set denoted by the Pure Identity Pattern URI if
 2610 and only if C_i matches P_i for all $1 \leq i \leq n$.

2611 9 Memory Organisation of Gen 2 RFID tags

2612 9.1 Types of Tag Data

2613 RFID Tags, particularly Gen 2 RFID tags, may carry data of three different kinds:

- 2614 ■ **Business Data:** Information that describes the physical object to which the tag is affixed. This
 2615 information includes the EPC that uniquely identifies the physical object, and may also include
 2616 other data elements carried on the tag. This information is what business applications act upon,
 2617 and so this data is commonly transferred between the data capture level and the business
 2618 application level in a typical implementation architecture. Most standardised business data on an
 2619 RFID tag is equivalent to business data that may be found in other data carriers, such as
 2620 barcodes. Business data can also include sensor data (e.g., as encoded in the XPC bits).
- 2621 ■ **Control Information:** Information that is used by data capture applications to help control the
 2622 process of interacting with tags. Control Information includes data that helps a capturing
 2623 application filter out tags from large populations to increase read efficiency, special handling

information that affects the behaviour of capturing application, information that controls tag security features, and so on. Control Information is typically *not* passed directly to business applications, though Control Information may influence how a capturing application presents business data to the business application level. Unlike Business Data, Control Information has no equivalent in barcodes or other data carriers.

- Tag Manufacture Information:** Information that describes the Tag itself, as opposed to the physical object to which the tag is affixed. Tag Manufacture information includes a manufacturer ID and a code that indicates the tag model. It may also include information that describes tag capabilities, as well as a unique serial number assigned at manufacture time. Usually, Tag Manufacture Information is like Control Information in that it is used by capture applications but not directly passed to business applications. In some applications, the unique serial number that may be a part of Tag Manufacture Information is used in addition to the EPC, and so acts like Business Data. Like Control Information, Tag Manufacture Information has no equivalent in barcodes or other data carriers.

It should be noted that these categories are slightly subjective, and the lines may be blurred in certain applications. However, they are useful for understanding how TDS is structured, and are a good guide for their effective and correct use.

The following table summarises the information above.

Table 9-1 Kinds of Data on a Gen 2 RFID Tag

| Information type | Description | Where on Gen 2 Tag | Where typically used | Bar Code Equivalent |
|------------------------------------|---|---|---|--|
| <i>Business Data</i> | Describes the physical object to which the tag is affixed. | EPC Bank (excluding PC and XPC bits, and filter value within EPC) User Memory Bank | Data Capture layer and Business Application layer | Yes: GS1 keys, Application Identifiers (AIs) |
| <i>Control Information</i> | Facilitates efficient tag interaction | Reserved Bank EPC Bank: PC and XPC bits, and filter value within EPC | Data Capture layer | No |
| <i>Tag Manufacture Information</i> | Describes the tag itself, as opposed to the physical object to which the tag is affixed | TID Bank | Data Capture layer Unique tag manufacture serial number may reach Business Application layer | No |

9.2 Gen 2 Tag Memory Map

Binary data structures defined in TDS are intended for use in RFID Tags, particularly in UHF Class 1 Gen 2 tags (also known as ISO/IEC 18000-63 [ISO18000-63] tags). The air interface standard [UHFC1G2] specifies the structure of memory on Gen 2 tags, as shown in Figure 9-1. Specifically, it specifies that memory in these tags consists of four separately addressable banks, numbered 00, 01, 10, and 11. It also specifies the intended use of each bank, and constraints upon the content of each bank dictated by the behaviour of the air interface. For example, the layout and meaning of the Reserved bank (bank 00), which contains passwords that govern certain air interface commands, is fully specified in [UHFC1G2].

For those memory banks and memory locations that have no special meaning to the air interface (i.e., are "just data" as far as the air interface is concerned), TDS normatively specifies the content and meaning of these memory locations.

Following the convention established in [UHFC1G2], memory addresses are described using hexadecimal bit addresses, where each bank begins with bit 00_h and extends upward to as many bits as each bank contains, the capacity of each bank being constrained in some respects by [UHFC1G2] but ultimately may vary with each tag make and model. Bit 00_h is considered the most significant bit of each bank, and when binary fields are laid out into tag memory the most significant bit of any given field occupies the lowest-numbered bit address occupied by that field.

NOTE: For reasons of TDS 1.x continuity, with respect to individual fields, the least significant bit of individual TDS 1.x fields is numbered zero. For example, the TDS 1.x-era specification of Access

2663
2664
2665
2666
2667
2668
2669

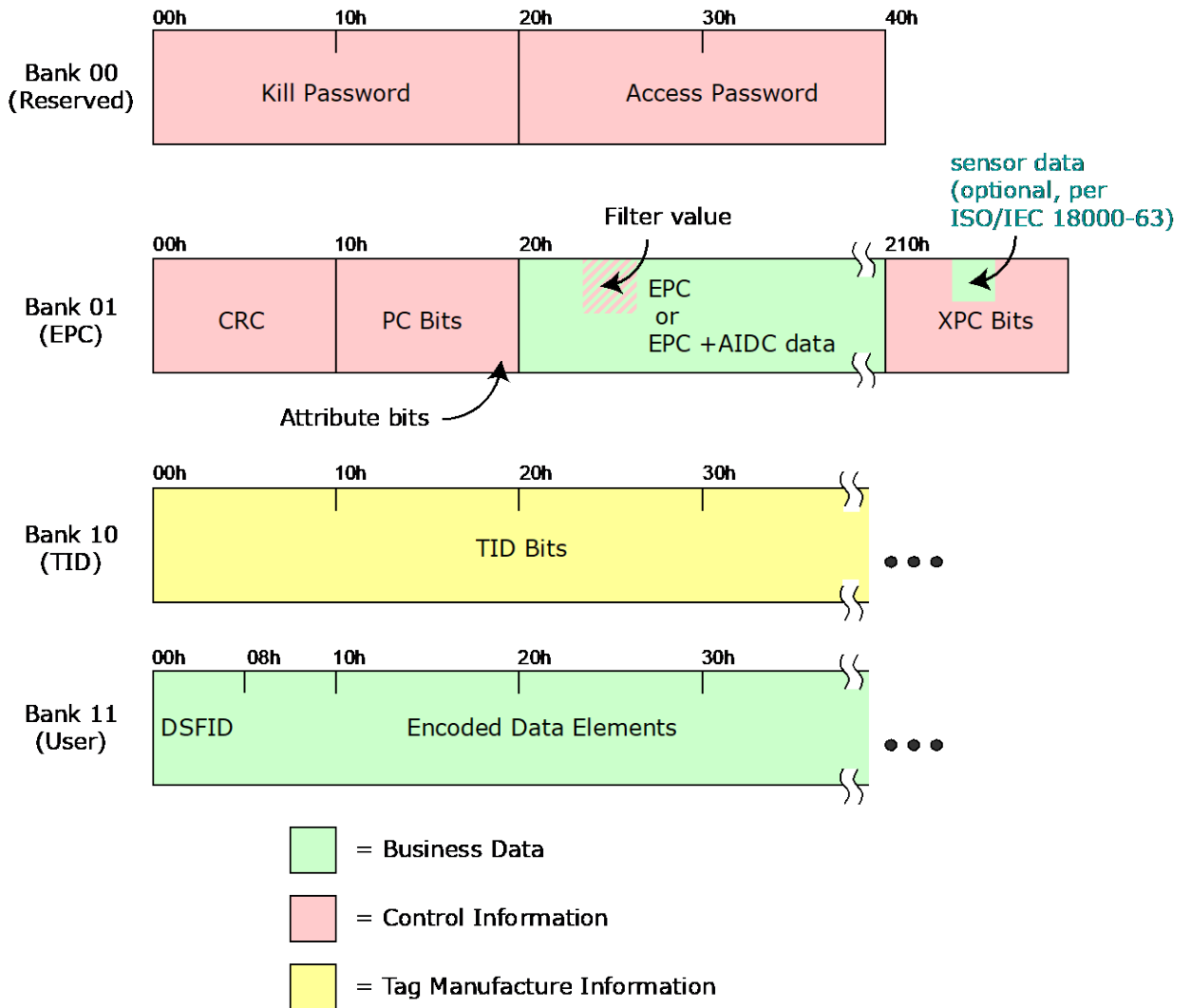
Password is a 32-bit unsigned integer consisting of bits $b_{31}b_{30}...b_0$, where b_{31} is the most significant bit and b_0 is the least significant bit. When the Access Password is stored at address $20_h - 3F_h$ (inclusive) in the Reserved bank of a Gen 2 tag, the most significant bit b_{31} is stored at tag address 20_h and the least significant bit b_0 is stored at address $3F_h$.

NOTE: Encodings new to TDS 2.0 are described counting bits from left to right.

The following figure shows the layout of memory on a Gen 2 tag, The colours indicate the type of data following the categorisation in [Figure 3-1](#).

2670

Figure 9-1 Gen 2 Tag Memory Map

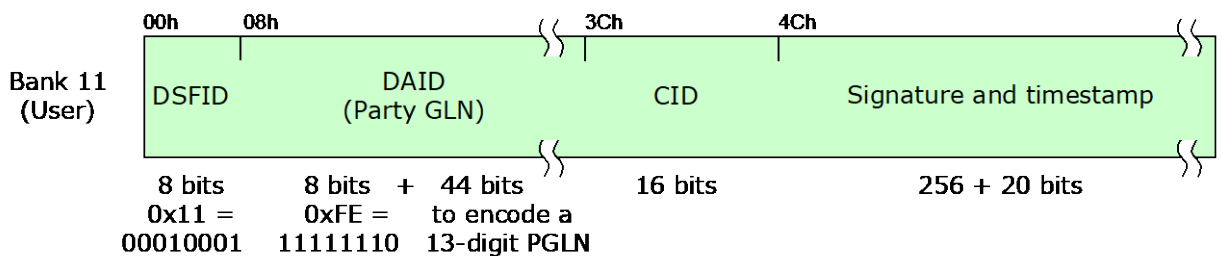


2671

2672

2673

Encoding an ISO/IEC 20248 DigSig in user memory using DSFID = 0x11 (Data Format 17)



The following table describes the fields in the memory map above.

Table 9-2 Gen 2 Memory Map

| Bank | Bits | Field | Description | Category | Where Specified |
|-----------------------|-------------------------------------|---------------|---|--|---|
| Bank 00 (Reserved) | 00 _h – 1F _h | Kill Passwd | A 32-bit password that must be presented to the tag in order to complete the Gen 2 "kill" command. | Control Info | [UHFC1G2] |
| | 20 _h – 2F _h | Access Passwd | A 32-bit password that must be presented to the tag in order to perform privileged operations | Control Info | [UHFC1G2] |
| Bank 01 (EPC) | 00 _h – 0F _h | CRC | A 16-bit Cyclic Redundancy Check computed over the contents of the EPC bank. | Control Info | [UHFC1G2] |
| | 10 _h – 1F _h | PC Bits | Protocol Control bits (see below) | Control Info | (see below) |
| | 20 _h – end | EPC | Electronic Product Code, plus filter value and any optionally included "AIDC data" (normatively specified in TDS 2.0) appended to the EPC itself. Note that the DSGTIN+ scheme supports the expression of a prioritised date field ahead of the GTIN within its binary encoding. This is then zero-filled to the word boundary . The Electronic Product code is a globally unique identifier for the physical object to which the tag is affixed. The filter value provides a means to improve tag read efficiency by selecting a subset of tags of interest. | Business Data (except filter value, which is Control Info) | The EPC is defined in Sections 6 , 7 , and 13 . The filter values are defined in Section 10 . |
| | 210 _h – 21F _h | XPC Bits | Extended Protocol Control bits. If bit 16 _h of the EPC bank is set to one, then bits 210 _h – 21F _h (inclusive) contain additional protocol control bits as specified in [UHFC1G2] | Control Info | [UHFC1G2] |
| Bank 10 (TID) | 00 _h – end | TID Bits | Tag Identification bits, which provide information about the tag itself, as opposed to the physical object to which the tag is affixed. | Tag Manufacture Info | Section 16 |

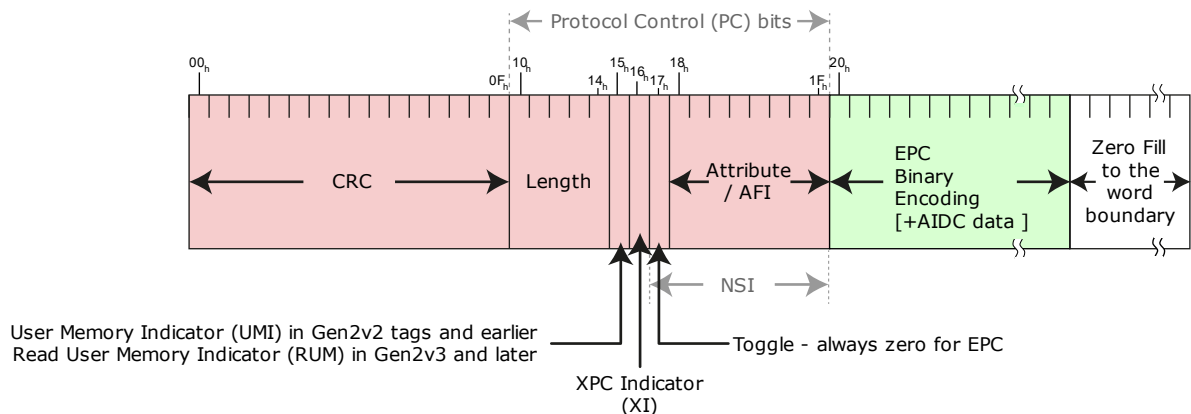
| Bank | Bits | Field | Description | Category | Where Specified |
|----------------|-----------------------|-------|---|---------------|------------------------------------|
| Bank 11 (User) | 00 _h – end | DSFID | <p>Logically, the content of user memory is a set of name-value pairs, where the name part is an OID [ASN.1] and the value is a character string. Physically, the first few bits are a Data Storage Format Identifier as specified in ISO/IEC 15961 [ISO15961] and ISO/IEC 15962 [ISO15962]. The DSFID specifies the format for the remainder of the user memory bank. The DSFID is typically eight bits in length, but may be extended further as specified in [ISO15961].</p> <p>When the DSFID specifies Access Method 2, the format of the remainder of user memory is "Packed Objects" as specified in Section 17. This format is recommended for use in EPC applications. The physical encoding in the Packed Objects data format is as a sequence of "Packed Objects," where each Packed Object includes one or more name-value pairs whose values are compacted together.</p> <p>When the DSFID specifies Access Method 17, the format of the remainder of user memory after the 8-bit DSFID (set to 00010001) is an ISO/IEC 20248 DigSig (digital signature data structure) consisting of: Domain Authority ID (DAID) = 8 bits (set to 11111110) +44 bits to encode the GS1 Party GLN (417) of the organisation that is accountable for the signature, Certificate ID (CID) = 16 bits, Signature and timestamp = 256+20 bits. A 20 bit timestamp supports a signing period of one year, with a resolution of minutes.</p> | Business Data | [ISO15961], [ISO15962], Section 17 |

2675
2676

The following figure illustrates in greater detail the first few bits of the EPC Bank (Bank 01), and in particular shows the various fields within the Protocol Control bits (bits 10_h – 1F_h, inclusive).

2677

Figure 9-2 Gen 2 Protocol Control (PC) Bits Memory Map



2678

2679 **9.3 PC bits**

2680 The following table specifies the meaning of the PC bits:

2681 **Table 9-3** Gen 2 Protocol Control (PC) Bits Memory Map

| Bits | Field | Name | Description |
|-----------------------------------|---|----------------------------|--|
| 10 _h – 14 _h | L4-L0 | Length | Represents the number of 16-bit words comprising the EPC field (below), beginning with the 8-bit, EPC Binary Header at 20 _h and including any optional "AIDC data" (normatively specified in TDS 2.0) appended to the EPC itself. Note that the DSGTIN+ scheme enables a prioritised date value to be encoded before the GTIN in the binary encoding. See discussion in Section 15.1.1 for the encoding of this field. |
| 15 _h | UMI (Gen2v2 tags and earlier) | User Memory Indicator | <p>(for Gen2v2 tags and earlier)</p> <p>Bit 15_h may be fixed by the Tag manufacturer or computed by the Tag.</p> <p>If UMI=0: If fixed, the Tag does not have File_0 (User Memory) and is incapable of allocating memory to it. If computed, then File_0 (User Memory) is not allocated or does not contain data.</p> <p>If UMI=1: If fixed, the Tag has File 0 (User Memory) or is capable of allocating memory to it. If computed, then File_0 (User Memory) is allocated and contains data.</p> |
| | RUM (Gen2v3 tags and later) | Read User Memory indicator | <p>(for Gen2v3 tags and later)</p> <p>Bit 15_h indicates that a Tag has memory allocated to File_0 and, if the Interrogator initiated the inventory round using a <i>QueryX</i>, that the Tag has encoded data in File_0. A Tag shall compute RUM according to Table 6-17 of [UHFC1G2] regardless of the lock or permalock status of EPC memory or the untraceability status of File_0.</p> <p>If an Interrogator changes a Tag's User Word Count (UWC) value (see [UHFC1G2]) or changes the number of words allocated to File_0 memory, then a Tag's RUM may be incorrect until the Interrogator power-cycles the Tag. Additionally, RUM may change without power cycling; for example, a Tag with memory allocated to File_0 and with UWC=0 will have RUM=0₂ after <i>QueryX</i> begins initializing an inventory round, but after a <i>Write</i> to the StoredPC, then RUM may change since the Tag may recompute its StoredCRC.</p> |

| Bits | Field | Name | Description |
|--|-------|---|---|
| 16 _h | XI | XPC W1 Indicator | <p>Indicates whether an XPC W1 is present for the specific circumstances described below.</p> <p>If XI=0: Either (i) Tag has no XPC_W1, or (ii) T=0 and either bits 210h–217h or bits 210h–218h (at tag manufacturer's option) of EPC memory are all zero, or (iii) T=1 and bits 210h–21Fh of EPC memory are all zero.</p> <p>If XI=1: Tag has an XPC_W1 and either (i) T=0 and at least one bit of 210h–217h or 210h–218h (at tag manufacturer's option) of EPC memory is nonzero, or (ii) T=1 and at least one bit of 210h–21Fh of EPC memory is nonzero.</p> |
| 17 _h | T | Numbering System Identifier Toggle | <p>If T=0: Indicates a GS1 EPCglobal application, encoded in compliance with TDS.</p> <p>If T=1: Indicates a non-GS1 EPCglobal application, not encoded in compliance with TDS. In particular, indicates that bits 18_h – 1F_h contain the ISO Application Family Identifier (AFI) as defined in [ISO15961] and the remainder of the EPC bank contains a Unique Item Identifier (UII) appropriate for that AFI.</p> |
| 18 _h – 1F _h (if toggle=0) | | RFU (Gen2v2, Gen2v3 tags) or Attribute bits (Gen v1.x tags) | <p>Gen2 v1.x tags: Bits that may guide the handling of the physical object to which the tag is affixed.</p> |
| 18 _h – 1F _h (if toggle=1) | AFI | Application Family Identifier | <p>An Application Family Identifier that specifies a non-GS1 EPCglobal application, not encoded in compliance with TDS, for which the remainder of the EPC bank contains a Unique Item Identifier (UII) appropriate for that AFI. (see [ISO15961])</p> |

2682 Bits 17_h – 1F_h (inclusive) are collectively known as the Numbering System Identifier (NSI). It should
 2683 be noted, however, that when the toggle bit (bit 17_h) is zero, the numbering system is always the
 2684 Electronic Product Code (EPC), and bits 18_h – 1F_h contain the Attribute bits whose purpose is
 2685 completely unrelated to identifying the numbering system being used.

2686 The Attribute bits are "control information" that may be used by capturing applications to guide the
 2687 capture process. Attribute Bits may be used to determine whether the physical object to which a tag
 2688 is affixed requires special handling of any kind.

2689 Attribute bits are available for all EPC types. The Attribute bit definitions specified here apply
 2690 regardless of which EPC scheme is used.

2691 Because Attribute bits are not part of the EPC, they are not included when the EPC is represented as
 2692 a pure identity URI **or as a GS1 Digital Link URI**, nor should the Attribute bits be considered as
 2693 part of the EPC by business applications. Capturing applications may, however, read the Attribute
 2694 bits and pass them upwards to business applications in some data field other than the EPC. It should
 2695 be recognised, however, that the purpose of the Attribute bits is to assist in the data capture and
 2696 physical handling process, and in most cases the Attribute bits will be of limited or no value to
 2697 business applications. The Attribute bits are not intended to provide reliable master data or product
 2698 descriptive attributes for business applications to use.

2699

9.4 XPC bits

2700

2701

The following table specifies the meaning of the XPC bits for tags whose Numbering System Identifier Toggle (T, bit 17h) is zero.

2702

For tags whose Numbering System Identifier Toggle is non-zero, please refer to [ISO18000-63] for XPC bit assignments.

2703

2704

Table 9-4 Gen 2 Extended Protocol Control (XPC) Bits Memory Map

| Bits | Field | Description | Settings |
|-------------------------------------|--------------------------------------|------------------------------------|---|
| 210 _h | XEB | XPC_W2 indicator | 0: Tag has no XPC_W2 or all bits of XPC_W2 are zero-valued 1: Tag has an XPC_W2 and at least one bit of XPC_W2 is nonzero |
| 211 _h – 213 _h | RFU | Reserved for future use | Annex L of Gen2 v2 permits using the ISO XPC bit definitions; accordingly, bits 211 _h -217 _h might not be fixed zeroes. Specifically, bits 214 _h to 217 _h are used by sensor tags |
| 214 _h – 217 _h | RFU (Gen2v2 tags and earlier) | | |
| 214 _h | SA (Gen2v3 tags and later) | Sensor Alarm indicator | 0: Tag is not reporting an alarm condition or does not support the SA flag 1: Tag is reporting an alarm condition |
| 215 _h | SS (Gen2v3 tags and later) | Simple Sensor indicator | 0: Tag does not have a Simple Sensor 1: Tag has a Simple Sensor |
| 216 _h | FS (Gen2v3 tags and later) | Full Function Sensor indicator | 0: Tag does not have a Full Function Sensor 1: Tag has a Full Function Sensor |
| 217 _h | SN (Gen2v3 tags and later) | Snapshot Sensor indicator | 0: Tag does not have a Snapshot Sensor 1: Tag has a Snapshot Sensor |
| 218 _h | B | Battery-assisted passive indicator | 0: Tag is passive or does not support the B flag 1: Tag is battery-assisted |
| 219 _h | C | Computed response indicator | 0: ResponseBuffer is empty or Tag does not support a ResponseBuffer 1: ResponseBuffer contains a response |
| 21A _h | SLI | SL indicator | 0: Tag has a deasserted SL flag or does not support the SLI bit 1: Tag has an asserted SL flag |
| 21B _h | TN | Tag Notification indicator | 0: Tag does not assert a notification or does not support the TN bit 1: Tag asserts a notification |
| 21C _h | U | Untraceable indicator | 0: Tag is traceable or does not support the U bit 1: Tag is untraceable |
| 21D _h | K | Killable indicator | 0: Tag is not killable by Kill command or does not support the K bit 1: Tag can be killed by Kill command. |
| 21E _h | NR | Non-Removable indicator | 0: Tag is removable from its host item or does not support the NR bit 1: Tag is not removable from its host item |

| Bits | Field | Description | Settings |
|------------------|-------|------------------|---|
| 21F _h | H | Hazmat indicator | 0: Tagged item is not hazardous material or Tag does not support the H bit 1: Tagged item is hazardous material Hazardous materials are defined by government regulations. Generally, a hazardous material (HazMat) is any item or agent (biological, chemical, radiological, and/or physical), which has the potential to cause harm to humans, animals, or the environment, either by itself or through interaction with other factors. |

NOTE:

Per section 6.3.2.1.2.2 Protocol-control (PC) word (StoredPC and PacketPC) of Gen2v2:
"If a Tag has T=0, XI=0, implements an XPC_W1, and is not truncating then the Tag substitutes the 8 LSBs of XPC_W1 (i.e. EPC memory 218h – 21Fh) for the 8 LSBs of the StoredPC (i.e. PC memory 18h – 1Fh) in its reply."

ALSO NOTE:

Gen2 *Inventory* operations do not use the READ, WRITE, or BLOCKWRITE commands for obtaining the contents of the EPC memory bank. Instead, Gen2 *Inventory* operations use the ACK command, and the host will only receive the PacketPC, which combines info from both the StoredPC and XPC_W1. The ACK command may also include the XPC_W1 in its entirety for a sensor tag.

Capture of the EPC memory bank (MB01) is a process that is optimized by the air protocol. As such, what is commonly referred to as the "PC word" during capture is really the 8 most significant bits (MSBs) of the Protocol Control (PC) bits, concatenated with 8 least significant bits (LSBs) of the Extended Protocol Control (XPC) bits when XI=0; when XI=1, the "PC word" during capture consists of all 16 PC bits, along with all 16 XPC bits.

10 Filter Value

The filter value is additional control information that may be included in the EPC memory bank of a Gen 2 tag. The intended use of the filter value is to allow an RFID reader to select or deselect the tags corresponding to certain physical objects, to make it easier to read the desired tags in an environment where there may be other tags present in the environment. For example, if the goal is to read the single tag on a pallet, and it is expected that there may be hundreds or thousands of item-level tags present, the performance of the capturing application may be improved by using the Gen 2 air interface to select the pallet tag and deselect the item-level tags.

Filter values are available for all EPC types except for the General Identifier (GID). There is a different set of standardised filter value values associated with each type of EPC, as specified below.

It is essential to understand that the filter value is additional "control information" that is *not* part of the Electronic Product Code. The filter value does not contribute to the unique identity of the EPC. For example, it is *not* permissible to attach two RFID tags to different physical objects where both tags contain the same EPC, even if the filter values are different on the two tags.

Because the filter value is not part of the EPC, the filter value is *not* included when the EPC is represented as a pure identity URI, element string or GS1 Digital Link URI, nor should the filter value be considered as part of the EPC by business applications. It is also important to note that filter values can only be used within EPC RFID data carriers and there is no barcode equivalent. Nor should filter values be confused with the indicator digit of a GTIN nor the extension digit of an SSCC.

Capturing applications may, however, read the filter value and pass it upwards to business applications in some data field other than the EPC. It should be recognised, however, that the purpose of the filter values is to assist in the data capture process, and in most cases the filter value will be of limited or no value to business applications. The filter value is *not* intended to provide a reliable packaging-level indicator for business applications to use.

2744 **10.1 Use of "Reserved" and "All Others" Filter Values**

2745 In the following sections, filter values marked as "reserved" are reserved for assignment by GS1 in
 2746 future versions of this specification. Implementations of the encoding and decoding rules specified
 2747 herein SHALL accept any value of the filter values, whether reserved or not. Applications, however,
 2748 SHOULD NOT direct an encoder to write a reserved value to a tag, nor rely upon a reserved value
 2749 decoded from a tag, as doing so may cause interoperability problems if a reserved value is assigned
 2750 in a future revision to this specification.

2751 Each EPC scheme includes a filter value identified as "All Others." This filter value means that the
 2752 object to which the tag is affixed does not match the description of any of the other filter values
 2753 defined for that EPC scheme. In some cases, the "All Others" filter value may appear on a tag that
 2754 was encoded to conform to an earlier version of this specification, at which time no other suitable
 2755 filter value was available. When encoding a new tag, the filter value should be set to match the
 2756 description of the object to which the tag is affixed, with "All Others" being used only if a suitable
 2757 filter value for the object is not defined in this specification.

2758 **10.2 Filter Values for SGTIN and DSGTIN+ EPC Tags**

2759 The normative specifications for Filter Values for SGTIN EPC Tags are specified below.

2760 **Table 10-1** SGTIN Filter Values

| Type | Filter Value | Binary Value |
|---|--------------|--------------|
| All Others (see Section 10.1) | 0 | 000 |
| Point of Sale (POS) Trade Item | 1 | 001 |
| Full Case for Transport * | 2 | 010 |
| Reserved (see Section 10.1) | 3 | 011 |
| Inner Pack Trade Item Grouping for Handling | 4 | 100 |
| Reserved (see Section 10.1) | 5 | 101 |
| Unit Load ** | 6 | 110 |
| Unit inside Trade Item or component inside a product not intended for individual sale | 7 | 111 |

2761 * When used as the EPC Filter Value for an SGTIN, "**Full Case for Transport**" denotes a case or
 2762 carton whose composition of multiple POS trade items is standardised via master data and can be
 2763 consistently (re-) ordered in this configuration by referencing a single GTIN.

2764 ** When used as the EPC Filter Value for an SGTIN, "**Unit Load**" denotes one or more trade items
 2765 contained on a pallet or other type of load carrier (e.g. roolly, dolly, tote, garment rack, bag, sack,
 2766 etc.) *, making them suitable for transport, stacking, and storage as a unit, whose composition is
 2767 standardised via master data and can be consistently (re-)ordered in this configuration by
 2768 referencing a single GTIN.

2769 **10.3 Filter Values for SSCC EPC Tags**

2770 The normative specifications for Filter Values for SSCC EPC Tags are specified below.

2771 **Table 10-2** SSCC Filter Values

| Type | Filter Value | Binary Value |
|--|--------------|--------------|
| All Others (see Section 10.1) | 0 | 000 |
| Reserved (see Section 10.1) | 1 | 001 |
| Full Case for Transport | 2 | 010 |
| Reserved (see Section 10.1) | 3 | 011 |
| Reserved (see Section 10.1) | 4 | 100 |
| Reserved (see Section 10.1) | 5 | 101 |

| Type | Filter Value | Binary Value |
|--|--------------|--------------|
| Unit Load | 6 | 110 |
| Reserved (see Section 10.1) | 7 | 111 |

2772 **10.4 Filter Values for SGLN EPC Tags**

2773 **Table 10-3** SGLN Filter Values

| Type | Filter Value | Binary Value |
|--|--------------|--------------|
| All Others (see Section 10.1) | 0 | 000 |
| Reserved (see Section 10.1) | 1 | 001 |
| Reserved (see Section 10.1) | 2 | 010 |
| Reserved (see Section 10.1) | 3 | 011 |
| Reserved (see Section 10.1) | 4 | 100 |
| Reserved (see Section 10.1) | 5 | 101 |
| Reserved (see Section 10.1) | 6 | 110 |
| Reserved (see Section 10.1) | 7 | 111 |

2774 **10.5 Filter Values for GRAI EPC Tags**

2775 **Table 10-4** GRAI Filter Values

| Type | Filter Value | Binary Value |
|--|--------------|--------------|
| All Others (see Section 10.1) | 0 | 000 |
| Reserved (see Section 10.1) | 1 | 001 |
| Reserved (see Section 10.1) | 2 | 010 |
| Reserved (see Section 10.1) | 3 | 011 |
| Reserved (see Section 10.1) | 4 | 100 |
| Reserved (see Section 10.1) | 5 | 101 |
| Reserved (see Section 10.1) | 6 | 110 |
| Reserved (see Section 10.1) | 7 | 111 |

2776 **10.6 Filter Values for GIAI EPC Tags**

2777 **Table 10-5** GIAI Filter Values

| Type | Filter Value | Binary Value |
|--|--------------|--------------|
| All Others (see Section 10.1) | 0 | 000 |
| Rail Vehicle | 1 | 001 |
| Reserved (see Section 10.1) | 2 | 010 |
| Reserved (see Section 10.1) | 3 | 011 |
| Reserved (see Section 10.1) | 4 | 100 |
| Reserved (see Section 10.1) | 5 | 101 |
| Reserved (see Section 10.1) | 6 | 110 |
| Reserved (see Section 10.1) | 7 | 111 |

2778 **10.7 Filter Values for GSRN and GSRNP EPC Tags**

2779 **Table 10-6** GSRN and GSRNP Filter Values

| Type | Filter Value | Binary Value |
|--|--------------|--------------|
| All Others (see Section 10.1) | 0 | 000 |
| Reserved (see Section 10.1) | 1 | 001 |
| Reserved (see Section 10.1) | 2 | 010 |
| Reserved (see Section 10.1) | 3 | 011 |
| Reserved (see Section 10.1) | 4 | 100 |
| Reserved (see Section 10.1) | 5 | 101 |
| Reserved (see Section 10.1) | 6 | 110 |
| Reserved (see Section 10.1) | 7 | 111 |

2780 **10.8 Filter Values for GDTI EPC Tags**

2781 **Table 10-7** GDTI Filter Values

| Type | Filter Value | Binary Value |
|--|--------------|--------------|
| All Others (see Section 10.1) | 0 | 000 |
| Travel Document * | 1 | 001 |
| Reserved (see Section 10.1) | 2 | 010 |
| Reserved (see Section 10.1) | 3 | 011 |
| Reserved (see Section 10.1) | 4 | 100 |
| Reserved (see Section 10.1) | 5 | 101 |
| Reserved (see Section 10.1) | 6 | 110 |
| Reserved (see Section 10.1) | 7 | 111 |

2782 * A **Travel Document** is an identity document issued by a government or international treaty
 2783 organisation to facilitate the movement of individuals across international boundaries.

2784 **10.9 Filter Values for CPI EPC Tags**

2785 **Table 10-8** CPI Filter Values

| Type | Filter Value | Binary Value |
|--|--------------|--------------|
| All Others (see Section 10.1) | 0 | 000 |
| Reserved (see Section 10.1) | 1 | 001 |
| Reserved (see Section 10.1) | 2 | 010 |
| Reserved (see Section 10.1) | 3 | 011 |
| Reserved (see Section 10.1) | 4 | 100 |
| Reserved (see Section 10.1) | 5 | 101 |
| Reserved (see Section 10.1) | 6 | 110 |
| Reserved (see Section 10.1) | 7 | 111 |

2786 **10.10 Filter Values for SGCN EPC Tags**

2787 **Table 10-9** SGCN Filter Values

| Type | Filter Value | Binary Value |
|--|--------------|--------------|
| All Others (see Section 10.1) | 0 | 000 |
| Reserved (see Section 10.1) | 1 | 001 |
| Reserved (see Section 10.1) | 2 | 010 |
| Reserved (see Section 10.1) | 3 | 011 |
| Reserved (see Section 10.1) | 4 | 100 |
| Reserved (see Section 10.1) | 5 | 101 |
| Reserved (see Section 10.1) | 6 | 110 |
| Reserved (see Section 10.1) | 7 | 111 |

2788 **10.11 Filter Values for ITIP EPC Tags**

2789 **Table 10-10** ITIP Filter Values

| Type | Filter Value | Binary Value |
|--|--------------|--------------|
| All Others (see Section 10.1) | 0 | 000 |
| Reserved (see Section 10.1) | 1 | 001 |
| Reserved (see Section 10.1) | 2 | 010 |
| Reserved (see Section 10.1) | 3 | 011 |
| Reserved (see Section 10.1) | 4 | 100 |
| Reserved (see Section 10.1) | 5 | 101 |
| Reserved (see Section 10.1) | 6 | 110 |
| Reserved (see Section 10.1) | 7 | 111 |

2790 **10.12 Filter Values for GID EPC Tags**

2791 The GID EPC scheme does not provide for the use of filter values.

2792 **10.13 Filter Values for DOD EPC Tags**

2793 Filter values for US DoD EPC Tags are as specified in [USDOD].

2794 **10.14 Filter Values for ADI EPC Tags**

2795 **Table 10-11** ADI Filter Values

| Type | Filter Value | Binary Value |
|--|--------------|--------------------|
| All Others (see Section 10.1) | 0 | 000000 |
| Item, other than an item to which filter values 8 through 63 apply | 1 | 000001 |
| Carton | 2 | 000010 |
| Reserved (see Section 10.1) | 3 thru 5 | 000011 thru 000101 |
| Pallet | 6 | 000110 |
| Reserved (see Section 10.1) | 7 | 000111 |
| Seat cushions | 8 | 001000 |

| Type | Filter Value | Binary Value |
|--|--------------|--------------------|
| Seat covers | 9 | 001001 |
| Seat belts | 10 | 001010 |
| Galley, Galley carts and other Galley Service Equipment | 11 | 001011 |
| Unit Load Devices, cargo containers | 12 | 001100 |
| Aircraft Security items (life vest boxes, rear lavatory walls, lavatory ceiling access hatches) | 13 | 001101 |
| Life vests | 14 | 001110 |
| Oxygen generators | 15 | 001111 |
| Engine components | 16 | 010000 |
| Avionics | 17 | 010001 |
| Experimental ("flight test") equipment | 18 | 010010 |
| Other emergency equipment (smoke masks, PBE, crash axes, medical kits, smoke detectors, flashlights, safety cards, etc.) | 19 | 010011 |
| Other rotables; e.g., line or base replaceable | 20 | 010100 |
| Other repairable | 21 | 010101 |
| Other cabin interior | 22 | 010110 |
| Other repair (exclude component); e.g., structure item repair | 23 | 010111 |
| Passenger Seats (structure) | 24 | 011000 |
| IFEs (In-Flight Entertainment) Systems | 25 | 011001 |
| Reserved (see Section 10.1) | 26 thru 55 | 011010 thru 110111 |
| Location Identifier (*) | 56 | 111000 |
| Documentation | 57 | 111001 |
| Tools | 58 | 111010 |
| Ground Support Equipment | 59 | 111011 |
| Other Non-flyable equipment | 60 | 111100 |
| Reserved for internal company use | 61 thru 63 | 111101 thru 111111 |

2796
2797
2798
2799
2800

! **Non-Normative:** When assigning filter values to tagged parts, the filter values chosen should be as specific as possible. For example, a filter value of 17 (Avionics) is a better choice for a radar black box than the more general category of 20 (Other Rotables). On the other hand, a filter value of 20 (Other Rotables) would be appropriate for a radar antenna in the nose cone of a plane since 17 (Avionics) would not be accurate.

2801
2802
2803
2804

✓ **Note:** location identifier may act differently from an item "identifying" tag in that it identifies a location that may be referenced by other items. Thus, an item might have an identification tag, but also a location tag. An example might be a particular part of an aircraft or even the entire aircraft.

2805
2806
2807
2808
2809
2810
2811

! **Non-Normative:** One example of "location" could be a particular airplane "tail number". For example, Airline XYZ has a fleet of 200 737s with the same interior configuration, and once you are inside of it, you can't tell which particular 737 you are in. This Airline wants to place RFID "location marker(s)" with the tail number encoded, and place them inside the passenger doors, or cargo hold doors. The doors could end up having two tags, one is for the door itself, i.e. it has the door part number, serial number, and things, and another tag is for "location" purpose.

11 Attribute bits (refer to 9.3 and 9.4)

This contents of this section have now been subsumed into sections [9.3](#) and [9.4](#).

12 EPC Tag URI and EPC Raw URI

The EPC memory bank of a Gen 2 tag contains a binary-encoded EPC, along with other control information. Applications do not normally process binary data directly. An application wishing to read the EPC may receive the EPC as a Pure Identity EPC URI, as defined in Section 6. In other situations, however, a capturing application may be interested in the control information on the tag as well as the EPC. Also, an application that writes the EPC memory bank needs to specify the values for control information that are written along with the EPC. In both of these situations, the EPC Tag URI and EPC Raw URI may be used.

For EPC schemes defined in TDS before TDS v2.0, the EPC Tag URI specifies both the EPC and the values of control information in the EPC memory bank. It also specifies which of several variant binary coding schemes is to be used (e.g., the choice between SGTIN-96 and SGTIN-198). As such, an EPC Tag URI completely and uniquely specifies the contents of the EPC memory bank for those EPC schemes for which it is defined. The EPC Raw URI also specifies the complete contents of the EPC memory bank, but represents the memory contents as a single decimal or hexadecimal numeral. The new EPC schemes defined in TDS v2.0 have not defined an equivalent EPC Tag URI syntax; instead the encoding/decoding is between the binary string and the corresponding GS1 element string, GS1 Digital Link URI or equivalently, the set of GS1 Application Identifiers and their values, as shown in [Figure 3-1](#). It should also be noted that the new EPC schemes defined in TDS 2.0 all permit the encoding of additional AIDC data after the EPC within the EPC/UII memory bank, as an alternative to encoding such data in the user memory bank.

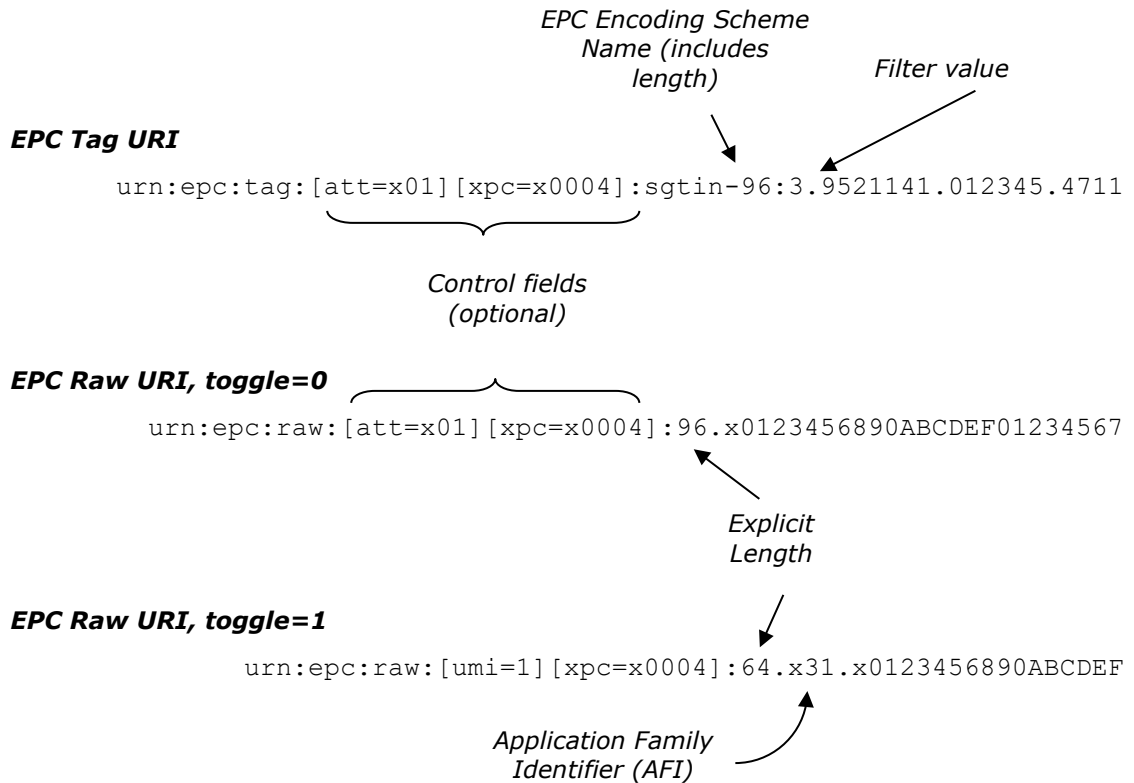
12.1 Structure of the EPC Tag URI and EPC Raw URI

The EPC Tag URI begins with `urn:epc:tag:`, and is used when the EPC memory bank contains a valid EPC. EPC Tag URIs resemble Pure Identity EPC URIs, but with added control information. The EPC Raw URI begins with `urn:epc:raw:`, and is used when the EPC memory bank does not contain a valid EPC. This includes situations where the toggle bit (bit 17_n) is set to one, as well as situations where the toggle bit is set to zero but the remainder of the EPC bank does not conform to the coding rules specified in Section 14, either because the header bits are unassigned or the remainder of the binary encoding violates a validity check for that header.

The following figure illustrates these URI forms.

2843

Figure 12-1 Illustration of EPC Tag URI and EPC Raw URI



2844

2845

2846

2847

2848

2849

2850

The first form in the figure, the EPC Tag URI, is used for a valid EPC. It resembles the Pure Identity EPC URI, with the addition of optional control information fields as specified in Section 12.2.2 and a (non-optional) filter value. The EPC scheme name (`sgtin-96` in the example above) specifies a particular binary encoding scheme, and so it includes the length of the encoding. This is in contrast to the Pure Identity EPC URI which identifies an EPC scheme but not a specific binary encoding (e.g., `sgtin` but not specifically `sgtin-96`).

2851

2852

2853

2854

2855

2856

2857

The EPC raw URI illustrated by the second example in the figure can be used whenever the toggle bit (bit 17_h) is zero, but is typically only used if the first form cannot (that is, if the contents of the EPC bank cannot be decoded according to Section 14.3.9). It specifies the contents of bit 20_h onward as a single hexadecimal numeral. The number of bits in this numeral is determined by the "length" field in the EPC bank of the tag (bits 10_h – 14_h). (The grammar in Section 12.4 includes a variant of this form in which the contents are specified as a decimal numeral. This form is deprecated.)

2858

2859

2860

The EPC Raw URI illustrated by the third example in the figure is used when the toggle bit (bit 17_h) is one. It is similar to the second form, but with an additional field between the length and payload that reports the value of the AFI field (bits 18_h – 1F_h) as a hexadecimal numeral.

2861

2862

Each of these forms is fully defined by the encoding and decoding procedures specified in Sections 14.3 and 14.4.

2863

12.2 Control Information

2864

2865

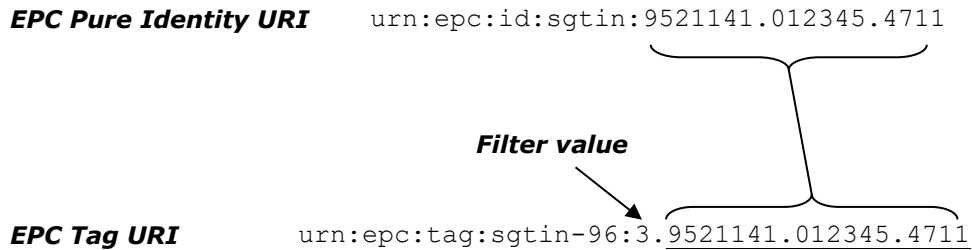
2866

The EPC Tag URI and EPC Raw URI specify the complete contents of the Gen 2 EPC memory bank, including control information such as filter values and Attribute bits. This section specifies how control information is included in these URIs.

2867 **12.2.1 Filter Values**

2868 Filter values are only available when the EPC bank contains a valid EPC, and only then when the EPC
 2869 is an EPC scheme other than GID. In the EPC Tag URI, the filter value is indicated as an additional
 2870 field following the scheme name and preceding the remainder of the EPC, as illustrated below:

2871 **Figure 12-2** Illustration of Filter Value within EPC Tag URI



2872
 2873 The filter value is a decimal integer. The allowed values of the filter value are specified in
 2874 Section [10](#).

2875 **12.2.2 Other control information fields**

2876 Control information in the EPC bank apart from the filter values is stored separately from the EPC.
 2877 Such information can be represented both in the EPC Tag URI and the EPC Raw URI, using the
 2878 name-value pair syntax described below.

2879 In both URI forms, control field name-value pairs may occur following the urn:epc:tag: or
 2880 urn:epc:raw:, as illustrated below:

2881 urn:epc:tag:[att=x01][xpc=x0004]:sgtin-96:3.9521141.112345.400

2882 urn:epc:raw:[att=x01][xpc=x0004]:96.x012345689ABCDEF01234567

2883 Each element in square brackets specifies the value of one control information field. An omitted field
 2884 is equivalent to specifying a value of zero. As a limiting case, if no control information fields are
 2885 specified in the URI it is equivalent to specifying a value of zero for all fields. This provides back-
 2886 compatibility with earlier versions of TDS.

2887 The available control information fields are specified in the following table.

2888 **Table 12-1** Control information fields

| Field | Syntax | Description | Read/Write |
|-----------------------|-------------|---|---|
| Attribute Bits | [att=xNN] | The value of the Attribute bits (bits 18 _n - 1F _n), as a two-digit hexadecimal numeral NN. This field is only available if the toggle bit (bit 17 _n) is zero. | Read / Write |
| User Memory Indicator | [umi=B] | The value of the user memory indicator bit (bit 15 _n). The value B is either the digit 0 or the digit 1. | Read / Write Note that certain Gen 2 Tags may ignore the value written to this bit, and some may calculate the value of the bit from the contents of user memory. See [UHFC1G2]. |
| Extended PC Bits | [xpc=xNNNN] | The value of the XPC bits (bits 210 _n -21F _n) as a four-digit hexadecimal numeral NNNN. | Read only |

2889 The user memory indicator and extended PC bits are calculated by the tag as a function of other
 2890 information on the tag or based on operations performed to the tag. Therefore, these fields cannot
 2891 be written directly. When reading from a tag, any of the control information fields may appear in the

2892 URI that results from decoding the EPC memory bank. When writing a tag, the `umi` and `xpc` fields
 2893 will be ignored when encoding the URI into the tag.
 2894 To aid in decoding, any control information fields that appear in a URI must occur in alphabetical
 2895 order (the same order as in the table above).

2896 **!** **Non-Normative:** Examples: The following examples illustrate the use of control information
 2897 fields in the EPC Tag URI and EPC Raw URI.

2898 `urn:epc:tag:sgtin-96:3.9521141.112345.400`

2899 This is a tag with an SGTIN EPC, filter bits = 3, the hazardous material Attribute bit set to
 2900 zero, no user memory (user memory indicator = 0), and not recommissioned (extended PC =
 2901 0). This illustrates back-compatibility with earlier versions of the Tag Data Standard.

2902 This is a tag with an SGTIN EPC, filter bits = 3, the hazardous material Attribute bit set to
 2903 one, no user memory (user memory indicator = 0), and not recommissioned (extended PC =
 2904 0). This URI might be specified by an application wishing to commission a tag with the
 2905 hazardous material bit set to one and the filter bits and EPC as shown.

2906 `urn:epc:raw:[att=x01][umi=1][xpc=x0004]:96.x1234567890ABCDEF01234567`

2907 This is a tag with toggle=0, random data in bits 20_h onward (not decodable as an EPC), the
 2908 hazardous material Attribute bit set to one, non-zero contents in user memory, and has been
 2909 recommissioned (as indicated by the extended PC).

2910 `urn:epc:raw:[xpc=x0001]:96.xC1.x1234567890ABCDEF01234567`

2911 This is a tag with toggle=1, Application Family Indicator = C1 (hexadecimal), and has had its
 2912 user memory killed (as indicated by the extended PC).

2913 12.3 EPC Tag URI and EPC Pure Identity URI

2914 The Pure Identity EPC URI as defined in Section 6 is a representation of an EPC for use in
 2915 information systems. The only information in a Pure Identity EPC URI is the EPC itself. The EPC Tag
 2916 URI, in contrast, contains additional information: it specifies the contents of all control information
 2917 fields in the EPC memory bank, and it also specifies which encoding scheme is used to encode the
 2918 EPC into binary. Therefore, to convert a Pure Identity EPC URI to an EPC Tag URI, additional
 2919 information must be provided. Conversely, to extract a Pure Identity EPC URI from an EPC Tag URI,
 2920 this additional information is removed. The procedures in this section specify how these conversions
 2921 are done.

2922 12.3.1 EPC Binary Coding Schemes

2923 For each EPC scheme as specified in Section 6, there are one or more corresponding EPC Binary
 2924 Coding Schemes that determine how the EPC is encoded into binary representation for use in RFID
 2925 tags. When there is more than one EPC Binary Coding Scheme available for a given EPC scheme, a
 2926 user must choose which binary coding scheme to use. In general, the shorter binary coding schemes
 2927 result in fewer bits and therefore permit the use of less expensive RFID tags containing less
 2928 memory, but are restricted in the range of serial numbers that are permitted. The longer binary
 2929 coding schemes allow for the full range of serial numbers permitted by the GS1 General
 2930 Specifications, but require more bits and therefore more expensive RFID tags. TDS 2.0 introduces
 2931 several new EPC schemes and corresponding binary encodings that support simpler
 2932 encoding/decoding rules and efficient variable-length encoding using the most efficient character set
 2933 for the actual value being encoded. The new EPC schemes and binary encodings introduced in TDS
 2934 2.0 do not use partition tables and require no knowledge of the length of the GS1 Company Prefix;
 2935 this is intended to improve interoperability between EPC and other data carriers such as 1D and 2D
 2936 barcodes, in which the length of the GS1 Company Prefix is not considered to be significant.

2937 For EPC schemes defined before TDS 2.0, it is important to note that two EPCs are the same if and
 2938 only if the Pure Identity EPC URIs are character for character identical. A long binary encoding (e.g.,
 2939 SGTIN-198) is *not* a different EPC from a short binary encoding (e.g., SGTIN-96) if the GS1
 2940 Company Prefix, item reference with indicator, and serial numbers are identical. The new EPC
 2941 binary encodings introduced in TDS v2.0 do not define corresponding Pure Identity EPC URIs but

2942
2943
2944
2945

their values are considered to be equivalent to those encoded in a short binary encoding (e.g., SGTIN-96) or a long binary encoding (e.g., SGTIN-198) if they all correspond to the same canonical GS1 Digital Link URI or the same GS1 element string, e.g. if the SGTIN-96, SGTIN-198, SGTIN+ or DSGTIN+ all express the same value for GTIN, AI (01) and Serial Number, AI (21).

2946
2947

All EPC schemes defined before TDS 2.0 remain valid in TDS 2.0. However, the new EPC schemes and binary encodings introduced in TDS 2.0 may be particularly suitable for the following scenarios:

2948
2949

1. When there is a desire/need to encode additional AIDC data after the EPC within the EPC/UII memory bank

2950
2951

2. When there is a desire or need to simplify encoding/decoding or difficulty in determining the length of a GS1 Company Prefix.

2952
2953
2954
2955
2956

3. When there is a desire to use fewer bits than the maximum when using alphanumeric values with a constrained character set or where a variable-length value is significantly shorter than its maximum permitted length. In such situations, the encoding indicators and length indicators in the new EPC schemes may result in a lower total bit count than for the equivalent "long" EPC schemes defined before TDS 2.0.

2957
2958

The following table enumerates the available EPC binary coding schemes, and indicates the limitations imposed on serial numbers.

2959

Table 12-2 EPC Binary Coding Schemes and their limitations

| EPC Scheme | EPC Binary Coding Scheme | EPC + Filter Bit Count | Includes Filter Value | Serial number limitation |
|------------|--------------------------|------------------------|-----------------------|---|
| sgtin | sgtin-96 | 96 | Yes | Numeric-only, no leading zeros, decimal value must be less than 2^{38} (i.e., decimal value less than or equal to 274,877,906,943). |
| | sgtin-198 | 198 | Yes | All values permitted by GS1 General Specifications (up to 20 alphanumeric characters) |
| | sgtin+ | Variable up to 216 | | |
| | dsgtin+ | Variable up to 236 | | |
| sscc | sscc-96 | 96 | Yes | All values permitted by GS1 General Specifications (11 – 5 decimal digits including extension digit, depending on GS1 Company Prefix length) |
| | sscc+ | 84 | | |
| sgln | sgln-96 | 96 | Yes | Numeric-only, no leading zeros, decimal value must be less than 2^{41} (i.e., decimal value less than or equal to 2,199,023,255,551). |
| | sgln-195 | 195 | Yes | All values permitted by GS1 General Specifications (up to 20 alphanumeric characters) |
| | sgln+ | Variable up to 212 | | |
| grai | grai-96 | 96 | Yes | Numeric-only, no leading zeros, decimal value must be less than 2^{38} (i.e., decimal value less than or equal to 274,877,906,943). |
| | grai-170 | 170 | Yes | All values permitted by GS1 General Specifications (up to 16 alphanumeric characters) |
| | grai+ | Variable up to 188 | | |
| giai | giai-96 | 96 | Yes | Numeric-only, no leading zeros, decimal value must be less than a limit that varies according to the length of the GS1 Company Prefix. See Section 14.6.5.1 . |
| | giai-202 | 202 | Yes | All values permitted by GS1 General Specifications (up to 18 – 24 alphanumeric characters, depending on company prefix length) |
| | giai+ | Variable up to 216 | | |

| EPC Scheme | EPC Binary Coding Scheme | EPC + Filter Bit Count | Includes Filter Value | Serial number limitation |
|------------|--|------------------------|--|--|
| gsrn | gsrn-96 | 96 | Yes | All values permitted by GS1 General Specifications (11 – 5 decimal digits, depending on GS1 Company Prefix length) |
| | gsrn+ | 84 | | |
| gsrnp | gsrnp-96 | 96 | Yes | All values permitted by GS1 General Specifications (11 – 5 decimal digits, depending on GS1 Company Prefix length) |
| | gsrnp+ | 84 | | |
| gdti | gdti-96 | 96 | Yes | Numeric-only, no leading zeros, decimal value must be less than 2^{41} (i.e., decimal value less than or equal to 2,199,023,255,551). |
| | gdti-113 (DEPRECATED as of TDS 1.9) | 113 | Yes | All values permitted by GS1 General Specifications prior to [GS1GS12.0] (up to 17 decimal digits, with or without leading zeros) |
| | gdti-174 | 174 | Yes | All values permitted by GS1 General Specifications (up to 17 alphanumeric characters) |
| | gdti+ | Variable up to 191 | | |
| sgcn | sgcn-96 | 96 | Yes | Numeric only, up to 12 decimal digits, with or without leading zeros. |
| | sgcn+ | Variable up to 108 | | |
| itip | itip-110 | 110 | Yes | Numeric-only, no leading zeros, decimal value must be less than 2^{38} (i.e., decimal value less than or equal to 274,877,906,943). |
| | itip-212 | 212 | Yes | All values permitted by GS1 General Specifications (up to 20 alphanumeric characters) |
| | itip+ | Variable up to 232 | | |
| gid | gid-96 | 96 | No | Numeric-only, no leading zeros, decimal value must be less than 2^{36} (i.e., decimal value must be less than or equal to 68,719,476,735). |
| usdod | usdod-96 | 96 | See "United States Department of Defense Supplier's Passive RFID Information Guide" [USDOD]. | |
| adi | adi-var | Variable | Yes | See Section 14.6.14.1 |
| cpi | cpi-96 | 96 | Yes | Serial Number: Numeric-only, no leading zeros, decimal value must be less than 2^{31} (i.e., decimal value less than or equal to 2,147,483,647). The component/part reference is also limited to values that are numeric-only, with no leading zeros, and whose length is less than or equal to 15 minus the length of the GS1 Company Prefix |
| | cpi-var | Variable | Yes | All values permitted by GS1 General Specifications (up to 12 decimal digits, no leading zeros). |
| | cpi+ | Variable up to 274 | | |

! **Non-Normative:** Explanation: For the SGTIN, SGLN, GRAI, and GIAI EPC schemes, the serial number according to the GS1 General Specifications is a variable length, alphanumeric string. This means that serial number 34, 034, 0034, etc, are all different serial numbers, as are P34, 34P, 0P34, P034, and so forth. In order to provide for up to 20 alphanumeric characters, 140 bits are required to encode the serial number within schemes such as SGTIN-198 that were defined before TDS 2.0. This is why the "long" binary encodings all have such a large number of bits. Similar considerations apply to the GDTI EPC scheme, except that the

2960
2961
2962
2963
2964
2965
2966

2967
2968
2969
2970
2971
2972
2973

GDTI only allows digit characters (but still permits leading zeros). For the new EPC binary encodings introduced in TDS 2.0, instead of allocating sufficient bit capacity to accommodate the maximum permitted length of serial number components and all permitted characters, the new EPC schemes use encoding indicators and length indicators to enable fewer bits to be used if the actual value of a serial number component is shorter than the maximum permitted length or if it uses a more constrained character set (e.g. only uses numeric digits even where alphanumeric characters are permitted). This is explained in further detail in section [14.5](#).

2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984

In order to accommodate the very common 96-bit RFID tag, additional binary coding schemes are introduced that only require 96 bits. In order to fit within 96 bits, some serial numbers have to be excluded. The 96-bit encodings of SGTIN, SGLN, GRAI, GIAI, and GDTI are limited to serial numbers that consist only of digits, which do not have leading zeros (unless the serial number consists in its entirety of a single 0 digit), and whose value when considered as a decimal numeral is less than 2^B , where B is the number of bits available in the binary coding scheme. The choice to exclude serial numbers with leading zeros was an arbitrary design choice at the time the 96-bit encodings were first defined; for example, an alternative would have been to permit leading zeros, at the expense of excluding other serial numbers. But it is impossible to escape the fact that in B bits there can be no more than 2^B different serial numbers.

2985
2986
2987
2988
2989
2990
2991

When decoding a "long" binary encoding defined before TDS 2.0 or any of the new EPC binary encodings introduced in TDS 2.0, it is not permissible to strip off leading zeros when the binary encoding includes leading zero characters. Likewise, when encoding an EPC into either the "short" or "long" form or new EPC binary encodings introduced in TDS 2.0, it is not permissible to strip off leading zeros prior to encoding. This means that EPCs whose serial numbers have leading zeros can only be encoded in the "long" form or in the new EPC binary encodings introduced in TDS 2.0, which are also capable of preserving leading zeros.

2992
2993
2994
2995
2996

In certain applications, it is desirable for the serial number to always contain a specific number of characters. Reasons for this may include wanting a predictable length for the EPC URI string, or for having a predictable size for a corresponding barcode encoding of the same identifier. In certain barcode applications, this is accomplished through the use of leading zeros. If 96-bit tags are used, however, the option to use leading zeros does not exist.

2997
2998
2999
3000
3001
3002
3003
3004

Therefore, in applications that both require 96-bit tags and require that the serial number be a fixed number of characters, it is recommended that numeric serial numbers be used that are in the range $10^D \leq \text{serial} < 10^{D+1}$, where D is the desired number of digits. For example, if 11-digit serial numbers are desired, an application can use serial numbers in the range 10,000,000,000 through 99,999,999,999. Such applications must take care to use serial numbers that fit within the constraints of 96-bit tags. For example, if 12-digit serial numbers are desired for SGTIN-96 encodings, then the serial numbers must be in the range 100,000,000,000 through 274,877,906,943.

3005
3006
3007

It should be remembered, however, that many applications do not require a fixed number of characters in the serial number, and so all serial numbers from 0 through the maximum value (without leading zeros) may be used with 96-bit tags.

3008 **12.3.2 EPC Pure Identity URI to EPC Tag URI**

3009

Given:

3010
3011
3012
3013
3014
3015
3016
3017

- An EPC Pure Identity URI as specified in Section [6.3](#). This is a string that matches the EPC-URI production of the grammar in Section [6.3](#).
- A selection of a binary coding scheme to use. This is one of the binary coding schemes specified in the "EPC Binary Coding Scheme" column of [Table 12-2](#). The chosen binary coding scheme must be one that corresponds to the EPC scheme in the EPC Pure Identity URI.
- A filter value, if the "Includes Filter Value" column of [Table 12-2](#) indicates that the binary encoding includes a filter value.
- The value of the Attribute bits.

3018

- The value of the user memory indicator.

3019

Validation:

3020

- The serial number portion of the EPC (the characters following the rightmost dot character) must conform to any restrictions implied by the selected binary coding scheme, as specified by the "Serial Number Limitation" column of [Table 12-2](#).

3021

3022

- The filter value must be in the range $0 \leq filter \leq 7$.

3023

Procedure:

3024

1. Starting with the EPC Pure Identity URI, replace the prefix `urn:epc:id:` with `urn:epc:tag:`.

3025

2. Replace the EPC scheme name with the selected EPC binary coding scheme name. For example, replace `sgtin` with `sgtin-96` or `sgtin-198`.

3026

3027

3. If the selected binary coding scheme includes a filter value, insert the filter value as a single decimal digit following the rightmost colon (":") character of the URI, followed by a dot (".") character.

3028

3029

3030

4. If the Attribute bits are non-zero, construct a string `[att=xNN]`, where NN is the value of the Attribute bits as a 2-digit hexadecimal numeral.

3031

3032

5. If the user memory indicator is non-zero, construct a string `[umi=1]`.

3033

6. If Step 4 or Step 5 yielded a non-empty string, insert those strings following the rightmost colon (":") character of the URI, followed by an additional colon character.

3034

3035

7. The resulting string is the EPC Tag URI.

3036

3037

12.3.3 EPC Tag URI to EPC Pure Identity URI

Given:

3038

1. An EPC Tag URI as specified in Section 12. This is a string that matches the `TagURI` production of the grammar in Section 12.4.

3039

3040

Procedure:

3041

1. Starting with the EPC Tag URI, replace the prefix `urn:epc:tag:` with `urn:epc:id:`.

3042

2. Replace the EPC binary coding scheme name with the corresponding EPC scheme name. For example, replace `sgtin-96` or `sgtin-198` with `sgtin`.

3043

3044

3. If the coding scheme includes a filter value, remove the filter value (the digit following the rightmost colon character) and the following dot (".") character.

3045

3046

4. If the URI contains one or more control fields as specified in Section 12.2.2, remove them and the following colon character.

3047

3048

5. The resulting string is the Pure Identity EPC URI.

3049

3050

12.4 Grammar

The following grammar specifies the syntax of the EPC Tag URI and EPC Raw URI. The grammar makes reference to grammatical elements defined in Sections 5 and 6.3.

3051

3052

`TagOrRawURI = TagURI / RawURI`

3053

`TagURI = %s"urn:epc:tag:" TagURIControlBody`

3054

`TagURIControlBody = 0*1(ControlField+ ":") TagURIBody`

3055

`TagURIBody = SGTINTagURIBody / SSCCTagURIBody / SGLNTagURIBody /`

3056

`GRAITagURIBody / GIAITagURIBody / GDTITagURIBody /`

3057

`GSRNTagURIBody / GSRNPTagURIBody / ITIPTagURIBody /`

3058

`GIDTagURIBody / SGCNTagURIBody / DODTagURIBody /`

3059

`ADITagUriBody / CPITagURIBody`

3060

```

3061
3062 SGTINTagURIBody = SGTINEncName ":" NumericComponent "." SGTINURIBody
3063 SGTINEncName = %s"sgtin-96" / %s"sgtin-198"
3064 SSCCTagURIBody = SSCCEncName ":" NumericComponent "." SSCCURIBody
3065 SSCCEncName = %s"sscc-96"
3066 SGLNTagURIBody = SGLNEncName ":" NumericComponent "." SGLNURIBody
3067 SGLNEncName = %s"sgln-96" / %s"sgln-195"
3068 GRAITagURIBody = GRAIEncName ":" NumericComponent "." GRAIURIBody
3069 GRAIEncName = %s"grai-96" / %s"grai-170"
3070 GIAITagURIBody = GIAIEncName ":" NumericComponent "." GIAIURIBody
3071 GIAIEncName = %s"giai-96" / %s"giai-202"
3072 GSRNTagURIBody = GSRNEncName ":" NumericComponent "." GSRNURIBody
3073 GSRNEncName = %s"gsrn-96"
3074 GSRNPEncName = %s"gsrnp-96"
3075 GDTITagURIBody = GDTIEncName ":" NumericComponent "." GDTIURIBody
3076 GDTIEncName = %s"gdtd-96" / %s"gdtd-113" / %s"gdtd-174"
3077 CPITagURIBody = CPIEncName ":" NumericComponent "." CPIURIBody
3078 CPIEncName = %s"cp-96" / %s"cp-var"
3079 SGCNTagURIBody = SGCNEncName ":" NumericComponent "." SGCNURIBody
3080 SGCNEncName = %s"sgcn-96"
3081 ITIPTagURIBody = ITIPEncName ":" NumericComponent "." ITIPURIBody
3082 ITIPEncName = %s"itip-110" / %s"itip-212"
3083 GIDTagURIBody = GIDEncName ":" GIDURIBody
3084 GIDEncName = %s"gid-96"
3085 DODTagURIBody = DODEncName ":" NumericComponent "." DODURIBody
3086 DODEncName = %s"usdod-96"
3087 ADITagURIBody = ADIEncName ":" NumericComponent "." ADIURIBody
3088 ADIEncName = %s"adi-var"
3089 RawURI = %s"urn:epc:raw:" RawURIControlBody
3090 RawURIControlBody = 0*1( ControlField+ ":") RawURIBody
3091 RawURIBody = DecimalRawURIBody / HexRawURIBody / AFIRawURIBody
3092 DecimalRawURIBody = NonZeroComponent "." NumericComponent
3093 HexRawURIBody = NonZeroComponent ".x" HexComponentOrEmpty
3094 AFIRawURIBody = NonZeroComponent ".x" HexComponent ".x" HexComponentOrEmpty
3095 ControlField = "[" ControlName "=" ControlValue "]"
3096 ControlName = %s"att" / %s"umi" / %s"xpc"
3097 ControlValue = BinaryControlValue / HexControlValue
3098 BinaryControlValue = "0" / "1"
3099 HexControlValue = %s"x" HexComponent
  
```

13 URIs for EPC Tag Encoding patterns

3100

3101 Certain software applications need to specify rules for filtering lists of tags according to various
 3102 criteria. This specification provides an EPC Tag Pattern URI for this purpose. An EPC Tag Pattern URI
 3103 does not represent a single tag encoding, but rather refers to a set of tag encodings. A typical
 3104 pattern looks like this:

```

3105 urn:epc:pat:sgtin-96:3.0652642.[102400-204700].*
  
```

3106 This pattern refers to any tag containing a 96-bit SGTIN EPC Binary Encoding, whose Filter field is 3,
 3107 whose GS1 Company Prefix is 0652642, whose Item Reference is in the range $102400 \leq$
 3108 *itemReference* ≤ 204700 , and whose Serial Number may be anything at all.

3109 In general, for all EPC schemes defined before TDS v2.0, there is an EPC Tag Pattern URI scheme
 3110 corresponding to each of those EPC Binary Encoding schemes, whose syntax is essentially identical
 3111 except that ranges or the star (*) character may be used in each field.

3112 The new EPC schemes defined in TDS v2.0 have not defined an equivalent EPC Tag URI syntax nor a
 3113 corresponding EPC Tag Pattern URI syntax; instead the encoding/decoding is between the binary
 3114 string and the corresponding GS1 element string, GS1 Digital Link URI or equivalently, the set of
 3115 GS1 Application Identifiers and their values, as shown in [Figure 3-1](#)

3116 For the SGTIN, SSCC, SGLN, GRAI, GIAI, GSRN, GDTI, SGCN and ITIP patterns, the pattern syntax
 3117 slightly restricts how wildcards and ranges may be combined. Only two possibilities are permitted
 3118 for the `CompanyPrefix` field. One, it may be a star (*), in which case the following field
 3119 (`ItemReference`, `SerialReference`, `LocationReference`,
 3120 `AssetType`, `IndividualAssetReference`, `ServiceReference`, `DocumentType`,
 3121 `CouponReference`, `Piece` or `Total`) must also be a star. Two, it may be a specific company
 3122 prefix, in which case the following field may be a number, a range, or a star. A range may not be
 3123 specified for the `CompanyPrefix`.

3124 **!** **Non-Normative:** Explanation: Because the company prefix is variable length, a range may
 3125 not be specified, as the range might span different lengths. When a particular company prefix
 3126 is specified, however, it is possible to match ranges or all values of the following field,
 3127 because its length is fixed for a given company prefix. The other case that is allowed is when
 3128 both fields are a star, which works for all tag encodings because the corresponding tag fields
 3129 (including the `Partition` field, where present) are simply ignored.

3130 The pattern URI for the DoD Construct is as follows:

3131 `urn:epc:pat:usdod-96:filterPat.CAGECodeOrDODAACPat.serialNumberPat`

3132 where `filterPat` is either a filter value, a range of the form `[lo-hi]`, or a * character;
 3133 `CAGECodeOrDODAACPat` is either a CAGE Code/DODAAC or a * character; and `serialNumberPat`
 3134 is either a serial number, a range of the form `[lo-hi]`, or a * character.

3135 The pattern URI for the Aerospace and Defense (ADI) identifier is as follows:

3136 `urn:epc:pat:adi-`
 3137 `var:filterPat.CAGECodeOrDODAACPat.partNumberPat.serialNumberPat`

3138 where `filterPat` is either a filter value, a range of the form `[lo-hi]`, or a * character;
 3139 `CAGECodeOrDODAACPat` is either a CAGE Code/DODAAC or a * character; `partNumberPat` is
 3140 either an empty string, a part number, or a * character; and `serialNumberPat` is either a serial
 3141 number or a * character.

3142 The pattern URI for the Component / Part (CPI) identifier is as follows:

3143 `urn:epc:pat:cpi-96:filterPat.CPI96PatBody.serialNumberPat`

3144 or

3145 `urn:epc:pat:cpi-var:filterPat.CPIVarPatBody`

3146 where `filterPat` is either a filter value, a range of the form `[lo-hi]`, or a * character;
 3147 `CPI96PatBody` is either *.* or a GS1 Company Prefix followed by a dot and either a numeric
 3148 component/part number, a range in the form `[lo-hi]`, or a * character; `serialNumberPat` is
 3149 either a serial number or a * character or a range in the form `[lo-hi]`; and `CPIVarPatBody` is
 3150 either *.*.* or a GS1 Company Prefix followed by a dot followed by a component/part reference
 3151 followed by a dot followed by either a component/part serial number, a range in the form `[lo-hi]` or
 3152 a * character.

3153 13.1 Syntax

3154 The syntax of EPC Tag Pattern URIs is defined by the grammar below.

3155 `PatURI = %s"urn:epc:pat:" PatBody`

3156 `PatBody =`

3157 `GIDPatURIBody /`
 3158 `SGTINPatURIBody /`
 3159 `SGTINAlphaPatURIBody /`
 3160 `SGLNGRAI96PatURIBody /`
 3161 `SGLNGRAIAlphaPatURIBody /`
 3162 `SSCCPatURIBody /`


```

3163          GIAI96PatURIBody /
3164          GIAIAlphaPatURIBody /
3165          GSRNPatURIBody /
3166          GSRNPPatURIBody /
3167          GDTIPatURIBody /
3168          CPIVarPatURIBody /
3169          SGCNPatURIBody /
3170          ITIPPatURIBody /
3171          USDOD96PatURIBody /
3172          ITIP212PatURIBody /
3173          ADIVarPatURIBody /
3174          CPI96PatURIBody
3175  GIDPatURIBody = %s"gid-96:" 2(PatComponent ".") PatComponent
3176  SGTIN96PatURIBody = %s"sgtin-96:" PatComponent "." GS1PatBody "."
3177  PatComponent
3178  SGTINAlphaPatURIBody = %s"sgtin-198:" PatComponent "." GS1PatBody "."
3179  GS3A3PatComponent
3180  SGLNGRAI96PatURIBody = SGLNGRAI96TagEncName ":" PatComponent "." GS1EpatBody
3181  "." PatComponent
3182  SGLNGRAI96TagEncName = %s"sgln-96" / %s"grai-96"
3183  SGLNGRAIAlphaPatURIBody = SGLNGRAIAlphaTagEncName ":" PatComponent "."
3184  GS1EpatBody "." GS3A3PatComponent
3185  SGLNGRAIAlphaTagEncName = %s"sgln-195" / %s"grai-170"
3186  SSSCPatURIBody = %s"sscc-96:" PatComponent "." GS1PatBody
3187  GIAI96PatURIBody = %s"giai-96:" PatComponent "." GS1PatBody
3188  GIAIAlphaPatURIBody = %s"giai-202:" PatComponent "." GS1GS3A3PatBody
3189  GSRNPatURIBody = %s"gsrn-96:" PatComponent "." GS1PatBody
3190  GSRNPPatURIBody = %s"gsrnp-96:" PatComponent "." GS1PatBody
3191  GDTIPatURIBody = GDTI96PatURIBody / GDTI113PatURIBody/ GDTI174PatURIBody
3192  GDTI96PatURIBody = %s"gdti-96:" PatComponent "." GS1EpatBody "."
3193  PatComponent
3194  GDTI113PatURIBody = %s"gdti-113:" PatComponent "." GS1EpatBody "."
3195  PaddedNumericOrStarComponent
3196  GDTI174PatURIBody = %s"gdti-174:" PatComponent "." GS1EpatBody "."
3197  GS3A3PatComponent
3198  CPI96PatURIBody = %s"cpi-96:" PatComponent "." GS1PatBody "." PatComponent
3199  CPIVarPatURIBody = %s"cpi-var:" PatComponent "." CPIVarPatBody
3200  CPIVarPatBody = "*.*.*"
3201  / PaddedNumericComponent "." CPreComponent "." PatComponent
3202  SGCNPatURIBody = SGCN96PatURIBody
3203  SGCN96PatURIBody = %s"sgcn-96:" PatComponent "." GS1EpatBody "."
3204  PaddedNumericOrStarComponent
3205  ITIP110PatURIBody = %s"itip-110:" PatComponent "." GS1PatBody "."
3206  PatComponent "." PatComponent "." PatComponent
3207  ITIP212PatURIBody = %s"itip-212:" PatComponent "." GS1PatBody "."
3208  PatComponent "." PatComponent "." GS3A3PatComponent
3209  USDOD96PatURIBody = %s"usdod-96:" PatComponent "." CAGECodeOrDODAACPat "."
3210  PatComponent
3211  ADIVarPatURIBody = %s"adi-var:" PatComponent "." CAGECodeOrDODAACPat "."
3212  ADIPatComponent "." ADIExtendedPatComponent
3213  PaddedNumericOrStarComponent = PaddedNumericComponent / StarComponent
3214  GS1PatBody = "*.*" / ( PaddedNumericComponent "." PaddedPatComponent )
3215  GS1EpatBody = "*.*" / ( PaddedNumericComponent "." PaddedOrEmptyPatComponent
3216  )
3217  GS1GS3A3PatBody = "*.*" / ( PaddedNumericComponent "." GS3A3PatComponent )
3218  PatComponent = NumericComponent / StarComponent / RangeComponent
3219  PaddedPatComponent = PaddedNumericComponent / StarComponent / RangeComponent
3220  PaddedOrEmptyPatComponent = PaddedNumericComponentOrEmpty
3221  / StarComponent
3222  / RangeComponent

```

3223 `GS3A3PatComponent = GS3A3Component / StarComponent`
 3224 `CAGECodeOrDODAACPat = CAGECodeOrDODAAC / StarComponent`
 3225 `ADIPatComponent = ADIComponent / StarComponent`
 3226 `ADIExtendedPatComponent = ADIExtendedComponent / StarComponent`
 3227 `StarComponent = "*"`
 3228 `RangeComponent = "[" NumericComponent "-" NumericComponent "]"`

3229 For a `RangeComponent` to be legal, the numeric value of the first `NumericComponent` must be
 3230 less than or equal to the numeric value of the second `NumericComponent`.

3231 13.2 Semantics

3232 The meaning of an EPC Tag Pattern URI (`urn:epc:pat:`) is formally defined as denoting a set of
 3233 EPC Tag URIs.

3234 The set of EPCs denoted by a specific EPC Tag Pattern URI is defined by the following decision
 3235 procedure, which says whether a given EPC Tag URI belongs to the set denoted by the EPC Tag
 3236 Pattern URI.

3237 Let `urn:epc:pat:EncName:P1.I..Pn` be an EPC Tag Pattern URI. Let
 3238 `urn:epc:tag:EncName:IC2...Cn` be an EPC Tag URI, where the `EncName` field of both URIs is
 3239 the same. The number of components (n) depends on the value of `EncName`.

3240 First, any EPC Tag URI component C_i is said to *match* the corresponding EPC Tag Pattern URI
 3241 component P_i if:

- 3242 ■ P_i is a `NumericComponent`, and C_i is equal to P_i ; or
- 3243 ■ P_i is a `PaddedNumericComponent`, and C_i is equal to P_i both in numeric value as well as in
3244 length; or
- 3245 ■ P_i is a `GS3A3Component`, `ADIExtendedComponent`, `ADIComponent`, or `CPreComponent`
3246 and C_i is equal to P_i , character for character; or
- 3247 ■ P_i is a `CAGECodeOrDODAAC`, and C_i is equal to P_i ; or
- 3248 ■ P_i is a `RangeComponent` [`lo-hi`], and $lo \leq C_i \leq hi$; or
- 3249 ■ P_i is a `StarComponent` (and C_i is anything at all)

3250 Then the EPC Tag URI is a member of the set denoted by the EPC Pattern URI if and only if C_i
 3251 matches P_i for all $1 \leq i \leq n$.

3252 14 EPC Binary Encoding

3253 This section specifies how EPC Tag URIs or element strings (GS1 Application Identifiers and their
 3254 values) are encoded into binary strings, and conversely how a binary string is decoded into an EPC
 3255 Tag URI (if possible) or element string (GS1 Application Identifiers and their values). The binary
 3256 strings defined by the encoding and decoding procedures in this section are suitable for use in the
 3257 EPC memory bank of a Gen 2 tag.

3258 The general structure of an EPC Binary Encoding as used on a tag is as a string of bits (i.e., a binary
 3259 representation), consisting of a fixed length header followed by a series of fields whose overall
 3260 length, structure, and function are determined by the header value. The assigned header values are
 3261 specified in Section 14.2. Both the encoding and decoding procedures are driven by coding tables
 3262 specified in Section 14.6. Each coding table specifies, for a given header value, the structure of the
 3263 fields following the header.

3264 EPC schemes are defined for most of the globally unique instance identifiers that can be constructed
 3265 using GS1 identification keys – so not only for GTIN but also SSCC, GRAI, GIAI etc. However,
 3266 binary encodings have only been defined for those where there is a strong case for encoding an EPC
 3267 in an RFID data carrier (e.g. for a serialised product instance or for a logistic unit, asset physical
 3268 location) but not for organisations nor for groupings of logistic units that correspond to
 3269 consignments or shipments.

3270 TDS 2.0 introduces alternative modernised EPC binary encodings for all EPC schemes based on GS1
 3271 identifiers, for which a binary encoding was already defined in TDS 1.13. These new EPC binary
 3272 encodings have much simpler translation to/from GS1 element strings on barcodes, with no need to
 3273 know the length of the GS1 Company Prefix, no omission of the check digit and no rearrangement of
 3274 the indicator digit of the GTIN nor the extension digit of the SSCC. The encoding/decoding is
 3275 between the binary string and the corresponding GS1 element string, GS1 Digital Link URI or
 3276 equivalently, the set of GS1 Application Identifiers and their values, as shown in [Figure 3-1](#). These
 3277 new EPC binary encodings all have names ending '+', to denote that they also offer the option of
 3278 encoding additional +AIDC data after the EPC binary string. No EPC Tag URI syntax is defined for
 3279 any of the new EPC schemes introduced in TDS 2.0, so instead of referring to Sections [14.3](#) and
 3280 [14.4](#) for the encoding and decoding procedures, Section [14.5](#) explains the encoding and decoding
 3281 procedures for the new EPC schemes introduced in TDS v2.0 and should be read in conjunction with
 3282 the relevant binary coding table from Section [14.6](#), which provides the binary coding tables for all
 3283 EPC schemes (old and new). A requirement for TDS 2.0 conformance is that implementations of
 3284 decoders SHALL support all of the new encoding and decoding methods in Section 14.5.
 3285 Implementers of encoders SHALL support all of the new encoding methods in Section 14.5 that are
 3286 explicitly mentioned within columns b or h of Table F in Section [15.3](#).

3287 The older EPC schemes defined before TDS 2.0 remain valid and for these EPC schemes, the
 3288 complete procedure for encoding an EPC Tag URI into the binary contents of the EPC memory bank
 3289 of a Gen 2 tag is specified in Section [15.1.1](#). The procedure in Section [15.1.1](#) uses the procedure
 3290 defined below in Section [14.3](#) (encoding URI to binary) to do the bulk of the work. Conversely, the
 3291 complete procedure for decoding the binary contents of the EPC memory bank of a Gen 2 tag into
 3292 an EPC Tag URI (or EPC Raw URI, if necessary) is specified in Section [15.2.2](#). The procedure in
 3293 Section [15.2.2](#) uses the procedure defined below in Section [14.4](#) (decoding binary to URI) to do the
 3294 bulk of the work.


3295 **14.1 Overview of Binary Encoding**

3296 To convert an EPC Tag URI to the EPC Binary Encoding, follow the procedure specified in
 3297 Section [14.3](#), which is summarised as follows. First, the appropriate coding table is selected from
 3298 among the tables specified in Section [14.4.9](#). The correct coding table is the one whose "URI
 3299 Template" entry matches the given EPC Tag URI. Each column in the coding table corresponds to a
 3300 bit field within the final binary encoding. Within each column, a "Coding Method" is specified that
 3301 says how to calculate the corresponding bits of the binary encoding, given some portion of the URI
 3302 as input. The encoding details for each "Coding Method" are given in subsections of Section [14.3](#).

3303 To convert an EPC Binary Encoding into an EPC Tag URI, follow the procedure specified in
 3304 Section [14.4](#), which is summarised as follows. First, the most significant eight bits are looked up in
 3305 the table of EPC binary headers ([Table 14-1](#) in Section [14.2](#)). This identifies the EPC coding scheme,
 3306 which in turn selects a coding table from among those specified in Section [14.6](#). Each column in the
 3307 coding table corresponds to a bit field in the input binary encoding. Within each column, a "Coding
 3308 Method" is specified that says how to calculate a corresponding portion of the output URI, given that
 3309 bit field as input. The decoding details for each "Coding Method" are given in subsections of
 3310 Section [14.4](#).

3311 **14.2 EPC Binary Headers**

3312 As already noted, the general structure of an EPC Binary Encoding as used on a tag is as a string of
 3313 bits (i.e., a binary representation), consisting of a fixed length, 8 bit, header followed by a series of
 3314 fields whose overall length, structure, and function are determined by the header value. For future
 3315 expansion purpose, a header value of 11111111 is defined, to indicate that longer headers beyond
 3316 8 bits is used; this provides for future expansion so that more than 256 header values may be
 3317 accommodated by using longer headers. Therefore, the present specification provides for up to 255
 3318 8-bit headers, plus a currently undetermined number of longer headers.

3319  **Non-Normative:** Back-compatibility note: In earlier versions of TDS, the header was of
 3320 variable length, using a tiered approach in which a zero value in each tier indicated that the
 3321 header was drawn from the next longer tier. For the encodings defined in the earlier
 3322 specification, headers were either 2 bits or 8 bits. Given that a zero value is reserved to
 3323 indicate a header in the next longer tier, the 2-bit header had 3 possible values (01, 10, and

3324
3325
3326

3327
3328
3329

3330
3331
3332
3333
3334

11, not 00), and the 8-bit header had 63 possible values (recognising that the first 2 bits must be 00 and 00000000 is reserved to allow headers that are longer than 8 bits). The 2-bit headers were only used in conjunction with certain 64-bit EPC Binary Encodings.

In more recent versions of TDS, the tiered header approach has been abandoned. Also, all 64-bit encodings (including all encodings that used 2-bit headers) have been deprecated, and should not be used in new applications.

The encoding schemes defined in this version of TDS are shown in [Table 14-1](#). The table also indicates currently unassigned header values that are "Reserved for Future Use" (RFU). All header values that had been reserved for legacy 64-bit encodings, defined in prior versions of the EPC Tag Data Standard, were sunset, effective 1 July, 2009, as previously announced by EPCglobal on 1 July, 2006.

Table 14-1 EPC Binary Header Values

| Header Value (binary) | Header Value (hexadecimal) | Encoding Length (bits) | Coding Scheme |
|------------------------------|----------------------------|------------------------|--|
| 0000 0000 | 00 | NA | Unprogrammed Tag |
| 0000 0001 | 01 | NA | Reserved for Future Use |
| 0000 001x | 02,03 | NA | Reserved for Future Use |
| 0000 01xx | 04,05 06,07 | NA NA | Reserved for Future Use Reserved for Future Use |
| 0000 1000 | 08 | | Reserved for Future Use |
| 0000 1001 | 09 | | Reserved for Future Use |
| 0000 1010 | 0A | | Reserved for Future Use |
| 0000 1011 | 0B | | Reserved for Future Use |
| 0000 1100 to 0000 1111 | 0C to 0F | | Reserved for Future Use |
| 0001 0000 to 0010 1011 | 10 to 2B | NA NA | Reserved for Future Use |
| 0010 1100 | 2C | 96 | GDTI-96 |
| 0010 1101 | 2D | 96 | GSRN-96 |
| 0010 1110 | 2E | 96 | GSRNP-96 |
| 0010 1111 | 2F | 96 | USDoD-96 |
| 0011 0000 | 30 | 96 | SGTIN-96 |
| 0011 0001 | 31 | 96 | SSCC-96 |
| 0011 0010 | 32 | 96 | SGLN-96 |
| 0011 0011 | 33 | 96 | GRAI-96 |
| 0011 0100 | 34 | 96 | GIAI-96 |
| 0011 0101 | 35 | 96 | GID-96 |
| 0011 0110 | 36 | 198 | SGTIN-198 |
| 0011 0111 | 37 | 170 | GRAI-170 |
| 0011 1000 | 38 | 202 | GIAI-202 |

3335

| Header Value (binary) | Header Value (hexadecimal) | Encoding Length (bits) | Coding Scheme |
|-------------------------------|----------------------------|------------------------|--|
| 0011 1001 | 39 | 195 | SGLN-195 |
| 0011 1010 | 3A | 113 | GDTI-113 (DEPRECATED as of TDS 1.9) |
| 0011 1011 | 3B | Variable | ADI-var |
| 0011 1100 | 3C | 96 | CPI-96 |
| 0011 1101 | 3D | Variable | CPI-var |
| 0011 1110 | 3E | 174 | GDTI-174 |
| 0011 1111 | 3F | 96 | SGCN-96 |
| 0100 0000 | 40 | 110 | ITIP-110 |
| 0100 0001 | 41 | 212 | ITIP-212 |
| 0100 0010 to 0111 1111 | 42 to 7F | | Reserved for Future Use |
| 1000 0000 to 1011 1111 | 80 to BF | | Reserved for Future Use |
| 1100 0000 to 1100 1101 | C0 to CD | | Reserved for Future Use |
| 1100 1110 | CE | | Reserved for Future Use |
| 1100 1111 to 1110 0001 | CF to E1 | | Reserved for Future Use |
| 1110 0010 | E2 | | E2 remains PERMANENTLY RESERVED to avoid confusion with the first eight bits of TID memory (Section 16). |
| 1110 0011 to 11010 1111 | E3 to EF | | Reserved for Future Use |
| 1111 0000 | F0 | variable | CPI+ |
| 1111 0001 | F1 | variable | GRAI+ |
| 1111 0010 | F2 | variable | SGLN+ |
| 1111 0011 | F3 | variable | ITIP+ |
| 1111 0100 | F4 | 84 | GSRN+ |
| 1111 0101 | F5 | 84 | GSRNP+ |
| 1111 0110 | F6 | variable | GDTI+ |
| 1111 0111 | F7 | variable | SGTIN+ |
| 1111 1000 | F8 | variable | SGCN+ |
| 1111 1001 | F9 | 84 | SSCC+ |
| 1111 1010 | FA | variable | GIAI+ |
| 1111 1011 | FB | variable | DSGTIN+ |
| 1111 1100 | FC | | RFU |

| Header Value (binary) | Header Value (hexadecimal) | Encoding Length (bits) | Coding Scheme |
|-----------------------|----------------------------|------------------------|--|
| 1111 1101 | FD | | RFU |
| 1111 1110 | FE | | 'Unspecified' / 'Pad' Header for use with optimised <i>Select</i> functionality tentatively planned for Gen2v3 |
| 1111 1111 | FF | NA | Reserved for Future Use (expressly reserved for headers longer than 8 bits) |

14.3 Encoding procedure

The following procedure encodes an EPC Tag URI into a bit string containing the encoded EPC and the filter value (for EPC schemes that have a filter value and for EPC schemes for which an EPC Tag URI is defined; no EPC Tag URI format is defined for new EPC schemes introduced in TDS 2.0 – for those schemes, the starting point for encoding is the corresponding GS1 element string or equivalently, the set of GS1 Application Identifiers and their values. For all new EPC schemes introduced in TDS 2.0, please refer to section [14.5](#) instead). This bit string is suitable for storing in the EPC memory bank of a Gen 2 Tag beginning at bit 20h. See Section [15.1.1](#) for the complete procedure for encoding the entire EPC memory bank, including control information that resides outside of the encoded EPC. (The procedure in Section [15.1.1](#) uses the procedure below as a subroutine.)

Given:

- An EPC Tag URI of the form `urn:epc:tag:scheme:remainder`

Yields:

- A bit string containing the EPC binary encoding of the specified EPC Tag URI, containing the encoded EPC together with the filter value (if applicable); OR
- An exception indicating that the EPC Tag URI could not be encoded.

Procedure:

1. Use the `scheme` to identify the coding table for this URI scheme. If no such scheme exists, stop: this URI is not syntactically legal.
2. Confirm that the URI syntactically matches the URI template associated with the coding table. If not, stop: this URI is not syntactically legal.
3. Read the coding table left-to-right, and construct the encoding specified in each column to obtain a bit string. If the "Coding Segment Bit Count" row of the table specifies a fixed number of bits, the bit string so obtained will always be of this length. The method for encoding each column depends on the "Coding Method" row of the table. If the "Coding Method" row specifies a specific bit string, use that bit string for that column. Otherwise, consult the following sections that specify the encoding methods. If the encoding of any segment fails, stop: this URI cannot be encoded.
4. Concatenate the bit strings from Step 3 to form a single bit string. If the overall binary length specified by the scheme is of fixed length, then the bit string so obtained will always be of that length. The position of each segment within the concatenated bit string is as specified in the "Bit Position" row of the coding table. Section [15.1.1](#) specifies the procedure that uses the result of this step for encoding the EPC memory bank of a Gen 2 tag.

The following sections specify the procedures to be used in Step 3.

14.3.1 "Integer" Encoding Method

The Integer encoding method is used for a segment that appears as a decimal integer in the URI, and as a binary integer in the binary encoding.

3374

Input:

 3375
3376

The input to the encoding method is the URI portion indicated in the "URI portion" row of the encoding table, a character string with no dot (".") characters.

3377

Validity Test:

3378

The input character string must satisfy the following:

3379

- It must match the grammar for `NumericComponent` as specified in Section 5.

3380

- The value of the string SHALL be considered as a decimal integer (i.e., leading zeros are not permitted) and SHALL be less than 2^b , where b is the value specified in the "Coding Segment Bit Count" row of the encoding table.

 3381
3382

3383

If any of the above tests fails, the encoding of the URI fails.

3384

Output:

 3385
3386
3387

The encoding of this segment is a b -bit integer (padded to the left with zero bits as necessary), where b is the value specified in the "Coding Segment Bit Count" row of the encoding table, whose value is the value of the input character string considered as a decimal integer.

3388

14.3.2 "String" Encoding method

 3389
3390

The String encoding method is used for a segment that appears as an alphanumeric string in the URI, and as an ISO/IEC 646 [ISO646] (ASCII) encoded bit string in the binary encoding.

3391

Input:

 3392
3393

The input to the encoding method is the URI portion indicated in the "URI portion" row of the encoding table, a character string with no dot (".") characters.

3394

Validity Test:

3395

The input character string must satisfy the following:

3396

- It must match the grammar for `GS3A3Component` as specified in Section 5.

3397

- For each portion of the string that matches the `Escape` production of the grammar specified in Section 5 (that is, a 3-character sequence consisting of a % character followed by two hexadecimal digits), the two hexadecimal characters following the % character must map to one of the 82 allowed characters specified in [Table I.3.1-1](#).

 3398
3399
3400

3401

- The number of characters must be less than or equal to $b/7$, where b is the value specified in the "Coding Segment Bit Count" row of the coding table.

3402

3403

If any of the above tests fails, the encoding of the URI fails.

3404

Output:

 3405
3406
3407
3408
3409
3410
3411
3412
3413

Consider the input to be a string of zero or more characters $s_1s_2...s_N$, where each character s_i is either a single character or a 3-character sequence matching the `Escape` production of the grammar (that is, a 3-character sequence consisting of a % character followed by two hexadecimal digits). Translate each character to a 7-bit string. For a single character, the corresponding 7-bit string is specified in [Table I.3.1-1](#). For an `Escape` sequence, the 7-bit string is the value of the two hexadecimal characters considered as a 7-bit integer. Concatenating those 7-bit strings in the order corresponding to the input, then pad to the right with zero bits as necessary to total b bits, where b is the value specified in the "Coding Segment Bit Count" row of the coding table. (The number of padding bits will be $b - 7N$.) The resulting b -bit string is the output.

3414

14.3.3 "Partition Table" Encoding method

 3415
3416

The Partition Table encoding method is used for a segment that appears in the URI as a pair of variable-length numeric fields separated by a dot (".") character, and in the binary encoding as a 3-

3417 bit "partition" field followed by two variable length binary integers. The number of characters in the
 3418 two URI fields always totals to a constant number of characters, and the number of bits in the
 3419 binary encoding likewise totals to a constant number of bits.

3420 The Partition Table encoding method makes use of a "partition table." The specific partition table to
 3421 use is specified in the coding table for a given EPC scheme.

3422 **Input:**

3423 The input to the encoding method is the URI portion indicated in the "URI portion" row of the
 3424 encoding table. This consists of two strings of digits separated by a dot (".") character. For the
 3425 purpose of this encoding procedure, the digit strings to the left and right of the dot are denoted *C*
 3426 and *D*, respectively.

3427 **Validity Test:**

3428 The input must satisfy the following:

- 3429 ■ *C* must match the grammar for `PaddedNumericComponent` as specified in Section 5.
- 3430 ■ *D* must match the grammar for `PaddedNumericComponentOrEmpty` as specified in Section 5.
- 3431 ■ The number of digits in *C* must match one of the values specified in the "GS1 Company Prefix
 3432 Digits (L)" column of the partition table. The corresponding row is called the "matching partition
 3433 table row" in the remainder of the encoding procedure.
- 3434 ■ The number of digits in *D* must match the corresponding value specified in the other field digits
 3435 column of the matching partition table row. Note that if the other field digits column specifies
 3436 zero, then *D* must be the empty string, implying the overall input segment ends with a "dot"
 3437 character.

3438 **Output:**

3439 Construct the output bit string by concatenating the following three components:

- 3440 ■ The value *P* specified in the "partition value" column of the matching partition table row, as a 3-
 3441 bit binary integer.
- 3442 ■ The value of *C* considered as a decimal integer, converted to an *M*-bit binary integer, where *M* is
 3443 the number of bits specified in the "GS1 Company Prefix bits" column of the matching partition
 3444 table row.
- 3445 ■ The value of *D* considered as a decimal integer, converted to an *N*-bit binary integer, where *N* is
 3446 the number of bits specified in the other field bits column of the matching partition table row. If
 3447 *D* is the empty string, the value of the *N*-bit integer is zero.

3448 The resulting bit string is $(3 + M + N)$ bits in length, which always equals the "Coding Segment Bit
 3449 Count" for this segment as indicated in the coding table.

3450 **14.3.4 "Unpadded Partition Table" Encoding method**

3451 The Unpadded Partition Table encoding method is used for a segment that appears in the URI as a
 3452 pair of variable-length numeric fields separated by a dot (".") character, and in the binary encoding
 3453 as a 3-bit "partition" field followed by two variable length binary integers. The number of characters
 3454 in the two URI fields is always less than or equal to a known limit, and the number of bits in the
 3455 binary encoding is always a constant number of bits.

3456 The Unpadded Partition Table encoding method makes use of a "partition table." The specific
 3457 partition table to use is specified in the coding table for a given EPC scheme.

3458 **Input:**

3459 The input to the encoding method is the URI portion indicated in the "URI portion" row of the
 3460 encoding table. This consists of two strings of digits separated by a dot (".") character. For the
 3461 purpose of this encoding procedure, the digit strings to the left and right of the dot are denoted *C*
 3462 and *D*, respectively.

3463
3464
3465
3466
3467
3468
3469
3470
3471

Validity Test:

The input must satisfy the following:

- *C* must match the grammar for `PaddedNumericComponent` as specified in Section 5.
- *D* must match the grammar for `NumericComponent` as specified in Section 5.
- The number of digits in *C* must match one of the values specified in the "GS1 Company Prefix Digits (L)" column of the partition table. The corresponding row is called the "matching partition table row" in the remainder of the encoding procedure.
- The value of *D*, considered as a decimal integer, must be less than 2^N , where *N* is the number of bits specified in the other field bits column of the matching partition table row.

3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483

Output:

Construct the output bit string by concatenating the following three components:

- The value *P* specified in the "partition value" column of the matching partition table row, as a 3-bit binary integer.
- The value of *C* considered as a decimal integer, converted to an *M*-bit binary integer, where *M* is the number of bits specified in the "GS1 Company Prefix bits" column of the matching partition table row.
- The value of *D* considered as a decimal integer, converted to an *N*-bit binary integer, where *N* is the number of bits specified in the other field bits column of the matching partition table row. If *D* is the empty string, the value of the *N*-bit integer is zero.

The resulting bit string is $(3 + M + N)$ bits in length, which always equals the "Coding Segment Bit Count" for this segment as indicated in the coding table.

3484

14.3.5 "String Partition Table" Encoding method

3485
3486
3487
3488
3489
3490
3491

The String Partition Table encoding method is used for a segment that appears in the URI as a variable-length numeric field and a variable-length string field separated by a dot (".") character, and in the binary encoding as a 3-bit "partition" field followed by a variable length binary integer and a variable length binary-encoded character string. The number of characters in the two URI fields is always less than or equal to a known limit (counting a 3-character escape sequence as a single character), and the number of bits in the binary encoding is padded if necessary to a constant number of bits.

3492
3493

The Partition Table encoding method makes use of a "partition table." The specific partition table to use is specified in the coding table for a given EPC scheme.

3494
3495
3496
3497
3498

Input:

The input to the encoding method is the URI portion indicated in the "URI portion" row of the encoding table. This consists of two strings separated by a dot (".") character. For the purpose of this encoding procedure, the strings to the left and right of the dot are denoted *C* and *D*, respectively.

3499
3500
3501
3502
3503
3504
3505
3506
3507
3508

Validity Test:

The input must satisfy the following:

- *C* must match the grammar for `PaddedNumericComponent` as specified in Section 5.
- *D* must match the grammar for `GS3A3Component` as specified in Section 5.
- The number of digits in *C* must match one of the values specified in the "GS1 Company Prefix Digits (L)" column of the partition table. The corresponding row is called the "matching partition table row" in the remainder of the encoding procedure.
- The number of characters in *D* must be less than or equal to the corresponding value specified in the other field maximum characters column of the matching partition table row. For the purposes of this rule, an escape triplet (`%nn`) is counted as one character.

- 3509
- 3510
- 3511
- 3512
- For each portion of D that matches the `Escape` production of the grammar specified in Section 5 (that is, a 3-character sequence consisting of a % character followed by two hexadecimal digits), the two hexadecimal characters following the % character must map to one of the 82 allowed characters specified in [Table I.3.1-1](#).

3513 **Output:**

3514 Construct the output bit string by concatenating the following three components:

- 3515
- 3516
- 3517
- 3518
- 3519
- The value P specified in the "partition value" column of the matching partition table row, as a 3-bit binary integer.
 - The value of C considered as a decimal integer, converted to an M -bit binary integer, where M is the number of bits specified in the "GS1 Company Prefix bits" column of the matching partition table row.
 - The value of D converted to an N -bit binary string, where N is the number of bits specified in the other field bits column of the matching partition table row. This N -bit binary string is constructed as follows. Consider D to be a string of zero or more characters $s_1s_2...s_N$, where each character s_i is either a single character or a 3-character sequence matching the `Escape` production of the grammar (that is, a 3-character sequence consisting of a % character followed by two hexadecimal digits). Translate each character to a 7-bit string. For a single character, the corresponding 7-bit string is specified in [Table I.3.1-1](#). For an `Escape` sequence, the 7-bit string is the value of the two hexadecimal characters considered as a 7-bit integer. Concatenate those 7-bit strings in the order corresponding to the input, then pad with zero bits as necessary to total N bits.

3520 The resulting bit string is $(3 + M + N)$ bits in length, which always equals the "Coding Segment Bit
3531 Count" for this segment as indicated in the coding table.

3532 **14.3.6 "Numeric String" Encoding method**

3533 The Numeric String encoding method is used for a segment that appears as a numeric string in the
3534 URI, possibly including leading zeros. The leading zeros are preserved in the binary encoding by
3535 prepending a "1" digit to the numeric string before encoding.

3536 **Input:**

3537 The input to the encoding method is the URI portion indicated in the "URI portion" row of the
3538 encoding table, a character string with no dot (".") characters.

3539 **Validity Test:**

3540 The input character string must satisfy the following:

- 3541
- 3542
- 3543
- 3544
- 3545
- It must match the grammar for `PaddedNumericComponent` as specified in Section 5.
 - The number of digits in the string, D , must be such that $2 \times 10^D < 2^b$, where b is the value specified in the "Coding Segment Bit Count" row of the encoding table. (For the GDTI-113 scheme, $b = 58$ and therefore the number of digits D must be less than or equal to 17. GDTI-113 and SGCN-96 are the only schemes that uses this encoding method.)

3546 If any of the above tests fails, the encoding of the URI fails.

3547 **Output:**

3548 Construct the output bit string as follows:

- 3549
- 3550
- 3551
- 3552
- Prepend the character "1" to the left of the input character string.
 - Convert the resulting string to a b -bit integer (padded to the left with zero bits as necessary), where b is the value specified in the "bit count" row of the encoding table, whose value is the value of the input character string considered as a decimal integer.

14.3.7 "6-bit CAGE/DODAAC" Encoding method

The 6-Bit CAGE/DoDAAC encoding method is used for a segment that appears as a 5-character CAGE code or 6-character DoDAAC in the URI, and as a 36-bit encoded bit string in the binary encoding.

Input:

The input to the encoding method is the URI portion indicated in the "URI portion" row of the encoding table, a 5- or 6-character string with no dot (".") characters.

Validity Test:

The input character string must satisfy the following:

- It must match the grammar for `CAGECodeOrDODAAC` as specified in Section [6.3.17](#).

If the above test fails, the encoding of the URI fails.

Output:

Consider the input to be a string of five or six characters $d_1d_2\dots d_N$, where each character d_i is a single character. Translate each character to a 6-bit string using [Table I.3.1-1 \(G\)](#). Concatenate those 6-bit strings in the order corresponding to the input. If the input was five characters, prepend the 6-bit value 100000 to the left of the result. The resulting 36-bit string is the output.

14.3.8 "6-Bit Variable String" Encoding method

The 6-Bit Variable String encoding method is used for a segment that appears in the URI as a string field, and in the binary encoding as variable length null-terminated binary-encoded character string.

Input:

The input to the encoding method is the URI portion indicated in the "URI portion" row of the encoding table.

Validity Test:

The input must satisfy the following:

- The input must match the grammar for the corresponding portion of the URI as specified in the appropriate subsection of Section [6.3](#).
- The number of characters in the input must be greater than or equal to the minimum number of characters and less than or equal to the maximum number of characters specified in the footnote to the coding table for this coding table column. For the purposes of this rule, an escape triplet (`%nn`) is counted as one character.
- For each portion of the input that matches the `Escape` production of the grammar specified in Section [5](#) (that is, a 3-character sequence consisting of a `%` character followed by two hexadecimal digits), the two hexadecimal characters following the `%` character must map to one of the characters specified in [Table I.3.1-1 \(G\)](#), and the character so mapped must satisfy any other constraints specified in the coding table for this coding segment.
- For each portion of the input that is a single character (as opposed to a 3-character escape sequence), that character must satisfy any other constraints specified in the coding table for this coding segment.

Output:

Consider the input to be a string of zero or more characters $s_1s_2\dots s_N$, where each character s_i is either a single character or a 3-character sequence matching the `Escape` production of the grammar (that is, a 3-character sequence consisting of a `%` character followed by two hexadecimal digits). Translate each character to a 6-bit string. For a single character, the corresponding 6-bit string is specified in [Table I.3.1-1 \(G\)](#). For an `Escape` sequence, the corresponding 6-bit string is specified in [Table I.3.1-1 \(G\)](#) by finding the escape sequence in the "URI Form" column.

3598 Concatenate those 6-bit strings in the order corresponding to the input, then append six zero bits
 3599 (000000).

3600 The resulting bit string is of variable length, but is always at least 6 bits and is always a multiple of
 3601 6 bits.

3602 **14.3.9 "6-Bit Variable String Partition Table" Encoding method**

3603 The 6-Bit Variable String Partition Table encoding method is used for a segment that appears in the
 3604 URI as a variable-length numeric field and a variable-length string field separated by a dot (".")
 3605 character, and in the binary encoding as a 3-bit "partition" field followed by a variable length binary
 3606 integer and a null-terminated binary-encoded character string. The number of characters in the two
 3607 URI fields is always less than or equal to a known limit (counting a 3-character escape sequence as
 3608 a single character), and the number of bits in the binary encoding is also less than or equal to a
 3609 known limit.

3610 The 6-Bit Variable String Partition Table encoding method makes use of a "partition table." The
 3611 specific partition table to use is specified in the coding table for a given EPC scheme.

3612 **Input:**

3613 The input to the encoding method is the URI portion indicated in the "URI portion" row of the
 3614 encoding table. This consists of two strings separated by a dot (".") character. For the purpose of
 3615 this encoding procedure, the strings to the left and right of the dot are denoted *C* and *D*,
 3616 respectively.

3617 **Validity Test:**

3618 The input must satisfy the following:

- 3619 ■ The input must match the grammar for the corresponding portion of the URI as specified in the
 3620 appropriate subsection of Section 6.3.
- 3621 ■ The number of digits in *C* must match one of the values specified in the "GS1 Company Prefix
 3622 Digits (L)" column of the partition table. The corresponding row is called the "matching partition
 3623 table row" in the remainder of the encoding procedure.
- 3624 ■ The number of characters in *D* must be less than or equal to the corresponding value specified
 3625 in the other field maximum characters column of the matching partition table row. For the
 3626 purposes of this rule, an escape triplet (%nn) is counted as one character.
- 3627 ■ For each portion of *D* that matches the `Escape` production of the grammar specified in
 3628 Section 5 (that is, a 3-character sequence consisting of a % character followed by two
 3629 hexadecimal digits), the two hexadecimal characters following the % character must map to one
 3630 of the 39 allowed characters specified in [Table I.3.1-1 \(G\)](#).

3631 **Output:**

3632 Construct the output bit string by concatenating the following three components:

- 3633 ■ The value *P* specified in the "partition value" column of the matching partition table row, as a 3-
 3634 bit binary integer.
- 3635 ■ The value of *C* considered as a decimal integer, converted to an *M*-bit binary integer, where *M* is
 3636 the number of bits specified in the "GS1 Company Prefix bits" column of the matching partition
 3637 table row.
- 3638 ■ The value of *D* converted to an *N*-bit binary string, where *N* is less than or equal to the number
 3639 of bits specified in the other field maximum bits column of the matching partition table row. This
 3640 binary string is constructed as follows. Consider *D* to be a string of one or more characters
 3641 $s_1s_2\dots s_N$, where each character s_i is either a single character or a 3-character sequence
 3642 matching the `Escape` production of the grammar (that is, a 3-character sequence consisting of
 3643 a % character followed by two hexadecimal digits). Translate each character to a 6-bit string. For
 3644 a single character, the corresponding 6-bit string is specified in [Table I.3.1-1 \(G\)](#). For an
 3645 `Escape` sequence, the 6-bit string is the value of the two hexadecimal characters considered as

3646 a 6-bit integer. Concatenate those 6-bit strings in the order corresponding to the input, then
 3647 add six zero bits.

3648 The resulting bit string is $(3 + M + N)$ bits in length, which is always less than or equal to the
 3649 maximum "Coding Segment Bit Count" for this segment as indicated in the coding table.

3650 **14.3.10 "Fixed Width Integer" Encoding Method**

3651 The Fixed Width Integer encoding method is used for a segment that appears as a zero-padded
 3652 decimal integer in the URI, and as a binary integer in the binary encoding.

3653 **Input:**

3654 The input to the encoding method is the URI portion indicated in the "URI portion" row of the
 3655 encoding table, an all-numeric character string with no dot (".") characters.

3656 **Validity Test:**

3657 The input character string must satisfy the following:

- 3658 ■ It must match the grammar for `PaddedNumericComponent` as specified in Section 5.
- 3659 ■ The value of the string when considered as a non-negative decimal integer must be less than
 3660 $((10^D) - 1)$ where $D = \text{int}(b \cdot \log(2) / \log(10))$, where b is the value specified in the "Coding
 3661 Segment Bit Count" row of the encoding table.

3662 If any of the above tests fails, the encoding of the URI fails.

3663 **Output:**

3664 The encoding of this segment is a b -bit integer (padded to the left with zero bits as necessary),
 3665 where b is the value specified in the "Coding Segment Bit Count" row of the encoding table, whose
 3666 value is the value of the input character string considered as a decimal integer.

3667 **14.4 Decoding procedure**

3668 This procedure decodes a bit string as found beginning at bit 20_h in the EPC memory bank of a Gen
 3669 2 Tag into an EPC Tag URI (This section only applies for EPC schemes for which an EPC Tag URI is
 3670 defined; no EPC Tag URI format is defined for new EPC schemes introduced in TDS 2.0 – for those
 3671 schemes, the result of decoding is the corresponding GS1 element string or equivalently, the set of
 3672 GS1 Application Identifiers and their values. For all new EPC schemes introduced in TDS 2.0, please
 3673 refer to section 14.5 instead). This procedure only decodes the EPC and filter value (if applicable).
 3674 Section 15.2.2 gives the complete procedure for decoding the entire contents of the EPC memory
 3675 bank, including control information that is stored outside of the encoded EPC. The procedure in
 3676 Section 15.2.2 should be used by most applications. (The procedure in Section 15.2.2 uses the
 3677 procedure below as a subroutine.)

3678 **Given:**

- 3679 ■ A bit string consisting of N bits $b_{N-1} b_{N-2} \dots b_0$

3680 **Yields:**

- 3681 ■ An EPC Tag URI beginning with `urn:epc:tag:`, which does not contain control information
 3682 fields (other than the filter value if the EPC scheme includes a filter value); OR
- 3683 ■ An exception indicating that the bit string cannot be decoded into an EPC Tag URI.

3684 **Procedure:**

- 3685 1. Extract the most significant eight bits, the EPC header: $b_{N-1} b_{N-2} \dots b_{N-8}$. Referring to [Table 14-1](#) in
 3686 Section 14.2, use the header to identify the coding table for this binary encoding and the
 3687 encoding bit length B . If no coding table exists for this header, stop: this binary encoding cannot
 3688 be decoded.

- 3689
3690
2. Confirm that the total number of bits N is greater than or equal to the total number of bits B specified for this header in [Table 14-1](#). If not, stop: this binary encoding cannot be decoded.
- 3691
3692
3693
3694
3695
3. If necessary, truncate the least significant bits of the input to match the number of bits specified in [Table 14-1](#). That is, if [Table 14-1](#) specifies B bits, retain bits $b_{N-1} b_{N-2} \dots b_{N-B}$. For the remainder of this procedure, consider the remaining bits to be numbered $b_{B-1} b_{B-2} \dots b_0$. (The purpose of this step is to remove any trailing zero padding bits that may have been read due to word-oriented data transfer.)
- 3696
3697
3698
4. For a variable-length coding scheme, there is no B specified in [Table 14-1](#) and so this step must be omitted. There may be trailing zero padding bits remaining after all segments are decoded in Step 4, below; if so, ignore them.
- 3699
3700
3701
3702
3703
3704
3705
5. Separate the bits of the binary encoding into segments according to the "bit position" row of the coding table. For each segment, decode the bits to obtain a character string that will be used as a portion of the final URI. The method for decoding each column depends on the "coding method" row of the table. If the "coding method" row specifies a specific bit string, the corresponding bits of the input must match those bits exactly; if not, stop: this binary encoding cannot be decoded. Otherwise, consult the following sections that specify the decoding methods. If the decoding of any segment fails, stop: this binary encoding cannot be decoded.
- 3706
3707
3708
3709
3710
6. For a variable-length coding segment, the coding method is applied beginning with the bit following the bits consumed by the previous coding column. That is, if the previous coding column (the column to the left of this one) consumed bits up to and including bit b_i , then the most significant bit for decoding this segment is bit b_{i-1} . The coding method will determine where the ending bit for this segment is.
- 3711
3712
3713
7. Concatenate the following strings to obtain the final URI: the string `urn:epc:tag:`, the scheme name as specified in the coding table, a colon (":") character, and the strings obtained in Step 4, inserting a dot (".") character between adjacent strings.

3714 The following sections specify the procedures to be used in Step 4.

3715 **14.4.1 "Integer" Decoding method**

3716 The Integer decoding method is used for a segment that appears as a decimal integer in the URI,
3717 and as a binary integer in the binary encoding.

3718 **Input:**

3719 The input to the decoding method is the bit string identified in the "bit position" row of the coding
3720 table.

3721 **Validity Test:**

3722 There are no validity tests for this decoding method.

3723 **Output:**

3724 The decoding of this segment is a decimal numeral whose value is the value of the input considered
3725 as an unsigned binary integer. The output shall not begin with a zero character if it is two or more
3726 digits in length.

3727 **14.4.2 "String" Decoding method**

3728 The String decoding method is used for a segment that appears as an alphanumeric string in the
3729 URI, and as an ISO/IEC 646 [ISO646] (ASCII) encoded bit string in the binary encoding.

3730 **Input:**

3731 The input to the decoding method is the bit string identified in the "bit position" row of the coding
3732 table. This length of this bit string is always a multiple of seven.

3733

Validity Test:

3734

The input bit string must satisfy the following:

3735

- Each 7-bit segment must have a value corresponding to a character specified in [Table I.3.1-1](#), or be all zeros.

3736

3737

- All 7-bit segments following an all-zero segment must also be all zeros.

3738

- The first 7-bit segment must not be all zeros. (In other words, the string must contain at least one character.)

3739

3740

If any of the above tests fails, the decoding of the segment fails.

3741

Output:

3742

Translate each 7-bit segment, up to but not including the first all-zero segment (if any), into a single character or 3-character escape triplet by looking up the 7-bit segment in [Table I.3.1-1](#), and using the value found in the "URI Form" column. Concatenate the characters and/or 3-character triplets in the order corresponding to the input bit string. The resulting character string is the output. This character string matches the `GS3A3` production of the grammar in Section [5](#).

3743

3744

3745

3746

3747

14.4.3 "Partition Table" Decoding method

3748

The Partition Table decoding method is used for a segment that appears in the URI as a pair of variable-length numeric fields separated by a dot (".") character, and in the binary encoding as a 3-bit "partition" field followed by two variable length binary integers. The number of characters in the two URI fields always totals to a constant number of characters, and the number of bits in the binary encoding likewise totals to a constant number of bits.

3749

3750

3751

3752

3753

The Partition Table decoding method makes use of a "partition table." The specific partition table to use is specified in the coding table for a given EPC scheme.

3754

3755

Input:

3756

The input to the decoding method is the bit string identified in the "bit position" row of the coding table. Logically, this bit string is divided into three substrings, consisting of a 3-bit "partition" value, followed by two substrings of variable length.

3757

3758

3759

Validity Test:

3760

The input must satisfy the following:

3761

- The three most significant bits of the input bit string, considered as a binary integer, must match one of the values specified in the "partition value" column of the partition table. The corresponding row is called the "matching partition table row" in the remainder of the decoding procedure.

3762

3763

3764

3765

- Extract the M next most significant bits of the input bit string following the three partition bits, where M is the value specified in the "Company Prefix Bits" column of the matching partition table row. Consider these M bits to be an unsigned binary integer, C . The value of C must be less than 10^L , where L is the value specified in the "GS1 Company Prefix Digits (L)" column of the matching partition table row.

3766

3767

3768

3769

3770

- There are N bits remaining in the input bit string, where N is the value specified in the other field bits column of the matching partition table row. Consider these N bits to be an unsigned binary integer, D . The value of D must be less than 10^K , where K is the value specified in the other field digits (K) column of the matching partition table row. Note that if $K = 0$, then the value of D must be zero.

3771

3772

3773

3774

3775

Output:

3776

Construct the output character string by concatenating the following three components:

3777

- The value C converted to a decimal numeral, padding on the left with zero ("0") characters to make L digits in total.

3778

3779

- A dot (".") character.

- 3780
- 3781
- 3782
- 3783
- The value D converted to a decimal numeral, padding on the left with zero ("0") characters to make K digits in total. If $K = 0$, append no characters to the dot above (in this case, the final URI string will have two adjacent dot characters when this segment is combined with the following segment).

3784 **14.4.4 "Unpadded Partition Table" Decoding method**

3785 The Unpadded Partition Table decoding method is used for a segment that appears in the URI as a
 3786 pair of variable-length numeric fields separated by a dot (".") character, and in the binary encoding
 3787 as a 3-bit "partition" field followed by two variable length binary integers. The number of characters
 3788 in the two URI fields is always less than or equal to a known limit, and the number of bits in the
 3789 binary encoding is always a constant number of bits.

3790 The Unpadded Partition Table decoding method makes use of a "partition table." The specific
 3791 partition table to use is specified in the coding table for a given EPC scheme.

3792 **Input:**

3793 The input to the decoding method is the bit string identified in the "bit position" row of the coding
 3794 table. Logically, this bit string is divided into three substrings, consisting of a 3-bit "partition" value,
 3795 followed by two substrings of variable length.

3796 **Validity Test:**

3797 The input must satisfy the following:

- 3798
- 3799
- 3800
- The three most significant bits of the input bit string, considered as a binary integer, must match one of the values specified in the "partition value" column of the partition table. The corresponding row is called the "matching partition table row" in the remainder of the decoding procedure.
 - Extract the M next most significant bits of the input bit string following the three partition bits, where M is the value specified in the "Company Prefix Bits" column of the matching partition table row. Consider these M bits to be an unsigned binary integer, C . The value of C must be less than 10^L , where L is the value specified in the "GS1 Company Prefix Digits (L)" column of the matching partition table row.
 - There are N bits remaining in the input bit string, where N is the value specified in the other field bits column of the matching partition table row. Consider these N bits to be an unsigned binary integer, D .

3809 **Output:**

3810 Construct the output character string by concatenating the following three components:

- 3811
- 3812
- 3813
- The value C converted to a decimal numeral, padding on the left with zero ("0") characters to make L digits in total.
 - A dot (".") character.
 - The value D converted to a decimal numeral, with no leading zeros (except that if $D = 0$ it is converted to a single zero digit).
- 3814
- 3815

3816 **14.4.5 "String Partition Table" Decoding method**

3817 The String Partition Table decoding method is used for a segment that appears in the URI as a
 3818 variable-length numeric field and a variable-length string field separated by a dot (".") character,
 3819 and in the binary encoding as a 3-bit "partition" field followed by a variable length binary integer
 3820 and a variable length binary-encoded character string. The number of characters in the two URI
 3821 fields is always less than or equal to a known limit (counting a 3-character escape sequence as a
 3822 single character), and the number of bits in the binary encoding is padded if necessary to a constant
 3823 number of bits.

3824 The Partition Table decoding method makes use of a "partition table." The specific partition table to
 3825 use is specified in the coding table for a given EPC scheme.

3826

Input:

3827

The input to the decoding method is the bit string identified in the "bit position" row of the coding table. Logically, this bit string is divided into three substrings, consisting of a 3-bit "partition" value, followed by two substrings of variable length.

3828

3829

3830

Validity Test:

3831

The input must satisfy the following:

3832

- The three most significant bits of the input bit string, considered as a binary integer, must match one of the values specified in the "partition value" column of the partition table. The corresponding row is called the "matching partition table row" in the remainder of the decoding procedure.

3833

3834

3835

3836

- Extract the M next most significant bits of the input bit string following the three partition bits, where M is the value specified in the "Company Prefix Bits" column of the matching partition table row. Consider these M bits to be an unsigned binary integer, C . The value of C must be less than 10^L , where L is the value specified in the "GS1 Company Prefix Digits (L)" column of the matching partition table row.

3837

3838

3839

3840

3841

- There are N bits remaining in the input bit string, where N is the value specified in the other field bits column of the matching partition table row. These bits must consist of one or more non-zero 7-bit segments followed by zero or more all-zero bits.

3842

3843

3844

- The number of non-zero 7-bit segments that precede the all-zero bits (if any) must be less or equal to than K , where K is the value specified in the "Maximum Characters" column of the matching partition table row.

3845

3846

3847

- Each of the non-zero 7-bit segments must have a value corresponding to a character specified in [Table I.3.1-1](#).

3848

3849

Output:

3850

Construct the output character string by concatenating the following three components:

3851

- The value C converted to a decimal numeral, padding on the left with zero ("0") characters to make L digits in total.

3852

3853

- A dot (".") character.

3854

- A character string determined as follows. Translate each non-zero 7-bit segment as determined by the validity test into a single character or 3-character escape triplet by looking up the 7-bit segment in [Table I.3.1-1](#), and using the value found in the "URI Form" column. Concatenate the characters and/or 3-character triplet in the order corresponding to the input bit string.

3855

3856

3857

3858 14.4.6 "Numeric String" Decoding method

3859

The Numeric String decoding method is used for a segment that appears as a numeric string in the URI, possibly including leading zeros. The leading zeros are preserved in the binary encoding by prepending a "1" digit to the numeric string before encoding.

3860

3861

3862

Input:

3863

The input to the decoding method is the bit string identified in the "bit position" row of the coding table.

3864

3865

Validity Test:

3866

The input must be such that the decoding procedure below does not fail.

3867

Output:

3868

Construct the output string as follows.

3869

- Convert the input bit string to a decimal numeral without leading zeros whose value is the value of the input considered as an unsigned binary integer.

3870

- 3871
- 3872
- If the numeral from the previous step does not begin with a "1" character, stop: the input is invalid.
- 3873
- 3874
- If the numeral from the previous step consists only of one character, stop: the input is invalid (because this would correspond to an empty numeric string).
- 3875
- Delete the leading "1" character from the numeral.
- 3876
- The resulting string is the output.

3877 **14.4.7 "6-Bit CAGE/DoDAAC" Decoding method**

3878 The 6-Bit CAGE/DoDAAC decoding method is used for a segment that appears as a 5-character
3879 CAGE code or 6-character DoDAAC code in the URI, and as a 36-bit encoded bit string in the binary
3880 encoding.

3881 **Input:**

3882 The input to the decoding method is the bit string identified in the "bit position" row of the coding
3883 table. This length of this bit string is always 36 bits.

3884 **Validity Test:**

3885 The input bit string must satisfy the following:

- 3886
- 3887
- 3888
- When the bit string is considered as consisting of six 6-bit segments, each 6-bit segment must have a value corresponding to a character specified in [Table I.3.1-1 \(G\)](#) except that the first 6-bit segment may also be the value 100000.
 - The first 6-bit segment must be the value 100000, or correspond to a digit character, or an uppercase alphabetic character excluding the letters I and O.
 - The remaining five 6-bit segments must correspond to a digit character or an uppercase alphabetic character excluding the letters I and O.
- 3889
- 3890
- 3891
- 3892

3893 If any of the above tests fails, the decoding of the segment fails.

3894 **Output:**

3895 Disregard the first 6-bit segment if it is equal to 100000. Translate each of the remaining five or six
3896 6-bit segments into a single character by looking up the 6-bit segment in [Table I.3.1-1 \(G\)](#) and
3897 using the value found in the "URI Form" column. Concatenate the characters in the order
3898 corresponding to the input bit string. The resulting character string is the output. This character
3899 string matches the CAGECodeOrDODAAC production of the grammar in Section [6.3.17](#).

3900 **14.4.8 "6-Bit Variable String" Decoding method**

3901 The 6-Bit Variable String decoding method is used for a segment that appears in the URI as a
3902 variable-length string field, and in the binary encoding as a variable-length null-terminated binary-
3903 encoded character string.

3904 **Input:**

3905 The input to the decoding method is the bit string that begins in the next least significant bit
3906 position following the previous coding segment. Only a portion of this bit string is consumed by this
3907 decoding method, as described below.

3908 **Validity Test:**

3909 The input must be such that the decoding procedure below does not fail.

3910 **Output:**

3911 Construct the output string as follows.

- 3912
- 3913
- 3914
- Beginning with the most significant bit of the input, divide the input into adjacent 6-bit segments, until a terminating segment consisting of all zero bits (000000) is found. If the input is exhausted before an all-zero segment is found, stop: the input is invalid.
- 3915
- 3916
- 3917
- 3918
- The number of 6-bit segments preceding the terminating segment must be greater than or equal to the minimum number of characters and less than or equal to the maximum number of characters specified in the footnote to the coding table for this coding table column. If not, stop: the input is invalid.
- 3919
- 3920
- 3921
- For each 6-bit segment preceding the terminating segment, consult [Table I.3.1-1 \(G\)](#) to find the character corresponding to the value of the 6-bit segment. If there is no character in the table corresponding to the 6-bit segment, stop: the input is invalid.
- 3922
- If the input violates any other constraint indicated in the coding table, stop: the input is invalid.
- 3923
- 3924
- 3925
- 3926
- 3927
- Translate each 6-bit segment preceding the terminating segment into a single character or 3-character escape triplet by looking up the 6-bit segment in [Table I.3.1-1 \(G\)](#) and using the value found in the "URI Form" column. Concatenate the characters and/or 3-character triplets in the order corresponding to the input bit string. The resulting string is the output of the decoding procedure.
- 3928
- 3929
- If any columns remain in the coding table, the decoding procedure for the next column resumes with the next least significant bit after the terminating 000000 segment.

3930 **14.4.9 "6-Bit Variable String Partition Table" Decoding method**

3931 The 6-Bit Variable String Partition Table decoding method is used for a segment that appears in the

3932 URI as a variable-length numeric field and a variable-length string field separated by a dot (".")

3933 character, and in the binary encoding as a 3-bit "partition" field followed by a variable length binary

3934 integer and a null-terminated binary-encoded character string. The number of characters in the two

3935 URI fields is always less than or equal to a known limit (counting a 3-character escape sequence as

3936 a single character), and the number of bits in the binary encoding is also less than or equal to a

3937 known limit.

3938 The 6-Bit Variable String Partition Table decoding method makes use of a "partition table." The

3939 specific partition table to use is specified in the coding table for a given EPC scheme.

3940 **Input:**

3941 The input to the decoding method is the bit string identified in the "bit position" row of the coding

3942 table. Logically, this bit string is divided into three substrings, consisting of a 3-bit "partition" value,

3943 followed by two substrings of variable length.

3944 **Validity Test:**

3945 The input must satisfy the following:

- 3946
- 3947
- 3948
- 3949
- The three most significant bits of the input bit string, considered as a binary integer, must match one of the values specified in the "partition value" column of the partition table. The corresponding row is called the "matching partition table row" in the remainder of the decoding procedure.
- 3950
- 3951
- 3952
- 3953
- 3954
- Extract the M next most significant bits of the input bit string following the three partition bits, where M is the value specified in the "Company Prefix Bits" column of the matching partition table row. Consider these M bits to be an unsigned binary integer, C . The value of C must be less than 10^L , where L is the value specified in the "GS1 Company Prefix Digits (L)" column of the matching partition table row.
- 3955
- 3956
- 3957
- 3958
- There are up to N bits remaining in the input bit string, where N is the value specified in the other field maximum bits column of the matching partition table row. These bits must begin with one or more non-zero 6-bit segments followed by six all-zero bits. Any additional bits after the six all-zero bits belong to the next coding segment in the coding table.
- 3959
- 3960
- 3961
- The number of non-zero 6-bit segments that precede the all-zero bits must be less or equal to than K , where K is the value specified in the "Maximum Characters" column of the matching partition table row.

- 3962 ■ Each of the non-zero 6-bit segments must have a value corresponding to a character specified
 3963 in [Table I.3.1-1](#) (G)

3964 **Output:**

3965 Construct the output character string by concatenating the following three components:

- 3966 ■ The value *C* converted to a decimal numeral, padding on the left with zero ("0") characters to
 3967 make *L* digits in total.
- 3968 ■ A dot (".") character.
- 3969 ■ A character string determined as follows. Translate each non-zero 6-bit segment as determined
 3970 by the validity test into a single character or 3-character escape triplet by looking up the 6-bit
 3971 segment in [Table I.3.1-1](#) (G) and using the value found in the "URI Form" column. Concatenate
 3972 the characters and/or 3-character triplet in the order corresponding to the input bit string.

3973 **14.4.10 "Fixed Width Integer" Decoding method**

3974 The Integer decoding method is used for a segment that appears as a zero-padded decimal integer
 3975 in the URI, and as a binary integer in the binary encoding.

3976 **Input:**

3977 The input to the decoding method is the bit string identified in the "bit position" row of the coding
 3978 table.

3979 **Validity Test:**

3980 Given a sequence of bits of length *b*, calculate i_{max} as follows:

3981 $D = \text{int}(b \cdot \log(2) / \log(10))$

3982 $i_{max} = 10^D - 1$

3984 Interpret the sequence of bits of length *b* as a non-negative integer value, *i*

3985 If $i > i_{max}$ then decoding fails because the bits correspond to a value that cannot be expressed in *D*
 3986 digits.

3987 **Output:**

3988 The decoding of this segment is a decimal numeral whose value is the value of the input considered
 3989 as an unsigned binary integer. The output is padded to the left, so that the total number of digits *D*
 3990 is given by $D = \text{int}(b \cdot \log(2) / \log(10))$.

3991 **14.5 Encoding/Decoding methods introduced in TDS 2.0**

3992 TDS 2.0 introduces several new binary encoding/decoding methods that are used both within the
 3993 construction and parsing of the new EPC identifiers as well as for the expression of additional AIDC
 3994 data beyond the end of the EPC identifier, as summarised in the table below and detailed in the
 3995 following subsections, which explain the encoding and decoding methods for each:

3996 **Table 14-2 Summary of Encoding/Decoding methods introduced in TDS 2.0**

| Method name | Section | Used within binary encoding of new EPC identifiers | Used within binary encoding of '+AIDC data' |
|--|------------------------|---|--|
| "+AIDC Data Toggle Bit" | 14.5.1 | Yes – to indicate whether additional AIDC data follows after the EPC identifier | No |
| "Fixed-Bit-Length Integer" | 14.5.2 | Yes – for filter value | Yes – e.g. for (20) Internal Product Variant |
| "Prioritised Date" | 14.5.3 | Yes – within DSGTIN+ | No |

| Method name | Section | Used within binary encoding of new EPC identifiers | Used within binary encoding of '+AIDC data' |
|--|--------------------------|--|--|
| "Fixed-Length Numeric" | 14.5.4 | Yes for most primary GS1 identification keys (e.g. GTIN, SSCC etc.). Not used by GIAI or CPI | Yes – when expressing additional GS1 identification keys within +AIDC data (e.g. expressing a GRAI in conjunction with an SGTIN+ EPC) |
| "Delimited/Terminated Numeric" | 14.5.5 | Yes – used for GIAI or CPI | Yes – used for GIAI or CPI |
| "Variable-length alphanumeric" | 14.5.6 | Yes – e.g. for (21) Serial Number within SGTIN+, DSGTIN+, ITIP+ | Yes – e.g. for (10) Batch/Lot Number |
| "Variable-length integer" | 14.5.6.1 | Yes – if value uses only 0-9 (leading zero digits are preserved) | Yes – if value uses only 0-9 (leading zero digits are preserved) |
| "Variable-length upper case hexadecimal" | 14.5.6.2 | Yes – if value uses only characters 0123456789ABCDEF | Yes – if value uses only characters 0123456789ABCDEF |
| "Variable-length lower case hexadecimal" | 14.5.6.3 | Yes – if value uses only characters 0123456789abcdef | Yes – if value uses only characters 0123456789abcdef |
| "Variable-length 6-bit file-safe URI-safe base 64" | 14.5.6.4 | Yes – if value uses only characters 0-9 A-Z a-z hyphen or underscore | Yes – if value uses only characters 0-9 A-Z a-z hyphen or underscore |
| "Variable-length URN Code 40" | 14.5.6.5 | Yes – if value uses only 0-9 A-Z colon, dot or hyphen | Yes – if value uses only 0-9 A-Z colon, dot or hyphen |
| "Variable-length 7-bit ASCII" | 14.5.6.6 | Yes – if value contains characters within the 82-character GS1 invariant subset of [ISO646] OTHER than digits 0-9 or letters A-Z a-z or hyphen, or underscore. | Yes – if value contains characters within the 82-character GS1 invariant subset of [ISO646] OTHER than digits 0-9 or letters A-Z a-z or hyphen, or underscore. |
| "Single data bit" | 14.5.7 | No | Yes – e.g. for AI (4321), (4322), (4323) |
| "6-digit date YYMMDD" | 14.5.8 | No – but see Prioritised Date within DSGTIN+, section 14.5.3 | Yes – e.g. for AI (17) |
| "10-digit date+time YYMMDDhhmm" | 14.5.9 | No | Yes – e.g. for AI (4324), (4325), (7003) |
| "Variable-format date / date range" | 14.5.10 | No | Yes – e.g. for AI (7007) = Harvest date / Harvest date range |
| "Variable-precision date+time" | 14.5.11 | No | Yes – e.g. for AI (8008) = Production date+time |
| "Country code (ISO 3166-1 alpha-2)" | 14.5.12 | No | Yes –for AI (4307) and (4317) |
| "Variable-length integer without encoding indicator" | 14.5.13 | Yes – in CPI+ and SGCN+ | Yes – for (255),(30),(37), (3900)-(3909), (3910)-(3919), (3920)-(3929), (3930)-(3939), (423), (425), (7004), (8011) and (8019) |

3997 **14.5.1 "+AIDC Data Toggle Bit"**

3998 The Data Toggle Bit encoding method is used for a segment that appears as a single bit in the
 3999 binary encoding that indicates whether or not additional AIDC data is encoded after the EPC within
 4000 the EPC/UII memory bank. This is primarily useful for 'Select' filtering over the air interface.

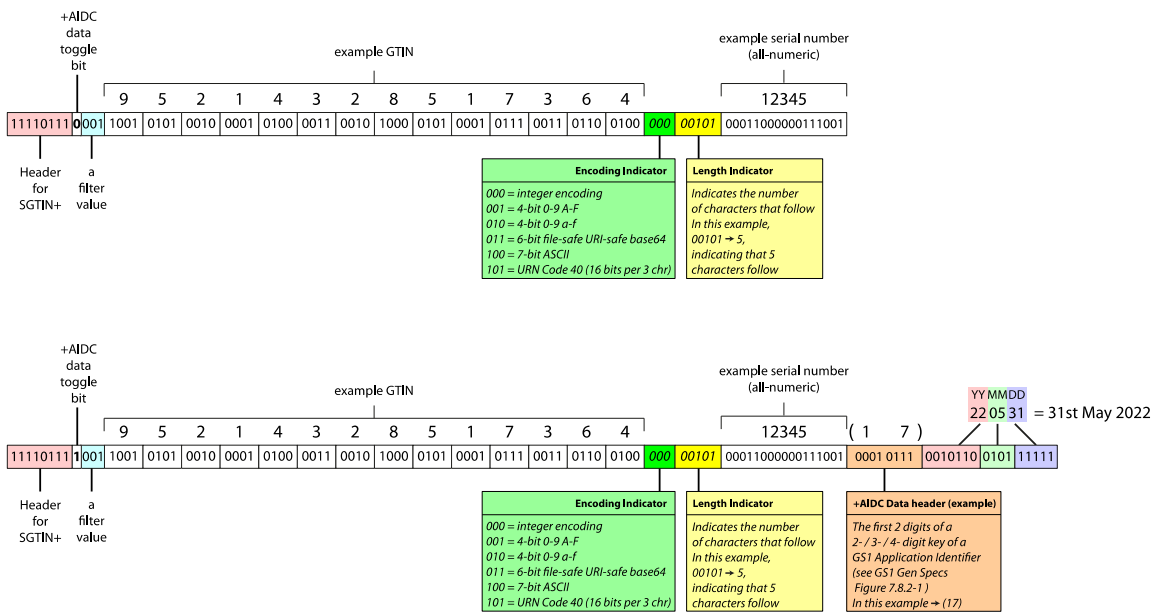
4001 The data toggle bit is a single bit that appears immediately after the 8-bit header of the new EPC
 4002 schemes and before the 3-bit filter value. Whoever / whatever encodes an EPC identifier into an
 4003 RFID tag has the responsibility to set the +AIDC data toggle bit correctly. Note that the +AIDC data
 4004 toggle bit is primarily used for selection of tag populations via the air interface and a non-essential
 4005 role in the decoding procedure if the guidance at the end of Section 15.3 is followed, to determine
 4006 whether or not any additional +AIDC data has been encoded after the end of the EPC identifier.

4007 If no additional AIDC data is encoded, the data toggle bit SHALL be set to 0.

4008 If additional AIDC is encoded, the data toggle bit SHALL be set to 1.

4009 The figure below shows an example of the use of the +AIDC data toggle bit.

4010 **Figure 14-1** Example of the use of the +AIDC data toggle bit



4011
 4012 **14.5.1.1 Encoding:**

4013 **Input:**

4014 The input to the encoding method is a Boolean value, in which:
 4015 true = additional AIDC data is to be encoded after the EPC within the EPC/UII memory bank
 4016 false = no additional AIDC data is to be encoded after the EPC within the EPC/UII memory bank

4017 **Validity Test:**

4018 The input must be either true or false, otherwise the encoding fails.

4019 **Output:**

4020 The encoding of this segment is a single bit, in which true is encoded as 1 while false is encoded as
 4021 0.

4022 **14.5.1.2 Decoding:**

4023 **Input:**

4024 The input to the decoding method is a single bit, which is interpreted as follows:
4025 1 = additional AIDC data is to be encoded after the EPC within the EPC/UII memory bank
4026 0 = no additional AIDC data is to be encoded after the EPC within the EPC/UII memory bank

4027 **Validity Test:**

4028 The output must be either true or false, otherwise the decoding fails.

4029 **Output:**

4030 The encoding of this segment is a Boolean value, in which 0 is interpreted as false (i.e. no additional
4031 AIDC data is to be encoded after the EPC within the EPC/UII memory bank), whereas 1 is
4032 interpreted as true (i.e. additional AIDC data is to be encoded after the EPC within the EPC/UII
4033 memory bank). If the +AIDC data toggle bit is set to 1, then refer to section [15.3](#) for further details
4034 about extraction of AIDC data that follows after new EPC schemes within the EPC/UII memory bank.

4035 **14.5.2 "Fixed-Bit-Length Integer"**

4036 The Fixed-Bit-Length-Integer encoding method is used for a segment that can represent numeric
4037 digits 1-9 using approximately 3.32 bits per digit, but using 3 bits in the case of a single digit filter
4038 value. When this method is used to encode the value of a GS1 Application Identifier, it is necessary
4039 to use Table F to determine the expected bit length, by locating the row for which the GS1
4040 Application Identifier key is shown in column a, then reading the expected bit length from column d.

4041 **14.5.2.1 Encoding**

4042 **Input:**

4043 The input to the encoding method is an integer. The expected number of bits must be determined
4044 from Table F (see introduction above) unless this method is being used to encode the filter value as
4045 3 bits.

4046 **Validity Test:**

4047 The input must be an integer, with no leading zeros, otherwise the encoding fails.

4048 **Output:**

4049 Convert the base 10 value to binary and if necessary left-pad with '0' bits to reach the expected bit
4050 length. This is the output of this encoding method.

4051 **14.5.2.2 Decoding**

4052 **Input:**

4053 The input to the decoding method is a fixed-length binary string of N bits, where N is determined
4054 from Table F (see introduction above) unless this method is being used to decode the filter value as
4055 3 bits.

4056 **Validity Test:**

4057 The output must be an integer.

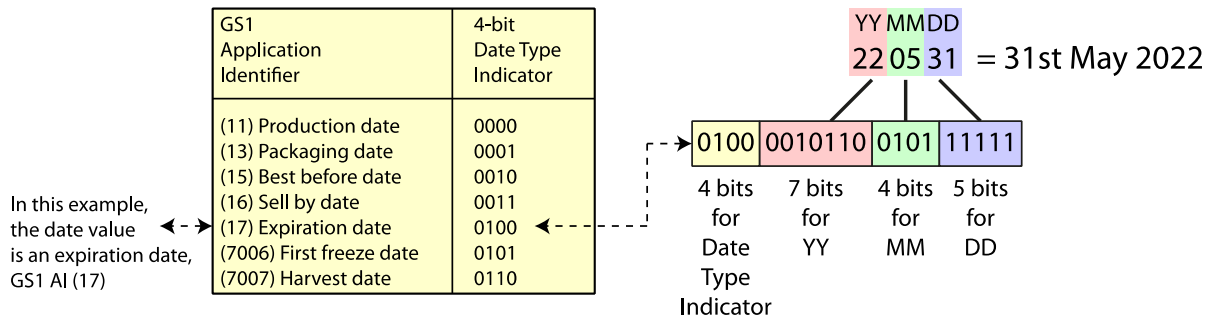
4058 **Output:**

4059 Read N bits and convert the value to an unsigned base 10 integer. Refer to Table F to determine
4060 the expected length in digits, shown in column c for the row that includes the GS1 Application
4061 Identifier key in column a. Convert the base 10 integer value to a numeric string and if
4062 necessary, left-pad with digits of '0' to reach the expected number of digits, as shown in column c of
4063 Table F. The result is the output of this decoding method.

4064 **14.5.3 "Prioritised Date"**

4065 The Prioritised Date encoding method is used within the DSGTIN+ scheme for a segment that
 4066 represents a date value in a well-defined position within the binary string (irrespective of the length
 4067 or character set used for the serial number), to support air interface filtering on a date of interest.
 4068 This is particularly useful to enable efficient scanning of perishable items with limited remaining
 4069 shelf life or to ensure that all expired / expiring products have been removed from sale. The
 4070 prioritised date format only supports 6-digit date values (YYMMDD) and includes a four-bit date type
 4071 indicator to express the meaning of the value – whether it corresponds to (11) production date, (17)
 4072 expiration date, (7007) harvest date, (16) sell-by date etc, as illustrated in the figure below.

4073 **Figure 14-2** Prioritised date format support for 6-digit date values



4074
 4075 Within the binary encoding of the DSGTIN+ scheme, the 4-bit date type indicator appears
 4076 immediately after the filter bits, i.e. 12 bits after the start of the EPC, starting at 2Ch.

4077 Its 4-bit string value must be one of the values shown in the table below. All other values are
 4078 reserved for future use.

| GS1 Application Identifier | 4-bit string for date type indicator |
|----------------------------|--------------------------------------|
| (11) Production date | 0000 |
| (13) Packaging date | 0001 |
| (15) Best before date | 0010 |
| (16) Sell by date | 0011 |
| (17) Expiration date | 0100 |
| (7006) First freeze date | 0101 |
| (7007) Harvest date | 0110 |

4079 **14.5.3.1 Encoding**

4080 **Input:**

4081 The input to the encoding method is a date-related GS1 Application Identifier and a 6-digit numeric
 4082 string representing a date value in the format YYMMDD, as expected in the GS1 General
 4083 Specifications.

4084 **Validity Test:**

4085 The GS1 Application Identifier must appear listed within the table above and the 6-digit numeric
 4086 string must only consist of digits 0-9 and is further constrained to be a plausible date value,
 4087 meaning that the third and fourth digits are always in the range 01-12 and the fifth and sixth digits
 4088 are always in the range 00-31 and do not indicate a day-of-month value that is greater than the
 4089 number of days in the month indicated by the third and fourth Digits. e.g. if the third and fourth
 4090 digits are "09" then a value of "31" for the fifth and sixth digits would be invalid because September
 4091 can only contain 30 days.

4092 **Output:**

4093 Create an empty binary string buffer to receive the output. Lookup the GS1 Application Identifier in
4094 the table below and append the corresponding four bits to the binary string buffer as the date type
4095 indicator.

4096 Consider the input string as pairs of digits in which the first two digits are YY, the next two digits are
4097 MM and the final two digits are DD.

4098 Convert YY to a decimal integer (e.g. '22' → 22) and convert this to an unsigned binary value, then
4099 if the resulting binary string for YY is less than seven bits in length, pad to the left with bits set to '0'
4100 to reach a total of seven bits. Append these seven bits to the binary string buffer.

4101 Convert MM to a decimal integer (e.g. '05' → 5) and convert this to an unsigned binary value, then
4102 if the resulting binary string for MM is less than four bits in length, pad to the left with bits set to '0'
4103 to reach a total of four bits. Append these four bits to the binary string buffer.

4104 Convert DD to a decimal integer (e.g. '31' → 31) and convert this to an unsigned binary value, then
4105 if the resulting binary string for DD is less than five bits in length, pad to the left with bits set to '0'
4106 to reach a total of five bits. Append these five bits to the binary string buffer.

4107 The binary string buffer should now consist of a total of 20 bits and should be considered as the
4108 output of this encoding method.

4109 **14.5.3.2 Decoding**

4110 **Input:**

4111 The input to the decoding method is a binary string of 20 bits.

4112 **Validity Test:**

4113 The left-most four bits must appear in the date table above, to indicate a specific date type,
4114 otherwise encoding fails. The next sixteen bits will be decoded as a 6-digit numeric string
4115 representing a date formatted as YYMMDD. After decoding, the third and fourth digits are always in
4116 the range 01-12 and the fifth and sixth digits are always in the range 00-31 and do not indicate a
4117 day-of-month value that is greater than the number of days in the month indicated by the third and
4118 fourth Digits. e.g. if the third and fourth digits are "09" then a value of "31" for the fifth and sixth
4119 digits would be invalid because September can only contain 30 days.

4120 **Output:**

4121 Lookup the left-most four bits in the table above to identify the GS1 Application Identifier to which
4122 the YYMMDD value corresponds.

4123 Create an empty string buffer to receive the six-digit output value YYMMDD.

4124 Treat the remaining sixteen bits as an encoding of the value.

4125 Working from left to right, read the next 7 bits as unsigned binary integer y, then convert to a base
4126 10 value YY, padding to the left with a single '0' digit if the initial result after conversion to base 10
4127 was in the range 0-9.

4128 Read the next 4 bits as unsigned binary integer m, then convert to a base 10 value MM, padding to
4129 the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9.

4130 Read the next 5 bits as unsigned binary integer d, then convert to a base 10 value DD, padding to
4131 the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9.

4132 Check that MM is within the range 01-12 and that DD is within the range 00-31 and does not exceed
4133 the number of days in the month for the month indicated by MM. Otherwise decoding fails.

4134 Concatenate YY MM and DD in sequence as the output value YYMMDD for the date-related GS1
4135 Application Identifier identified by the date type indicator (the left-most four bits of the binary input
4136 string).

4137 **14.5.4 "Fixed-Length Numeric"**

4138 The Fixed-Length Numeric encoding method is used for a segment that can represent numeric digits
 4139 0-9 using 4 bits per digit/character, preserving leading zero digits and (where possible) aligning with
 4140 nibble (half-byte) boundaries to support air interface filtering on a known sequence of digits (such
 4141 as a known GS1 Company Prefix), irrespective of any initial indicator digit or extension digit that
 4142 may be present. The encoding and decoding methods use the following table:
 4143

4144 **Table 14-3 "Fixed-Length Numeric" encoding table**

| Numeric character | 4-bit sequence |
|-------------------|----------------|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

4145 **14.5.4.1 Encoding**

4146 **Input:**

4147 The input to the encoding method is a fixed-length string of N characters, each of which is either a
 4148 numeric digit in the range 0-9.

4149 **Validity Test:**

4150 The input must not contain any characters except for digits 0-9, otherwise the encoding fails.

4151 **Output:**

4152 Create an empty binary string buffer to receive the output. Working from left to right, consider
 4153 each character of the input string. Lookup the character in the table above and append the
 4154 corresponding sequence of four bits to the binary string buffer. Continue until each character of the
 4155 input string has been processed. For an input string of N digits, the binary string buffer should now
 4156 contain 4N bits and is considered to be the output of this encoding method.

4157 **14.5.4.2 Decoding**

4158 **Input:**

4159 The input to the decoding method is a fixed-length binary string of 4N bits, considered as a
 4160 concatenation of N groups of 4-bit sequences

4161 **Validity Test:**

4162 Each of the 4-bit sequences in the input must appear within the table above, otherwise decoding
 4163 fails. The output must not contain any characters except for digits 0-9, otherwise the decoding fails

4164 **Output:**

4165 Create an empty string buffer to receive the numeric string output. Working from left to right,
 4166 consider each set of four bits of the input string, moving the cursor to the right by four bits each

4167 time. Lookup the four bit sequence in the table above and append the corresponding character to
 4168 the output string buffer. Continue until no further bits remain to be processed in the binary input
 4169 string. For a binary input string of 4N bits, the output string buffer should now contain N digits 0-9
 4170 and is considered to be the output of this decoding method.

4171 **14.5.5 "Delimited/Terminated Numeric"**

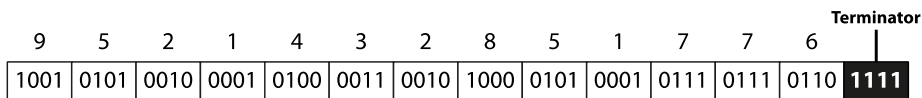
4172 The Delimited/Terminated 4-bit Integer encoding method is used for a segment that can represent a
 4173 variable-length string that begins with numeric digits 0-9, preserving leading zero digits and (where
 4174 possible) aligning with nibble (half-byte) boundaries to support air interface filtering on a known
 4175 sequence of digits, irrespective of any initial indicator digit or extension digit that may be present.

4176 If the string contains no characters except digits 0-9, a 4-bit terminator '1111' indicates the end of
 4177 the string.

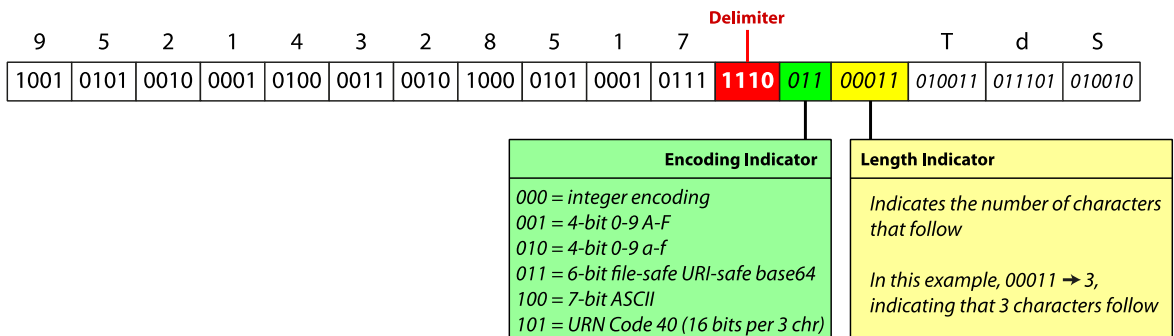
4178 If the string contains characters other than numeric digits 0-9, a 4-bit delimiter indicates the end of
 4179 the initial all-numeric substring, with the remainder of the string (starting with the first character
 4180 that is not a digit 0-9) being encoded using the variable-length alphanumeric method.

4181 **Figure 14-3** Example of numeric delimiter and terminator

(a) All-numeric values always end with the 4-bit terminator '1111'



(b) For other values that are not all-numeric, a 4-bit delimiter '1110' indicates the end of the initial all-numeric part



4182 The encoding and decoding methods use the following table for all of the initial digits:
 4183
 4184

4185 **Table 14-4** Encoding table for initial digits of "Delimited/Terminated Numeric" encoding method

| Numeric character | 4-bit sequence | Interpretation |
|-------------------|----------------|-------------------|
| 0 | 0000 | Numeric digit '0' |
| 1 | 0001 | Numeric digit '1' |
| 2 | 0010 | Numeric digit '2' |
| 3 | 0011 | Numeric digit '3' |
| 4 | 0100 | Numeric digit '4' |
| 5 | 0101 | Numeric digit '5' |
| 6 | 0110 | Numeric digit '6' |

| Numeric character | 4-bit sequence | Interpretation |
|-------------------|----------------|---|
| 7 | 0111 | Numeric digit '7' |
| 8 | 1000 | Numeric digit '8' |
| 9 | 1001 | Numeric digit '9' |
| <i>Delimiter</i> | 1110 | End of the initial all-numeric substring; the remainder of the string uses the variable-length alphanumeric – see section 14.5.6 and its subsections. |
| <i>Terminator</i> | 1111 | End of a string that is all-numeric |

4186 **14.5.5.1 Encoding**

4187 **Input:**

4188 The input to the encoding method is a string of characters, either consisting only of digits 0-9 or
4189 with an initial substring that consists only of digits 0-9.

4190 **Validity Test:**

4191 The input must begin with a sequence of numeric digits 0-9, preserving leading zero digits, but may
4192 be followed by a string of alphanumeric or symbol characters that are permitted for the value of this
4193 GS1 Application Identifier.

4194 **Output:**

4195 Create an empty binary string buffer to receive the output. Working from left to right, consider
4196 each character of the input string. If the character is a digit 0-9, lookup the

4197 Lookup the digit in the table below and append the corresponding sequence of four bits to the binary
4198 string buffer. Continue until each character of the input string has been processed. Finally, if no
4199 variable-length alphanumeric segment follows, append a terminator sequence of four bits ('1111')
4200 otherwise, if a variable-length alphanumeric segment follows, append a delimiter sequence of four
4201 bits ('1110'). For an input string of N digits, the binary string buffer should now contain (4N+4) bits
4202 and is considered to be the output of this encoding method. If the input string was not all-numeric,
4203 the binary string buffer should be further appended with the output of applying the variable-length
4204 alphanumeric method to the remaining characters– see section [14.5.6](#)

4205 **14.5.5.2 Decoding**

4206 **Input:**

4207 The input to the encoding method is a binary string

4208 **Validity Test:**

4209 The output must begin with a sequence of numeric digits 0-9, preserving leading zero digits, but
4210 may be followed by a string of alphanumeric or symbol characters that are permitted for the value
4211 of this GS1 Application Identifier.

4212 **Output:**

4213 Create an empty string buffer to receive the output. Working from left to right, consider each
4214 excessive group of four bits as a hexadecimal character.

4215 If the four bits correspond to a digit 0-9, append this character to the output buffer. If the four bits
4216 are '1111' (hexadecimal character F), the final terminator has been read and indicates the end of an
4217 all-numeric value; the output is the all-numeric contents of the output string buffer. If the four bits
4218 are '1110' (hexadecimal character E), the delimiter character has now been read, indicating that the
4219 next character is not a digit but instead decoding switches after reading the delimiter '1110' to the

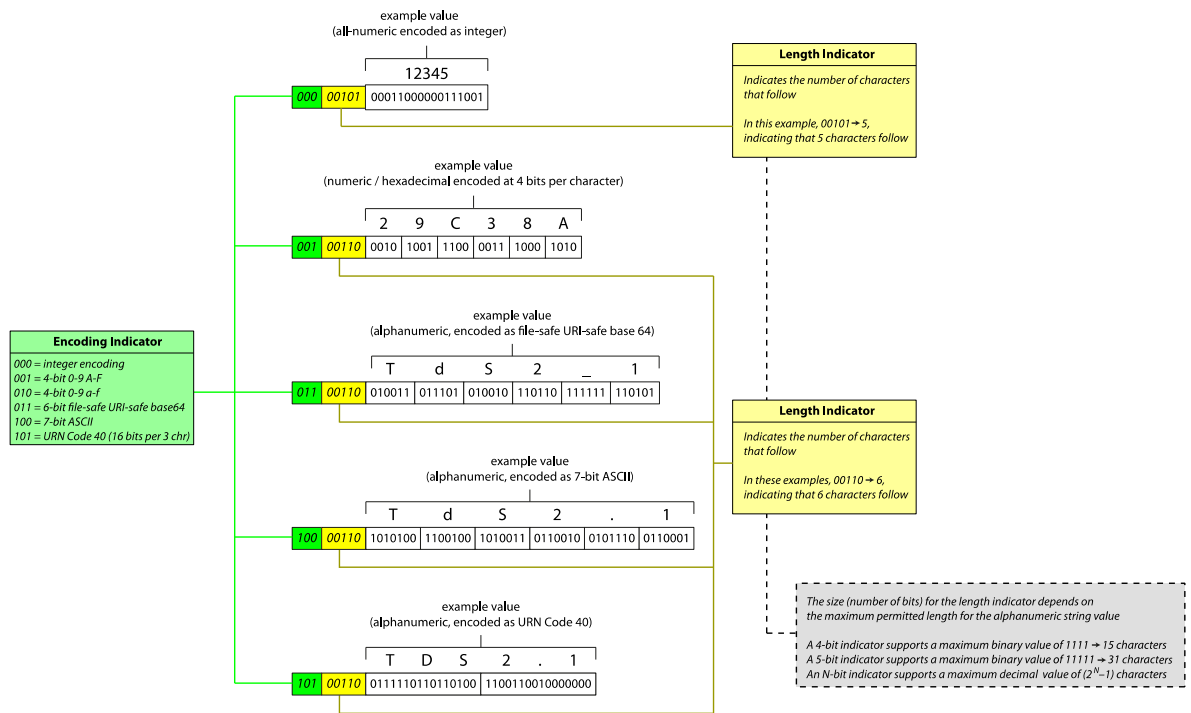
4220
4221
4222
4223

variable-length alphanumeric method and the next bits are a 3-bit encoding indicator, followed by a length indicator (see column f of Table F). The final output consists of the all-numeric contents of the output string buffer from this method, concatenated with with the output of the variable length alphanumeric method used to decode the remaining bits.

4224 **14.5.6 "Variable-length alphanumeric"**

4225 The Variable-length Alphanumeric encoding method is used to encode variable-length alphanumeric
4226 strings using the minimum number of bits. This requires knowledge of the length of the string to be
4227 encoded, as well as analysis of the character set required to express the value. Shorter lengths and
4228 more restricted character sets result in fewer bits.
4229

Figure 14-4 Examples of "Variable-length alphanumeric" encoding method



4230

4231 When encoding, implementations may use **the decision tree below**, to determine the most
4232 efficient encoding method to use, based on the characters actually present in the value to be
4233 encoded, then use that method specified in the relevant subsection. Having said that, a tag that is
4234 encoded using a less efficient encoding method may still conform to TDS 2.0 provided that the
4235 actual encoding method used has been correctly indicated via the three encoding indicator bits.

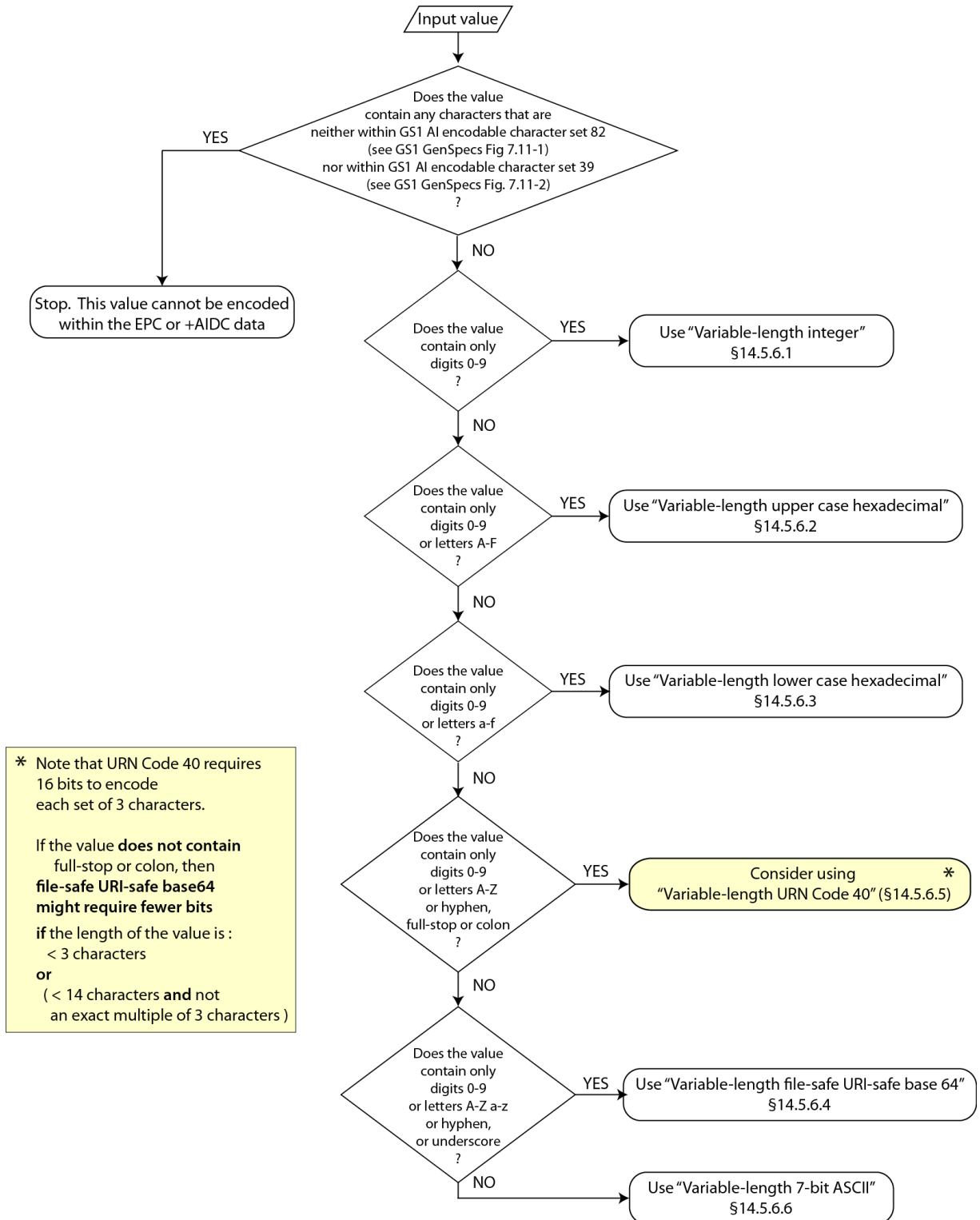
4236 When decoding, the first three bits are the encoding indicator. Refer to the decision tree flowchart or
4237 Table E (encoding indicator values) to determine which subsection to use for the value of the
4238 encoding indicator.

4239 Although the decision tree flowchart and Table E provide guidance about which encoding method is
4240 likely to require the fewest bits for the actual value being encoded, the use of a less efficient
4241 encoding method is permitted, provided that the encoding indicator is set correctly.

4242 Note also that although the ["Variable-length URN Code 40 \(§14.5.6.5\)](#) method is slightly more
4243 efficient (at 16 bits per 3 characters) than the ["Variable-length 6-bit file-safe URI-safe base 64](#)
4244 [\(§14.5.6.4\)](#) method (at 6 bits per character), there are situations where use of the latter may result
4245 in fewer bits, particularly if the length of the value is less than 3 characters or if it is less than 14
4246 characters and not an exact multiple of 3 characters. For values longer than 13 characters,
4247 ["Variable-length URN Code 40 \(§14.5.6.5\)](#) may be more efficient, if its more restricted character set
4248 is sufficient to express the value being encoded.

4249
4250

Figure 14-5 Decision tree flowchart to select the most efficient encoding method based on the value being encoded



* Note that URN Code 40 requires 16 bits to encode each set of 3 characters.

If the value **does not** contain full-stop or colon, then **file-safe URI-safe base64** might require fewer bits if the length of the value is :
 < 3 characters
 or
 (< 14 characters **and** not an exact multiple of 3 characters)

4251
4252
4253
4254
4255

4256
4257

Table E – lists the permitted values for **encoding indicator** together with the encoding methods and the character ranges supported by each method

| 3-bit encoding indicator | Coding method name | Defined in TDS section | Supported characters | Number of bits per character |
|--------------------------|---|--------------------------|--|--|
| 000 = 0 | Variable-length integer | 14.5.6.1 | 0-9 | ≈ 3.32 bits per digit, rounded up to next integer |
| 001 = 1 | Variable-length upper case hexadecimal | 14.5.6.2 | 0-9 A-F | 4 bits per digit or hexadecimal character |
| 010 = 2 | Variable-length lower case hexadecimal | 14.5.6.3 | 0-9 a-f | 4 bits per digit or hexadecimal character |
| 011 = 3 | Variable-length file-safe URI-safe base 64 | 14.5.6.4 | 0-9 A-Z a-z _ - | 6 bits per character |
| 100 = 4 | Variable-length 7-bit ASCII | 14.5.6.6 | All 82 characters within GS1 Gen Specs Fig 7.11-1 OR All 39 characters within GS1 Gen Specs Fig 7.11-2 | 7 bits per character |
| 101 = 5 | Variable-length URN Code 40 | 14.5.6.5 | 0-9 A-Z . : - | ≈ 5.33 bits per character (16 bits per 3 characters) |
| 110 = 6 | Reserved for future use | | | |
| 111 = 7 | Reserved for encoding indicators longer than 3 bits | | | |

4258 **14.5.6.1 "Variable-length integer"**

4259 The Variable-length Integer encoding method is used to encode variable-length numeric strings as
4260 unsigned binary integers using the minimum number of bits. It preserves leading zeros, since the
4261 decoding method is required to left-pad the decoded integer to the number of digits indicated by the
4262 length indicator that was encoded. This method requires knowledge of L, the length of the string to
4263 be encoded, as well as L_{max}, the maximum permitted length for such a string.

4264 Note: this is similar to the Fixed-Bit-Length Integer method (§14.5.2) except that the binary value
4265 is appended after appropriate encoding indicator (three bits set to 000) and length indicator.

4266 **14.5.6.1.1 Encoding**

4267 **Input:**

4268 The input to the encoding method is a numeric string of length L consisting only of digits 0-9.

4269 **Validity Test:**

4270 If the input string contains characters other than digits 0-9 or length L > L_{max}, encoding fails.

4271 **Output:**

4272 Create an empty binary string buffer to receive the output. Append three bits '000' to the binary
4273 string buffer, to set an encoding indicator value of '0'.

4274 Lookup b_{LI}, the number of bits for expressing the length indicator in Table F.

4275 Convert the actual length L from a base 10 integer to a binary value, then if necessary, pad to the
 4276 left with bits of '0' to reach a total length b_{LI} for the binary string representing the length indicator.
 4277 If $L_{max} = 1$, the binary string representing the length indicator is empty, of zero length.
 4278 Append the binary string representing the length indicator to the binary string buffer.
 4279 Convert the input string of L digits 0-9 to a base10 integer then convert this to an unsigned binary
 4280 integer, v .
 4281 Calculate b_v , the number of bits for expressing the value either via a lookup of L in table B and
 4282 reading the value in the column titled 'Integer encoding' or using the following formula:
 4283 $b_v = \text{ceiling}(L \cdot \log(10) / \log(2))$
 4284
 4285 If necessary, pad the binary string v with bits of '0' to reach a total length b_v for the binary string
 4286 representing the numeric string value.
 4287 After any necessary padding, append binary string v (of length b_v) to the binary string buffer.
 4288 The contents of the binary string buffer is now the binary output of this encoding method.

4289 **14.5.6.1.2 Decoding**

4290 **Input:**

4291 The input to the decoding method is a binary string for which the leftmost three bits must be '000'.

4292 **Validity Test:**

4293 If the leftmost three bits of the input binary string do not match '000', decoding fails.

4294 If the output string contains characters other than digits 0-9 or if length $L > L_{max}$, decoding fails.

4295 **Output:**

4296 Create an empty binary string buffer to receive the output.

4297 Read the first three bits of the input binary string as the encoding indicator and check that these
 4298 match '000', otherwise this decoding method cannot be used.

4299 Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4300 Read the next b_{LI} bits of the binary input string as the length indicator and convert this binary value
 4301 to an unsigned base 10 integer L , the number of characters that are encoded. Within the binary
 4302 input string, move the cursor past the b_{LI} length indicator bits to begin decoding the actual value.

4303 Calculate b_v , the number of bits for expressing the value either via a lookup of L in table B and
 4304 reading the value in the column titled 'Integer encoding' or using the following formula:
 4305 $b_v = \text{ceiling}(L \cdot \log(10) / \log(2))$
 4306

4307

4307 Read the next b_v bits from the binary string and convert this to an unsigned base 10 integer V .

4308 Convert V to a numeric string. If V is fewer than L digits in length, left-pad V with digits of '0' to
 4309 reach a total of L digits. The resulting L -digit numeric string value V (with any necessary left-
 4310 padding) is the output of this decoding method.

4311 **14.5.6.2 "Variable-length upper case hexadecimal"**

4312 The Variable-length upper case hexadecimal method is used to encode variable-length strings
 4313 consisting of digits 0-9 and letters A-F as unsigned binary integers using four bits per character.
 4314 This requires knowledge of L , the length of the string to be encoded, as well as L_{max} , the maximum
 4315 permitted length for such a string.

4316
4317

This method uses the following table to map each character 0-9 A-F to a 4 bit binary string:

Table 14-5 Mapping table for "Variable-length upper case hexadecimal" encoding method

| Character | 4-bit binary string |
|-----------|---------------------|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |

| Character | 4-bit binary string |
|-----------|---------------------|
| 8 | 1000 |
| 9 | 1001 |
| A | 1010 |
| B | 1011 |
| C | 1100 |
| D | 1101 |
| E | 1110 |
| F | 1111 |

4318

14.5.6.2.1 Encoding

4319

Input:

4320
4321

The input to the encoding method is a numeric string of length L consisting only of digits 0-9 or letters A-F.

4322

Validity Test:

4323
4324

If the input string contains characters other than digits 0-9 or letters A-F or length $L > L_{max}$, encoding fails.

4325

Output:

4326
4327

Create an empty binary string buffer to receive the output. Append three bits '001' to the binary string buffer, to set an encoding indicator value of '1'.

4328

Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4329
4330

Convert the actual length L from a base 10 integer to a binary value, then if necessary, pad to the left with bits of '0' to reach a total length b_{LI} for the binary string representing the length indicator.

4331

If $L_{max} = 1$, the binary string representing the length indicator is empty, of zero length.

4332

Append the binary string representing the length indicator to the binary string buffer.

4333
4334
4335

Working from left to right across the input string, lookup each character in the table above and append the corresponding four bits to the binary string buffer. Repeat until all L characters of the input string have been processed.

4336

The contents of the binary string buffer is now the output of this encoding method.

4337

14.5.6.2.2 Decoding

4338

Input:

4339
4340

The input to the encoding method is a binary string whose leftmost three bits are '001', corresponding to an encoding indicator value '1' for this method.

4341

Validity Test:

4342

If the input binary string does not begin with bits '001' this decoding method cannot be used.

4343
4344

If the output string contains characters other than digits 0-9 or letters A-F or is of length $L > L_{max}$, decoding fails.

4345

Output:

4346

Create an empty string buffer to receive the output.

4347

Read three bits from the binary input string and check that these match '001', otherwise decoding fails. Within the binary input string, advance the cursor beyond those leftmost three bits.

4348

4349

Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4350

Read b_{LI} bits from the binary input string and convert this unsigned integer value to base 10 value L , the number of characters that are to be decoded. Within the binary input string, advance the cursor beyond the b_{LI} length indicator bits. Repeat the follow procedure L times, once per character to be decoded:

4351

4352

4353

4354

Read the next four bits from the binary input string and advance the cursor beyond the bits that have just been read. Lookup the four bits in the table above and append the corresponding character to the output string buffer.

4355

4356

4357

When L characters have been decoded, the contents of the output string buffer is the output of this decoding method.

4358

4359

14.5.6.3 "Variable-length lower case hexadecimal"

4360

The Variable-length lower case hexadecimal method is used to encode variable-length strings consisting of digits 0-9 and letters a-f as unsigned binary integers using four bits per character. This requires knowledge of L , the length of the string to be encoded, as well as L_{max} , the maximum permitted length for such a string.

4361

4362

4363

4364

This method uses the following table to map each character 0-9 a-f to a 4 bit binary string:

4365

Table 14-6 Mapping table for "Variable-length lower case hexadecimal" encoding method

| Character | 4-bit binary string |
|-----------|---------------------|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |

| Character | 4-bit binary string |
|-----------|---------------------|
| 8 | 1000 |
| 9 | 1001 |
| a | 1010 |
| b | 1011 |
| c | 1100 |
| d | 1101 |
| e | 1110 |
| f | 1111 |

4366

14.5.6.3.1 Encoding

4367

Input:

4368

The input to the encoding method is a numeric string of length L consisting only of digits 0-9 or letters a-f.

4369

4370

Validity Test:

4371

If the input string contains characters other than digits 0-9 or letters a-f or length $L > L_{max}$, encoding fails.

4372

4373

Output:

4374

Create an empty binary string buffer to receive the output. Append three bits '010' to the binary string buffer, to set an encoding indicator value of '2'.

4375

4376 Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.
 4377 Convert the actual length L from a base 10 integer to a binary value, then if necessary, pad to the
 4378 left with bits of '0' to reach a total length b_{LI} for the binary string representing the length indicator.
 4379 If $L_{max} = 1$, the binary string representing the length indicator is empty, of zero length.
 4380 Append the binary string representing the length indicator to the binary string buffer.
 4381 Working from left to right across the input string, lookup each character in the table above and
 4382 append the corresponding four bits to the binary string buffer. Repeat until all L characters of the
 4383 input string have been processed.
 4384 The contents of the binary string buffer is now the output of this encoding method.

4385 **14.5.6.3.2 Decoding**

4386 **Input:**

4387 The input to the encoding method is a binary string whose leftmost three bits are '010',
 4388 corresponding to an encoding indicator value '2' for this method.

4389 **Validity Test:**

4390 If the input binary string does not begin with bits '010' this decoding method cannot be used.

4391 If the output string contains characters other than digits 0-9 or letters a-f or is of length $L > L_{max}$,
 4392 decoding fails.

4393 **Output:**

4394 Create an empty string buffer to receive the output.

4395 Read three bits from the binary input string and check that these match '010', otherwise decoding
 4396 fails. Within the binary input string, advance the cursor beyond those leftmost three bits.

4397 Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4398 Read b_{LI} bits from the binary input string and convert this unsigned integer value to base 10 value
 4399 L , the number of characters that are to be decoded. Within the binary input string, advance the
 4400 cursor beyond the b_{LI} length indicator bits. Repeat the follow procedure L times, once per character
 4401 to be decoded:

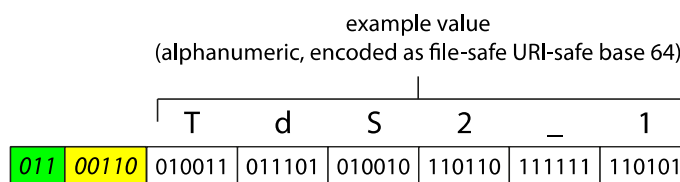
4402 Read the next four bits from the binary input string and advance the cursor beyond the bits that
 4403 have just been read. Lookup the four bits in the table above and append the corresponding
 4404 character to the output string buffer.

4405 When L characters have been decoded, the contents of the output string buffer is the output of this
 4406 decoding method.

4407 **14.5.6.4 "Variable-length 6-bit file-safe URI-safe base 64"**

4408 The Variable-length file-safe base64 encoding method is used to encode variable-length strings of
 4409 digits 0-9, upper case letters A-Z, lower case letters a-z, hyphen or underscore characters using 6
 4410 bits per character. This requires knowledge of L , the length of the string to be encoded, as well as
 4411 L_{max} , the maximum permitted length for such a string.

4412 **Figure 14-6** Example value - alphanumeric, encoded as file-safe URI-safe base 64



4413

4414

Table 14-7 Mapping table for "Variable-length 6-bit file-safe URI-safe base 64" encoding method

| Character | 6-bit binary string |
|-----------|---------------------|
| A | 000000 |
| B | 000001 |
| C | 000010 |
| D | 000011 |
| E | 000100 |
| F | 000101 |
| G | 000110 |
| H | 000111 |
| I | 001000 |
| J | 001001 |
| K | 001010 |
| L | 001011 |
| M | 001100 |
| N | 001101 |
| O | 001110 |
| P | 001111 |
| Q | 010000 |
| R | 010001 |
| S | 010010 |
| T | 010011 |
| U | 010100 |
| V | 010101 |
| W | 010110 |
| X | 010111 |
| Y | 011000 |
| Z | 011001 |
| a | 011010 |
| b | 011011 |
| c | 011100 |
| d | 011101 |

| Character | 6-bit binary string |
|-----------|---------------------|
| g | 100000 |
| h | 100001 |
| i | 100010 |
| j | 100011 |
| k | 100100 |
| l | 100101 |
| m | 100110 |
| n | 100111 |
| o | 101000 |
| p | 101001 |
| q | 101010 |
| r | 101011 |
| s | 101100 |
| t | 101101 |
| u | 101110 |
| v | 101111 |
| w | 110000 |
| x | 110001 |
| y | 110010 |
| z | 110011 |
| 0 | 110100 |
| 1 | 110101 |
| 2 | 110110 |
| 3 | 110111 |
| 4 | 111000 |
| 5 | 111001 |
| 6 | 111010 |
| 7 | 111011 |
| 8 | 111100 |
| 9 | 111101 |

| | |
|---|--------|
| e | 011110 |
| f | 011111 |

| | |
|----------------|--------|
| - (hyphen) | 111110 |
| _ (underscore) | 111111 |

4415 **14.5.6.4.1 Encoding**

4416 **Input:**

4417 The input to the encoding method is a string of length L consisting only of digits 0-9 or upper case
4418 letters A-Z, colon, hyphen and full-stop (period/dot).

4419 **Validity Test:**

4420 If the input string contains characters other than digits 0-9 or upper case letters A-Z, colon, hyphen
4421 and full-stop (period/dot) or length $L > L_{max}$, encoding fails.

4422 **Output:**

4423 Create an empty binary string buffer to receive the output. Append three bits '011' to the binary
4424 string buffer, to set an encoding indicator value of '3'.

4425 Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4426 Convert the actual length L from a base 10 integer to a binary value, then if necessary, pad to the
4427 left with bits of '0' to reach a total length b_{LI} for the binary string representing the length indicator.

4428 If $L_{max} = 1$, the binary string representing the length indicator is empty, of zero length.

4429 Append the binary string representing the length indicator to the binary string buffer.

4430 Starting at the beginning of the input string and moving left-to-right, considering each character in
4431 turn until no further characters remain to be encoded, lookup the character in the table below and
4432 append the corresponding set of six bits to the binary string buffer.

4433 The contents of the binary string buffer is now the binary output of this encoding method.

4434 **14.5.6.4.2 Decoding**

4435 **Input:**

4436 The input to the encoding method is a binary string whose leftmost three bits are '011',
4437 corresponding to an encoding indicator value '3' for this method.

4438 **Validity Test:**

4439 If the input binary string does not begin with bits '011' this decoding method cannot be used.

4440 If the output string contains characters other than digits 0-9 or letters A-Z a-z, hyphen or
4441 underscore or is of length $L > L_{max}$, decoding fails.

4442 **Output:**

4443 Create an empty string buffer to receive the output.

4444 Read three bits from the binary input string and check that these match '011', otherwise decoding
4445 fails. Within the binary input string, advance the cursor beyond those leftmost three bits.

4446 Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4447 Read b_{LI} bits from the binary input string and convert this unsigned integer value to base 10 value
4448 L, the number of characters that are to be decoded. Within the binary input string, advance the
4449 cursor beyond the b_{LI} length indicator bits. Repeat the follow procedure L times, once per character
4450 to be decoded:

4451 Read the next six bits from the binary input string and advance the cursor beyond the bits that have
4452 just been read. Lookup the six bits in the table above and append the corresponding character to
4453 the output string buffer.

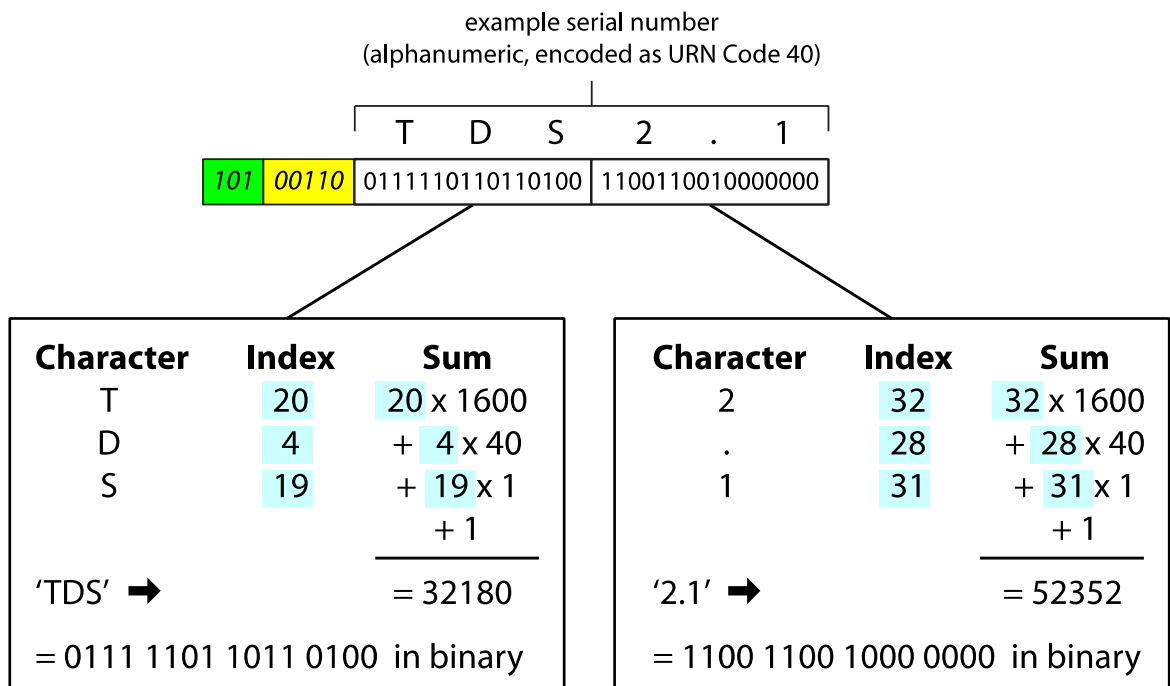
4454 When L characters have been decoded, the contents of the output string buffer is the output of this
 4455 decoding method.

4456 **14.5.6.5 "Variable-length URN Code 40"**

4457 The Variable-length URN Code 40 encoding method is used to encode variable-length strings of
 4458 digits 0-9, upper case letters A-Z, colon, hyphen and full-stop (period/dot) using 16 bits for each set
 4459 of 3 characters. This requires knowledge of L, the length of the string to be encoded, as well as
 4460 L_{max} , the maximum permitted length for such a string.

4461 The figure below illustrates the use of the variable-length URN Code 40 method to encode 6
 4462 characters.

4463 **Figure 14-7** Use of the "Variable-length URN Code 40" method to encode 6 characters



4464
 4465 URN Code 40 uses the following character table to map supportable characters to index values that
 4466 are used in the calculation:

4467 **Table 14-8 URN Code 40 character table**

| Character | Index |
|---------------|-------|
| PAD character | 0 |
| A | 1 |
| B | 2 |
| C | 3 |
| D | 4 |
| E | 5 |
| F | 6 |
| G | 7 |
| H | 8 |

| Character | Index |
|---------------|-------|
| T | 20 |
| U | 21 |
| V | 22 |
| W | 23 |
| X | 24 |
| Y | 25 |
| Z | 26 |
| - (hyphen) | 27 |
| . (full stop) | 28 |

| | |
|---|----|
| I | 9 |
| J | 10 |
| K | 11 |
| L | 12 |
| M | 13 |
| N | 14 |
| O | 15 |
| P | 16 |
| Q | 17 |
| R | 18 |
| S | 19 |

| | |
|-----------|----|
| : (colon) | 29 |
| 0 | 30 |
| 1 | 31 |
| 2 | 32 |
| 3 | 33 |
| 4 | 34 |
| 5 | 35 |
| 6 | 36 |
| 7 | 37 |
| 8 | 38 |
| 9 | 39 |

4468 **14.5.6.5.1 Encoding**

4469 **Input:**

4470 The input to the encoding method is a string of length L consisting only of digits 0-9 or upper case
 4471 letters A-Z, colon, hyphen and full-stop (period/dot). The maximum permitted length for the value
 4472 (L_{max}) must also be known.

4473 **Validity Test:**

4474 If the input string contains characters other than digits 0-9 or upper case letters A-Z, colon, hyphen
 4475 and full-stop (period/dot) or length $L > L_{max}$, encoding fails.

4476 **Output:**

4477 Create an empty binary string buffer to receive the output. Append three bits '101' to the binary
 4478 string buffer, to set an encoding indicator value of '5'.

4479 Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4480 Convert the actual length L from a base 10 integer to a binary value, then if necessary, pad to the
 4481 left with bits of '0' to reach a total length b_{LI} for the binary string representing the length indicator.

4482 If $L_{max} = 1$, the binary string representing the length indicator is empty, of zero length.

4483 Append the binary string representing the length indicator to the binary string buffer.

4484 Working from left to right across the input string, consider each successive group of three
 4485 characters. If the final group only contains one or two characters, consider the final group to be
 4486 appended at the right with two or one pad characters respectively, to reach a total of three
 4487 characters.

4488 Within each group of three characters, lookup the corresponding index values for each character. i_1
 4489 is the index value for the first character, i_2 the index for the second character and i_3 is the index for
 4490 the third character. Calculate $r = (1600i_1 + 40i_2 + i_3 + 1)$. Convert r to binary and if necessary,
 4491 left-pad with bits of '0' to reach a total of 16 bits. Append this 16 bit string to the binary string
 4492 buffer and repeat this process for the next group of three characters until no further groups remain
 4493 to be processed.

4494 The contents of the binary string buffer is now the binary output of this encoding method.

4495 **14.5.6.5.2 Decoding**

4496 **Input:**

4497 The input to the decoding method is a binary string. The maximum permitted length for the value (
4498 L_{max}) must also be known.

4499 **Validity Test:**

4500 If the leftmost three bits of the binary input string are not '101' then this method cannot be used
4501 because the encoding indicator does not correspond to this method.

4502 If the output string contains characters other than digits 0-9 or upper case letters A-Z, colon,
4503 hyphen and full-stop (period/dot) or length $L > L_{max}$, encoding fails.

4504 **Output:**

4505 Create an empty string buffer to receive the output. Working from left to right across the binary
4506 input string, read the first three bits and check that these are '101', the encoding indicator value for
4507 this method. Otherwise, this method cannot be used.

4508 Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4509 Read b_{LI} bits as the length indicator and convert that unsigned binary integer to a base 10 value L ,
4510 the number of characters to be read. Move the cursor of the binary string past the three-bit
4511 encoding indicator '101' and the length indicator of b_{LI} bits to begin reading the encoded data.

4512 If L is exactly divisible by 3, the number of iterations $n = L/3$, otherwise $n = \text{ceiling}(L/3)$.

4513 Repeat the following procedure n times, reading and processing 16 bits from the input binary string
4514 on each iteration and advancing the cursor accordingly:

4515 For each iteration, convert the 16 bit string to a base 10 unsigned integer r .

4516 Calculate $i_3 = (r-1)\%40$ where $\%$ is the modulo division operator and $(r-1)\%40$ is the
4517 remainder of $(r-1)$ after division by 40.

4518 Calculate $i_2 = ((r-1 - i_3)/40)\%40$

4519 Calculate $i_1 = ((r-1 - i_3 - 40i_2)/1600)$

4520 Lookup i_1 in the table above and append the corresponding character to the output string buffer.

4521 If $i_2 > 0$, lookup i_2 in the table above and append the corresponding character to the output string
4522 buffer.

4523 If $i_3 > 0$, lookup i_3 in the table above and append the corresponding character to the output string
4524 buffer.

4525 After all n iterations have been completed, the contents of the output string buffer are considered to
4526 be the output of this decoding method.

4527 **14.5.6.6 "Variable-length 7-bit ASCII"**

4528 The Variable-length 7-bit ASCII encoding method is used to encode variable-length strings of
4529 characters within the 82-character GS1 invariant subset of ISO/IEC 646 [ISO646] or within the 39
4530 character GS1 invariant subset of ISO/IEC 646 using 7 bits per character. This requires knowledge
4531 of L , the length of the string to be encoded, as well as L_{max} , the maximum permitted length for such
4532 a string.

4533 This method uses the following character table, mapping characters to 7 bit sequences.

4534 **Table 14-9 Character table for "Variable-length 7-bit ASCII" encoding method**

| Character | 7-bit binary string |
|-----------|---------------------|
| ! | 0100001 |

| Character | 7-bit binary string |
|-----------|---------------------|
| M | 1001101 |

| Character | 7-bit binary string |
|-----------|---------------------|
| " | 0100010 |
| # | 0100011 |
| % | 0100101 |
| & | 0100110 |
| ' | 0100111 |
| (| 0101000 |
|) | 0101001 |
| * | 0101010 |
| + | 0101011 |
| , | 0101100 |
| - | 0101101 |
| . | 0101110 |
| / | 0101111 |
| 0 | 0110000 |
| 1 | 0110001 |
| 2 | 0110010 |
| 3 | 0110011 |
| 4 | 0110100 |
| 5 | 0110101 |
| 6 | 0110110 |
| 7 | 0110111 |
| 8 | 0111000 |
| 9 | 0111001 |
| : | 0111010 |
| ; | 0111011 |
| < | 0111100 |
| = | 0111101 |
| > | 0111110 |
| ? | 0111111 |
| A | 1000001 |
| B | 1000010 |

| Character | 7-bit binary string |
|-----------|---------------------|
| N | 1001110 |
| O | 1001111 |
| P | 1010000 |
| Q | 1010001 |
| R | 1010010 |
| S | 1010011 |
| T | 1010100 |
| U | 1010101 |
| V | 1010110 |
| W | 1010111 |
| X | 1011000 |
| Y | 1011001 |
| Z | 1011010 |
| _ | 1011111 |
| a | 1100001 |
| b | 1100010 |
| c | 1100011 |
| d | 1100100 |
| e | 1100101 |
| f | 1100110 |
| g | 1100111 |
| h | 1101000 |
| i | 1101001 |
| j | 1101010 |
| k | 1101011 |
| l | 1101100 |
| m | 1101101 |
| n | 1101110 |
| o | 1101111 |
| p | 1110000 |
| q | 1110001 |

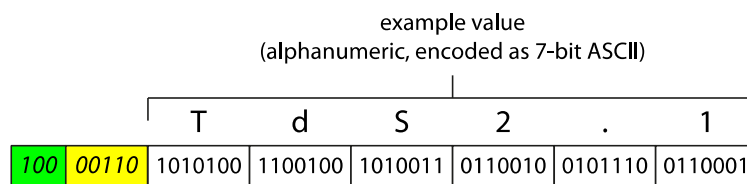
| Character | 7-bit binary string |
|-----------|---------------------|
| C | 1000011 |
| D | 1000100 |
| E | 1000101 |
| F | 1000110 |
| G | 1000111 |
| H | 1001000 |
| I | 1001001 |
| J | 1001010 |
| K | 1001011 |
| L | 1001100 |

| Character | 7-bit binary string |
|-----------|---------------------|
| r | 1110010 |
| s | 1110011 |
| t | 1110100 |
| u | 1110101 |
| v | 1110110 |
| w | 1110111 |
| x | 1111000 |
| y | 1111001 |
| z | 1111010 |
| | |

4535
4536

The following figure provides a worked example to illustrate this method.

Figure 14-8 Example of alphanumeric encoded as 7-bit ASCII



4537

4538 **14.5.6.6.1 Encoding**

4539

Input:

4540
4541
4542

The input to the encoding method is a string of length L consisting only of characters appearing within the 82-character GS1 invariant subset of ISO/IEC 646 or within the 39 character GS1 invariant subset of ISO/IEC 646. See GS1 General Specifications, Figures 7.11-1 and 7.11-2.

4543

Validity Test:

4544
4545
4546

If the input string contains characters other than those appearing within the 82-character GS1 invariant subset of ISO/IEC 646 or within the 39 character GS1 invariant subset of ISO/IEC 646 or length $L > L_{max}$, encoding fails.

4547

Output:

4548
4549

Create an empty binary string buffer to receive the output. Append three bits '100' to the binary string buffer, to set an encoding indicator value of '4'.

4550

Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4551
4552

Convert the actual length L from a base 10 integer to a binary value, then if necessary, pad to the left with bits of '0' to reach a total length b_{LI} for the binary string representing the length indicator.

4553

If $L_{max} = 1$, the binary string representing the length indicator is empty, of zero length.

4554

Append the binary string representing the length indicator to the binary string buffer.

4555
4556
4557

Starting at the beginning of the input string and moving left-to-right, considering each character in turn until no further characters remain to be encoded, lookup the character in the table below and append the corresponding set of seven bits to the binary string buffer.

4558 The contents of the binary string buffer is now the binary output of this encoding method.

4559 **14.5.6.6.2 Decoding**

4560 **Input:**

4561 The input to the decoding method is a binary string. The maximum permitted length for the value (L_{max}) must also be known.
4562

4563 **Validity Test:**

4564 If the leftmost three bits of the binary input string are not '100' then this method cannot be used
4565 because the encoding indicator does not correspond to this method.

4566 If the output string contains characters other than digits 0-9 or letters A-Z a-z,
4567 h147ninitialiunderscore or if its length $L > L_{max}$, decoding fails.

4568 **Output:**

4569 Create an empty string buffer to receive the output. Working from left to right across the binary
4570 input string, read the first three bits and check that these are '100', the encoding indicator value for
4571 this method. Otherwise, this method cannot be used.

4572 Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4573 Read b_{LI} bits from the binary input string and convert this unsigned integer value to base 10 value
4574 L , the number of characters that are to be decoded. Within the binary input string, advance the
4575 cursor beyond the leftmost encoding indicator bits '100' and the b_{LI} length indicator bits. Repeat the
4576 follow procedure L times, once per character to be decoded:

4577 Read the next seven bits from the binary input string and advance the cursor beyond the bits that
4578 have just been read. Lookup the seven bits in the table above and append the corresponding
4579 character to the output string buffer.

4580 When L characters have been decoded, the contents of the output string buffer is the output of this
4581 decoding method.

4582 **14.5.7 "Single data bit"**

4583 GS1 Application Identifiers (4321), (4322), (4323) use a single digit of '0' or '1' to represent a single
4584 bit Boolean value in which '0' indicates false, whereas '1' indicates true.

4585 **14.5.7.1 Encoding**

4586 **Input:**

4587 The input to the encoding method is one decimal digit, 0 ("false") or 1 ("true").

4588 **Validity Test:**

4589 The input must consist of exactly one decimal digit, which must be 0 or 1,

4590 **Output:**

4591 The output is a lone bit, 0 or 1.

4592 **14.5.7.2 Decoding**

4593 **Input:**

4594 The input to the encoding method is a lone bit, 0 or 1.

4595 **Validity Test:**

4596 The input must consist of exactly one bit, otherwise the encoding fails.

4597
4598
4599

Output:

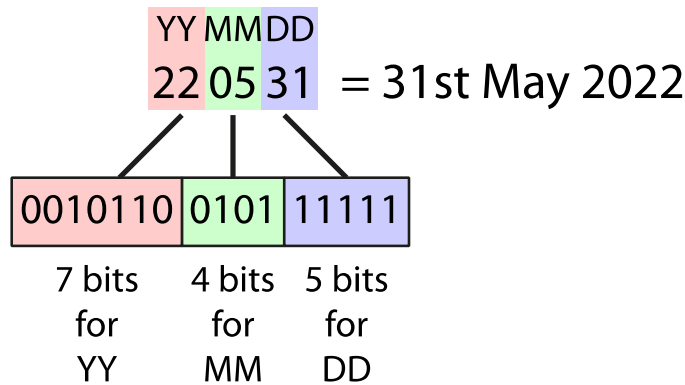
If the single bit is 0, it is decoded as decimal value 0. If the single bit is 1, it is decoded as decimal value 1. 0 = false, 1 = true.

4600
4601
4602
4603
4604
4605

14.5.8 "6-digit date YYMMDD"

Several GS1 Application Identifiers express a date value as a six-digit numeric string formatted as YYMMDD, in which YY represents the year, MM represents the month and DD represents the day of the month. Such a numeric string value can be efficiently encoded using 16 bits as shown in the figure below, using 7 bits to encode YY, 4 bits to encode MM and 5 bits to encode DD:

Figure 14-9 Efficient encoding of YYMMDD date value using 16 bits



4606

14.5.8.1 Encoding

4608
4609
4610

Input:

The input to the encoding method is a 6-digit numeric string representing a date value in the format YYMMDD, as expected in the GS1 General Specifications.

4611

Validity Test:

The 6-digit numeric string must only consist of digits 0-9 and is further constrained to be a plausible date value, meaning that the third and fourth digits are always in the range 01-12 and the fifth and sixth digits are always in the range 00-31 and do not indicate a day-of-month value that is greater than the number of days in the month indicated by the third and fourth Digits. e.g. if the third and fourth digits are "09" then a value of "31" for the fifth and sixth digits would be invalid because September can only contain 30 days.

4618

Output:

4619 Create an empty binary string buffer to receive the output.

4620 Consider the input string as pairs of digits in which the first two digits are YY, the next two digits are MM and the final two digits are DD.

4621

4622 Convert YY to a decimal integer (e.g. '22' → 22) and convert this to an unsigned binary value, then
 4623 if the resulting binary string for YY is less than seven bits in length, pad to the left with bits set to '0'
 4624 to reach a total of seven bits. Append these seven bits to the binary string buffer.

4625 Convert MM to a decimal integer (e.g. '05' → 5) and convert this to an unsigned binary value, then
 4626 if the resulting binary string for MM is less than four bits in length, pad to the left with bits set to '0'
 4627 to reach a total of four bits. Append these four bits to the binary string buffer.

4628 Convert DD to a decimal integer (e.g. '31' → 31) and convert this to an unsigned binary value, then
 4629 if the resulting binary string for DD is less than five bits in length, pad to the left with bits set to '0'
 4630 to reach a total of five bits. Append these five bits to the binary string buffer.

4631 The binary string buffer should now consist of a total of 16 bits and should be considered as the
 4632 output of this encoding method.

4633 **14.5.8.2 Decoding**

4634 **Input:**

4635 The input to the decoding method is a binary string of 16 bits.

4636 **Validity Test:**

4637 The sixteen bits will be decoded as a 6-digit numeric string representing a date formatted as
 4638 YYMMDD. After decoding, the third and fourth digits must always be in the range 01-12 and the
 4639 fifth and sixth digits must always be in the range 00-31 and must not indicate a day-of-month value
 4640 that is greater than the number of days in the month indicated by the third and fourth Digits. e.g. if
 4641 the third and fourth digits are "09" then a value of "31" for the fifth and sixth digits would be invalid
 4642 because September can only contain 30 days.

4643 **Output:**

4644 Create an empty string buffer to receive the six-digit output value YYMMDD.

4645 Treat the sixteen bits as an encoding of the date value.

4646 Working from left to right, read the first 7 bits as unsigned binary integer y, then convert to a base
 4647 10 value YY, padding to the left with a single '0' digit if the initial result after conversion to base 10
 4648 was in the range 0-9.

4649 Read the next 4 bits as unsigned binary integer m, then convert to a base 10 value MM, padding to
 4650 the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9.

4651 Read the next 5 bits as unsigned binary integer d, then convert to a base 10 value DD, padding to
 4652 the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9.

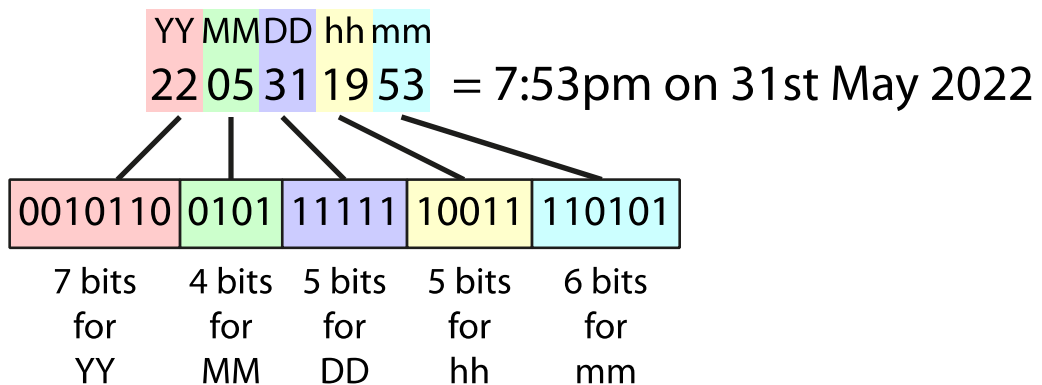
4653 Check that MM is within the range 01-12 and that DD is within the range 00-31 and does not exceed
 4654 the number of days in the month for the month indicated by MM. Otherwise decoding fails.

4655 Concatenate YY MM and DD in sequence as the output value YYMMDD.

4656 **14.5.9 "10-digit date+time YYMMDDhhmm"**

4657 GS1 Application Identifiers (4324), (4325), (7003) use a 10-digit numeric string to express a date
 4658 format YYMMDDhhmm in which YY represents the year, MM represents the month, DD represents
 4659 the day of the month, hh represents the hour of the day and mm represents the minutes. Such a
 4660 numeric string value can be efficiently encoded using 27 bits as shown in the figure below, using 7
 4661 bits to encode YY, 4 bits to encode MM, 5 bits to encode DD, 5 bits to encode hh and 6 bits to
 4662 encode mm:

4663 **Figure 14-10** Encoding of YYMMDDhhmm date time value using 27 bits



4664

4665 14.5.9.1 Encoding

4666 **Input:**

4667 The input to the encoding method is a 10-digit numeric string representing a date value in the
4668 format YYMMDDhhmm, as expected in the GS1 General Specifications.

4669 **Validity Test:**

4670 The 10-digit numeric string must only consist of digits 0-9 and is further constrained to be a
4671 plausible date+time value, meaning that the third and fourth digits are always in the range 01-12
4672 and the fifth and sixth digits are always in the range 00-31 and do not indicate a day-of-month
4673 value that is greater than the number of days in the month indicated by the third and fourth Digits.
4674 e.g. if the third and fourth digits are "09" then a value of "31" for the fifth and sixth digits would be
4675 invalid because September can only contain 30 days. The seventh and eighth digits must be in the
4676 range 00-24, while the ninth and tenth digits must be in the range 00-59.

4677 **Output:**

4678 Create an empty binary string buffer to receive the output.

4679 Consider the input string as pairs of digits in which the first two digits are YY, the next two digits are
4680 MM, followed by two digits DD, a further two digits hh and a final two digits mm.

4681 Convert YY to a decimal integer (e.g. '22' → 22) and convert this to an unsigned binary value, then
4682 if the resulting binary string for YY is less than seven bits in length, pad to the left with bits set to '0'
4683 to reach a total of seven bits. Append these seven bits to the binary string buffer.

4684 Convert MM to a decimal integer (e.g. '05' → 5) and convert this to an unsigned binary value, then
4685 if the resulting binary string for MM is less than four bits in length, pad to the left with bits set to '0'
4686 to reach a total of four bits. Append these four bits to the binary string buffer.

4687 Convert DD to a decimal integer (e.g. '31' → 31) and convert this to an unsigned binary value, then
4688 if the resulting binary string for DD is less than five bits in length, pad to the left with bits set to '0'
4689 to reach a total of five bits. Append these five bits to the binary string buffer.

4690 Convert hh to a decimal integer (e.g. '07' → 7) and convert this to an unsigned binary value, then if
4691 the resulting binary string for hh is less than five bits in length, pad to the left with bits set to '0'
4692 to reach a total of five bits. Append these five bits to the binary string buffer.

4693 Convert mm to a decimal integer (e.g. '59' → 59) and convert this to an unsigned binary value,
4694 then if the resulting binary string for mm is less than six bits in length, pad to the left with bits set
4695 to '0' to reach a total of six bits. Append these six bits to the binary string buffer.

4696 The binary string buffer should now consist of a total of 27 bits and should be considered as the
4697 output of this encoding method.

4698 14.5.9.2 Decoding

4699 **Input:**

4700 The input to the decoding method is a binary string of 27 bits.

4701 **Validity Test:**

4702 The sixteen bits will be decoded as a 10-digit numeric string representing a date formatted as
4703 YYMMDDhhmm. After decoding, the third and fourth digits must always be in the range 01-12 and
4704 the fifth and sixth digits must always be in the range 00-31 and must not indicate a day-of-month
4705 value that is greater than the number of days in the month indicated by the third and fourth Digits.
4706 e.g. if the third and fourth digits are "09" then a value of "31" for the fifth and sixth digits would be
4707 invalid because September can only contain 30 days. The seventh and eighth digits must be in the
4708 range 00-24, while the ninth and tenth digits must be in the range 00-59.

4709 **Output:**

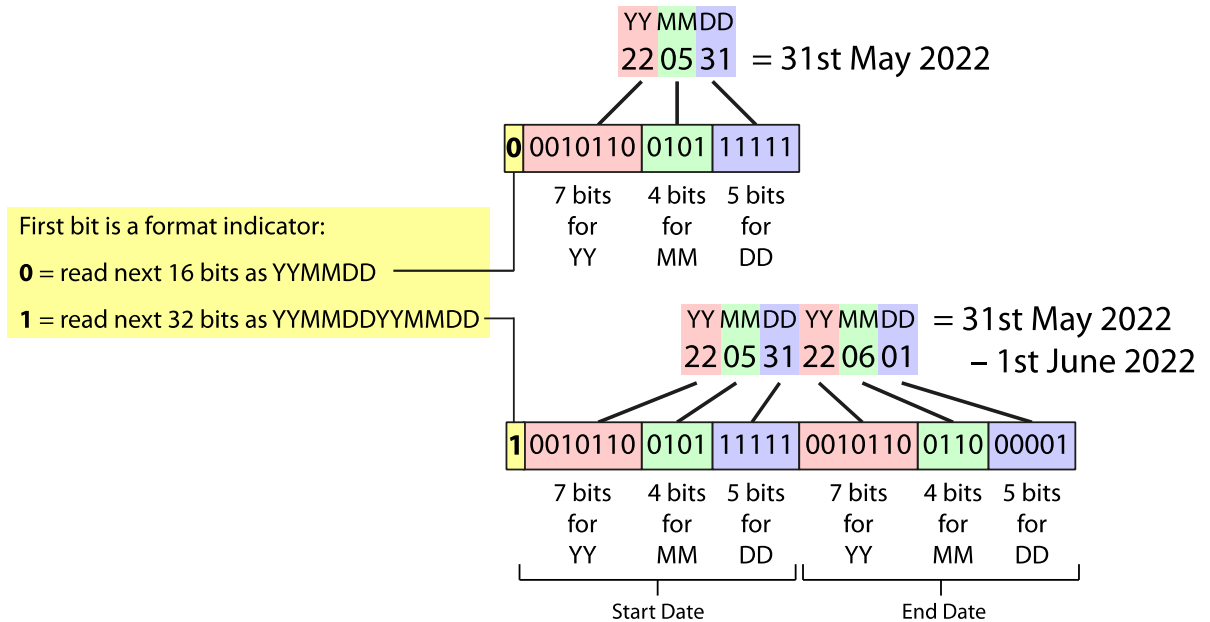
4710 Create an empty string buffer to receive the ten-digit output value YYMMDDhhmm.

4711 Treat the 27 bits as an encoding of the date+time value.
 4712 Working from left to right, read the first 7 bits as unsigned binary integer *y*, then convert to a base
 4713 10 value *YY*, padding to the left with a single '0' digit if the initial result after conversion to base 10
 4714 was in the range 0-9.
 4715 Read the next 4 bits as unsigned binary integer *m*, then convert to a base 10 value *MM*, padding to
 4716 the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9.
 4717 Read the next 5 bits as unsigned binary integer *d*, then convert to a base 10 value *DD*, padding to
 4718 the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9.
 4719 Read the next 5 bits as unsigned binary integer *h*, then convert to a base 10 value *hh*, padding to
 4720 the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9.
 4721 Read the next 6 bits as unsigned binary integer *n*, then convert to a base 10 value *mm*, padding to
 4722 the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9.
 4723 Check that *MM* is within the range 01-12 and that *DD* is within the range 00-31 and does not exceed
 4724 the number of days in the month for the month indicated by *MM*. Otherwise decoding fails.
 4725 Check that *hh* is within the range 00-24 and that *mm* is within the range 00-59. If *hh* is '24' then
 4726 *mm* must be '00' otherwise decoding fails
 4727 Concatenate *YY MM DD hh mm* in sequence as the output value *YYMMDDhhmm*.

4728 **14.5.10 "Variable-format date / date range"**

4729 GS1 Application Identifier (7007) expresses either a harvest date or a harvest date range (indicating
 4730 a start date then an end date). A single *YYMMDD* date value can be efficiently encoded using 16
 4731 bits, whereas a date range consisting of a start date and end date will require 32 bits. In order to
 4732 distinguish between these two possibilities, this method uses a single bit format indicator as shown
 4733 in the figure below. If that single bit format indicator is set to 0, a single date value *YYMMDD* is
 4734 expected. If the single bit format indicator is set to 1, a pair of date values *YYMMDD YYMMDD* is
 4735 expected, to express a date range.
 4736

Figure 14-11 Encoding of "Variable-format date / date range"



4737

4738 **14.5.10.1 Encoding**4739 **Input:**

4740 The input to the encoding method is either a 6-digit numeric string representing a date value in the
4741 format YYMMDD, or a 12 digit numeric string representing a date range in the format
4742 YYMMDDYYMMDD as expected in the GS1 General Specifications.

4743 **Validity Test:**

4744 A 6-digit numeric string must only consist of digits 0-9 and is further constrained to be a plausible
4745 date value, meaning that the third and fourth digits are always in the range 01-12 and the fifth and
4746 sixth digits are always in the range 00-31 and do not indicate a day-of-month value that is greater
4747 than the number of days in the month indicated by the third and fourth Digits. e.g. if the third and
4748 fourth digits are "09" then a value of "31" for the fifth and sixth digits would be invalid because
4749 September can only contain 30 days. A 12-digit numeric string must only consist of digits 0-9 and
4750 both the first six digits and last six digits are further constrained to be a plausible date value, as
4751 previously explained.

4752 **Output:**

4753 Create an empty binary string buffer to receive the output.

4754 If the input is a 6-digit string in the format YYMMDD, append a single bit of '0' to the binary string
4755 buffer. If the input is a 12-digit string in the format YYMMDD, append a single bit of '1' to the
4756 binary string buffer.

4757 Perform the following procedure once if the input is a 6-digit string YYMMDD or perform it twice,
4758 with each set of six digits YYMMDD for the date range if the input is a 12-digit string
4759 YYMMDDYYMMDD.

4760 Consider the input string as pairs of digits in which the first two digits are YY, the next two digits are
4761 MM and the final two digits are DD.

4762 Convert YY to a decimal integer (e.g. '22' → 22) and convert this to an unsigned binary value, then
4763 if the resulting binary string for YY is less than seven bits in length, pad to the left with bits set to '0'
4764 to reach a total of seven bits. Append these seven bits to the binary string buffer.

4765 Convert MM to a decimal integer (e.g. '05' → 5) and convert this to an unsigned binary value, then
4766 if the resulting binary string for MM is less than four bits in length, pad to the left with bits set to '0'
4767 to reach a total of four bits. Append these four bits to the binary string buffer.

4768 Convert DD to a decimal integer (e.g. '31' → 31) and convert this to an unsigned binary value, then
4769 if the resulting binary string for DD is less than five bits in length, pad to the left with bits set to '0'
4770 to reach a total of five bits. Append these five bits to the binary string buffer.

4771 The binary string buffer should now consist of a total of 17 bits (for a 6-digit input of YYMMDD) or
4772 33 bits (for a 12-digit input of YYMMDDYYMMDD) and should be considered as the output of this
4773 encoding method.

4774 **14.5.10.2 Decoding**4775 **Input:**

4776 The input to the decoding method is a binary string of 17 bits or 33 bits, of which the first bit is a
4777 date format indicator, where '0' indicates that 16 bits follow, to be decoded as a 6-digit date string
4778 YYMMDD, whereas '1' indicates that 32 bits follow, to be decoded as a 12-digit date range string
4779 YYMMDDYYMMDD.

4780 **Validity Test:**

4781 Each set of sixteen bits will be decoded as a 6-digit numeric string representing a date formatted as
4782 YYMMDD. After decoding, the third and fourth digits must always be in the range 01-12 and the
4783 fifth and sixth digits must always be in the range 00-31 and must not indicate a day-of-month value
4784 that is greater than the number of days in the month indicated by the third and fourth Digits. e.g. if

4785 the third and fourth digits are "09" then a value of "31" for the fifth and sixth digits would be invalid
 4786 because September can only contain 30 days.

4787 **Output:**

4788 Create an empty string buffer to receive the six-digit output value YYMMDD or the twelve-digit
 4789 output value YYMMDDYYMMDD.

4790 Read the left-most bit of the binary input string and move the cursor beyond it, to begin reading
 4791 data. If the single bit value is '0', perform the following procedure once. If the single bit value is
 4792 '1', perform the following procedure twice.

4793 Treat the next sixteen bits as an encoding of a date value.

4794 Working from left to right, read the first 7 bits as unsigned binary integer *y*, then convert to a base
 4795 10 value YY, padding to the left with a single '0' digit if the initial result after conversion to base 10
 4796 was in the range 0-9.

4797 Read the next 4 bits as unsigned binary integer *m*, then convert to a base 10 value MM, padding to
 4798 the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9.

4799 Read the next 5 bits as unsigned binary integer *d*, then convert to a base 10 value DD, padding to
 4800 the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9.

4801 Check that MM is within the range 01-12 and that DD is within the range 00-31 and does not exceed
 4802 the number of days in the month for the month indicated by MM. Otherwise decoding fails.

4803 Concatenate YY MM and DD in sequence as the output value YYMMDD and append this to the output
 4804 string buffer.

4805 If the initial bit of the binary input string was set to '1', ensure that the procedure above has been
 4806 performed twice, for both the start date and the end date, both formatted as YYMMDD.

4807 The output string buffer should now consist of either a 6-digit numeric string representing a date
 4808 formatted as YYMMDD or a 12-digit numeric string representing a date range formatted as
 4809 YYMMDDYYMMDD. This is the output of this decoding method.

4810 **14.5.11 "Variable-precision date+time"**

4811 GS1 Application Identifier (8008) expresses a production date and time with a choice of three
 4812 formats that differ in the precision of the time value, either hours, hours and minutes or hours,
 4813 minutes and seconds, as shown in the figure below.

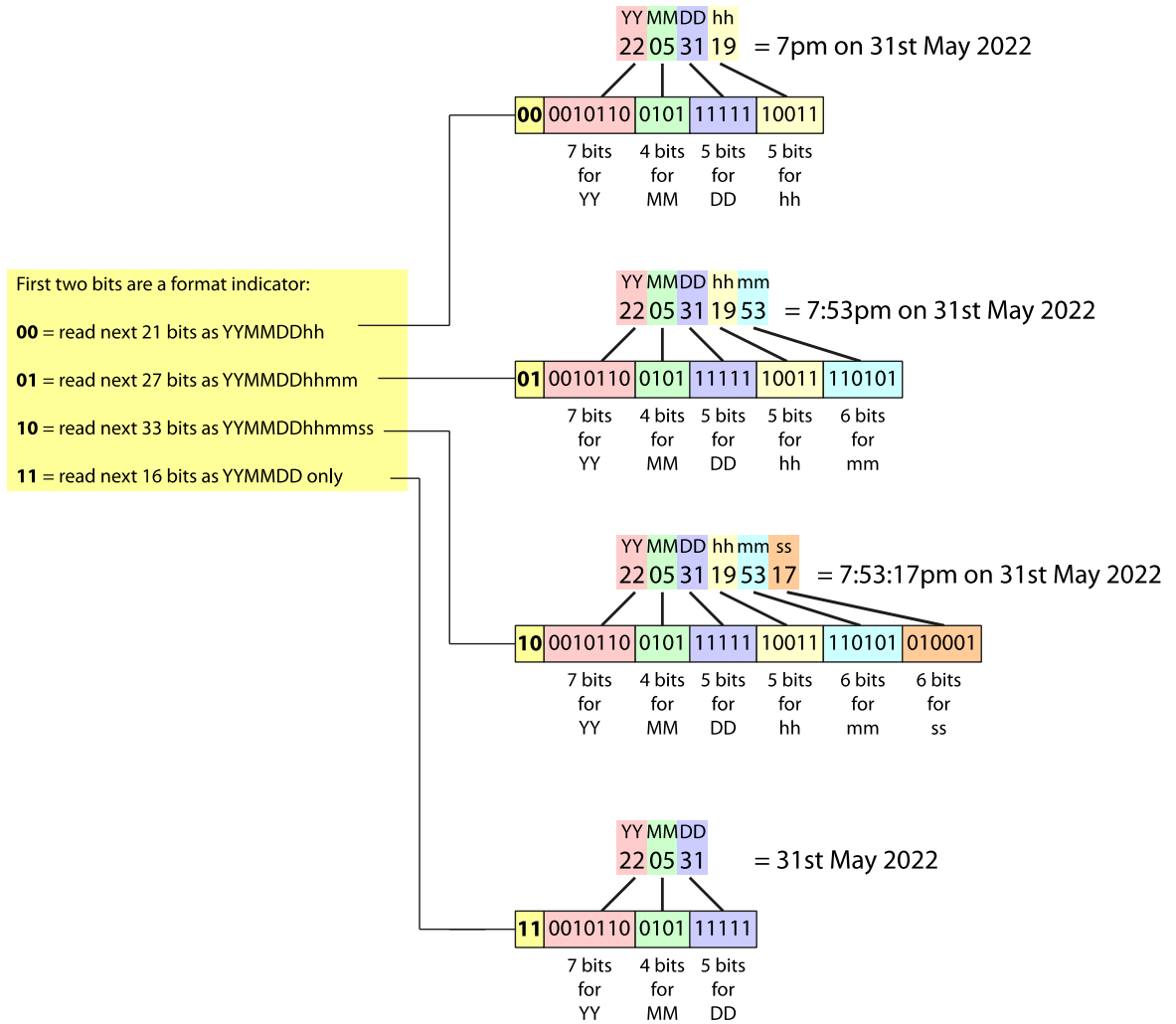
4814 GS1 Application Identifier (7011) expresses a test-by date, either as a date in YYMMDD format or as
 4815 a date-time that also expresses hours and minutes,

4816 A numeric string representing a date formatted as YYMMDD can be encoded in 16 bits.
 4817 A numeric string representing a date+hours formatted as YYMMDDhh can be encoded in 21 bits.
 4818 A numeric string representing a date+hours+minutes formatted as YYMMDDhhmm can be encoded
 4819 in 27 bits.
 4820 A numeric string representing a date+hours+minutes+seconds formatted as YYMMDDhhmmss can
 4821 be encoded in 33 bits.

4822 To distinguish between these four alternatives, the binary encoding begins with a two-bit format
 4823 indicator whose value is '00' for YYMMDDhh, '01' for YYMMDDhhmm, '10' for YYMMDDhhmmss or
 4824 '11' for YYMMDD.

4825

Figure 14-12 Encoding of "Variable-precision date+time"



4826

4827 **14.5.11.1 Encoding**

4828 **Input:**

4829 The input to the encoding method is either a 6-digit numeric string representing a date in the format
 4830 YYMMDD, a 8-digit numeric string representing a date+time value in the format YYMMDDhh, a 10-
 4831 digit numeric string representing a date+time value in the format YYMMDDhhmm or a 12-digit
 4832 numeric string representing a date+time value in the format YYMMDDhhmmss, as expected in the
 4833 GS1 General Specifications.

4834 **Validity Test:**

4835 The numeric string must only consist of digits 0-9 and is further constrained to be a plausible date
 4836 or date+time value, meaning that the third and fourth digits are always in the range 01-12 and the
 4837 fifth and sixth digits are always in the range 00-31 and do not indicate a day-of-month value that is
 4838 greater than the number of days in the month indicated by the third and fourth Digits. e.g. if the
 4839 third and fourth digits are "09" then a value of "31" for the fifth and sixth digits would be invalid
 4840 because September can only contain 30 days. The seventh and eighth digits (if present) must be in
 4841 the range 00-24, while the ninth and tenth digits (if present) must be in the range 00-59 and the
 4842 eleventh and twelfth digits (if present) must also be in the range 00-59.

4843

Output:

4844

Create an empty binary string buffer to receive the output.

4845

If the input string was a 6-digit numeric string formatted as YYMMDD, append '11' to the binary string buffer. If the input string was a 8-digit numeric string formatted as YYMMDDhh, append '00' to the binary string buffer. If the input string was 10-digit numeric string formatted as YYMMDDhhmm, append '01' to the binary string buffer. If the input string was 12-digit numeric string formatted as YYMMDDhhmmss, append '10' to the binary string buffer.

4846

4847

4848

4849

4850

Consider the input string as pairs of digits in which the first two digits are YY, the next two digits are MM, followed by two digits DD, then (if present) a further two digits hh and (if present) two digits mm and (if present) two digits ss.

4851

4852

4853

Convert YY to a decimal integer (e.g. '22' → 22) and convert this to an unsigned binary value, then if the resulting binary string for YY is less than seven bits in length, pad to the left with bits set to '0' to reach a total of seven bits. Append these seven bits to the binary string buffer.

4854

4855

4856

Convert MM to a decimal integer (e.g. '05' → 5) and convert this to an unsigned binary value, then if the resulting binary string for MM is less than four bits in length, pad to the left with bits set to '0' to reach a total of four bits. Append these four bits to the binary string buffer.

4857

4858

4859

Convert DD to a decimal integer (e.g. '31' → 31) and convert this to an unsigned binary value, then if the resulting binary string for DD is less than five bits in length, pad to the left with bits set to '0' to reach a total of five bits. Append these five bits to the binary string buffer.

4860

4861

4862

If present, convert hh to a decimal integer (e.g. '07' → 7) and convert this to an unsigned binary value, then if the resulting binary string for hh is less than five bits in length, pad to the left with bits set to '0' to reach a total of five bits. Append these five bits to the binary string buffer.

4863

4864

4865

If present, convert mm to a decimal integer (e.g. '59' → 59) and convert this to an unsigned binary value, then if the resulting binary string for mm is less than six bits in length, pad to the left with bits set to '0' to reach a total of six bits. Append these six bits to the binary string buffer.

4866

4867

4868

If present, convert ss to a decimal integer (e.g. '59' → 59) and convert this to an unsigned binary value, then if the resulting binary string for ss is less than six bits in length, pad to the left with bits set to '0' to reach a total of six bits. Append these six bits to the binary string buffer.

4869

4870

4871

The binary string buffer should now consist of a total of either 18 bits (for a 6-digit input YYMMDD) or 23 bits (for an 8-digit input YYMMDDhh) or 29 bits (for a 10-digit input YYMMDDhhmm) or 35 bits (for a 12-digit input YYMMDDhhmmss) and should be considered as the output of this encoding method.

4872

4873

4874

4875

14.5.11.2 Decoding

4876

Input:

4877

The input to the decoding method is a binary string of either 18, 23, 29 or 35 bits.

4878

Validity Test:

4879

The leftmost two bits are a date+time format indicator. As shown in Figure 14-12, the value of these two bits determine how many further bits should be read and how they should be interpreted.

4880

4881

In all situations, the next 16 bits will be decoded as a 6-digit numeric string representing a date formatted as YYMMDD, using 7 bit for YY, followed by 4 bits for MM, then 5 bits for DD. If the initial two bits for the date+time format indicator have a value other than '11', further groups of bits shall be read and interpreted as follows, in sequence: 5 bits for hh, 6 bits for mm and 6 bits for ss.

4882

4883

4884

4885

After decoding the initial 16 bits after the two-bit indicator, the third and fourth digits must always be in the range 01-12 for MM and the fifth and sixth digits must always be in the range 00-31 for DD and must not indicate a day-of-month value that is greater than the number of days in the month indicated by the third and fourth digits. e.g. if the third and fourth digits are "09" then a value of "31" for the fifth and sixth digits would be invalid because September can only contain 30 days. The seventh and eighth digits (if present) must be in the range 00-24 for hh, while the ninth and tenth digits (if present) must be in the range 00-59 for mm and the eleventh and twelfth digits (if present) must also be in the range 00-59 for ss.

4886

4887

4888

4889

4890

4891

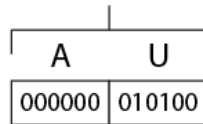
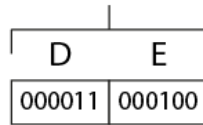
4892

| | |
|------|---|
| 4893 | Output: |
| 4894 | Create an empty string buffer to receive the output value. |
| 4895 | Read the leftmost two bits of the binary input string and move the cursor beyond those initial two bits. If the value is '00', the next 21 bits will be decoded to an 8-digit numeric string YYMMDDhh. |
| 4896 | If the value is '01', the next 27 bits will be decoded to a 10-digit numeric string YYMMDDhhmm. |
| 4897 | If the value is '10', the next 33 bits will be decoded to a 12-digit numeric string YYMMDDhhmmss. |
| 4898 | If the value is '11', the next 16 bits will be decoded to a 6-digit numeric string YYMMDD. |
| 4899 | |
| 4900 | Working from left to right, read the first 7 bits as unsigned binary integer <i>y</i> , then convert to a base 10 value YY, padding to the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9. |
| 4901 | |
| 4902 | |
| 4903 | Read the next 4 bits as unsigned binary integer <i>m</i> , then convert to a base 10 value MM, padding to the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9. |
| 4904 | |
| 4905 | If present, read the next 5 bits as unsigned binary integer <i>d</i> , then convert to a base 10 value DD, padding to the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9. |
| 4906 | |
| 4907 | |
| 4908 | If present, read the next 5 bits as unsigned binary integer <i>h</i> , then convert to a base 10 value hh, padding to the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9. |
| 4909 | |
| 4910 | |
| 4911 | If present, read the next 6 bits as unsigned binary integer <i>n</i> , then convert to a base 10 value mm, padding to the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9. |
| 4912 | |
| 4913 | |
| 4914 | If present, read the next 6 bits as unsigned binary integer <i>s</i> , then convert to a base 10 value ss, padding to the left with a single '0' digit if the initial result after conversion to base 10 was in the range 0-9. |
| 4915 | |
| 4916 | |
| 4917 | Check that MM is within the range 01-12 and that DD is within the range 00-31 and does not exceed the number of days in the month for the month indicated by MM. Otherwise decoding fails. |
| 4918 | |
| 4919 | Check that hh (if present) is within the range 00-24 and that mm (if present) is within the range 00-59 and that ss (if present) is also within the range 00-59. If hh is '24' then both mm and ss (if present) must be '00', otherwise decoding fails. |
| 4920 | |
| 4921 | |
| 4922 | If the initial two-bit date indicator was '00', concatenate YY MM DD hh in sequence as the output value YYMMDDhh. |
| 4923 | |
| 4924 | If the initial two-bit date indicator was '01', concatenate YY MM DD hh mm in sequence as the output value YYMMDDhhmm. |
| 4925 | |
| 4926 | If the initial two-bit date indicator was '10', concatenate YY MM DD hh mm ss in sequence as the output value YYMMDDhhmmss. |
| 4927 | |
| 4928 | If the initial two-bit date indicator was '11', concatenate YY MM DD in sequence as the output value YYMMDD. |
| 4929 | |
| 4930 | |
| 4931 | 14.5.12 "Country code (ISO 3166-1 alpha-2)" |
| 4932 | The Country code (ISO 3166-1 alpha-2) encoding method is used to encode two-letter strings of upper case letters A-Z using 6 bits per character, using the file-safe URI-safe base64 alphabet for the binary encoding of each letter. |
| 4933 | |
| 4934 | |

4935

Figure 14-13 ISO 3166-1 alpha-2 country code encoded as file-safe URI base 64

Two letters
(encoded as file-safe URI-safe base 64)



4936

4937

4938

Table 14-10 Encoding table for "Country code (ISO 3166-1 alpha-2)"

| Character | 6-bit binary string |
|-----------|---------------------|
| A | 000000 |
| B | 000001 |
| C | 000010 |
| D | 000011 |
| E | 000100 |
| F | 000101 |
| G | 000110 |
| H | 000111 |
| I | 001000 |
| J | 001001 |
| K | 001010 |
| L | 001011 |
| M | 001100 |

| Character | 6-bit binary string |
|-----------|---------------------|
| N | 001101 |
| O | 001110 |
| P | 001111 |
| Q | 010000 |
| R | 010001 |
| S | 010010 |
| T | 010011 |
| U | 010100 |
| V | 010101 |
| W | 010110 |
| X | 010111 |
| Y | 011000 |
| Z | 011001 |

4939

14.5.12.1 Encoding

4940

Input:

4941

The input to the encoding method is a string of two upper case letters A-Z.

4942

Validity Test:

4943

4944

If the input string contains characters other than upper case letters A-Z or is not exactly two characters in length, encoding fails.

4945

Output:

4946

Create an empty binary string buffer to receive the output.

4947

4948

Lookup the first character in the table above and append the corresponding set of six bits to the binary string buffer.

4949 Lookup the second character in the table above and append the corresponding set of six bits to the
4950 binary string buffer.
4951 The contents of the binary string buffer is now the binary output of this encoding method.

4952 **14.5.12.2 Decoding**

4953 **Input:**

4954 The input to the encoding method is a binary string of 12 bits.

4955 **Validity Test:**

4956 If the output string contains characters other than upper case letters A-Z, decoding fails.

4957 **Output:**

4958 Create an empty string buffer to receive the output.

4959 Read the first six bits from the binary input string. Lookup the six bits in the table above and
4960 append the corresponding character to the output string buffer.

4961 Read the next (final) six bits from the binary input string. Lookup the six bits in the table above and
4962 append the corresponding character to the output string buffer.

4963 The contents of the output string buffer is the output of this decoding method.

4964 **14.5.13 "Variable-length integer without encoding indicator"**

4965 The 'Variable-length Integer without encoding indicator' encoding method is used to encode
4966 variable-length numeric strings as unsigned binary integers using the minimum number of bits.

4967 It is very similar to the method ["Variable-length integer" \(§14.5.6.1\)](#) option within ["Variable-length
4968 alphanumeric" \(§14.5.6\)](#) but is used in situations where the value is defined within the GS1 General
4969 Specifications to be strictly numeric rather than alphanumeric, so no encoding indicator is used
4970 within this method.

4971 It preserves leading zeros, since the decoding method is required to left-pad the decoded integer to
4972 the number of digits indicated by the length indicator that was encoded. This method requires
4973 knowledge of L , the length of the string to be encoded, as well as L_{\max} , the maximum permitted
4974 length for such a string.

4975 Note: this is also similar to the ["Fixed-Bit-Length Integer" method \(§14.5.2\)](#) except that the length
4976 is not fixed and the binary value is appended after an appropriate length indicator (but no encoding
4977 indicator).

4978 **14.5.13.1 Encoding**

4979 **Input:**

4980 The input to the encoding method is a numeric string of length L consisting only of digits 0-9.

4981 **Validity Test:**

4982 If the input string contains characters other than digits 0-9 or length $L > L_{\max}$, encoding fails.

4983 **Output:**

4984 Create an empty binary string buffer to receive the output.

4985 Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

4986 Convert the actual length L from a base 10 integer to a binary value, then if necessary, pad to the
4987 left with bits of '0' to reach a total length b_{LI} for the binary string representing the length indicator.

4988 If $L_{\max} = 1$, the binary string representing the length indicator is empty, of zero length.

4989 Append the binary string representing the length indicator to the binary string buffer.

4990 Convert the input string of L digits 0-9 to a base10 integer then convert this to an unsigned binary
4991 integer, v.

4992 Calculate b_v , the number of bits for expressing the value either via a lookup of L in table B and
4993 reading the value in the column titled 'Integer encoding' or using the following formula:
4994
4995 $b_v = \text{ceiling}(L \cdot \log(10) / \log(2))$

4996 If necessary, pad the binary string v with bits of '0' to reach a total length b_v for the binary string
4997 representing the numeric string value.

4998 After any necessary padding, append binary string v (of length b_v) to the binary string buffer.

4999 The contents of the binary string buffer is now the binary output of this encoding method.

5000 **14.5.13.2 Decoding**

5001 **Input:**

5002 The input to the decoding method is a binary string.

5003 **Validity Test:**

5004 If the output string contains characters other than digits 0-9 or if length $L > L_{\text{max}}$, decoding fails.

5005 **Output:**

5006 Create an empty binary string buffer to receive the output.

5007 Lookup b_{LI} , the number of bits for expressing the length indicator in Table F.

5008 Read the next b_{LI} bits of the binary input string as the length indicator and convert this binary value
5009 to an unsigned base 10 integer L, the number of characters that are encoded. Within the binary
5010 input string, move the cursor past the b_{LI} length indicator bits to begin decoding the actual value.

5011 Calculate b_v , the number of bits for expressing the value either via a lookup of L in table B and
5012 reading the value in the column titled 'Integer encoding' or using the following formula:
5013

5014 $b_v = \text{ceiling}(L \cdot \log(10) / \log(2))$

5015 Read the next b_v bits from the binary string and convert this to an unsigned base 10 integer V.

5016 Convert V to a numeric string. If V is fewer than L digits in length, left-pad V with digits of '0' to
5017 reach a total of L digits. The resulting L-digit numeric string value V (with any necessary left-
5018 padding) is the output of this decoding method.

5019 **14.5.14 "Optional minus sign in 1 bit"**

5020 GS1 Application Identifiers (4330), (4331), (4332), (4333) express a 6 digit value for
5021 maximum/minimum temperature in hundredths of degrees Celsius or Fahrenheit and use an
5022 optional trailing minus sign to indicate if the temperature is negative.

5023 To support efficient encoding of the optional trailing minus sign, this method uses a single bit value
5024 in which '0' indicates an empty string (used for positive temperature values in the Celsius and
5025 Fahrenheit scales), whereas '1' indicates the presence of a trailing minus sign (used for negative
5026 temperature values in the Celsius and Fahrenheit scales).

5027 **14.5.14.1 Encoding**

5028 **Input:**

5029 The input to the encoding method is a string, either the empty string "" or a single minus/hyphen
5030 character "-". The empty string will be mapped to a single bit with value 0. The single
5031 minus/hyphen character will be mapped to a single bit with value 1

5032

Validity Test:

5033

The input must consist of either the empty string "" or a single minus/hyphen character "-"

5034

Output:

5035

The output is a single bit, 0 or 1.

5036

If the input is the empty string "", the output shall be a single bit set to value 0.

5037

If the input is a single minus/hyphen character "-", the output shall be a single bit set to value 1.

5038

14.5.14.2 Decoding

5039

Input:

5040

The input to the encoding method is a single bit, 0 or 1.

5041

Validity Test:

5042

The input must consist of exactly one bit, otherwise the encoding fails.

5043

Output:

5044

If the single bit is 0, it is decoded as an empty string "".

5045

If the single bit is 1, it is decoded as a single minus/hyphen character "-".

5046

5047

14.6 EPC Binary coding tables

5048

This section specifies coding tables for use with the encoding procedure of Section [14.3](#) and the decoding procedure of Section [14.3.4](#).

5049

5050

For EPC schemes defined before TDS 2.0, the "Bit Position" row of each coding table illustrates the relative bit positions of segments within each binary encoding. Before TDS 2.0, the "Bit Position" row only took a 'counting down' approach, in which the highest subscript indicates the most significant bit, and subscript 0 indicates the least significant bit. Note that this is opposite to the way RFID tag memory bank bit addresses are normally indicated, where address 0 is the most significant bit. In TDS 2.0, for the older EPC schemes, two "Bit Position" rows are shown, one taking the previous 'counting down' approach, from most significant bit to least significant bit, with the bit count decreasing from left to right, as well as separate row using the 'counting up' approach, in which b_0 is the left-most bit and b_0-b_7 always correspond to the EPC header bits, with the bit count increasing from left to right.

5051

5052

5053

5054

5055

5056

5057

5058

5059

5060

For new EPC schemes defined in TDS 2.0 (those whose name ends with '+', e.g. SGTIN+), because many of these involve variable-length components and multiple alternative encodings and the possibility of additional +AIDC data appended after the EPC, the "Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right, in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits. Note that this 'counting up' approach is different from the 'counting down' approach taken for the older EPC schemes because the total bit count for most of the new EPC schemes is variable, typically depending on the length and character set used in the actual value being encoded for the serial component, so for most of the new EPC schemes introduced in TDS 2.0, 'counting down' from the most significant bit at the left to least significant bit at the right cannot even provide a consistent formula or expression for the numbering the bits that correspond to the header, +AIDC toggle bit, filter bit or primary GS1 identification key.

5061

5062

5063

5064

5065

5066

5067

5068

5069

5070

5071

5072

14.6.1 Serialised Global Trade Item Number (SGTIN)

5073

Two coding schemes for the SGTIN are specified, a 96-bit encoding (SGTIN-96) and a 198-bit encoding (SGTIN-198). The SGTIN-198 encoding allows for the full range of serial numbers up to 20 alphanumeric characters as specified in [GS1GS]. The SGTIN-96 encoding allows for numeric-only

5074

5075

5076 serial numbers, without leading zeros, whose value is less than 2^{38} (that is, from 0 through
 5077 274,877,906,943, inclusive).

5078 Both SGTIN coding schemes make reference to the following partition table.

5079 **Table 14-11** SGTIN Partition Table

| Partition Value (<i>P</i>) | GS1 Company Prefix | | Indicator/Pad Digit and Item Reference | |
|------------------------------|--------------------|---------------------|--|--------|
| | Bits (<i>M</i>) | Digits (<i>L</i>) | Bits (<i>N</i>) | Digits |
| 0 | 40 | 12 | 4 | 1 |
| 1 | 37 | 11 | 7 | 2 |
| 2 | 34 | 10 | 10 | 3 |
| 3 | 30 | 9 | 14 | 4 |
| 4 | 27 | 8 | 17 | 5 |
| 5 | 24 | 7 | 20 | 6 |
| 6 | 20 | 6 | 24 | 7 |

5080 **14.6.1.1 SGTIN-96 coding table**

5081 **Table 14-12** SGTIN-96 coding table

| Scheme | SGTIN-96 | | | | | |
|--|------------------------------|----------------------|-------------------------|------------------------|---------------------------------|--|
| URI Template | urn:epc:tag:sgtin-96:F.C.I.S | | | | | |
| Total Bits | 96 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix (*) | Indicator (**) / Item Reference | Serial |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 24-4 | 38 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 7-1 digits | up to 12 digits in range 0 – 274,877,906,943 without preservation of leading zeros |
| Coding Segment | EPC Header | Filter | GTIN | | | Serial |
| URI portion | | F | C . I | | | S |
| Coding Segment Bit Count | 8 | 3 | 47 | | | 38 |
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}b_{85}$ | $b_{84}b_{83}...b_{38}$ | | | $b_{37}b_{36}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{57}$ | | | $b_{58}b_{59}...b_{95}$ |

| Scheme | SGTIN-96 | | | |
|----------------------|----------|-------------------------------|---|-------------------------------|
| Coding Method | 00110000 | Integer §14.3.1 §14.4.1 | Partition Table 14-11 §14.3.3 §14.4.3 | Integer §14.3.1 §14.4.1 |

5082 (*) See Section [7.3.2](#) for the case of an SGTIN derived from a GTIN-8.

5083 (**) Note that in the case of an SGTIN derived from a GTIN-12 or GTIN-13, a zero pad digit takes
5084 the place of the Indicator Digit. In all cases, see Section [7.2.3](#) for the definition of how the Indicator
5085 Digit (or zero pad) and the Item Reference are combined into this segment of the EPC.

5086 **14.6.1.2 SGTIN-198 coding table**

5087 **Table 14-13** SGTIN-198 coding table

| Scheme | SGTIN-198 | | | | | |
|--|-------------------------------|-------------------------------|---|------------------------|--------------------------------|------------------------------|
| URI Template | urn:epc:tag:sgtin-198:F.C.I.S | | | | | |
| Total Bits | 198 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix (*) | Indicator (**)/ Item Reference | Serial |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 24-4 | 140 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 7-1 digits | up to 20 characters |
| Coding Segment | EPC Header | Filter | GTIN | | | Serial |
| URI portion | | F | C . I | | | S |
| Coding Segment Bit Count | 8 | 3 | 47 | | | 140 |
| Bit Position (counting down) | $b_{197}b_{196}...b_{190}$ | $b_{189}b_{188}b_{187}$ | $b_{186}b_{185}...b_{140}$ | | | $b_{139}b_{138}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{57}$ | | | $b_{58}b_{59}...b_{197}$ |
| Coding Method | 00110110 | Integer §14.3.1 §14.4.1 | Partition Table 14-11 §14.3.3 §14.4.3 | | | String §14.3.2 §14.4.2 |

5088 (*) See Section [7.3.2](#) for the case of an SGTIN derived from a GTIN-8.

5089 (**) Note that in the case of an SGTIN derived from a GTIN-12 or GTIN-13, a zero pad digit takes
5090 the place of the Indicator Digit. In all cases, see Section [7.2.3](#) for the definition of how the Indicator
5091 Digit (or zero pad) and the Item Reference are combined into this segment of the EPC.

5092 **14.6.1.3 SGTIN+**

5093 The **SGTIN+** coding scheme uses the following **coding** table.

5094

Table 14-5 SGTIN+ coding table

| Scheme | SGTIN+ | | | | |
|--|--|-------------------------------|----------------------------------|------------------------------|--|
| GS1 Digital Link URI syntax | https://id.gs1.org/01/{gtin}/21/{serial} | | | | |
| Total Bits | Up to 216 bits | | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | GTIN | Serial Number |
| Corresponding GS1 AI | | | | (01) | (21) |
| Logical Segment Bit Count | 8 | 1 | 3 | 56 | 3 bit encoding indicator + 5 bit length indicator + up to 140 bits |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | 14 digits | up to 20 characters |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...b_{67}$ | $b_{68}b_{69}b_{70}...$ |
| Coding Method | 11110111 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Fixed-Length Numeric §14.5.4 | Variable-length alphanumeric §14.5.6 |

5095
5096
5097

* Note that for the SGTIN+ and all other EPC schemes new to TDS 2.0, the "Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right, in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits.

5098 **14.6.1.4 DSGTIN+**

5099 The **DSGTIN+** coding scheme uses the following **coding** table.

5100

Table 14-6 DSGTIN+ coding table

| Scheme | DSGTIN+ | | | | | |
|--|--|------------------|---------------|--|-----------|--|
| GS1 Digital Link URI syntax | https://id.gs1.org/01/{gtin}/21/{serial} | | | | | |
| Total Bits | Up to 236 bits | | | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | Date | GTIN | Serial Number |
| Corresponding GS1 AI | | | | One of (11),(13),(15),(16),(17),(7006),(7007) as indicated | (01) | (21) |
| Logical Segment Bit Count | 8 | 1 | 3 | 4 bit date type indicator + 16 bit date value | 56 | 3 bit encoding indicator + 5 bit length indicator + up to 140 bits |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | date type indicator and 6-digit date YYMMDD | 14 digits | up to 20 characters |

| Scheme | DSGTIN+ | | | | | |
|---------------------------------------|-------------------|-------------------------------|----------------------------------|----------------------------------|------------------------------|--------------------------------------|
| Bit Position (counting up)* | $b_0b_1\dots b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}\dots b_{30}b_{31}$ | $b_{32}b_{33}\dots b_{87}$ | $b_{88}b_{89}b_{90}\dots$ |
| Coding Method | 11111011 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Prioritised Date §14.5.3 | Fixed-Length Numeric §14.5.4 | Variable-length alphanumeric §14.5.6 |

* Note that for the DSGTIN+ and all other EPC schemes new to TDS 2.0, the "Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right, in which b_0 is the left-most bit and b_0 - b_7 bits always correspond to the EPC header bits.

5101
5102
5103

14.6.2 Serial Shipping Container Code (SSCC)

Two coding schemes for the SSCC are specified:

- **SSCC-96** (TDS 1.x) is fixed at 96 bits length, is GCP-partitioned, and allows for the full range of SSCCs as specified in [GS1GS].
- **SSCC+** is fixed at 84 bits length, is not GCP-partitioned, and allows for simplified interoperability with the full range of SSCCs in their GS1 element string form, as specified in [GS1GS].

5104
5105
5106
5107
5108
5109
5110

14.6.2.1 SSCC-96

The **SSCC-96** coding scheme uses the following **partition** table.

Table 14-7 SSCC Partition Table

| Partition Value (P) | GS1 Company Prefix | | Extension Digit and Serial Reference | |
|------------------------|--------------------|---------------|--------------------------------------|--------|
| | Bits (M) | Digits (L) | Bits (N) | Digits |
| 0 | 40 | 12 | 18 | 5 |
| 1 | 37 | 11 | 21 | 6 |
| 2 | 34 | 10 | 24 | 7 |
| 3 | 30 | 9 | 28 | 8 |
| 4 | 27 | 8 | 31 | 9 |
| 5 | 24 | 7 | 34 | 10 |
| 6 | 20 | 6 | 38 | 11 |

The **SSCC-96** coding scheme uses the following **coding** table.

Table 14-8 SSCC-96 coding table

| Scheme | SSCC-96 | | | | | |
|------------------------|---------------------------|--------|-----------|--------------------|------------------------------|------------|
| URI Template | urn:epc:tag:sscc-96:F.C.S | | | | | |
| Total Bits | 96 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Extension / Serial Reference | (Reserved) |

5114
5115

| Scheme | SSCC-96 | | | | | |
|--|-------------------------|-------------------------------|--|-------------|-------------|-------------------------|
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 38-18 | 24 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 11-5 digits | |
| Coding Segment | EPC Header | Filter | SSCC | | | (Reserved) |
| URI portion | | F | C . S | | | |
| Coding Segment Bit Count | 8 | 3 | 61 | | | 24 |
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}b_{85}$ | $b_{84}b_{83}...b_{24}$ | | | $b_{23}b_{36}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{71}$ | | | $b_{72}b_{73}...b_{95}$ |
| Coding Method | 00110001 | Integer §14.3.1 §14.4.1 | Partition Table 14-7 §14.3.3 §14.4.3 | | | 00...0 (24 zero bits) |

5116 **14.6.2.2 SSCC+**

5117 The **SSCC+** coding scheme uses the following **coding** table.

5118 **Table 14-9** SSCC+ coding table

| Scheme | SSCC+ | | | |
|--|------------------------------|----------------------------------|-------------------------------------|---------------------------------|
| GS1 Digital Link URI syntax | https://id.gs1.org/00/{sscc} | | | |
| Total Bits | 84 | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | SSCC |
| Corresponding GS1 AI | | | | (00) |
| Logical Segment Bit Count | 8 | 1 | 3 | 72 |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | 18 digits |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...b_{83}$ |
| Coding Method | 11111001 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Fixed-Length Numeric §14.5.4 |

5119 * Note that for the SSCC+ and other other EPC schemes new to TDS 2.0, the **"Bit Position" row**
 5120 **of each new EPC coding table is shown only with a 'counting up' approach from left to**
 5121 **right**, in which b_0 is the left-most bit and b_0 - b_7 bits always correspond to the EPC header bits.

5122 **14.6.3 Global Location Number with or without Extension (SGLN)**

5123 Two coding schemes for the SGLN are specified, a 96-bit encoding (SGLN-96) and a 195-bit
 5124 encoding (SGLN-195). The SGLN-195 encoding allows for the full range of GLN extensions up to 20
 5125 alphanumeric characters as specified in [GS1GS]. The SGLN-96 encoding allows for numeric-only
 5126 GLN extensions, without leading zeros, whose value is less than 2^{41} (that is, from 0 through
 5127 2,199,023,255,551, inclusive). Note that an extension value of 0 is reserved to indicate that the
 5128 SGLN is equivalent to the GLN indicated by the GS1 Company Prefix and location reference; this
 5129 value is available in both the SGLN-96 and the SGLN-195 encodings.

5130 Both SGLN coding schemes make reference to the following partition table.

5131 **Table 14-10** SGLN Partition Table

| Partition Value (<i>P</i>) | GS1 Company Prefix | | Location Reference | |
|---------------------------------|----------------------|------------------------|----------------------|--------|
| | Bits (<i>M</i>) | Digits (<i>L</i>) | Bits (<i>N</i>) | Digits |
| 0 | 40 | 12 | 1 | 0 |
| 1 | 37 | 11 | 4 | 1 |
| 2 | 34 | 10 | 7 | 2 |
| 3 | 30 | 9 | 11 | 3 |
| 4 | 27 | 8 | 14 | 4 |
| 5 | 24 | 7 | 17 | 5 |
| 6 | 20 | 6 | 21 | 6 |

5132 **14.6.3.1 SGLN-96 coding table**

5133 **Table 14-11** SGLN-96 coding table

| Scheme | SGLN-96 | | | | | |
|--|-----------------------------|---------------|---------------|--------------------|--------------------|--|
| URI Template | urn:epc:tag:sgln-96:F.C.L.E | | | | | |
| Total Bits | 96 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Location Reference | Extension |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 21-1 | 41 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 6-0 digits | Up to 13 digits in range 0 – 2,199,023,255,551 without preservation of leading zeros |
| Coding Segment | EPC Header | Filter | GLN | | | Extension |
| URI portion | | F | C.L | | | E |
| Coding Segment Bit Count | 8 | 3 | 44 | | | 41 |

| Scheme | SGLN-96 | | | |
|--|-------------------------|---|---|---|
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}b_{85}$ | $b_{84}b_{83}...b_{41}$ | $b_{40}b_{39}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{54}$ | $b_{55}b_{56}...b_{95}$ |
| Coding Method | 00110010 | Integer §14.3.1 §14.4.1 | Partition Table 14-10 §14.3.3 §14.4.3 | Integer §14.3.1 §14.4.1 |

5134 **14.6.3.2 SGLN-195 coding table**

5135 **Table 14-12** SGLN-195 coding table

| Scheme | SGLN-195 | | | | | |
|--|------------------------------|---|---|--------------------|--------------------|--|
| URI Template | urn:epc:tag:sgln-195:F.C.L.E | | | | | |
| Total Bits | 195 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Location Reference | Extension |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 21-1 | 140 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 6-0 digits | up to 20 characters |
| Coding Segment | EPC Header | Filter | GLN | | | Extension |
| URI portion | | F | C.L | | | E |
| Coding Segment Bit Count | 8 | 3 | 44 | | | 140 |
| Bit Position (counting down) | $b_{194}b_{193}...b_{187}$ | $b_{186}b_{185}b_{184}$ | $b_{183}b_{182}...b_{140}$ | | | $b_{139}b_{138}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{54}$ | | | $b_{55}b_{56}...b_{194}$ |
| Coding Method | 00111001 | Integer §14.3.1 §14.4.1 | Partition Table 14-10 §14.3.3 §14.4.3 | | | String §14.3.2 §14.4.2 |

5136 **14.6.3.3 SGLN+**

5137 The **SGLN+** coding scheme uses the following **coding** table.

5138

Table 14-13 SGLN+ coding table

| Scheme | SGLN+ | | | | |
|--|---|---|--|--|--|
| GS1 Digital Link URI syntax | https://id.gs1.org/414/{gln}/254/{glnextension} | | | | |
| Total Bits | Up to 212 bits | | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | GLN | GLN Extension |
| Corresponding GS1 AI | | | | (414) | (254) |
| Logical Segment Bit Count | 8 | 1 | 3 | 52 | 3 bit encoding indicator + 5 bit length indicator + up to 140 bits for GLN Extension |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | 13 digits | up to 20 characters |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...b_{63}$ | $b_{64}b_{65}b_{66}...$ |
| Coding Method | 11110010 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Fixed-Length Numeric §14.5.4 | Variable-length alphanumeric §14.5.6 |

5139
5140
5141

* Note that for the SGLN+ and other other EPC schemes new to TDS 2.0, the **"Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right**, in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits.

5142 **14.6.4 Global Returnable Asset Identifier (GRAI)**

5143 Two coding schemes for the GRAI are specified, a 96-bit encoding (GRAI-96) and a 170-bit encoding
5144 (GRAI-170). The GRAI-170 encoding allows for the full range of serial numbers up to 16
5145 alphanumeric characters as specified in [GS1GS]. The GRAI-96 encoding allows for numeric-only
5146 serial numbers, without leading zeros, whose value is less than 2^{38} (that is, from 0 through
5147 274,877,906,943, inclusive).

5148 Only GRAIs that include the optional serial number may be represented as EPCs. A GRAI without a
5149 serial number represents an asset class, rather than a specific instance, and therefore may not be
5150 used as an EPC (just as a non-serialised GTIN may not be used as an EPC).

5151 Both GRAI coding schemes make reference to the following partition table.

5152 **Table 14-14** GRAI Partition Table

| Partition Value (P) | Company Prefix | | Asset Type | |
|---------------------|----------------|------------|------------|--------|
| | Bits (M) | Digits (L) | Bits (N) | Digits |
| 0 | 40 | 12 | 4 | 0 |
| 1 | 37 | 11 | 7 | 1 |
| 2 | 34 | 10 | 10 | 2 |
| 3 | 30 | 9 | 14 | 3 |

| Partition Value (P) | Company Prefix | | Asset Type | |
|---------------------|----------------|---|------------|---|
| 4 | 27 | 8 | 17 | 4 |
| 5 | 24 | 7 | 20 | 5 |
| 6 | 20 | 6 | 24 | 6 |

5153 **14.6.4.1 GRAI-96 coding table**

5154 **Table 14-15** GRAI-96 coding table

| Scheme | GRAI-96 | | | | | |
|--|-----------------------------|-------------------------------|---|--------------------|------------|--|
| URI Template | urn:epc:tag:grai-96:F.C.A.S | | | | | |
| Total Bits | 96 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Asset Type | Serial |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 24-4 | 38 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digit | 6-0 digits | Up to 12 digits in range 0 – 274,877,906,943 without preservation of leading zeros |
| Coding Segment | EPC Header | Filter | Partition + Company Prefix + Asset Type | | | Serial |
| URI portion | | F | C . A | | | S |
| Coding Segment Bit Count | 8 | 3 | 47 | | | 38 |
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}b_{85}$ | $b_{84}b_{83}...b_{38}$ | | | $b_{37}b_{36}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{57}$ | | | $b_{58}b_{59}...b_{95}$ |
| Coding Method | 00110011 | Integer §14.3.1 §14.4.1 | Partition Table 14-14 §14.3.3 §14.4.3 | | | Integer §14.3.1 §14.4.1 |

5155 **14.6.4.2 GRAI-170 coding table**

5156 **Table 14-15** GRAI-170 coding table

| Scheme | GRAI-170 |
|---------------------|------------------------------|
| URI Template | urn:epc:tag:grai-170:F.C.A.S |
| Total Bits | 170 |

| Scheme GRAI-170 | | | | | | |
|--|----------------------------|-------------------------------|---|--------------------|------------------------------|---------------------|
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Asset Type | Serial |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 24-4 | 112 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 6-0 digits | Up to 16 characters |
| Coding Segment | EPC Header | Filter | Partition + Company Prefix + Asset Type | | Serial | |
| URI portion | | F | C . A | | S | |
| Coding Segment Bit Count | 8 | 3 | 47 | | 112 | |
| Bit Position (counting down) | $b_{169}b_{168}...b_{162}$ | $b_{161}b_{160}b_{159}$ | $b_{158}b_{157}...b_{112}$ | | $b_{111}b_{110}...b_0$ | |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{57}$ | | $b_{58}b_{59}...b_{169}$ | |
| Coding Method | 00110111 | Integer §14.3.1 §14.4.1 | Partition Table 14-14 §14.3.3 §14.4.3 | | String §14.3.2 §14.4.2 | |

5157 **14.6.4.3 GRAI+**

5158 The **GRAI+** coding scheme uses the following **coding** table.

5159 **Table 14-16** GRAI+ coding table

| Scheme GRAI+ | | | | | |
|--|--------------------------------|------------------|-------------------|------------------------------------|--|
| GS1 Digital Link URI syntax | https://id.gs1.org/8003/{grai} | | | | |
| Total Bits | Up to 188 bits | | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | Leading pad '0' then 13-digit GRAI | GRAI Serial Component |
| Corresponding GS1 AI | | | | (8003) | |
| Logical Segment Bit Count | 8 | 1 | 3 | 56 | 3 bit encoding indicator + 5 bit length indicator + up to 112 bits |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | 14 digits | Up to 16 characters |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...b_{67}$ | $b_{68}b_{69}b_{70}...$ |

| Scheme | GRAI+ | | | | |
|----------------------|----------|----------------------------------|-------------------------------------|---------------------------------|---|
| Coding Method | 11110001 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Fixed-Length Numeric §14.5.4 | Variable-length alphanumeric §14.5.6 |

5160
5161
5162

* Note that for the GRAI+ and other other EPC schemes new to TDS 2.0, the **"Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right**, in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits.

5163

14.6.5 Global Individual Asset Identifier (GIAI)

5164
5165
5166
5167
5168

Two coding schemes for the GIAI are specified, a 96-bit encoding (GIAI-96) and a 202-bit encoding (GIAI-202). The GIAI-202 encoding allows for the full range of serial numbers up to 24 alphanumeric characters as specified in [GS1GS]. The GIAI-96 encoding allows for numeric-only serial numbers, without leading zeros, whose value is, up to a limit that varies with the length of the GS1 Company Prefix.

5169
5170

Each GIAI coding schemes make reference to a different partition table, specified alongside the corresponding coding table in the subsections below.

5171

14.6.5.1 GIAI-96 Partition Table and coding table

5172

The GIAI-96 coding scheme makes use of the following partition table.

5173

Table 14-17 GIAI-96 Partition Table

| Partition Value (<i>P</i>) | Company Prefix | | Individual Asset Reference | |
|------------------------------|-------------------|---------------------|----------------------------|-------------------------|
| | Bits (<i>M</i>) | Digits (<i>L</i>) | Bits (<i>N</i>) | Max Digits (<i>K</i>) |
| 0 | 40 | 12 | 42 | 13 |
| 1 | 37 | 11 | 45 | 14 |
| 2 | 34 | 10 | 48 | 15 |
| 3 | 30 | 9 | 52 | 16 |
| 4 | 27 | 8 | 55 | 17 |
| 5 | 24 | 7 | 58 | 18 |
| 6 | 20 | 6 | 62 | 19 |

5174

Table 14-18 GIAI-96 coding table

| Scheme | GIAI-96 | | | | |
|--|---------------------------|---------------|---------------|--------------------|--|
| URI Template | urn:epc:tag:giai-96:F.C.A | | | | |
| Total Bits | 96 | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Individual Asset Reference |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 62-42 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 19-13 digits without preservation of leading zeros |
| Coding Segment | EPC Header | Filter | GIAI | | |

| Scheme | GIAI-96 | | |
|--|-------------------------|-------------------------------|--|
| URI portion | | F | C . A |
| Coding Segment Bit Count | 8 | 3 | 85 |
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}b_{85}$ | $b_{84}b_{83}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{95}$ |
| Coding Method | 00110100 | Integer §14.3.1 §14.4.1 | Unpadded Partition Table 14-17 §14.3.4 §14.4.4 |

5175 **14.6.5.2 GIAI-202 Partition Table and coding table**

5176 The GIAI-202 coding scheme makes use of the following partition table.

5177 **Table 14-20** GIAI-202 Partition Table

| Partition Value (<i>P</i>) | Company Prefix | | Individual Asset Reference | |
|---------------------------------|----------------------|------------------------|----------------------------|-----------------------|
| | Bits (<i>M</i>) | Digits (<i>L</i>) | Bits (<i>N</i>) | Maximum Characters |
| 0 | 40 | 12 | 148 | 18 |
| 1 | 37 | 11 | 151 | 19 |
| 2 | 34 | 10 | 154 | 20 |
| 3 | 30 | 9 | 158 | 21 |
| 4 | 27 | 8 | 161 | 22 |
| 5 | 24 | 7 | 164 | 23 |
| 6 | 20 | 6 | 168 | 24 |

5178 **Table 14-21** GIAI-202 coding table

| Scheme | GIAI-202 | | | | |
|--|----------------------------|------------------|------------------|--------------------|----------------------------|
| URI Template | urn:epc:tag:giai-202:F.C.A | | | | |
| Total Bits | 202 | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Individual Asset Reference |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 168-148 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 24-18 characters |
| Coding Segment | EPC Header | Filter | GIAI | | |
| URI portion | | F | C . A | | |
| Coding Segment Bit Count | 8 | 3 | 191 | | |

| Scheme | GIAI-202 | | |
|--|----------------------------|---|--|
| Bit Position (counting down) | $b_{201}b_{200}...b_{194}$ | $b_{193}b_{192}b_{191}$ | $b_{190}b_{189}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{201}$ |
| Coding Method | 00111000 | Integer §14.3.1 §14.4.1 | String Partition Table 14-20 §14.3.5 §14.4.5 |

5179 **14.6.5.3 GIAI+ Coding table**

5180 The GIAI+ coding scheme makes use of the following coding table.

5181 **Table 14-22** GIAI+ coding table

| Scheme | GIAI+ | | | |
|--|--|---|--|--|
| GS1 Digital Link URI syntax | https://id.gs1.org/8004/{gai} | | | |
| Total Bits | Up to 222 bits (assuming shortest initial all-numeric sequence to be 4 digits) | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | GIAI |
| Corresponding GS1 AI | | | | (8004) |
| Logical Segment Bit Count | 8 | 1 | 3 | 4n (for initial n digits) + 4 bit terminator OR 4n (for initial n digits) + 4 bit delimiter + 3 bit encoding indicator + 5 bit length indicator + up to (210-7n) bits |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | Up to 30 characters |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...$ |
| Coding Method | 11111010 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit- Length Integer §14.5.2 | Delimited/terminated Numeric (§14.5.5) (followed by Variable-length alphanumeric (§14.5.6) for any characters after the initial n digits) |

5182 * Note that for the GIAI+ and other other EPC schemes new to TDS 2.0, the **"Bit Position" row of**
 5183 **each new EPC coding table is shown only with a 'counting up' approach from left to right,**
 5184 in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits.

5185 **14.6.6 Global Service Relation Number - Recipient (GSRN)**

5186 Two encoding schemes for the GSRN are specified:

- 5187 • **GSRN-96** (TDS 1.x) is fixed at 96 bits length, is GCP-partitioned, and allows for the full
 5188 range of "Recipient" GSRNs corresponding to AI (8018), as specified in [GS1GS].
- 5189 • **GSRN+** is fixed at 84 bits length, is not GCP-partitioned, and allows for simplified
 5190 interoperability with the full range of "Recipient" GSRNs corresponding to AI (8018), in their
 5191 GS1 element string form, as specified in [GS1GS].

5192 **14.6.6.1 GSRN-96**

 5193 The **GSRN-96** coding scheme uses the following **partition** table.

 5194 **Table 14-23** GSRN Partition Table

| Partition Value (<i>P</i>) | Company Prefix | | Service Reference | |
|---------------------------------|----------------------|------------------------|----------------------|--------|
| | Bits (<i>M</i>) | Digits (<i>L</i>) | Bits (<i>N</i>) | Digits |
| 0 | 40 | 12 | 18 | 5 |
| 1 | 37 | 11 | 21 | 6 |
| 2 | 34 | 10 | 24 | 7 |
| 3 | 30 | 9 | 28 | 8 |
| 4 | 27 | 8 | 31 | 9 |
| 5 | 24 | 7 | 34 | 10 |
| 6 | 20 | 6 | 38 | 11 |

 5195 The **GSRN-96** coding scheme uses the following **coding** table.

 5196 **Table 14-24** GSRN-96 coding table

| Scheme | GSRN-96 | | | | | |
|--|----------------------------|-------------------------------|---|--------------------|-------------------|--------------------------|
| URI Template | urn:epc:tag:gsrcn-96:F.C.S | | | | | |
| Total Bits | 96 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Service Reference | (Reserved) |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 38-18 | 24 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 11-5 digits | |
| Coding Segment | EPC Header | Filter | GSRN | | | (Reserved) |
| URI portion | | F | C.S | | | |
| Coding Segment Bit Count | 8 | 3 | 61 | | | 24 |
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}b_{85}$ | $b_{84}b_{83}...b_{24}$ | | | $b_{23}b_{22}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{71}$ | | | $b_{72}b_{73}...b_{95}$ |
| Coding Method | 00101101 | Integer §14.3.1 §14.4.1 | Partition Table 14-23 §14.3.3 §14.4.3 | | | 00...0 (24 zero bits) |

 5197 **14.6.6.2 GSRN+**

 5198 The **GSRN+** coding scheme uses the following **coding** table.

5199

Table 14-25 GSRN+ coding table

| Scheme | GSRN+ | | | |
|--|--------------------------------|-------------------------------|----------------------------------|------------------------------|
| GS1 Digital Link URI syntax | https://id.gs1.org/8018/{gsrn} | | | |
| Total Bits | 84 | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | GSRN |
| Corresponding GS1 AI | | | | 8018 |
| Logical Segment Bit Count | 8 | 1 | 3 | 72 |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | 18 digits |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...b_{83}$ |
| Coding Method | 11110100 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Fixed-Length Numeric §14.5.4 |

5200
5201
5202

* Note that for the GSRN+ and other other EPC schemes new to TDS 2.0, the **"Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right**, in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits.

5203 **14.6.7 Global Service Relation Number - Provider (GSRNP)**

5204

Two encoding schemes for the GSRNP are specified:

5205
5206

- **GSRNP-96** (TDS 1.x) is fixed at 96 bits length, is GCP-partitioned, and allows for the full range of "Provider" GSRNs corresponding to AI (8017), as specified in [GS1GS].
- **GSRNP+** is fixed at 84 bits length, is not GCP-partitioned, and allows for simplified interoperability with the full range of "Provider" GSRNs corresponding to AI (8018), in their GS1 element string form, as specified in [GS1GS].

5207
5208
5209

5210 **14.6.7.1 GSRNP-96**

5211

The **GSRNP-96** coding scheme uses the following **partition** table.

5212

Table 14-26 GSRNP Partition Table

| Partition Value (P) | Company Prefix | | Service Reference | |
|---------------------|----------------|------------|-------------------|--------|
| | Bits (M) | Digits (L) | Bits (N) | Digits |
| 0 | 40 | 12 | 18 | 5 |
| 1 | 37 | 11 | 21 | 6 |
| 2 | 34 | 10 | 24 | 7 |
| 3 | 30 | 9 | 28 | 8 |
| 4 | 27 | 8 | 31 | 9 |
| 5 | 24 | 7 | 34 | 10 |

| Partition Value (P) | Company Prefix | | Service Reference | |
|---------------------|----------------|---|-------------------|----|
| 6 | 20 | 6 | 38 | 11 |

5213 The **GSRNP-96** coding scheme uses the following **coding** table.

5214 **Table 14-27** GSRNP-96 coding table

| Scheme | GSRNP-96 | | | | | |
|--|-----------------------------|-------------------------------|---|--------------------|-------------------|--------------------------|
| URI Template | urn:epc:tag:gsrcnp-96:F.C.S | | | | | |
| Total Bits | 96 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Service Reference | (Reserved) |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 38-18 | 24 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 11-5 digits | |
| Coding Segment | EPC Header | Filter | GSRN | | | (Reserved) |
| URI portion | | F | C.S | | | |
| Coding Segment Bit Count | 8 | 3 | 61 | | | 24 |
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}b_{85}$ | $b_{84}b_{83}...b_{24}$ | | | $b_{23}b_{22}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{71}$ | | | $b_{72}b_{73}...b_{95}$ |
| Coding Method | 00101110 | Integer §14.3.1 §14.4.1 | Partition Table 14-23 §14.3.3 §14.4.3 | | | 00...0 (24 zero bits) |

5215 **14.6.7.2 GSRNP+**

5216 The **GSRNP+** coding scheme uses the following **coding** table.

5217 **Table 14-28** GSRNP+ coding table

| Scheme | GSRNP+ | | | |
|------------------------------------|----------------------------------|--------------|--------|------|
| GS1 Digital Link URI syntax | https://id.gs1.org/8017/{gsrcnp} | | | |
| Total Bits | 84 | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | GSRN |
| Corresponding GS1 AI | | | | 8017 |
| Logical Segment Bit Count | 8 | 1 | 3 | 72 |

| Scheme | GSRNP+ | | | |
|--|----------------|-------------------------------|----------------------------------|------------------------------|
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | 18 digits |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...b_{83}$ |
| Coding Method | 11110101 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Fixed-Length Numeric §14.5.4 |

* Note that for the GSRNP+ and other other EPC schemes new to TDS 2.0, the "Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right, in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits.

14.6.8 Global Document Type Identifier (GDTI)

Three coding schemes for the GDTI specified, a 96-bit encoding (GDTI-96), a 113-bit encoding (GDTI-113, DEPRECATED as of TDS 1.9), and a 174-bit encoding (GDTI-174). The GDTI-174 encoding allows for the full range of document serialisation up to 17 alphanumeric characters, as specified in [GS1GS]. The deprecated GDTI-113 encoding allows for a reduced range of document serial numbers up to 17 numeric characters (including leading zeros) as originally specified in [GS1GS]. The GDTI-96 encoding allows for document serial numbers without leading zeros whose value is less than 2^{41} (that is, from 0 through 2,199,023,255,551, inclusive).

Only GDTIs that include the optional serial number may be represented as EPCs. A GDTI without a serial number represents a document class, rather than a specific document, and therefore may not be used as an EPC (just as a non-serialised GTIN may not be used as an EPC).

Both GDTI coding schemes make reference to the following partition table.

Table 14-29 GDTI Partition Table

| Partition Value (P) | Company Prefix | | Document Type | |
|---------------------|----------------|------------|---------------|--------|
| | Bits (M) | Digits (L) | Bits (N) | Digits |
| 0 | 40 | 12 | 1 | 0 |
| 1 | 37 | 11 | 4 | 1 |
| 2 | 34 | 10 | 7 | 2 |
| 3 | 30 | 9 | 11 | 3 |
| 4 | 27 | 8 | 14 | 4 |
| 5 | 24 | 7 | 17 | 5 |
| 6 | 20 | 6 | 21 | 6 |

14.6.8.1 GDTI-96 coding table

Table 14-30 GDTI-96 coding table

| Scheme | GDTI-96 | | | | | |
|------------------------|-----------------------------|--------|-----------|--------------------|---------------|--------|
| URI Template | urn:epc:tag:gdti-96:F.C.D.S | | | | | |
| Total Bits | 96 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Document Type | Serial |

| Scheme | GDTI-96 | | | | | |
|--|-------------------------|-------------------------------|---|-------------|------------|--|
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 21-1 | 41 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 6-0 digits | Up to 13 digits in range 0 – 2,199,023,255,551 without preservation of leading zeros |
| Coding Segment | EPC Header | Filter | Partition + Company Prefix + Document Type | | | Serial |
| URI portion | | F | C . D | | | S |
| Coding Segment Bit Count | 8 | 3 | 44 | | | 41 |
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}b_{85}$ | $b_{84}b_{83}...b_{41}$ | | | $b_{40}b_{39}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{54}$ | | | $b_{55}b_{56}...b_{95}$ |
| Coding Method | 00101100 | Integer §14.3.1 §14.4.1 | Partition Table 14-29 §14.3.3 §14.4.3 | | | Integer §14.3.1 §14.4.1 |

5236 **14.6.8.2 GDTI-113 coding table**

5237 **Table 14-31** GDTI-113 coding table

| Scheme | GDTI-113 | | | | | |
|--|------------------------------|---------------|--|--------------------|---------------|---|
| URI Template | urn:epc:tag:gdti-113:F.C.D.S | | | | | |
| Total Bits | 113 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Document Type | Serial |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 21-1 | 58 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 6-0 digits | Up to 17 digits without preservation of leading zeros |
| Coding Segment | EPC Header | Filter | Partition + Company Prefix + Document Type | | | Serial |
| URI portion | | F | C . D | | | S |

| Scheme | GDTI-113 | | | |
|--|----------------------------|---|---------------------------------------|---|
| Coding Segment Bit Count | 8 | 3 | 44 | 58 |
| Bit Position (counting down) | $b_{112}b_{111}...b_{105}$ | $b_{104}b_{103}b_{102}$ | $b_{101}b_{100}...b_{58}$ | $b_{57}b_{56}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{54}$ | $b_{55}b_{56}...b_{112}$ |
| Coding Method | 00111010 | Integer §14.3.1 §14.4.1 | Partition Table 14-29 | Numeric String §14.3.6 |

5238 **14.6.8.3 GDTI-174 coding table**

5239 **Table 14-32** GDTI-174 coding table

| Scheme | GDTI-174 | | | | | |
|--|------------------------------|---|---|--------------------|---------------|--|
| URI Template | urn:epc:tag:gdti-174:F.C.A.S | | | | | |
| Total Bits | 174 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Document Type | Serial |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 21-1 | 119 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 6-0 digits | Up to 17 characters |
| Coding Segment | EPC Header | Filter | Partition + Company Prefix + Asset Type | | | Serial |
| URI portion | | F | C.A | | | S |
| Coding Segment Bit Count | 8 | 3 | 44 | | | 119 |
| Bit Position (counting down) | $b_{173}b_{172}...b_{166}$ | $b_{165}b_{164}b_{163}$ | $b_{162}b_{161}...b_{119}$ | | | $b_{118}b_{117}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{54}$ | | | $b_{55}b_{56}...b_{173}$ |
| Coding Method | 00111110 | Integer §14.3.1 §14.4.1 | Partition Table 14-29 §14.3.3 §14.4.3 | | | String §14.3.2 §14.4.2 |

5240 **14.6.8.4 GDTI+**

5241 The **GDTI+** coding scheme uses the following **coding** table.

5242

Table 14-33 GDTI+ coding table

| Scheme | GDTI+ | | | | |
|--|-------------------------------|-------------------------------|----------------------------------|------------------------------|--|
| GS1 Digital Link URI syntax | https://id.gs1.org/253/{gdti} | | | | |
| Total Bits | Up to 191 bits | | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | GDTI | GDTI Serial Component |
| Corresponding GS1 AI | | | | (253) | |
| Logical Segment Bit Count | 8 | 1 | 3 | 52 | 3 bit encoding indicator + 5 bit length indicator + up to 119 bits |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | 13 digits | Up to 17 characters |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...b_{63}$ | $b_{64}b_{65}... b_{(B-1)}$ |
| Coding Method | 11110110 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Fixed-Length Numeric §14.5.4 | Variable-length alphanumeric §14.5.6 |

5243
5244
5245

* Note that for the GDTI+ and other EPC schemes new to TDS 2.0, the **"Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right**, in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits.

5246

14.6.9 CPI Identifier (CPI)

5247
5248
5249

Two coding schemes for the CPI identifier are specified: the 96-bit scheme CPI-96 and the variable-length encoding CPI-var. CPI-96 makes use of Partition Table 14-34 and CPI-var makes use of Partition Table 14-35.

5250

Table 14-34 CPI-96 Partition Table

| Partition Value (P) | GS1 Company Prefix | | Component/Part Reference | |
|---------------------|--------------------|------------|--------------------------|----------------|
| | Bits (M) | Digits (L) | Bits (N) | Maximum Digits |
| 0 | 40 | 12 | 11 | 3 |
| 1 | 37 | 11 | 14 | 4 |
| 2 | 34 | 10 | 17 | 5 |
| 3 | 30 | 9 | 21 | 6 |
| 4 | 27 | 8 | 24 | 7 |
| 5 | 24 | 7 | 27 | 8 |
| 6 | 20 | 6 | 31 | 9 |

5251

Table 14-35 CPI-var Partition Table

| Partition Value (P) | GS1 Company Prefix | | Component/Part Reference | |
|---------------------|--------------------|------------|--------------------------|--------------------|
| | Bits (M) | Digits (L) | Maximum Bits ** (N) | Maximum Characters |
| | | | | |

| Partition Value (P) | GS1 Company Prefix | | Component/Part Reference | |
|---------------------|--------------------|----|--------------------------|----|
| 0 | 40 | 12 | 114 | 18 |
| 1 | 37 | 11 | 120 | 19 |
| 2 | 34 | 10 | 126 | 20 |
| 3 | 30 | 9 | 132 | 21 |
| 4 | 27 | 8 | 138 | 22 |
| 5 | 24 | 7 | 144 | 23 |
| 6 | 20 | 6 | 150 | 24 |

** The number of bits depends on the number of characters in the Component/Part Reference; see Sections [14.3.9](#) and [14.4.9](#).

5252
5253

5254 **14.6.9.1 CPI-96 coding table**

5255

Table 14-19 CPI-96 coding table

| Scheme | CPI-96 | | | | | |
|--|----------------------------|---|--|--------------------|---|--|
| URI Template | urn:epc:tag:cpi-96:F.C.P.S | | | | | |
| Total Bits | 96 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Component / Part Reference | Serial |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 31-11 | 31 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 9-3 digits without preservation of leading zeros | Up to 10 digits in range 0 - 2,147,483,647 without preservation of leading zeros |
| Coding Segment | EPC Header | Filter | Component/Part Identifier | | Component / Part Serial Number | |
| URI portion | | F | C.P | | S | |
| Coding Segment Bit Count | 8 | 3 | 54 | | 31 | |
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}b_{85}$ | $b_{84}b_{83}...b_{31}$ | | $b_{30}b_{29}...b_0$ | |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{64}$ | | $b_{65}b_{67}...b_{95}$ | |
| Coding Method | 00111100 | Integer §14.3.1 §14.4.1 | Unpadded Partition Table 14-34 §14.3.4 §14.4.4 | | Integer §14.3.1 §14.4.1 | |

5256 **14.6.9.2 CPI-var coding table**

5257 **Table 14-20** CPI-var coding table

| Scheme | CPI-var | | | | | |
|--|---|-------------------------------|--|--------------------|----------------------------|---|
| URI Template | urn:epc:tag:cpi-var:F.C.P.S | | | | | |
| Total Bits | Variable: between 86 and 224 bits (inclusive) | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Component / Part Reference | Serial |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 12-150 (variable) | 40 (fixed) |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 24-18 characters | Up to 12 digits without preservation of leading zeros |
| Coding Segment | EPC Header | Filter | Component/Part Identifier | | | Component / Part Serial Number |
| URI portion | | F | C.P | | | S |
| Coding Segment Bit Count | 8 | 3 | Up to 173 bits | | | 40 |
| Bit Position (counting down) | $b_{B-1}b_{B-2}...b_{B-8}$ | $b_{B-9}b_{B-10}b_{B-11}$ | $b_{B-12}b_{B-13}...b_{40}$ | | | $b_{39}b_{38}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{(B-41)}$ | | | $b_{(B-40)}b_{(B-39)}...b_{(B-1)}$ |
| Coding Method | 00111101 | Integer §14.3.1 §14.4.1 | 6-Bit Variable String Partition Table 14-35 §14.3.9 14.4.9 | | | Integer §14.3.1 §14.4.1 |

5258 **14.6.9.3 CPI+ coding table**

5259 **Table 14-21** CPI+ coding table

| Scheme | CPI+ | | | | |
|------------------------------------|--|--------------|--------|--------|------------|
| GS1 Digital Link URI syntax | https://id.gs1.org/8010/{cpi}/8011/{cpi_serial} | | | | |
| Total Bits | Up to 266 bits (if at least first 4 characters of CPI are all-numeric) | | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | CPI | CPI Serial |
| Corresponding GS1 AI | | | | (8010) | (8011) |

| Scheme | CPI+ | | | | |
|--|----------------|--|---|---|---|
| Logical Segment Bit Count | 8 | 1 | 3 | 4n (for initial n digits) + 4 bit terminator OR 4n (for initial n digits) + 4 bit delimiter + 3 bit encoding indicator + 5 bit length indicator + up to (210-7n) bits | 4 bit length indicator + up to 40 bits |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | Up to 30 characters with preservation of leading zeros | Up to 12 digits with preservation of leading zeros |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...$ | $..b_{(B-2)}b_{(B-1)}$ |
| Coding Method | 11110000 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Delimited/terminated Numeric (§14.5.5) (followed by Variable-length alphanumeric (§14.5.6) for any characters after the initial n digits) | Variable-length integer without encoding indicator §14.5.13 (using 4-bit length indicator, $b_{LI} = 4$) |

* Note that for the CPI+ and other other EPC schemes new to TDS 2.0, the "Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right, in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits.

5260
5261
5262

14.6.10 Global Coupon Number (SGCN)

A lone, 96-bit coding scheme (SGCN-96) is specified for the SGCN, allowing for the full range of coupon serial component numbers up to 12 numeric characters (including leading zeros) as specified in [GS1GS]. Only SGCNs that include the serial number may be represented as EPCs. A GCN without a serial number represents a coupon class, rather than a specific coupon, and therefore may not be used as an EPC (just as a non-serialised GTIN may not be used as an EPC).

The SGCN coding scheme makes reference to the following partition table.

Table 14-39 SGCN Partition Table

| Partition Value (P) | Company Prefix | | Coupon Reference | |
|------------------------|----------------|---------------|------------------|--------|
| | Bits (M) | Digits (L) | Bits (N) | Digits |
| 0 | 40 | 12 | 1 | 0 |
| 1 | 37 | 11 | 4 | 1 |
| 2 | 34 | 10 | 7 | 2 |
| 3 | 30 | 9 | 11 | 3 |
| 4 | 27 | 8 | 14 | 4 |

5263
5264
5265
5266
5267
5268
5269
5270

| Partition Value (P) | Company Prefix | | Coupon Reference | |
|---------------------|----------------|---|------------------|---|
| 5 | 24 | 7 | 17 | 5 |
| 6 | 20 | 6 | 21 | 6 |

5271 **14.6.10.1 SGCN-96 coding table**

5272 **Table 14-40** SGCN-96 coding table

| Scheme | SGCN-96 | | | | | |
|--|-----------------------------|---|---|--------------------|--|--|
| URI Template | urn:epc:tag:sgcn-96:F.C.D.S | | | | | |
| Total Bits | 96 | | | | | |
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix | Coupon Reference | Serial Component |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 21-1 | 41 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (6-0) | 6-12 digits | 6-0 digits | Up to 12 digits with preservation of leading zeros |
| Coding Segment | EPC Header | Filter | Partition + Company Prefix + Coupon Reference | | Serial | |
| URI portion | | F | C.D | | S | |
| Coding Segment Bit Count | 8 | 3 | 44 | | 41 | |
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}b_{85}$ | $b_{84}b_{83}...b_{41}$ | | $b_{40}b_{39}...b_0$ | |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{54}$ | | $b_{55}b_{56}...b_{95}$ | |
| Coding Method | 00111111 | Integer §14.3.1 §14.4.1 | Partition Table 14-39 §14.3.3 §14.4.3 | | Numeric String §14.3.6 §14.4.6 | |

5273 **14.6.10.2 SGCN+**

5274 The **SGCN+** coding scheme uses the following **coding** table.

5275 **Table 14-41** SGCN+ coding table

| Scheme | SGLN+ | | | | |
|------------------------------------|------------------------------|--------------|--------|---------------------------------------|----------------------|
| GS1 Digital Link URI syntax | https://id.gs1.org/255/{gcn} | | | | |
| Total Bits | Up to 108 bits | | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | GCN without optional serial component | GCN serial component |
| Corresponding GS1 AI | | | | | (255) |

| Scheme | SGLN+ | | | | |
|--|----------------|-------------------------------|----------------------------------|------------------------------|---|
| Logical Segment Bit Count | 8 | 1 | 3 | 52 | 4 bit length indicator + up to 40 bits |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | 13 digits | Up to 12 digits |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...b_{63}$ | $b_{64}b_{65}b_{66}...$ |
| Coding Method | 11111000 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Fixed-Length Numeric §14.5.4 | Variable-length integer without encoding indicator §14.5.13 (using 4-bit length indicator, $b_{LI} = 4$) |

* Note that for the SGCN+ and other other EPC schemes new to TDS 2.0, the "Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right, in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits.

14.6.11 Individual Trade Item Piece (ITIP)

Two coding schemes for the ITIP are specified, a 110-bit encoding (ITIP-110) and a 212-bit encoding (ITIP-212). The ITIP-212 encoding allows for the full range of serial numbers up to 20 alphanumeric characters as specified in [GS1GS]. The ITIP-110 encoding allows for numeric-only serial numbers, without leading zeros, whose value is less than 2^{38} (that is, from 0 through 274,877,906,943, inclusive).

Both ITIP coding schemes make reference to the following partition table.

Table 14-42 ITIP Partition Table

| Partition Value (<i>P</i>) | GS1 Company Prefix | | Indicator/Pad Digit and Item Reference | |
|------------------------------|--------------------|---------------------|--|--------|
| | Bits (<i>M</i>) | Digits (<i>L</i>) | Bits (<i>N</i>) | Digits |
| 0 | 40 | 12 | 4 | 1 |
| 1 | 37 | 11 | 7 | 2 |
| 2 | 34 | 10 | 10 | 3 |
| 3 | 30 | 9 | 14 | 4 |
| 4 | 27 | 8 | 17 | 5 |
| 5 | 24 | 7 | 20 | 6 |
| 6 | 20 | 6 | 24 | 7 |

14.6.11.1 ITIP-110 coding table

Table 14-43 ITIP-110 coding table

| | | |
|---------------------|---------------------------------|--|
| Scheme | ITIP-110 | |
| URI Template | urn:epc:tag:itip-110:F.C.I.PT.S | |
| Total Bits | 110 | |

| Scheme | | ITIP-110 | | | | | | |
|--|--------------------------------|-------------------------------|---|------------------------|---------------------------------|---|---|--|
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix (*) | Indicator (**) / Item Reference | Piece | Total | Serial |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 24-4 | 7 | 7 | 38 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (0-6) | 6-12 digits | 7-1 digits | 2 digits | 2 digits | up to 12 digits in range 0 – 274,877,906,943 without preservation of leading zeros |
| Coding Segment | EPC Header | Filter | GTIN | | | Piece | Total | Serial |
| URI portion | | F | C . I | | | P | T | S |
| Coding Segment Bit Count | 8 | 3 | 47 | | | 7 | 7 | 38 |
| Bit Position (counting down) | $b_{109}b_{108} \dots b_{102}$ | $b_{101}b_{100}b_{99}$ | $b_{98}b_{97} \dots b_{52}$ | | | $b_{51}b_{50} \dots b_{45}$ | $b_{44}b_{43} \dots b_{38}$ | $b_{37}b_{36} \dots b_0$ |
| Bit Position (counting up) | $b_0b_1 \dots b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12} \dots b_{57}$ | | | $b_{58}b_{59} \dots b_{64}$ | $b_{65}b_{66} \dots b_{71}$ | $b_{72}b_{73} \dots b_{109}$ |
| Coding Method | 01000000 | Integer §14.3.1 §14.4.1 | Partition Table 14-11 §14.3.3 §14.4.3 | | | Fixed Width Integer §14.3.1 0 §14.4.1 0 | Fixed Width Integer §14.3.10 §14.4.10 | Integer §14.3.1 §14.4.1 |

5289

(*) See Section [7.3.2](#) for the case of an SGTIN derived from a GTIN-8.

5290

5291

5292

(**) Note that in the case of an ITIP derived from a GTIN-12 or GTIN-13, a zero pad digit takes the place of the Indicator Digit. In all cases, see Section [7.2.3](#) for the definition of how the Indicator Digit (or zero pad) and the Item Reference are combined into this segment of the EPC.

5293

14.6.11.2 ITIP-212 coding table

5294

Table 14-44 ITIP-212 coding table

| Scheme | | ITIP-212 | |
|---------------------|---------------------------------|----------|--|
| URI Template | urn:epc:tag:itip-212:F.C.I.PT.S | | |
| Total Bits | 212 | | |

| Scheme ITIP-212 | | | | | | | | |
|--|----------------------------|-------------------------------|---|------------------------|---------------------------------|---|---|--|
| Logical Segment | EPC Header | Filter | Partition | GS1 Company Prefix (*) | Indicator (**) / Item Reference | Piece | Total | Serial |
| Logical Segment Bit Count | 8 | 3 | 3 | 20-40 | 24-4 | 7 | 7 | 140 |
| Logical Segment Character Count | | 1 digit (0-7) | 1 digit (0-6) | 6-12 digits | 7-1 digits | 2 digits | 2 digits | up to 20 characters with preservation of leading zeros |
| Coding Segment | EPC Header | Filter | GTIN | | | Piece | Total | Serial |
| URI portion | | F | C . I | | | P | T | S |
| Coding Segment Bit Count | 8 | 3 | 47 | | | 7 | 7 | 140 |
| Bit Position (counting down) | $b_{211}b_{210}...b_{204}$ | $b_{203}b_{202}b_{201}$ | $b_{200}b_{199}...b_{154}$ | | | $b_{153}b_{152}...b_{147}$ | $b_{146}b_{145}...b_{140}$ | $b_{139}b_{138}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9b_{10}$ | $b_{11}b_{12}...b_{57}$ | | | $b_{58}b_{59}...b_{64}$ | $b_{65}b_{66}...b_{71}$ | $b_{72}b_{73}...b_{211}$ |
| Coding Method | 01000001 | Integer §14.3.1 §14.4.1 | Partition Table 14-11 §14.3.3 §14.4.3 | | | Fixed Width Integer §14.3.10 §14.4.10 | Fixed Width Integer §14.3.10 §14.4.10 | String §14.3.2 §14.4.2 |

5295

(*) See Section [7.3.2](#) for the case of an SGTIN derived from a GTIN-8.

5296

(**) Note that in the case of an ITIP derived from a GTIN-12 or GTIN-13, a zero pad digit takes the place of the Indicator Digit. In all cases, see Section [7.2.3](#) for the definition of how the Indicator Digit (or zero pad) and the Item Reference are combined into this segment of the EPC.

5297

5298

5299

14.6.11.3 ITIP+

The **ITIP+** coding scheme uses the following **coding** table.

5300

Table 14-45 ITIP+ coding table

5301

| Scheme ITIP+ | | | | | |
|------------------------------------|--|--------------|--------|------|---------------|
| GS1 Digital Link URI syntax | https://id.gs1.org/8006/{itip}/21/{serial} | | | | |
| Total Bits | Up to 232 bits | | | | |
| Logical Segment | EPC Header | +Data Toggle | Filter | ITIP | Serial Number |

| Scheme | ITIP+ | | | | |
|--|----------------|--|---|---|--|
| Corresponding GS1 AI | | | | (8006) | (21) |
| Logical Segment Bit Count | 8 | 1 | 3 | 72 | 3 bit encoding indicator + 5 bit length indicator + up to 140 bits |
| Logical Segment Character Count | | 1 digit (0 or 1) | 1 digit (0-7) | 18 digits | up to 20 characters with preservation of leading zeros |
| Bit Position (counting up)* | $b_0b_1...b_7$ | b_8 | $b_9b_{10}b_{11}$ | $b_{12}b_{13}...b_{83}$ | $b_{84}b_{85}b_{86}...$ |
| Coding Method | 11110011 | +AIDC Data Toggle Bit §14.5.1 | Fixed-Bit-Length Integer §14.5.2 | Fixed-Length Numeric §14.5.4 | Variable-length alphanumeric §14.5.6 |

* Note that for the ITIP+ and other other EPC schemes new to TDS 2.0, the "Bit Position" row of each new EPC coding table is shown only with a 'counting up' approach from left to right, in which b_0 is the left-most bit and b_0-b_7 bits always correspond to the EPC header bits.

5302
5303
5304

14.6.12 General Identifier (GID)

One coding scheme for the GID is specified: the 96-bit encoding GID-96. No partition table is required.

5305
5306
5307

14.6.12.1 GID-96 coding table

Table 14-22 GID-96 coding table

| Scheme | GID-96 | | | |
|--|--------------------------|---|---|---|
| URI Template | urn:epc:tag:gid-96:M.C.S | | | |
| Total Bits | 96 | | | |
| Logical Segment | EPC Header | General Manager Number ³ | Object Class | Serial Number |
| Logical Segment Bit Count | 8 | 28 | 24 | 36 |
| Coding Segment | EPC Header | General Manager Number | Object Class | Serial Number |
| URI portion | | M | C | S |
| Coding Segment Bit Count | 8 | 28 | 24 | 36 |
| Bit Position (counting down) | $b_{95}b_{94}...b_{88}$ | $b_{87}b_{86}...b_{60}$ | $b_{59}b_{58}...b_{36}$ | $b_{35}b_{34}...b_0$ |
| Bit Position (counting up) | $b_0b_1...b_7$ | $b_8b_9...b_{35}$ | $b_{36}b_{37}...b_{59}$ | $b_{60}b_{61}...b_{95}$ |
| Coding Method | 00110101 | Integer §14.3.1 §14.4.1 | Integer §14.3.1 §14.4.1 | Integer §14.3.1 §14.4.1 |

³ **NOTE that General Manager Number issuance has been discontinued**, effective June 2023.

5310 **14.6.13 DoD Identifier**

5311 At the time of this writing, the details of the DoD encoding is explained in a document titled "United
 5312 States Department of Defense Supplier's Passive RFID Information Guide" that can be obtained at
 5313 the United States Department of Defense's web site
 5314 (https://www.dla.mil/Portals/104/Documents/TroopSupport/CloTex/CT_RFID_GUIDE_2011.pdf).

5315 **14.6.14 ADI Identifier (ADI)**

5316 One coding scheme for the ADI identifier is specified: the variable-length encoding ADI-var. No
 5317 partition table is required.

5318 **14.6.14.1 ADI-var coding table**

5319 **Table 14-23** ADI-var coding table

| Scheme | ADI-var | | | | |
|--|---|---|---|--|---|
| URI Template | urn:epc:tag:adi-var:F.D.P.S | | | | |
| Total Bits | Variable: between 68 and 434 bits (inclusive) | | | | |
| Logical Segment | EPC Header | Filter | CAGE/ DoDAAC | Part Number | Serial Number |
| Logical Segment Bit Count | 8 | 6 | 36 | Variable | Variable |
| Logical Segment Character Count | | | 6 characters | 1-33 characters | 2-31 characters |
| Coding Segment | EPC Header | Filter | CAGE/ DoDAAC | Part Number | Serial Number |
| URI Portion | | F | D | P | S |
| Coding Segment Bit Count | 8 | 6 | 36 | Variable (6 – 198) | Variable (12 – 186) |
| Bit Position (counting down) | $b_{B-1}b_{B-2}...b_{B-8}$ | $b_{B-9}b_{B-10}...b_{B-14}$ | $b_{B-15}b_{B-16}...b_{B-50}$ | $b_{B-51}b_{B-52}...$ | $...b_1b_0$ |
| Bit Position (counting up) | $b_0..b_7$ | $b_8..b_{13}$ | $b_{14}..b_{49}$ | $b_{50} b_{51}...$ | $...b_{B-2}b_{B-1}$ |
| Coding Method | 00111011 | Integer §14.3.1 §14.4.1 | 6-bit CAGE/ DoDAAC §14.3.7 §14.4.7 | 6-bit Variable String §14.3.8 §14.4.8 | 6-bit Variable String §14.3.8 §14.4.8 |

5320 **Notes:**

5321 The number of characters in the Part Number segment must be greater than or equal to zero and
 5322 less than or equal to 32. In the binary encoding, a 6-bit zero terminator is always present.

5323 The number of characters in the Serial Number segment must be greater than or equal to one and
 5324 less than or equal to 30. In the binary encoding, a 6-bit zero terminator is always present.

5325 The "#" character (represented in the URI by the escape sequence %23) may appear as the first
 5326 character of the Serial Number segment, but otherwise may not appear in the Part Number segment
 5327 or elsewhere in the Serial Number segment.

5328 **15 EPC Memory Bank contents**

5329 This section specifies how to translate the EPC Tag URI and EPC Raw URI into the binary contents of
5330 the EPC memory bank of a Gen 2 Tag, and vice versa.

5331 **15.1 Encoding procedures**

5332 This section specifies how to translate the EPC Tag URI and EPC Raw URI into the binary contents of
5333 the EPC memory bank of a Gen 2 Tag.

5334 **15.1.1 EPC Tag URI into Gen 2 EPC Memory Bank**

5335 **Given:**

- 5336 ■ An EPC Tag URI beginning with `urn:epc:tag:`

5337 **Encoding procedure:**

- 5338 1. If the URI is not syntactically valid according to Section 12.4, stop: this URI cannot be encoded.
- 5339 2. Apply the encoding procedure of Section 14.3 to the URI. The result is a binary string of N bits.
5340 If the encoding procedure fails, stop: this URI cannot be encoded.
- 5341 3. Fill in the Gen 2 EPC Memory Bank according to the following table:

5342 **Table 15-1** Recipe to Fill In Gen 2 EPC Memory Bank from EPC Tag URI

| Bits | Field | Contents |
|-----------------------------------|-----------------------|--|
| 00 _n – 0F _n | CRC | CRC code calculated from the remainder of the memory bank. (Normally, this is calculated automatically by the reader, and so software that implements this procedure need not be concerned with it.) |
| 10 _n – 14 _n | Length | The number of bits, N , in the EPC binary encoding determined in Step 2 above, divided by 16, and rounded up to the next higher integer if N was not a multiple of 16. |
| 15 _n | User Memory Indicator | If the EPC Tag URI includes a control field [<code>umi=1</code>], a one bit. If the EPC Tag URI includes a control field [<code>umi=0</code>] or does not contain a <code>umi</code> control field, a zero bit. Note that certain Gen 2 Tags may ignore the value written to this bit, and instead calculate the value of the bit from the contents of user memory. See [UHFC1G2]. |
| 16 _n | XPC Indicator | This bit is calculated by the tag and ignored by the tag when the tag is written, and so is disregarded by this encoding procedure. |
| 17 _n | Toggle | 0, indicating that the EPC bank contains an EPC |
| 18 _n – 1F _n | Attribute Bits | If the EPC Tag URI includes a control field [<code>att=xNN</code>], the value <code>NN</code> considered as an 8-bit hexadecimal number. If the EPC Tag URI does not contain such a control field, zero. |
| 20 _n – ? | EPC/UII | The N bits obtained from the EPC binary encoding procedure in Step 2 above, followed by enough zero bits to bring the total number of bits to a multiple of 16 (0 – 15 extra zero bits) |

5343 **15.1.2 EPC Raw URI into Gen 2 EPC Memory Bank**

5344 **Given:**

- 5345 ■ An EPC Raw URI beginning with `urn:epc:raw:.` Such a URI has one of the following three
5346 forms:

5347 `urn:epc:raw:OptionalControlFields:Length.xHexPayload`

5348 `urn:epc:raw:OptionalControlFields:Length.xAFI.xHexPayload`

urn:epc:raw:OptionalControlFields:Length.DecimalPayload

Encoding procedure:

1. If the URI is not syntactically valid according to the grammar in Section 12.4, stop: this URI cannot be encoded.
2. Extract the leftmost *NonZeroComponent* according to the grammar (the *Length* field in the templates above). This component immediately follows the rightmost colon (:) character. Consider this as a decimal integer, *N*. This is the number of bits in the raw payload.
3. Determine the toggle bit and AFI (if any):
 - a. If the body of the URI matches the *DecimalRawURIBody* or *HexRawURIBody* production of the grammar (the first and third templates above), the toggle bit is zero.
 - b. If the body of the URI matches the *AFIRawURIBody* production of the grammar (the second template above), the toggle bit is one. The AFI is the value of the leftmost *HexComponent* within the *AFIRawURIBody* (the *AFI* field in the template above), considered as an 8-bit unsigned hexadecimal integer. If the value of the *HexComponent* is greater than or equal to 256, stop: this URI cannot be encoded.
4. Determine the EPC/UII payload:
 - c. If the body of the URI matches the *HexRawURIBody* production of the grammar (first template above) or *AFIRawURIBody* production of the grammar (second template above), the payload is the rightmost *HexComponent* within the body (the *HexPayload* field in the templates above), considered as an *N*-bit unsigned hexadecimal integer, where *N* is as determined in Step 2 above. If the value of this *HexComponent* greater than or equal to 2^N , stop: this URI cannot be encoded.
 - d. If the body of the URI matches the *DecimalRawURIBody* production of the grammar (third template above), the payload is the rightmost *NumericComponent* within the body (the *DecimalPayload* field in the template above), considered as an *N*-bit unsigned decimal integer, where *N* is as determined in Step 2 above. If the value of this *NumericComponent* greater than or equal to 2^N , stop: this URI cannot be encoded.
5. Fill in the Gen 2 EPC Memory Bank according to the following table:

Table 15-2 Recipe to Fill In Gen 2 EPC Memory Bank from EPC Raw URI

| Bits | Field | Contents |
|-----------------------------------|-----------------------|---|
| 00 _h – 0F _h | CRC | CRC code calculated from the remainder of the memory bank. (Normally, this is calculated automatically by the reader, and so software that implements this procedure need not be concerned with it.) |
| 10 _h – 14 _h | Length | The number of bits, <i>N</i> , in the EPC binary encoding determined in Step 2 above, divided by 16, and rounded up to the next higher integer if <i>N</i> was not a multiple of 16. |
| 15 _h | User Memory Indicator | This bit is calculated by the tag and ignored by the tag when the tag is written, and so is disregarded by this encoding procedure. |
| 16 _h | XPC Indicator | This bit is calculated by the tag and ignored by the tag when the tag is written, and so is disregarded by this encoding procedure. |
| 17 _h | Toggle | The value determined in Step 3, above. |
| 18 _h – 1F _h | AFI / Attribute Bits | If the toggle determined in Step 3 is one, the value of the AFI determined in Step 3.2. Otherwise, If the URI includes a control field [att=xNN], the value NN considered as an 8-bit hexadecimal number. If the URI does not contain such a control field, zero. |
| 20 _h – ? | EPC/UII | The <i>N</i> bits determined in Step 4 above, followed by enough zero bits to bring the total number of bits to a multiple of 16 (0 – 15 extra zero bits) |

5378 15.2 Decoding procedures

5379 This section specifies how to translate the binary contents of the EPC memory bank of a Gen 2 Tag
5380 into the EPC Tag URI and EPC Raw URI.

5381 15.2.1 Gen 2 EPC Memory Bank into EPC Raw URI

5382 **Given:**

- 5383 ■ The contents of the EPC Memory Bank of a Gen 2 tag

5384 **Procedure:**

- 5385 1. Extract the length bits, bits 10_h – 14_h. Consider these bits to be an unsigned integer L .
- 5386 2. Calculate $N = 16L$.
- 5387 3. If bit 17_h is set to one, extract bits 18_h – 1F_h and consider them to be an unsigned integer A .
5388 Construct a string consisting of the letter "x", followed by A as a 2-digit hexadecimal numeral
5389 (using digits and uppercase letters only), followed by a period (".").
- 5390 4. Apply the decoding procedure of Section [15.2.4](#) to decode control fields.
- 5391 5. Extract N bits beginning at bit 20_h and consider them to be an unsigned integer V . Construct a
5392 string consisting of the letter "x" followed by V as a $(N/4)$ -digit hexadecimal numeral (using
5393 digits and uppercase letters only).
- 5394 6. Construct a string consisting of "urn:epc:raw:", followed by the result from Step 4 (if not
5395 empty), followed by N as a decimal numeral without leading zeros, followed by a period ("."),
5396 followed by the result from Step 3 (if not empty), followed by the result from Step 5. This is the
5397 final EPC Raw URI.

5398 15.2.2 Gen 2 EPC Memory Bank into EPC Tag URI

5399 This procedure decodes the contents of a Gen 2 EPC Memory bank into an EPC Tag URI beginning
5400 with urn:epc:tag: if the memory contains a valid EPC, or into an EPC Raw URI beginning
5401 urn:epc:raw: otherwise.

5402 **Given:**

- 5403 ■ The contents of the EPC Memory Bank of a Gen 2 tag

5404 **Procedure:**

- 5405 1. Extract the length bits, bits 10_h – 14_h. Consider these bits to be an unsigned integer L .
- 5406 2. Calculate $N = 16L$.
- 5407 3. Extract N bits beginning at bit 20_h. Apply the decoding procedure of Section [14.3.9](#), passing the
5408 N bits as the input to that procedure.
- 5409 4. If the decoding procedure of Section [14.3.9](#) fails, continue with the decoding procedure of
5410 Section [15.2.1](#) to compute an EPC Raw URI. Otherwise, the decoding procedure of
5411 Section [14.3.9](#) yielded an EPC Tag URI beginning urn:epc:tag:. Continue to the next step.
- 5412 5. Apply the decoding procedure of Section [15.2.4](#) to decode control fields.
- 5413 6. Insert the result from Section [15.2.4](#) (including any trailing colon) into the EPC Tag URI
5414 obtained in Step 4, immediately following the urn:epc:tag: prefix. (If Section [15.2.4](#) yielded
5415 an empty string, this result is identical to what was obtained in Step 4.) The result is the final
5416 EPC Tag URI.

5417 15.2.3 Gen 2 EPC Memory Bank into Pure Identity EPC URI

5418 This procedure decodes the contents of a Gen 2 EPC Memory bank into a Pure Identity EPC URI
 5419 beginning with `urn:epc:id:` if the memory contains a valid EPC, or into an EPC Raw URI beginning
 5420 `urn:epc:raw:` otherwise.

5421 **Given:**

- 5422 ■ The contents of the EPC Memory Bank of a Gen 2 tag

5423 **Procedure:**

- 5424 1. Apply the decoding procedure of Section [15.2.2](#) to obtain either an EPC Tag URI or an EPC Raw
 5425 URI. If an EPC Raw URI is obtained, this is the final result.
- 5426 2. Otherwise, apply the procedure of Section [12.3.3](#) to the EPC Tag URI from Step 1 to obtain a
 5427 Pure Identity EPC URI. This is the final result.

5428 15.2.4 Decoding of control information

5429 This procedure is used as a subroutine by the decoding procedures in Sections [15.2.1](#) and [15.2.2](#). It
 5430 calculates a string that is inserted immediately following the `urn:epc:tag:` or `urn:epc:raw:`
 5431 prefix, containing the values of all non-zero control information fields (apart from the filter value). If
 5432 all such fields are zero, this procedure returns an empty string, in which case nothing additional is
 5433 inserted after the `urn:epc:tag:` or `urn:epc:raw:` prefix.

5434 **Given:**

- 5435 ■ The contents of the EPC Memory Bank of a Gen 2 tag

5436 **Procedure:**

- 5437 1. If bit 17_h is zero, extract bits 18_h – 1F_h and consider them to be an unsigned integer *A*. If *A* is
 5438 non-zero, append the string `[att=xAA]` (square brackets included) to *CF*, where *AA* is the value
 5439 of *A* as a two-digit hexadecimal numeral.
- 5440 2. If bit 15_h is non-zero, append the string `[umi=1]` (square brackets included) to *CF*.
- 5441 3. If bit 16_h is non-zero, extract bits 210_h – 21F_h and consider them to be an unsigned integer *X*.
 5442 Append the string `[xpc-w1=xXXXX]` (square brackets included) to *CF*, where *XXXX* is the value
 5443 of *X* as a four-digit hexadecimal numeral. Note that in the Gen 2 air interface, bits 210_h – 21F_h
 5444 are inserted into the backscattered inventory data immediately following bit 1F_h, when bit 16_h is
 5445 non-zero. See [UHFC1G2]. If bit 210_h is non-zero, extract bits 220_h – 22F_h and consider them to
 5446 be an unsigned integer *Y*. Append the string `[xpc=xXXXXYYYY]` (square brackets included) to
 5447 *CF*, where *YYYY* is the value of *Y* as a four-digit hexadecimal numeral. Note that in the Gen 2 air
 5448 interface, bits 220_h – 22F_h are inserted into the backscattered inventory data immediately
 5449 following bit 21F_h, when bit 210_h is non-zero. See [UHFC1G2].
- 5450 4. Return the resulting string (which may be empty).

5451 15.3 '+AIDC data' following new EPC schemes in the EPC/UII memory bank

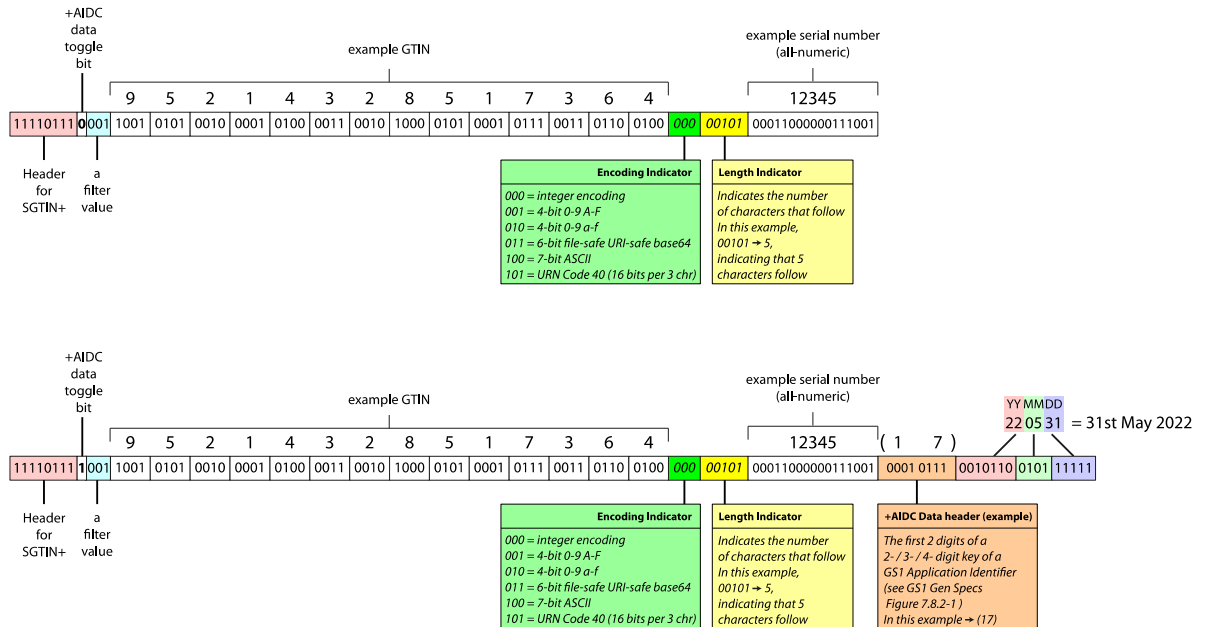
5452 All of the new EPC schemes introduced in TDS 2.0 (DSGTIN+, SGTIN+ etc.) support appending of a
 5453 AIDC data beyond the end of the EPC within the EPC/UII memory bank.

5454 A single bit that follows immediately after the 8-bit EPC header of the new EPC schemes serves as a
 5455 toggle bit for '+AIDC data'. If this bit is set 1, additional AIDC data is expected after the EPC. If this
 5456 bit is set to 0 no additional AIDC data is expected.

5457
5458

This is illustrated in the figure below:

Figure 15-1 Example of '+AIDC data' in EPC/UII memory



5459
5460
5461
5462
5463
5464
5465
5466
5467
5468
5469
5470
5471
5472

Each set of additional AIDC data begins with an 8-bit AIDC data header, which is interpreted as two 4-bit hexadecimal characters. If either or both of these characters are in the range A-F, these indicate a special header typically used for optimisation purposes or reserved for future use. Otherwise, if both of these characters are in the range 0 to 9, they should be interpreted as the first two digits of a GS1 Application Identifier key. GS1 Application Identifier keys consists of two, three or four digits, such as (01), (414), (8003). By consulting Figure 7.8.1-2 within the GS1 General Specifications, it is possible to determine whether additional digits need to be read for GS1 Application Identifier keys that are three or four digits in length.

For example, in Figure 7.8.1-2 within the GS1 General Specifications, 41 is always the start of a 3-digit key 41n, while 80 is always the start of a 4-digit key, 80nn. Table K is derived from GS1 Gen Specs Figure 7.8.1-2, adding an additional column to indicate how many additional bits need to be read beyond the initial eight bits of the data header.

5473
5474
5475

Table K is shown in full below. It is derived from Figure 7.8.1-2 of the GS1 General Specifications and includes an extra column that indicates the number of additional bits to be read.

| First two digits | GS1 AI length | Additional bits to read |
|------------------|---------------|-------------------------|
| 00 | 2 | 0 |
| 01 | 2 | 0 |
| 02 | 2 | 0 |
| 10 | 2 | 0 |
| 11 | 2 | 0 |
| 12 | 2 | 0 |
| 13 | 2 | 0 |
| 15 | 2 | 0 |
| 16 | 2 | 0 |
| 17 | 2 | 0 |
| 20 | 2 | 0 |
| 21 | 2 | 0 |
| 22 | 2 | 0 |
| 23 | 3 | 4 |
| 24 | 3 | 4 |
| 25 | 3 | 4 |
| 31 | 4 | 8 |
| 32 | 4 | 8 |
| 33 | 4 | 8 |
| 34 | 4 | 8 |
| 35 | 4 | 8 |
| 36 | 4 | 8 |

| First two digits | GS1 AI length | Additional bits to read |
|------------------|---------------|-------------------------|
| 37 | 2 | 0 |
| 39 | 4 | 8 |
| 40 | 3 | 4 |
| 41 | 3 | 4 |
| 42 | 3 | 4 |
| 43 | 4 | 8 |
| 70 | 4 | 8 |
| 71 | 3 | 4 |
| 72 | 4 | 8 |
| 80 | 4 | 8 |
| 81 | 4 | 8 |
| 82 | 4 | 8 |
| 90 | 2 | 0 |
| 91 | 2 | 0 |
| 92 | 2 | 0 |
| 93 | 2 | 0 |
| 94 | 2 | 0 |
| 95 | 2 | 0 |
| 96 | 2 | 0 |
| 97 | 2 | 0 |
| 98 | 2 | 0 |
| 99 | 2 | 0 |

5476
5477

If the first two digits are not shown in Table K, no GS1 Application Identifier key begins with those two digits.

5478
5479

If a 2-digit key is indicated, no additional bits must be read – the 8-bit data header is interpreted as a two-digit GS1 Application Identifier key.

5480
5481
5482

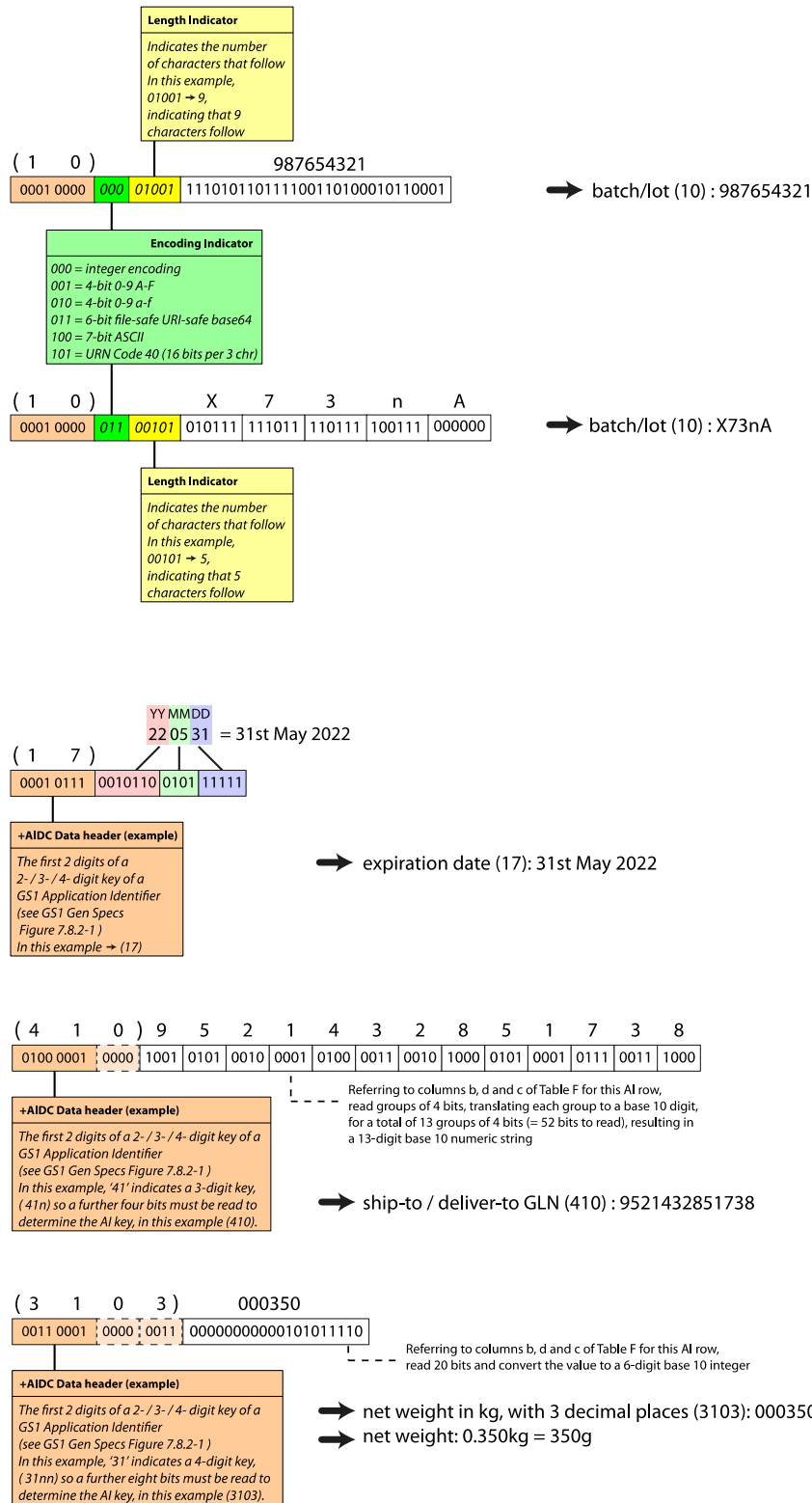
If a 3-digit key is indicated, four additional bits must be read beyond the 8-bit data header and interpreted as the third digit of the GS1 Application Identifier key.

If a 4-digit key is indicated, a further eight bits must be read after the 8-bit data header and

5483
5484
5485

interpreted as the third and fourth digits of the GS1 Application Identifier key. This is illustrated in the Figure below:

Figure 15-2 Reading and interpreting additional bits after the 8-bit data header



5486
5487
5488
5489

After determining the GS1 Application Identifier key (whether 2,3 or 4 digits), a lookup in column a of Table F explains how the corresponding value is to be encoded. Most values consist of a single component which is either numeric or alphanumeric and may be fixed length or variable length.

5490
5491
5492

However, a small number of values consist of two components where the second component is typically variable-length and maybe alphanumeric or numeric, while the first component is typically fixed length.

5493
5494

Locate the row containing GS1 Application Identifier key in column a of Table F, then read column b to determine the encoding for the first component of the value.

5495

5496
5497

If the first component is fixed-length, the number of characters is shown in column c and the number of bits is shown in column d. For the examples shown in the figure above, the extract of Table F is shown below:

5498
5499

If the value is variable-length, column g indicates the maximum number of characters permitted for the first component and column f specifies the number of bits for the length indicator.

5500
5501
5502
5503

Table F is shown in full below. Note that a small number of GS1 Application Identifiers have a second component in Table F, shown as values in columns h-o, which are analogous to columns b-g but apply to the second component that is encoded in binary immediately after the first component. The GS1 Application Identifiers that use a second component are the following: (253), (255), (3910)-(3919), (3930)-(3939), (421), (7030)-(7039), (7040), (8003).

| a | b | c | d | e | f | g | | h | j | k | m | n | o |
|----|--------------------------------------|-------------------|--------------------|--------------------------|------------------------|--------------------|--|------------------|-------------------|--------------------|--------------------------|---------------------------------|--------------------|
| AI | First component | | | | | | | Second component | | | | | |
| | Format | Fixed length #chr | Fixed length #bits | Encoding indicator #bits | Length indicator #bits | Max. Length (chrs) | | Format | Fixed length #chr | Fixed length #bits | Encoding indicator #bits | Length indicator #bits required | Max. Length (chrs) |
| | | L | | | b _{LI} | L _{max} | | | L | | | b _{LI} | L _{max} |
| 00 | Fixed-length numeric §14.5.4 | 18 | 72 | | | | | | | | | | |
| 01 | Fixed-length numeric §14.5.4 | 14 | 56 | | | | | | | | | | |
| 02 | Fixed-length numeric §14.5.4 | 14 | 56 | | | | | | | | | | |
| 10 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | |
| 11 | 6-digit date YYYYMM §14.5.8 | 6 | 16 | | | | | | | | | | |
| 12 | 6-digit date YYYYMM §14.5.8 | 6 | 16 | | | | | | | | | | |

| | | | | | | | | | | | | | | |
|-----|--|---|----|---|---|----|--|--|--|--|--|--|--|--|
| 13 | 6-digit date YYMMDD §14.5.8 | 6 | 16 | | | | | | | | | | | |
| 15 | 6-digit date YYMMDD §14.5.8 | 6 | 16 | | | | | | | | | | | |
| 16 | 6-digit date YYMMDD §14.5.8 | 6 | 16 | | | | | | | | | | | |
| 17 | 6-digit date YYMMDD §14.5.8 | 6 | 16 | | | | | | | | | | | |
| 20 | Fixed-Bit-Length Integer §14.5.2 | 2 | 7 | | | | | | | | | | | |
| 21 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | | |
| 22 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | | |
| 235 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 28 | | | | | | | | |
| 240 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | | |
| 241 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | | |
| 242 | Variable-length integer without encoding indicator §14.5.13 | | | | 3 | 6 | | | | | | | | |
| 243 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | | |
| 250 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | | |

| | | | | | | | | | | | | | |
|-----------|---|----|----|---|---|----|---|--|--|---|---|----|--|
| 251 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | |
| 253 | Fixed-length numeric §14.5.4 | 13 | 52 | | | | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 17 | |
| 254 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | |
| 255 | Fixed-length numeric §14.5.4 | 13 | 52 | | | | Variable-length integer without encoding indicator §14.5.13 | | | | 4 | 12 | |
| 30 | Variable-length integer without encoding indicator §14.5.13 | | | | 4 | 8 | | | | | | | |
| 3100-3105 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3110-3115 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3120-3125 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3130-3135 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3140-3145 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3150-3155 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |

| | | | | | | | | | | | | | |
|---------------|---|---|----|--|--|--|--|--|--|--|--|--|--|
| 3160 -3165 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3200 -3205 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3210 -3215 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3220 -3225 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3230 -3235 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3240 -3245 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3250 -3255 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3260 -3265 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3270 -3275 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3280 -3285 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3290 -3295 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3300 -3305 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3310 -3315 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |

| | | | | | | | | | | | | | |
|---------------|---|---|----|--|--|--|--|--|--|--|--|--|--|
| 3320 -3325 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3330 -3335 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3340 -3345 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3350 -3355 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3360 -3365 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3370 -3375 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3400 -3405 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3410 -3415 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3420 -3425 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3430 -3435 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3440 -3445 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3450 -3455 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3460 -3465 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |

| | | | | | | | | | | | | | |
|---------------|---|---|----|--|--|--|--|--|--|--|--|--|--|
| 3470 -3475 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3480 -3485 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3490 -3495 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3500 -3505 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3510 -3515 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3520 -3525 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3530 -3535 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3540 -3545 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3550 -3555 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3560 -3565 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3570 -3575 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3600 -3605 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 3610 -3615 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |

| | | | | | | | | | | | | |
|---------------|---|---|----|--|---|----|--|--|--|--|---|----|
| 3620 -3625 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | |
| 3630 -3635 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | |
| 3640 -3645 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | |
| 3650 -3655 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | |
| 3660 -3665 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | |
| 3670 -3675 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | |
| 3680 -3685 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | |
| 3690 -3695 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | |
| 37 | Variable-length integer without encoding indicator §14.5.13 | | | | 4 | 8 | | | | | | |
| 3900 -3909 | Variable-length integer without encoding indicator §14.5.13 | | | | 4 | 15 | | | | | | |
| 3910 -3919 | Fixed-Bit-Length Integer §14.5.2 | 3 | 10 | | | | | Variable- length integer without encoding indicator §14.5.13 | | | 4 | 15 |

| | | | | | | | | | | | | | |
|---------------|---|----|----|---|---|----|--|---|--|--|---|---|----|
| 3920 -3929 | Variable-length integer without encoding indicator §14.5.13 | | | | 4 | 15 | | | | | | | |
| 3930 -3939 | Fixed-Bit-Length Integer §14.5.2 | 3 | 10 | | | | | Variable-length integer without encoding indicator §14.5.13 | | | | 4 | 15 |
| 3940 -3943 | Fixed-Bit-Length Integer §14.5.2 | 4 | 14 | | | | | | | | | | |
| 3950 -3953 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 400 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | |
| 401 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | |
| 402 | Fixed-Bit-Length Integer §14.5.2 | 17 | 57 | | | | | | | | | | |
| 403 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | |
| 410 - 417 | Fixed-length numeric §14.5.4 | 13 | 52 | | | | | | | | | | |
| 420 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | |
| 421 | Fixed-Bit-Length Integer §14.5.2 | 3 | 10 | | | | | Variable-length alphanumeric §14.5.6 | | | 3 | 4 | 9 |

| | | | | | | | | | | | | | | |
|------|---|---|----|---|---|----|--|--|--|--|--|--|--|--|
| 422 | Fixed-Bit-Length Integer §14.5.2 | 3 | 10 | | | | | | | | | | | |
| 423 | Variable-length integer without encoding indicator §14.5.13 | | | | 4 | 15 | | | | | | | | |
| 424 | Fixed-Bit-Length Integer §14.5.2 | 3 | 10 | | | | | | | | | | | |
| 425 | Variable-length integer without encoding indicator §14.5.13 | | | | 4 | 15 | | | | | | | | |
| 426 | Fixed-Bit-Length Integer §14.5.2 | 3 | 10 | | | | | | | | | | | |
| 427 | Variable-length alphanumeric §14.5.6 | | | 3 | 2 | 3 | | | | | | | | |
| 4300 | Variable-length alphanumeric §14.5.6 | | | 3 | 6 | 35 | | | | | | | | |
| 4301 | Variable-length alphanumeric §14.5.6 | | | 3 | 6 | 35 | | | | | | | | |
| 4302 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | | |
| 4303 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | | |
| 4304 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | | |
| 4305 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | | |
| 4306 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | | |

| | | | | | | | | | | | | | | |
|------|--|----|----|---|---|----|--|--|--|--|--|--|--|--|
| 4307 | Country code (ISO 3166-1 alpha-2) §14.5.12 | 2 | 12 | | | | | | | | | | | |
| 4308 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | | |
| 4309 | Fixed-Bit-Length Integer §14.5.2 | 20 | 67 | | | | | | | | | | | |
| 4310 | Variable-length alphanumeric §14.5.6 | | | 3 | 6 | 35 | | | | | | | | |
| 4311 | Variable-length alphanumeric §14.5.6 | | | 3 | 6 | 35 | | | | | | | | |
| 4312 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | | |
| 4313 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | | |
| 4314 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | | |
| 4315 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | | |
| 4316 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | | |
| 4317 | Country code (ISO 3166-1 alpha-2) §14.5.12 | 2 | 12 | | | | | | | | | | | |
| 4318 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | | |
| 4319 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | | |

| | | | | | | | | | | | | | |
|------|---------------------------------------|----|----|---|---|----|---|--|---|--|--|--|---|
| 4320 | Variable-length alphanumeric §14.5.6 | | | 3 | 6 | 35 | | | | | | | |
| 4321 | Single data bit §14.5.7 | 1 | 1 | | | | | | | | | | |
| 4322 | Single data bit §14.5.7 | 1 | 1 | | | | | | | | | | |
| 4323 | Single data bit §14.5.7 | 1 | 1 | | | | | | | | | | |
| 4324 | 10-digit date+time YYMMDDhhmm §14.5.9 | 10 | 27 | | | | | | | | | | |
| 4325 | 10-digit date+time YYMMDDhhmm §14.5.9 | 10 | 27 | | | | | | | | | | |
| 4326 | 6-digit date YYMMDD §14.5.8 | 6 | 16 | | | | | | | | | | |
| 4330 | Fixed-Bit-Length Integer §14.5.2 | 6 | | | | | Optional minus sign in 1 bit (§14.5.14) | | 1 | | | | 1 |
| 4331 | Fixed-Bit-Length Integer §14.5.2 | 6 | | | | | Optional minus sign in 1 bit (§14.5.14) | | 1 | | | | 1 |
| 4332 | Fixed-Bit-Length Integer §14.5.2 | 6 | | | | | Optional minus sign in 1 bit (§14.5.14) | | 1 | | | | 1 |
| 4333 | Fixed-Bit-Length Integer §14.5.2 | 6 | | | | | Optional minus sign in 1 bit (§14.5.14) | | 1 | | | | 1 |

| | | | | | | | | | | | | | |
|------|--|----|----|---|---|----|--|--|--|--|--|--|--|
| 7001 | Fixed-Bit-Length Integer §14.5.2 | 13 | 44 | | | | | | | | | | |
| 7002 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | |
| 7003 | 10-digit date+time YYMMDDhhmm §14.5.9 | 10 | 27 | | | | | | | | | | |
| 7004 | Variable-length integer without encoding indicator §14.5.13 | | | | 3 | 4 | | | | | | | |
| 7005 | Variable-length alphanumeric §14.5.6 | | | 3 | 4 | 12 | | | | | | | |
| 7006 | 6-digit date YYMMDD §14.5.8 | 6 | 16 | | | | | | | | | | |
| 7007 | Variable-format date / date range §14.5.10 | | | | | | | | | | | | |
| 7008 | Variable-length alphanumeric §14.5.6 | | | 3 | 2 | 3 | | | | | | | |
| 7009 | Variable-length alphanumeric §14.5.6 | | | 3 | 4 | 10 | | | | | | | |
| 7010 | Variable-length alphanumeric §14.5.6 | | | 3 | 2 | 2 | | | | | | | |
| 7011 | Variable-precision date+time §14.5.11 | | | | | | | | | | | | |
| 7020 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | |
| 7021 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | |

| | | | | | | | | | | | | | |
|-----------|--------------------------------------|----|----|---|---|----|--------------------------------------|--|--|---|---|----|--|
| 7022 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | |
| 7023 | Delimited/terminated numeric §14.5.5 | | | 3 | 5 | 30 | | | | | | | |
| 7030-7039 | Fixed-Bit-Length Integer §14.5.2 | 3 | 10 | | | | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 27 | |
| 7040 | Variable-length alphanumeric §14.5.6 | | | 3 | 3 | 4 | | | | | | | |
| 710-715 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | |
| 7230-7239 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | |
| 7240 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | |
| 7241 | Fixed-length numeric §14.5.4 | 2 | 8 | | | | | | | | | | |
| 7242 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 25 | | | | | | | |
| 8001 | Fixed-Bit-Length Integer §14.5.2 | 14 | 47 | | | | | | | | | | |
| 8002 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | |
| 8003 | Fixed-length numeric §14.5.4 | 14 | 56 | | | | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 16 | |

| | | | | | | | | | | | | | |
|------|---|----|----|---|---|----|--|--|--|--|--|--|--|
| 8004 | Delimited/terminated numeric §14.5.5 | | | 3 | 5 | 30 | | | | | | | |
| 8005 | Fixed-Bit-Length Integer §14.5.2 | 6 | 20 | | | | | | | | | | |
| 8006 | Fixed-length numeric §14.5.4 | 18 | 72 | | | | | | | | | | |
| 8007 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 24 | | | | | | | |
| 8008 | Variable-precision date+time §14.5.11 | | | | | | | | | | | | |
| 8009 | Variable-length alphanumeric §14.5.6 | | | 3 | 6 | 50 | | | | | | | |
| 8010 | Delimited/terminated numeric §14.5.5 | | | 3 | 5 | 30 | | | | | | | |
| 8011 | Variable-length integer without encoding indicator §14.5.13 | | | | 4 | 12 | | | | | | | |
| 8012 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 20 | | | | | | | |
| 8013 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 25 | | | | | | | |
| 8017 | Fixed-length numeric §14.5.4 | 18 | 72 | | | | | | | | | | |
| 8018 | Fixed-length numeric §14.5.4 | 18 | 72 | | | | | | | | | | |
| 8019 | Variable-length integer without encoding indicator §14.5.13 | | | | 4 | 10 | | | | | | | |

| | | | | | | | | | | | | | |
|-------|--------------------------------------|----|----|---|---|----|--|--|--|--|--|--|--|
| 8020 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 25 | | | | | | | |
| 8026 | Fixed-length numeric §14.5.4 | 18 | 72 | | | | | | | | | | |
| 8030 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 90 | | | | | | | |
| 8110 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | |
| 8111 | Fixed-Bit-Length Integer §14.5.2 | 4 | 14 | | | | | | | | | | |
| 8112 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | |
| 8200 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 70 | | | | | | | |
| 90 | Variable-length alphanumeric §14.5.6 | | | 3 | 5 | 30 | | | | | | | |
| 91-99 | Variable-length alphanumeric §14.5.6 | | | 3 | 7 | 90 | | | | | | | |

5504

5505
5506
5507
5508
5509
5510
5511
5512
5513
5514
5515
5516
5517
5518
5519
5520
5521
5522

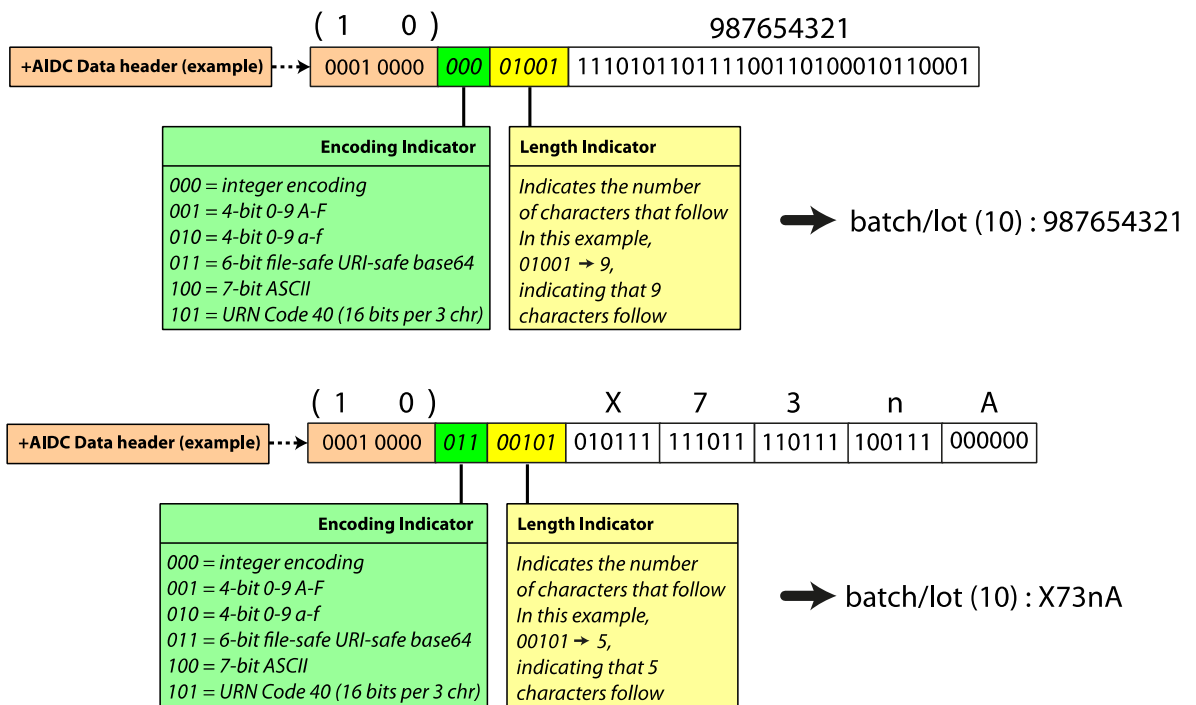
Table E (see Section 14.5.6) lists the permitted values for encoding indicator together with the encoding methods and the character ranges supported by each method.

Note that variable-length numeric values do not use an encoding indicator but typically do use a length indicator. The exception to the statement above is for the GIAI and CPI, which use the 'terminated/delimited' encoding method, in which a delimiter or terminator character marks the end of an initial all-numeric sequence. If the remainder is an alphanumeric sequence, the delimiter character is followed by an encoding indicator, length indicator and the encoding of the alphanumeric sequence.

Where present, the length indicator always indicates the total number of characters or digits for that value or component. For example a value 00101 indicates a length of 5 characters.

The figure below shows two examples for encoding a batch/lot number, one all-numeric, the other alphanumeric. The two examples illustrate different values of encoding indicator and length indicator, as well as the corresponding bit layouts. Note that because the first example is all-numeric, integer encoding at 3.32bits per digit can be used, whereas the second example is mixed case alphanumeric, but because it is not using any symbol characters, we can use file-safe URI-safe base64 encoding at 6 bits per character.

Figure 15-3 Examples of encoding all-numeric and alphanumeric batch/lot number



5523
5524
5525
5526
5527
5528
5529
5530
5531
5532
5533
5534

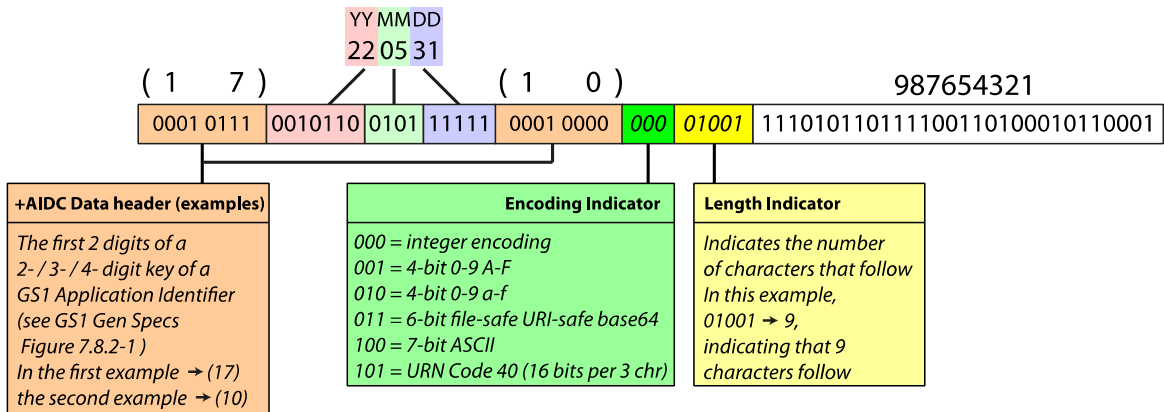
The number of bits required for the length indicator depends on the maximum permitted length for the value (or the value of the first / second component shown in Table F). Columns f and n of Table F indicate the number of bits to be used for the length indicator (where present), for the first and second components respectively.

Date values and date-time values use particularly optimised encodings to save bits and column b of Table F indicates dedicated methods for efficiently encoding/decoding date value or date+time values.

It is possible to encode more than one AIDC data value after the EPC by repeating the procedure and adding further data headers for each successive GS1 Application Identifier and its value. This is illustrated in the following figure. All remaining bits up to the next 16-bit word boundary SHALL be set to '0'.

5535

Figure 15-4 Encoding more than one AIDC data value after the EPC



5536

5537

5538

5539

5540

5541

When decoding +AIDC data encoded after the EPC, the decoding procedure should be repeated if the number of 16-bit words indicated by the Gen 2 Protocol Control bits 10_h – 14_h indicate that further bits have been encoded. If fewer than 8 bits remain before the indicated word count is reached, there can be no further +AIDC data. Otherwise, if at least 8 further bits remain, consider the following three options:

5542

5543

- If the next 8-bits are not '00000000', repeat the procedure, considering those 8 bits as the next +AIDC data header.
- If the next 8 bit are '00000000' and at least 72 bits remain, consider those 8 bits as a +AIDC data header for an SSCC (00) and decode the following 72 bits using the Fixed-length Numeric method described in §14.5.4.
- If the next 8 bit are '00000000' and fewer than 72 bits remain, stop, since this cannot be decoded as an SSCC (00).

5544

5545

5546

5547

5548

All additional AIDC data expressed within the EPC/UII memory bank SHALL observe the rules regarding mandatory associations and invalid pairs of GS1 Application Identifiers, defined in the GS1 General Specifications and considering the GS1 Application Identifiers that are effectively already expressed by the EPC identifier itself, e.g. (01) and (21) in the case of SGTIN+.

5553

5554

5555

The non-binary values decoded for AIDC data expressed within the EPC/UII memory bank SHALL observe the rules regarding format and content that are defined for the corresponding GS1 Application Identifier within the GS1 General Specifications.

5556

5557

5558

Table B (shown below) calculates the number of bits required to encode the value of a string of length L depending on the encoding method selected. This table may be used to avoid the need for floating-point arithmetic calculations.

| | Encoding indicator 000 | Encoding indicator 001 or 010 | Encoding indicator 101 | Encoding indicator 011 | Encoding indicator 100 |
|------------------------------------|---|---|--|--|---|
| L = Number of digits or characters | Integer encoding @ ≈ 3.32 bits / digit | Numeric string encoding @ 4 bits / digit | URN Code 40 encoding @ 16 bits per 3 characters | File-safe base 64 encoding @ 6 bits per character | Truncated ASCII encoding @7 bits per character |
| 1 | 4 | 4 | 16 | 6 | 7 |
| 2 | 7 | 8 | 16 | 12 | 14 |
| 3 | 10 | 12 | 16 | 18 | 21 |
| 4 | 14 | 16 | 32 | 24 | 28 |
| 5 | 17 | 20 | 32 | 30 | 35 |

| | Encoding indicator 000 | Encoding indicator 001 or 010 | Encoding indicator 101 | Encoding indicator 011 | Encoding indicator 100 |
|----|------------------------|-------------------------------|------------------------|------------------------|------------------------|
| 6 | 20 | 24 | 32 | 36 | 42 |
| 7 | 24 | 28 | 48 | 42 | 49 |
| 8 | 27 | 32 | 48 | 48 | 56 |
| 9 | 30 | 36 | 48 | 54 | 63 |
| 10 | 34 | 40 | 64 | 60 | 70 |
| 11 | 37 | 44 | 64 | 66 | 77 |
| 12 | 40 | 48 | 64 | 72 | 84 |
| 13 | 44 | 52 | 80 | 78 | 91 |
| 14 | 47 | 56 | 80 | 84 | 98 |
| 15 | 50 | 60 | 80 | 90 | 105 |
| 16 | 54 | 64 | 96 | 96 | 112 |
| 17 | 57 | 68 | 96 | 102 | 119 |
| 18 | 60 | 72 | 96 | 108 | 126 |
| 19 | 64 | 76 | 112 | 114 | 133 |
| 20 | 67 | 80 | 112 | 120 | 140 |
| 21 | 70 | 84 | 112 | 126 | 147 |
| 22 | 74 | 88 | 128 | 132 | 154 |
| 23 | 77 | 92 | 128 | 138 | 161 |
| 24 | 80 | 96 | 128 | 144 | 168 |
| 25 | 84 | 100 | 144 | 150 | 175 |
| 26 | 87 | 104 | 144 | 156 | 182 |
| 27 | 90 | 108 | 144 | 162 | 189 |
| 28 | 94 | 112 | 160 | 168 | 196 |
| 29 | 97 | 116 | 160 | 174 | 203 |
| 30 | 100 | 120 | 160 | 180 | 210 |
| 31 | 103 | 124 | 176 | 186 | 217 |
| 32 | 107 | 128 | 176 | 192 | 224 |
| 33 | 110 | 132 | 176 | 198 | 231 |
| 34 | 113 | 136 | 192 | 204 | 238 |
| 35 | 117 | 140 | 192 | 210 | 245 |
| 36 | 120 | 144 | 192 | 216 | 252 |
| 37 | 123 | 148 | 208 | 222 | 259 |
| 38 | 127 | 152 | 208 | 228 | 266 |
| 39 | 130 | 156 | 208 | 234 | 273 |
| 40 | 133 | 160 | 224 | 240 | 280 |
| 41 | 137 | 164 | 224 | 246 | 287 |
| 42 | 140 | 168 | 224 | 252 | 294 |
| 43 | 143 | 172 | 240 | 258 | 301 |
| 44 | 147 | 176 | 240 | 264 | 308 |

| | Encoding indicator 000 | Encoding indicator 001 or 010 | Encoding indicator 101 | Encoding indicator 011 | Encoding indicator 100 |
|----|------------------------|-------------------------------|------------------------|------------------------|------------------------|
| 45 | 150 | 180 | 240 | 270 | 315 |
| 46 | 153 | 184 | 256 | 276 | 322 |
| 47 | 157 | 188 | 256 | 282 | 329 |
| 48 | 160 | 192 | 256 | 288 | 336 |
| 49 | 163 | 196 | 272 | 294 | 343 |
| 50 | 167 | 200 | 272 | 300 | 350 |
| 51 | 170 | 204 | 272 | 306 | 357 |
| 52 | 173 | 208 | 288 | 312 | 364 |
| 53 | 177 | 212 | 288 | 318 | 371 |
| 54 | 180 | 216 | 288 | 324 | 378 |
| 55 | 183 | 220 | 304 | 330 | 385 |
| 56 | 187 | 224 | 304 | 336 | 392 |
| 57 | 190 | 228 | 304 | 342 | 399 |
| 58 | 193 | 232 | 320 | 348 | 406 |
| 59 | 196 | 236 | 320 | 354 | 413 |
| 60 | 200 | 240 | 320 | 360 | 420 |
| 61 | 203 | 244 | 336 | 366 | 427 |
| 62 | 206 | 248 | 336 | 372 | 434 |
| 63 | 210 | 252 | 336 | 378 | 441 |
| 64 | 213 | 256 | 352 | 384 | 448 |
| 65 | 216 | 260 | 352 | 390 | 455 |
| 66 | 220 | 264 | 352 | 396 | 462 |
| 67 | 223 | 268 | 368 | 402 | 469 |
| 68 | 226 | 272 | 368 | 408 | 476 |
| 69 | 230 | 276 | 368 | 414 | 483 |
| 70 | 233 | 280 | 384 | 420 | 490 |
| 71 | 236 | 284 | 384 | 426 | 497 |
| 72 | 240 | 288 | 384 | 432 | 504 |
| 73 | 243 | 292 | 400 | 438 | 511 |
| 74 | 246 | 296 | 400 | 444 | 518 |
| 75 | 250 | 300 | 400 | 450 | 525 |
| 76 | 253 | 304 | 416 | 456 | 532 |
| 77 | 256 | 308 | 416 | 462 | 539 |
| 78 | 260 | 312 | 416 | 468 | 546 |
| 79 | 263 | 316 | 432 | 474 | 553 |
| 80 | 266 | 320 | 432 | 480 | 560 |
| 81 | 270 | 324 | 432 | 486 | 567 |
| 82 | 273 | 328 | 448 | 492 | 574 |
| 83 | 276 | 332 | 448 | 498 | 581 |

| | Encoding indicator 000 | Encoding indicator 001 or 010 | Encoding indicator 101 | Encoding indicator 011 | Encoding indicator 100 |
|----|------------------------|-------------------------------|------------------------|------------------------|------------------------|
| 84 | 280 | 336 | 448 | 504 | 588 |
| 85 | 283 | 340 | 464 | 510 | 595 |
| 86 | 286 | 344 | 464 | 516 | 602 |
| 87 | 290 | 348 | 464 | 522 | 609 |
| 88 | 293 | 352 | 480 | 528 | 616 |
| 89 | 296 | 356 | 480 | 534 | 623 |
| 90 | 299 | 360 | 480 | 540 | 630 |

5559

5560 16 Tag Identification (TID) Memory Bank Contents

5561 To conform to this specification, the Tag Identification memory bank (bank 10) SHALL contain an 8
5562 bit ISO/IEC 15963 [ISO15963] allocation class identifier of E2_h at memory locations 00_h to 07_h. TID
5563 memory above location 07_h SHALL be configured as follows:

- 5564 ■ 08_h: XTID (**X**) indicator (whether a Tag implements Extended Tag Identification, XTID)
- 5565 ■ 09_h: Security (**S**) indicator (whether a Tag supports the *Authenticate* and/or *Challenge*
5566 commands)
- 5567 ■ 0A_h: File (**F**) indicator (whether a Tag supports the *FileOpen* command)
- 5568 ■ 0B_h to 13_h: a 9-bit mask-designer identifier (**MDID**) available from GS1
- 5569 ■ 14_h to 1F_h: a 12-bit, Tag-manufacturer-defined Tag Model Number (**TMN**)
- 5570 ■ above 1F_h: as defined in section [16.2](#) below

5571 The Tag model number (TMN) may be assigned any value by the holder of a given MDID. However,
5572 [UHFC1G2] states "TID memory locations above 07_h shall be defined according to the registration
5573 authority defined by this class identifier value and shall contain, at a minimum, sufficient identifying
5574 information for an Interrogator to uniquely identify the custom commands and/or optional features
5575 that a Tag supports." For the allocation class identifier of E2_h this information is the MDID and TMN,
5576 regardless of whether the extended TID is present or not. If two tags differ in custom commands
5577 and/or optional features, they must be assigned different MDID/TMN combinations. In particular, if
5578 two tags contain an extended TID and the values in their respective extended TIDs differ in any
5579 value other than the value of the serial number, they must be assigned a different MDID/TMN
5580 combination. (The serial number by definition must be different for any two tags having the same
5581 MDID and TMN, so that the Serialised Tag Identification specified in Section [16.2.6](#) is globally
5582 unique.) For tags that do not contain an extended TID, it should be possible in principle to use the
5583 MDID and TMN to look up the same information that would be encoded in the extended TID were it
5584 actually present on the tag, and so again a different MDID/TMN combination must be used if two
5585 tags differ in the capabilities as they would be described by the extended TID, were it actually
5586 present.

5587 TID memory locations 00_h to 1F_h SHALL be permalocked at time of manufacture. If the Tag
5588 implements an XTID then the entire XTID SHALL also be permalocked at time of manufacture.

5589 **As of Gen2v3, tags with allocation class identifier E2_h SHALL support a serialised TID by**
5590 **using a unique serial number**, as defined in section 16.2.2 below.

5591 16.1 Short Tag Identification (TID)

5592 If the XTID indicator ("X" bit 08_h of the TID bank) is set to zero, the TID bank only contains the
5593 allocation class identifier, XTID ("X"), Security ("S") and File ("F") indicators, the mask designer
5594 identifier (MDID), and Tag model number (TMN), as specified above. Readers and applications that

5595
5596

are not configured to handle the extended TID will treat all TIDs as short tag identification, regardless of whether the XTID indicator is zero or one.



5597
5598
5599
5600
5601
5602
5603
5604
5605
5606
5607

Note: The memory maps depicted in this document are identical to how they are depicted in [UHFC1G2]. The lowest word address starts at the bottom of the map and increases as you go up the map. The bit address reads from left to right starting with bit zero and ending with bit fifteen. The fields (MDID, TMN, etc) described in the document put their most significant bit (highest bit number) into the lowest bit address in memory and the least significant bit (bit zero) into the highest bit address in memory. Take the ISO/IEC 15963 [ISO15963] allocation class identifier of E2h = 111000102 as an example. The most significant bit of this field is a one and it resides at address 00h of the TID memory bank. The least significant bit value is a zero and it resides at address 07h of the TID memory bank. When tags backscatter data in response to a read command they transmit each word starting from bit address zero and ending with bit address fifteen.

5608

Table 16-1 Short TID format

| TID MEM BANK BIT ADDRESS | BIT ADDRESS WITHIN WORD (In Hexadecimal) | | | | | | | | | | | | | | | |
|----------------------------------|--|---|---|---|------------------------|---|---|---|---|---|---|------------|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 10 _h -1F _h | MDID[3:0] | | | | TAG MODEL NUMBER[11:0] | | | | | | | | | | | |
| 00 _h -0F _h | E2 _h | | | | | | | | X | S | F | MDID [8:4] | | | | |

5609

16.2 Extended Tag identification (XTID)

5610
5611
5612

The XTID is intended to provide more information to end users about the capabilities of tags that are observed in their RFID applications. The XTID extends the format by adding support for serialisation and information about key features implemented by the tag.

5613
5614
5615

If the XTID bit (bit 08_h of the TID bank) is set to one, the TID bank SHALL contain the allocation class identifier, mask designer identifier (MDID), and Tag model number (TMN) as specified above, and SHALL also contain additional information as specified in this section.

5616
5617
5618
5619
5620

If the XTID bit as defined above is one, TID memory locations 20_h to 2F_h SHALL contain a 16-bit XTID header as specified in Section 16.2.1. The values in the XTID header specify what additional information is present in memory locations 30_h and above. TID memory locations 00_h through 2F_h are the only fixed location fields in the extended TID; all fields following the XTID header can vary in their location in memory depending on the values in the XTID header.

5621
5622
5623
5624
5625
5626
5627
5628
5629

The information in the XTID following the XTID header SHALL consist of zero or more multi-word "segments," each segment being divided into one or more "fields," each field providing certain information about the tag as specified below. The XTID header indicates which of the XTID segments the tag mask-designer has chosen to include. The order of the XTID segments in the TID bank shall follow the order that they are listed in the XTID header from most significant bit to least significant bit. If an XTID segment is not present then segments at less significant bits in the XTID header shall move to lower TID memory addresses to keep the XTID memory structure contiguous. In this way a minimum amount of memory is used to provide a serial number and/or describe the features of the tag. A fully populated XTID is shown in the table below.



5630
5631
5632
5633
5634
5635
5636
5637

Non-Normative: The XTID header corresponding to this memory map would be 0011110000000000₂. If the tag only contained a 48 bit serial number the XTID header would be 0010000000000000₂. The serial number would start at bit address 30_h and end at bit address 5F_h. If the tag contained just the BlockWrite and BlockErase segment and the User Memory and BlockPermaLock segment the XTID header would be 0000110000000000₂. The BlockWrite and BlockErase segment would start at bit address 30_h and end at bit address 6F_h. The User Memory and BlockPermaLock segment would start at bit address 70_h and end at bit address 8F_h.

5638
5639

Table 16-2 Non-Normative example: Extended Tag Identification (XTID) format for the TID memory bank

| TDS Reference Section | TID MEM BANK BIT ADDRESS | BIT ADDRESS WITHIN WORD (In Hexadecimal) | | | | | | | | | | | | | | | |
|------------------------|----------------------------------|--|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 16.2.5 | C0 _n -CF _n | User Memory and BlockPermaLock Segment [15:0] | | | | | | | | | | | | | | | |
| | B0 _n -BF _n | User Memory and BlockPermaLock Segment [31:16] | | | | | | | | | | | | | | | |
| 16.2.4 | A0 _n -AF _n | BlockWrite and BlockErase Segment [15:0] | | | | | | | | | | | | | | | |
| | 90 _n -9F _n | BlockWrite and BlockErase Segment [31:16] | | | | | | | | | | | | | | | |
| | 80 _n -8F _n | BlockWrite and BlockErase Segment [47:32] | | | | | | | | | | | | | | | |
| | 70 _n -7F _n | BlockWrite and BlockErase Segment [63:48] | | | | | | | | | | | | | | | |
| 16.2.3 | 60 _n -6F _n | Optional Command Support Segment [15:0] | | | | | | | | | | | | | | | |
| 16.2.2 | 50 _n -5F _n | Serial Number Segment [15:0] | | | | | | | | | | | | | | | |
| | 40 _n -4F _n | Serial Number Segment [31:16] | | | | | | | | | | | | | | | |
| | 30 _n -3F _n | Serial Number Segment [47:32] | | | | | | | | | | | | | | | |
| 16.2.1 | 20 _n -2F _n | XTID Header Segment [15:0] | | | | | | | | | | | | | | | |
| 16.1 | 10 _n -1F _n | Refer to Table 16-1 | | | | | | | | | | | | | | | |
| | 00 _n -0F _n | | | | | | | | | | | | | | | | |

5640
5641
5642
5643
5644

Note that this example depicts the memory mapping when the serialisation bits in the XTID header (see Table 16-3), are set to 001, indicating the XTID Serial Number is 48 bits long. Other settings of the serialisation bits in the XTID header will shift the addresses of the Optional Command Support Segment, the BlockWrite and BlockErase Segment and the User Memory and BlockPermaLock Segment.

5645 **16.2.1 XTID Header**

5646
5647
5648
5649
5650
5651
5652
5653
5654

The XTID header is shown in [Table 16-3](#). It contains defined and reserved for future use (RFU) bits. The extended header bit and RFU bits (bits 9 through 0) shall be set to zero to comply with this version of the specification. Bits 15 through 13 of the XTID header word indicate the presence and size of serialisation on the tag. If they are set to zero then there is no serialisation in the XTID. If they are not zero then there is a tag serial number immediately following the header. The optional features currently in bits 12 through 10 are handled differently. A zero indicates the reader needs to perform a database look up or that the tag does not support the optional feature. A one indicates that the tag supports the optional feature and that the XTID contains the segment describing this feature.

5655
5656

Note that the contents of the XTID header uniquely determine the overall length of the XTID as well as the starting address for each included XTID segment.

5657

Table 16-3 The XTID header

| Bit Position in Word | Field | Description |
|----------------------|-------------------------|---|
| 0 | Extended Header Present | If non-zero, specifies that additional XTID header bits are present beyond the 16 XTID header bits specified herein. This provides a mechanism to extend the XTID in future versions of the EPC Tag Data Standard. This bit SHALL be set to zero to comply with this version of the EPC Tag Data Standard. If zero, specifies that the XTID header only contains the 16 bits defined herein. |
| 1 - 8 | RFU | Reserved for future use. These bits SHALL be zero to comply with this version of the EPC Tag Data Standard |

| Bit Position in Word | Field | Description |
|----------------------|--|---|
| 9 | Lock Bit Segment | If non-zero, specifies that the XTID includes the Lock Bit segment specified in Section 16.2.6. If zero, specifies that the XTID does not include the Lock Bit segment word. |
| 10 | User Memory and Block Perma Lock Segment Present | If non-zero, specifies that the XTID includes the User Memory and Block PermaLock segment specified in Section 16.2.5. If zero, specifies that the XTID does not include the User Memory and Block PermaLock words. |
| 11 | BlockWrite and BlockErase Segment Present | If non-zero, specifies that the XTID includes the BlockWrite and BlockErase segment specified in Section 16.2.4. If zero, specifies that the XTID does not include the BlockWrite and BlockErase words. |
| 12 | Optional Command Support Segment Present | If non-zero, specifies that the XTID includes the Optional Command Support segment specified in Section 16.2.3. If zero, specifies that the XTID does not include the Optional Command Support word. |
| 13 – 15 | Serialisation | If non-zero, specifies that the XTID includes a unique serial number, whose length in bits is $48 + 16(N - 1)$, where N is the value of this field. If zero, specifies that the XTID does not include a unique serial number. As of Gen2v3, tags with allocation class identifier E2 _h SHALL support a serialised TID by using a unique serial number. Bit 15 is the MSB; bit 13 is the LSB. |

5658 **16.2.2 XTID Serialisation**

5659 The length of the XTID serialisation is specified in the XTID header. The managing entity specified
5660 by the tag mask designer ID is responsible for assigning unique serial numbers for each tag model
5661 number. The length of the serial number uses the following algorithm:

- 5662 0: Indicates no serialisation
- 5663 1-7: Length in bits = $48 + ((\text{Value}-1) * 16)$

5664 **16.2.3 Optional Command Support segment**

5665 If bit twelve is set in the XTID header then the following word is added to the XTID. Bit fields that
5666 are left as zero indicate that the tag does not support that feature. The description of the features is
5667 as follows.

5668 **Table 16-4** Optional Command Support XTID Word

| Bit Position in Segment | Field | Description |
|-------------------------|-------------------|--|
| 0-4 | Max EPC Size | This five bit field shall indicate the maximum size that can be programmed into the first five bits of the PC. |
| 5 | Recom Support | If this bit is set, the tag supports recommissioning as specified in [UHFC1G2]. |
| 6 | Access | If this bit is set, it indicates that the tag supports the access command. |
| 7 | Separate Lockbits | If this bit is set, it means that the tag supports lock bits for each memory bank rather than the simplest implementation of a single lock bit for the entire tag. |
| 8 | Auto UMI Support | If this bit is set, it means that the tag automatically sets its user memory indicator bit in the PC word. |
| 9 | PJM Support | If this bit is set, it indicates that the tag supports phase jitter modulation. This is an optional modulation mode supported only in Gen 2 HF tags. |

| Bit Position in Segment | Field | Description |
|-------------------------|--------------------------|---|
| 10 | BlockErase Supported | If set, this indicates that the tag supports the BlockErase command. How the tag supports the BlockErase command is described in Section 16.2.4 . A manufacture may choose to set this bit, but not include the BlockWrite and BlockErase field if how to use the command needs further explanation through a database lookup. |
| 11 | BlockWrite Supported | If set, this indicates that the tag supports the BlockWrite command. How the tag supports the BlockErase command is described in Section 16.2.4 . A manufacture may choose to set this bit, but not include the BlockWrite and BlockErase field if how to use the command needs further explanation through a database lookup. |
| 12 | BlockPermaLock Supported | If set, this indicates that the tag supports the BlockPermaLock command. How the tag supports the BlockPermaLock command is described in Section 16.2.5 . A manufacture may choose to set this bit, but not include the BlockPermaLock and User Memory field if how to use the command needs further explanation through a database lookup. |
| 13-15 | [RFU] | These bits are RFU and should be set to zero. |

5669 **16.2.4 BlockWrite and BlockErase segment**

5670 If bit eleven of the XTID header is set then the XTID shall include the four-word BlockWrite and
 5671 BlockErase segment. To indicate that a command is not supported, the tag shall have all fields
 5672 related to that command set to zero. This SHALL always be the case when the Optional Command
 5673 Support Segment (Section [16.2.3](#)) is present and it indicates that BlockWrite or BlockErase is not
 5674 supported. The descriptions of the fields are as follows.

5675 **Table 16-5** XTID Block Write and Block Erase Information

| Bit Position in Segment | Field | Description |
|-------------------------|--------------------------------------|---|
| 0-7 | Block Write Size | Max block size that the tag supports for the BlockWrite command. This value should be between 1-255 if the BlockWrite command is described in this field. |
| 8 | Variable Size Block Write | This bit is used to indicate if the tag supports BlockWrite commands with variable sized blocks. If the value is zero the tag only supports writing blocks exactly the maximum block size indicated in bits [7-0]. If the value is one the tag supports writing blocks less than the maximum block size indicated in bits [7-0]. |
| 9-16 | Block Write EPC Address Offset | This indicates the starting word address of the first full block that may be written to using BlockWrite in the EPC memory bank. |
| 17 | No Block Write EPC address alignment | This bit is used to indicate if the tag memory architecture has hard block boundaries in the EPC memory bank. If the value is zero the tag has hard block boundaries in the EPC memory bank. The tag will not accept BlockWrite commands that start in one block and end in another block. These block boundaries are determined by the max block size and the starting address of the first full block. All blocks have the same maximum size. If the value is one the tag has no block boundaries in the EPC memory bank. It will accept all BlockWrite commands that are within the memory bank. |
| 18-25 | Block Write User Address Offset | This indicates the starting word address of the first full block that may be written to using BlockWrite in the User memory. |

| Bit Position in Segment | Field | Description |
|-------------------------|---------------------------------------|---|
| 26 | No Block Write User Address Alignment | <p>This bit is used to indicate if the tag memory architecture has hard block boundaries in the USER memory bank.</p> <p>If the value is zero the tag has hard block boundaries in the USER memory bank. The tag will not accept BlockWrite commands that start in one block and end in another block. These block boundaries are determined by the max block size and the starting address of the first full block. All blocks have the same maximum size.</p> <p>If the value is one the tag has no block boundaries in the USER memory bank. It will accept all BlockWrite commands that are within the memory bank.</p> |
| 27-31 | [RFU] | These bits are RFU and should be set to zero. |
| 32-39 | Size of Block Erase | Max block size that the tag supports for the BlockErase command. This value should be between 1-255 if the BlockErase command is described in this field. |
| 40 | Variable Size Block Erase | <p>This bit is used to indicate if the tag supports BlockErase commands with variable sized blocks.</p> <p>If the value is zero the tag only supports erasing blocks exactly the maximum block size indicated in bits [39-32].</p> <p>If the value is one the tag supports erasing blocks less than the maximum block size indicated in bits [39-32].</p> |
| 41-48 | Block Erase EPC Address Offset | This indicates the starting address of the first full block that may be erased in EPC memory bank. |
| 49 | No Block Erase EPC Address Alignment | <p>This bit is used to indicate if the tag memory architecture has hard block boundaries in the EPC memory bank.</p> <p>If the value is zero the tag has hard block boundaries in the EPC memory bank. The tag will not accept BlockErase commands that start in one block and end in another block. These block boundaries are determined by the max block size and the starting address of the first full block. All blocks have the same maximum size.</p> <p>If the value is one the tag has no block boundaries in the EPC memory bank. It will accept all BlockErase commands that are within the memory bank.</p> |
| 50-57 | Block Erase User Address Offset | This indicates the starting address of the first full block that may be erased in User memory bank. |
| 58 | No Block Erase User Address Alignment | <p>Bit 58: This bit is used to indicate if the tag memory architecture has hard block boundaries in the USER memory bank.</p> <p>If the value is zero the tag has hard block boundaries in the USER memory bank. The tag will not accept BlockErase commands that start in one block and end in another block. These block boundaries are determined by the max block size and the starting address of the first full block. All blocks have the same maximum size.</p> <p>If the value is one the tag has no block boundaries in the USER memory bank. It will accept all BlockErase commands that are within the memory bank.</p> |
| 59-63 | [RFU] | These bits are reserved for future use and should be set to zero. |

5676 **16.2.5 User Memory and BlockPermaLock segment**

5677 This two-word segment is present in the XTID if bit 10 of the XTID header is set. Bits 15-0 shall
 5678 indicate the size of user memory in words. Bits 31-16 shall indicate the size of the blocks in the
 5679 USER memory bank in words for the BlockPermaLock command. Note: These block sizes only apply
 5680 to the BlockPermaLock command and are independent of the BlockWrite and BlockErase commands.

5681

Table 16-6 XTID Block PermaLock and User Memory Information

| Bit Position in Segment | Field | Description |
|-------------------------|---------------------------|---|
| 0-15 | User Memory Size | Number of 16-bit words in user memory. |
| 16-31 | BlockPermaLock Block Size | <p>If non-zero, the size in words of each block that may be block perma-locked. That is, the block perma-lock feature allows blocks of $N \times 16$ bits to be locked, where N is the value of this field.</p> <p>If zero, then the XTID does not describe the block size for the BlockPermaLock feature. The tag may or may not support block perma-locking.</p> <p>This field SHALL be zero if the Optional Command Support Segment (Section 16.2.3) is present and its BlockPermaLockSupported bit is zero.</p> |

5682

16.2.6 Optional Lock Bit segment

This one-word segment is present in the XTID if bit 9 of the XTID header is set. Bits 0-5 shall indicate the current lock bit settings for the memory banks on the tag.

5683

5684

5685

Table 16-7 Lock Bit Information

| Bit Position in Segment | Field | Description |
|-------------------------|----------------------------|---|
| 0 | File_0 memory (perma-lock) | <p>The lock bits are defined by the Lock command in the air protocol specification available at https://www.gs1.org/standards/epc-rfid/uhf-air-interface-protocol</p> |
| 1 | File_0 memory (pwd write) | |
| 2 | TID memory (perma-lock) | |
| 3 | TID memory (pwd write) | |
| 4 | EPC memory (perma-lock) | |
| 5 | EPC memory (pwd writ-) | |
| 6-15 | [RFU] | These bits are reserved for future use and should be set to zero. |

5686

16.3 Serialised Tag Identification (STID)

This section specifies a URI form for the serialisation encoded within an XTID, called the Serialised Tag Identifier (STID). The STID URI form may be used by business applications that use the serialised TID to uniquely identify the tag onto which an EPC has been programmed. The STID URI is intended to supplement, not replace, the EPC for those applications that make use of RFID tag serialisation in addition to the EPC that uniquely identifies the physical object to which the tag is affixed; e.g., in an application that uses the STID to help ensure a tag has not been counterfeited.

5687

5688

5689

5690

5691

5692

5693

16.3.1 STID URI grammar

The syntax of the STID URI is specified by the following grammar:

STID-URI = %s"urn:epc:stid:" 2(%s"x" HexComponent ".") %s"x" HexComponent

where the first and second HexComponents SHALL consist of exactly three UpperHexChars and the third HexComponent SHALL consist of 12, 16, 20, 24, 28, 32, or 36 UpperHexChars.

The first HexComponent is the value of bits 08h-13h. For tags using the Gen2 v1.x air interface, this consists of the 12-bit Tag Mask Designer ID (MDID); for tags using Gen2 v2 and later versions of the air interface, these twelve bits consist of the three X, S and F indicators (bits 08h-0Ah), followed by the 9-bit MDID (bits 0Bh-13h) as specified in Section 16.1.

The second HexComponent is the value of the Tag Model Number as specified in Section 16.1.

5698

5699

5700

5701

5702

5703 The third `HexComponent` is the value of the XTID serial number as specified in Sections
 5704 [16.2.1](#) and [16.2.2](#). The number of `UpperHexChars` in the third `HexComponent` is equal to the
 5705 number of bits in the XTID serial number divided by four.

5706 **16.3.2 Decoding procedure: TID Bank Contents to STID URI**

5707 The following procedure specifies how to construct an STID URI given the contents of the TID bank
 5708 of a Gen 2 Tag.

5709 **Given:**

- 5710 ■ The contents of the TID memory bank of a Gen 2 Tag, as a bit string $b_0b_1\dots b_{N-1}$, where the
 5711 number of bits N is at least 48.

5712 **Yields:**

- 5713 ■ An STID-URI

5714 **Procedure:**

- 5715 1. Bits $b_0\dots b_7$ should match the value 11100010. If not, stop: this TID bank contents does not
 5716 contain a TDS-compliant XTID.
- 5717 2. Bit b_8 should be set to one. If not, stop: this TID bank contents does not contain a TDS-
 5718 compliant XTID.
- 5719 3. Consider bits $b_8\dots b_{19}$ as a 12-bit unsigned integer. For tags using the Gen2 v1.x air interface,
 5720 this consists of the 12-bit Tag Mask Designer ID (MDID); for tags using Gen2 v2 and later
 5721 versions of the air interface, these twelve bits consist of the three X, S and F indicators
 5722 (b_8, b_9, b_{10}), followed by the 9-bit MDID ($b_{11}\dots b_{19}$).
- 5723 4. Consider bits $b_{20}\dots b_{31}$ as a 12-bit unsigned integer. This is the Tag Model Number.
- 5724 5. Consider bits $b_{32}\dots b_{34}$ as a 3-bit unsigned integer V . If V equals zero, stop: this TID bank
 5725 contents does not contain a serial number. Otherwise, calculate the length of the serial number
 5726 $L = 4 + 16(V - 1)$. Consider bits $b_{48}b_{49}\dots b_{48+L-1}$ as an L -bit unsigned integer. This is the serial
 5727 number.
- 5728 6. Construct the STID-URI by concatenating the following strings: the prefix `urn:epc:stid:`, the
 5729 lowercase letter `x`, the value of $b_8\dots b_{19}$ from Step 3 as a 3-character hexadecimal numeral, a dot
 5730 (`.`) character, the lowercase letter `x`, the value of the Tag Model Number from Step 4 as a 3-
 5731 character hexadecimal numeral, a dot (`.`) character, the lowercase letter `x`, and the value of the
 5732 serial number from Step 5 as a $(L/4)$ -character hexadecimal numeral. Only uppercase letters `A`
 5733 through `F` shall be used in constructing the hexadecimal numerals.

5734 **17 User Memory Bank Contents**

5735 The User Memory Bank provides a variable size memory to store additional data attributes related to
 5736 the object identified in the EPC Memory Bank of the tag.

5737 User memory may or may not be present on a given tag. The User Memory Indicator (UMI), within
 5738 the PC bits, is specified in section [9.3](#).

5739 To conform with this specification, the first eight bits of the User Memory Bank SHALL contain a
 5740 Data Storage Format Identifier (DSFID) as specified in [ISO15962]. This maintains compatibility
 5741 with other standards. The DSFID consists of three logical fields: Access Method, Extended Syntax
 5742 Indicator, and Data Format. The Access Method is specified in the two most significant bits of the
 5743 DSFID, and is encoded with the value "10" to designate the "Packed Objects" Access Method as
 5744 specified in Annex I herein if the "Packed Objects" Access Method is employed, and is encoded with
 5745 the value "00" to designate the "No-Directory" Access Method as specified in [ISO15962] if the "No-
 5746 Directory" Access Method is employed. The next bit is set to one if there is a second DSFID byte
 5747 present. The five least significant bits specify the Data Format, which indicates what data system
 5748 predominates in the memory contents. If GS1 Application Identifiers (AIs) predominate, the value of
 5749 "01001" specifies the GS1 Data Format 9 as registered with ISO, which provides most efficient

5750 support for the use of AI data elements. Annex I through Annex M of this specification contain the
 5751 complete specification of the "Packed Objects" Access Method; this content appears in ISO/IEC
 5752 15962 [ISO15962] as Annex [I](#) through [M](#), respectively,. A complete definition of the DSFID is
 5753 specified in [ISO15962]. A complete definition of the table that governs the Packed Objects
 5754 encoding of Application Identifiers (AIs) is specified by GS1 and registered with ISO under the
 5755 procedures of [ISO15962], and is reproduced in [E.3](#). This table is similar in format to the
 5756 hypothetical example shown as Table L-1 in [L](#), but with entries to accommodate encoding of all valid
 5757 Application Identifiers.


5758 A tag whose User Memory Bank programming conforms to this specification SHALL be encoded
 5759 using either the Packed Objects Access Method or the No-Directory Access Method, provided that if
 5760 the No-Directory Access Method is used that the "application-defined" compaction mode as specified
 5761 in [ISO15962] SHALL NOT be used. A tag whose User Memory Bank programming conforms to this
 5762 specification MAY use any registered Data Format including Data Format 9.

5763 An ISO/IEC 20248 [ISO20248] digital signature (to authenticate the tag data) may be stored in
 5764 User Memory encoded as GS1 AI (8030) using Packed Objects (Data Format 9) or natively and more
 5765 efficiently using Data Format 17, since the CIDSnip is encoded as binary data. The CIDSnip
 5766 corresponds to the value of AI (8030) and consists of the [ISO20248] Domain Authority Identifier
 5767 (DAID – the party who is accountable for the digital signature), the Certificate Identifier (CID),
 5768 signature, timestamp and optional client-specific data fields, though these are typically absent. In
 5769 both cases the EPC is included in the signature using the [ISO20248] readmethod pragma. It is
 5770 recommended to include the TID (using the readmethod pragma) in the digital signature to provide
 5771 for tag data copy detection. The [ISO20248] Domain Authority Identifier (DAID – the party who is
 5772 accountable for the digital signature) and the GS1 Party GLN (PGLN) -- corresponding to GS1 AI
 5773 (417) -- are equivalent. Whenever a [ISO20248] digital signature is associated with a GS1 element
 5774 string, the DAID SHALL use the PGLN. See [ISO20248] clause 7.5.

5775 An ISO/IEC 20248 DigSig construct expressed using GS1 Application Identifier (8030) can be most
 5776 efficiently encoded in User Memory using Data Format 17 (rather than Packed Objects using Data
 5777 Format 9) which is a total length of 352 bits when the signing period is one calendar year with a
 5778 resolution of minutes. The length remains the same with any additional data signed, which is placed
 5779 elsewhere, for example an authentication code printed in UV-fluorescent ink or embedded in an
 5780 hologram or watermark. Such data is included in the signature, but not stored in the DigSig
 5781 construct.

5782 Where the Packed Objects specification in [I](#) makes reference to Extensible Bit Vectors (EBVs), the
 5783 format specified in Annex [D](#) SHALL be used.

5784 A hardware or software component that conforms to this specification for User Memory Bank
 5785 reading and writing SHALL fully implement the Packed Objects Access Method as specified in
 5786 Annexes [I](#) through [M](#) of this specification (implying support for all registered Data Formats), SHALL
 5787 implement the No-Directory Access Method as specified in [ISO15962], and MAY implement other
 5788 Access Methods defined in [ISO15962] and subsequent versions of that standard. A hardware or
 5789 software component NEED NOT, however, implement the "application-defined" compaction mode of
 5790 the No-Directory Access Method as specified in [ISO15962]. A hardware or software component
 5791 whose intended function is only to initialise tags (e.g., a printer) may conform to a subset of this
 5792 specification by implementing either the Packed Objects or the No-Directory access method, but in
 5793 this case NEED NOT implement both.

5794  **Non-Normative:** Explanation: This specification allows two methods of encoding data in user
 5795 memory. The ISO/IEC 15962 "No-Directory" Access Method has an installed base owing to its
 5796 longer history and acceptance within certain end user communities. The Packed Objects
 5797 Access Method was developed to provide for more efficient reading and writing of tags, and
 5798 less tag memory consumption.

5799 The "application-defined" compaction mode of the No-Directory Access Method is not allowed
 5800 because it cannot be understood by a receiving system unless both sides have the same
 5801 definition of how the compaction works.

5802 Note that the Packed Objects Access Method supports the encoding of data either with or
 5803 without a directory-like structure for random access. The fact that the other access method is

5804 named "No-Directory" in [ISO15962] should not be taken to imply that the Packed Objects
5805 Access Method always includes a directory.

5806 **18 Conformance**

5807 TDS by its nature has an impact on many parts of the GS1 System Architecture. Unlike other
5808 standards that define a specific hardware or software interface, TDS defines data formats, along
5809 with procedures for converting between equivalent formats. Both the data formats and the
5810 conversion procedures are employed by a variety of hardware, software, and data components in
5811 any given system.

5812 This section defines what it means to conform to TDs. As noted above, there are many types of
5813 system components that have the potential to conform to various parts of the TDS, and these are
5814 enumerated below.

5815 **18.1 Conformance of RFID Tag Data**

5816 The data programmed on a Gen 2 RFID tag may be in conformance with TDS as specified below.
5817 Conformance may be assessed separately for the contents of each memory bank.

5818 Each memory bank may be in an "uninitialised" state or an "initialised" state. The uninitialised state
5819 indicates that the memory bank contains no data, and is typically only used between the time a tag
5820 is manufactured and the time it is first programmed for use by an application. The conformance
5821 requirements are given separately for each state, where applicable.

5822 **18.1.1 Conformance of Reserved Memory Bank (Bank 00)**

5823 The contents of the Reserved memory bank (Bank 00) of a Gen 2 tag is not subject to conformance
5824 to the EPC Tag Data Standard. The contents of the Reserved memory bank is specified in
5825 [UHFC1G2].

5826 **18.1.2 Conformance of EPC Memory Bank (Bank 01)**

5827 The contents of the EPC memory bank (Bank 01) of a Gen 2 tag are subject to conformance to the
5828 EPC Tag Data Standard (TDS) as follows.

5829 The contents of the EPC memory bank conform to TDS in the uninitialised state if all of the following
5830 are true:

- 5831 ■ Bit 17_h SHALL be set to zero.
- 5832 ■ Bits 18_h through 1F_h (inclusive), the Attribute bits, SHALL be set to zero.
- 5833 ■ Bits 20_h through 27_h (inclusive) SHALL be set to zero, indicating an uninitialised EPC Memory
5834 Bank.
- 5835 ■ All other bits of the EPC memory bank SHALL be as specified in Section 9 and/or [UHFC1G2], as
5836 applicable.

5837 The contents of the EPC memory bank conform to TDS in the initialised state if all of the following
5838 are true:

- 5839 ■ Bit 17_h SHALL be set to zero.
- 5840 ■ Bits 18_h through 1F_h (inclusive), the Attribute bits, SHALL be as specified in Sections 9.3 and 0.
- 5841 ■ Bits 20_h through 27_h (inclusive) SHALL be set to a valid EPC header value as specified in [Table](#)
5842 [14-1](#) that is, a header value not marked as "reserved" or "unprogrammed tag" in the table.
- 5843 ■ Let N be the value of the "encoding length" column of the row of [Table 14-1](#) corresponding to
5844 the header value, and let M be equal to 20_h + N - 1. Bits 20_h through M SHALL be a valid EPC
5845 binary encoding; that is, the decoding procedure of Section [14.3.7](#) when applied to these bits
5846 SHALL NOT raise an exception.

- 5847
5848
- Bits M+1 through the end of the EPC memory bank or bit 20F_h (whichever occurs first) SHALL be set to zero.
- 5849
5850
- All other bits of the EPC memory bank SHALL be as specified in Section 9 and/or [UHFC1G2], as applicable.

5851
5852
5853

! **Non-Normative:** Explanation: A consequence of the above requirements is that to conform to this specification, no additional application data (such as a second EPC) may be put in the EPC memory bank beyond the EPC that begins at bit 20_h.

5854 **18.1.3 Conformance of TID Memory Bank (Bank 10)**

5855 The contents of the TID memory bank (Bank 10) of a Gen 2 tag is subject to conformance to TDS,
5856 as specified in Section 16.

5857 **18.1.4 Conformance of User Memory Bank (Bank 11)**

5858 The contents of the User memory bank (Bank 11) of a Gen 2 tag is subject to conformance to TDS,
5859 as specified in Section 17.

5860 **18.2 Conformance of Hardware and Software Components**

5861 Hardware and software components may process data that is read from or written to Gen 2 RFID
5862 tags. Hardware and software components may also manipulate Electronic Product Codes in various
5863 forms regardless of whether RFID tags are involved. All such uses may be subject to conformance to
5864 TDS as specified below. Exactly what is required to conform depends on what the intended or
5865 claimed function of the hardware or software component is.

5866 **18.2.1 Conformance of hardware and software Components That Produce or Consume**
5867 **Gen 2 Memory Bank Contents**

5868 This section specifies conformance of hardware and software components that produce and consume
5869 the contents of a memory bank of a Gen 2 tag. This includes components that interact directly with
5870 tags via the Gen 2 Air Interface as well as components that manipulate a software representation of
5871 raw memory contents

5872 **Definitions:**

- 5873
5874
5875
5876
5877
5878
- **Bank X Consumer** (where X is a specific memory bank of a Gen 2 tag): A hardware or software component that accepts as input via some external interface the contents of Bank X of a Gen 2 tag. This includes components that read tags via the Gen 2 Air Interface (i.e., readers), as well as components that manipulate a software representation of raw memory contents (e.g., "middleware" software that receives a hexadecimal-formatted image of tag memory from an interrogator as input).
 - **Bank X Producer** (where X is a specific memory bank of a Gen 2 tag): A hardware or software component that outputs via some external interface the contents of Bank X of a Gen 2. This includes components that interact directly with tags via the Gen 2 Air Interface (i.e., write-capable interrogators and printers – the memory contents delivered to the tag is an output via the air interface), as well as components that manipulate a software representation of raw memory contents (e.g., software that outputs a "write" command to an interrogator, delivering a hexadecimal-formatted image of tag memory as part of the command).

5886 A hardware or software component that "passes through" the raw contents of tag memory Bank X
5887 from one external interface to another is simultaneously a Bank X Consumer and a Bank X Producer.
5888 For example, consider a reader device that accepts as input from an application via its network "wire
5889 protocol" a command to write EPC tag memory, where the command includes a hexadecimal-
5890 formatted image of the tag memory that the application wishes to write, and then writes that image
5891 to a tag via the Gen 2 Air Interface. That device is a Bank 01 Consumer with respect to its "wire
5892 protocol," and a Bank 01 Producer with respect to the Gen 2 Air Interface. The conformance

5893 requirements below insure that such a device is capable of accepting from an application and writing
 5894 to a tag any EPC bank contents that is valid according to this specification.

5895 The following conformance requirements apply to Bank X Consumers and Producers as defined
 5896 above:

- 5897 ■ A Bank 01 (EPC bank) Consumer SHALL accept as input any memory contents that conforms to
 5898 this specification, as conformance is specified in Section [18.1.2](#).
- 5899 ■ If a Bank 01 Consumer interprets the contents of the EPC memory bank received as input, it
 5900 SHALL do so in a manner consistent with the definitions of EPC memory bank contents in this
 5901 specification.
- 5902 ■ A Bank 01 (EPC bank) Producer SHALL produce as output memory contents that conforms to
 5903 this specification, as conformance is specified in Section [18.1.2](#), whenever the hardware or
 5904 software component produces output for Bank 01 containing an EPC. A Bank 01 Producer MAY
 5905 produce output containing a non-EPC if it sets bit 17_h to one.
- 5906 ■ If a Bank 01 Producer constructs the contents of the EPC memory bank from component parts,
 5907 it SHALL do so in a manner consistent with this.
- 5908 ■ A Bank 10 (TID Bank) Consumer SHALL accept as input any memory contents that conforms to
 5909 this specification, as conformance is specified in Section [18.1.3](#).
- 5910 ■ If a Bank 10 Consumer interprets the contents of the TID memory bank received as input, it
 5911 SHALL do so in a manner consistent with the definitions of TID memory bank contents in this
 5912 specification.
- 5913 ■ A Bank 10 (TID bank) Producer SHALL produce as output memory contents that conforms to
 5914 this specification, as conformance is specified in Section [18.1.3](#).
- 5915 ■ If a Bank 10 Producer constructs the contents of the TID memory bank from component parts, it
 5916 SHALL do so in a manner consistent with this specification.
- 5917 ■ Conformance for hardware or software components that read or write the User memory bank
 5918 (Bank 11) SHALL be as specified in Section [17](#).

5919 **18.2.2 Conformance of hardware and software Components that Produce or Consume**
 5920 **URI Forms of the EPC**

5921 This section specifies conformance of hardware and software components that use URIs as specified
 5922 herein as inputs or outputs.

5923 **Definitions:**

- 5924 ■ **EPC URI Consumer:** A hardware or software component that accepts an EPC URI as input via
 5925 some external interface. An EPC URI Consumer may be further classified as a Pure Identity URI
 5926 EPC Consumer if it accepts an EPC Pure Identity URI as an input, or an EPC Tag/Raw URI
 5927 Consumer if it accepts an EPC Tag URI or EPC Raw URI as input.
- 5928 ■ **EPC URI Producer:** A hardware or software component that produces an EPC URI as output via
 5929 some external interface. An EPC URI Producer may be further classified as a Pure Identity URI
 5930 EPC Producer if it produces an EPC Pure Identity URI as an output, or an EPC Tag/Raw URI
 5931 Producer if it produces an EPC Tag URI or EPC Raw URI as output.

5932 A given hardware or software component may satisfy more than one of the above definitions, in
 5933 which case it is subject to all of the relevant conformance tests below.

5934 **The following conformance requirements apply to Pure Identity URI EPC Consumers:**

- 5935 ■ A Pure Identity URI EPC Consumer SHALL accept as input any string that satisfies the grammar
 5936 of Section [6](#), including all constraints on the number of characters in various components.
- 5937 ■ A Pure Identity URI EPC Consumer SHALL reject as invalid any input string that begins with the
 5938 characters `urn:epc:id:` that does not satisfy the grammar of Section [6](#), including all
 5939 constraints on the number of characters in various components.

5940
 5941
 5942

- If a Pure Identity URI EPC Consumer interprets the contents of a Pure Identity URI, it SHALL do so in a manner consistent with the definitions of the Pure Identity EPC URI in this specification and the specifications referenced herein (including the GS1 General Specifications).

5943

The following conformance requirements apply to Pure Identity URI EPC Producers:

 5944
 5945

- A Pure Identity EPC URI Producer SHALL produce as output strings that satisfy the grammar in Section 6, including all constraints on the number of characters in various components.

 5946
 5947
 5948

- A Pure Identity EPC URI Producer SHALL NOT produce as output a string that begins with the characters `urn:epc:id:` that does not satisfy the grammar of Section 6, including all constraints on the number of characters in various components.

 5949
 5950

- If a Pure Identity EPC URI Producer constructs a Pure Identity EPC URI from component parts, it SHALL do so in a manner consistent with this specification.

5951

The following conformance requirements apply to EPC Tag/Raw URI Consumers:

 5952
 5953
 5954

- An EPC Tag/Raw URI Consumer SHALL accept as input any string that satisfies the `TagURI` production of the grammar of Section 12.4, and that can be encoded according to Section 14.3 without causing an exception.

 5955
 5956

- An EPC Tag/Raw URI Consumer MAY accept as input any string that satisfies the `RawURI` production of the grammar of Section 12.4.

 5957
 5958
 5959

- An EPC Tag/Raw URI Consumer SHALL reject as invalid any input string that begins with the characters `urn:epc:tag:` that does not satisfy the grammar of Section 12.4, or that causes the encoding procedure of Section 14.3 to raise an exception.

 5960
 5961
 5962

- An EPC Tag/Raw URI Consumer that accepts EPC Raw URIs as input SHALL reject as invalid any input string that begins with the characters `urn:epc:raw:` that does not satisfy the grammar of Section 12.4.

 5963
 5964
 5965
 5966

- To the extent that an EPC Tag/Raw URI Consumer interprets the contents of an EPC Tag URI or EPC Raw URI, it SHALL do so in a manner consistent with the definitions of the EPC Tag URI and EPC Raw URI in this specification and the specifications referenced herein (including the GS1 General Specifications).

5967

The following conformance requirements apply to EPC Tag/Raw URI Producers:

 5968
 5969
 5970
 5971

- An EPC Tag/Raw URI Producer SHALL produce as output strings that satisfy the `TagURI` production or the `RawURI` production of the grammar of Section 12.4, provided that any output string that satisfies the `TagURI` production must be encodable according to the encoding procedure of Section 14.3 without raising an exception.

 5972
 5973

- An EPC Tag/Raw URI Producer SHALL NOT produce as output a string that begins with the characters `urn:epc:tag:` or `urn:epc:raw:` except as specified in the previous bullet.

 5974
 5975

- If an EPC Tag/Raw URI Producer constructs an EPC Tag URI or EPC Raw URI from component parts, it SHALL do so in a manner consistent with this specification.

 5976
 5977

18.2.3 Conformance of hardware and software components that translate between EPC Forms

 5978
 5979
 5980
 5981
 5982

This section specifies conformance for hardware and software components that translate between EPC forms, such as translating an EPC binary encoding to an EPC Tag URI, an EPC Tag URI to a Pure Identity EPC URI, a Pure Identity EPC URI to an EPC Tag URI, or an EPC Tag URI to the contents of the EPC memory bank of a Gen 2 tag. Any such component by definition accepts these forms as inputs or outputs, and is therefore also subject to the relevant parts of Sections 18.2.1 and 18.2.2.

 5983
 5984
 5985

- A hardware or software component that takes the contents of the EPC memory bank of a Gen 2 tag as input and produces the corresponding EPC Tag URI or EPC Raw URI as output SHALL produce an output equivalent to applying the decoding procedure of Section 15.2.2 to the input.

- 5986
5987
5988
- A hardware or software component that takes the contents of the EPC memory bank of a Gen 2 tag as input and produces the corresponding EPC Tag URI or EPC Raw URI as output SHALL produce an output equivalent to applying the decoding procedure of Section [15.2.3](#) to the input.
- 5989
5990
5991
- A hardware or software component that takes an EPC Tag URI as input and produces the corresponding Pure Identity EPC URI as output SHALL produce an output equivalent to applying the procedure of Section [12.3.3](#) to the input.
- 5992
5993
5994
5995
- A hardware or software component that takes an EPC Tag URI as input and produces the contents of the EPC memory bank of a Gen 2 tag as output (whether by actually writing a tag or by producing a software representation of raw memory contents as output) SHALL produce an output equivalent to applying the procedure of Section [15.1.1](#) to the input.

5996
5997

18.3 Conformance of Human Readable Forms of the EPC and of EPC Memory Bank contents

5998
5999
6000
6001
6002

This section specifies conformance for human readable representations of an EPC. Human readable representations may be used on printed labels, in documents, etc. This section does not specify the conditions under which a human readable representation of an EPC or RFID tag contents shall or should be printed on any label, packaging, or other medium; it only specifies what is a conforming human readable representation when it is desired to include one.

- 6003
6004
- To conform to this specification, a human readable representation of an electronic product code SHALL be a Pure Identity EPC URI as specified in Section [6](#).
- 6005
6006
6007
6008
- To conform to this specification, a human readable representation of the entire contents of the EPC memory bank of a Gen 2 tag SHALL be an EPC Tag URI or an EPC Raw URI as specified in Section [12](#). An EPC Tag URI SHOULD be used when it is possible to do so (that is, when the memory bank contents contains a valid EPC).

6009
6010
6011
6012
6013
6014
6015
6016
6017
6018
6019
6020
6021
6022

A Character Set for Alphanumeric Serial Numbers

The following table specifies the characters that are permitted by the GS1 General Specifications [GS1GS] for use in alphanumeric serial numbers. The columns are as follows:

- **Graphic symbol:** The printed representation of the character as used in human-readable forms.
- **Name:** The common name for the character
- **Hex Value:** A hexadecimal numeral that gives the 7-bit binary value for the character as used in EPC binary encodings. This hexadecimal value is always equal to the ISO/IEC 646 [ISO646] (ASCII) code for the character.
- **URI Form:** The representation of the character within Pure Identity EPC URI and EPC Tag URI forms. This is either a single character whose ASCII code is equal to the value in the "hex value" column, or an escape triplet consisting of a percent character followed by two characters giving the hexadecimal value for the character.

Table I.3.1-1 Characters Permitted in Alphanumeric Serial Numbers

| Graphic symbol | Name | Hex Value | URI Form | Graphic symbol | Name | Hex Value | URI Form |
|----------------|-------------------|-----------|----------|----------------|------------------|-----------|----------|
| ! | Exclamation Mark | 21 | ! | M | Capital Letter M | 4D | M |
| " | Quotation Mark | 22 | %22 | N | Capital Letter N | 4E | N |
| % | Percent Sign | 25 | %25 | O | Capital Letter O | 4F | O |
| & | Ampersand | 26 | %26 | P | Capital Letter P | 50 | P |
| ' | Apostrophe | 27 | ' | Q | Capital Letter Q | 51 | Q |
| (| Left Parenthesis | 28 | (| R | Capital Letter R | 52 | R |
|) | Right Parenthesis | 29 |) | S | Capital Letter S | 53 | S |
| * | Asterisk | 2A | * | T | Capital Letter T | 54 | T |
| + | Plus sign | 2B | + | U | Capital Letter U | 55 | U |
| , | Comma | 2C | , | V | Capital Letter V | 56 | V |
| - | Hyphen/ Minus | 2D | - | W | Capital Letter W | 57 | W |
| . | Full Stop | 2E | . | X | Capital Letter X | 58 | X |
| / | Solidus | 2F | %2F | Y | Capital Letter Y | 59 | Y |
| 0 | Digit Zero | 30 | 0 | Z | Capital Letter Z | 5A | Z |
| 1 | Digit One | 31 | 1 | _ | Low Line | 5F | _ |
| 2 | Digit Two | 32 | 2 | a | Small Letter a | 61 | a |
| 3 | Digit Three | 33 | 3 | b | Small Letter b | 62 | b |

| Graphic symbol | Name | Hex Value | URI Form | Graphic symbol | Name | Hex Value | URI Form |
|----------------|-------------------|-----------|----------|----------------|----------------|-----------|----------|
| 4 | Digit Four | 34 | 4 | c | Small Letter c | 63 | c |
| 5 | Digit Five | 35 | 5 | d | Small Letter d | 64 | d |
| 6 | Digit Six | 36 | 6 | e | Small Letter e | 65 | e |
| 7 | Digit Seven | 37 | 7 | f | Small Letter f | 66 | f |
| 8 | Digit Eight | 38 | 8 | g | Small Letter g | 67 | g |
| 9 | Digit Nine | 39 | 9 | h | Small Letter h | 68 | h |
| : | Colon | 3A | : | i | Small Letter i | 69 | i |
| ; | Semicolon | 3B | ; | j | Small Letter j | 6A | j |
| < | Less-than Sign | 3C | %3C | k | Small Letter k | 6B | k |
| = | Equals Sign | 3D | = | l | Small Letter l | 6C | l |
| > | Greater-than Sign | 3E | %3E | m | Small Letter m | 6D | m |
| ? | Question Mark | 3F | %3F | n | Small Letter n | 6E | n |
| A | Capital Letter A | 41 | A | o | Small Letter o | 6F | o |
| B | Capital Letter B | 42 | B | p | Small Letter p | 70 | p |
| C | Capital Letter C | 43 | C | q | Small Letter q | 71 | q |
| D | Capital Letter D | 44 | D | r | Small Letter r | 72 | r |
| E | Capital Letter E | 45 | E | s | Small Letter s | 73 | s |
| F | Capital Letter F | 46 | F | t | Small Letter t | 74 | t |
| G | Capital Letter G | 47 | G | u | Small Letter u | 75 | u |
| H | Capital Letter H | 48 | H | v | Small Letter v | 76 | v |
| I | Capital Letter I | 49 | I | w | Small Letter w | 77 | w |
| J | Capital Letter J | 4A | J | x | Small Letter x | 78 | x |
| K | Capital Letter K | 4B | K | y | Small Letter y | 79 | y |
| L | Capital Letter L | 4C | L | z | Small Letter z | 7A | z |

6023
6024

B Glossary (non-normative)

 Please refer to the www.gs1.org/glossary for the latest version of the glossary.

| Term | Defined Where | Meaning |
|--------------------------------|--|---|
| Application Identifier (AI) | [GS1GS] | A numeric code that identifies a data element within a GS1 element string. |
| Attribute Bits | Sections 9.3 and 0 | An 8-bit field of control information that is stored in the EPC Memory Bank of a Gen 2 RFID Tag when the tag contains an EPC. The Attribute Bits includes data that guides the handling of the object to which the tag is affixed, for example a bit that indicates the presence of hazardous material. |
| Barcode | | A data carrier that holds text data in the form of light and dark markings which may be read by an optical reader device. |
| Control Information | Section 9.1 | Information that is used by data capture applications to help control the process of interacting with RFID Tags. Control Information includes data that helps a capturing application filter out tags from large populations to increase read efficiency, special handling information that affects the behaviour of capturing application, information that controls tag security features, and so on. Control Information is typically <i>not</i> passed directly to business applications, though Control Information may influence how a capturing application presents business data to the business application level. Unlike Business Data, Control Information has no equivalent in bar codes or other data carriers. |
| Data Carrier | | Generic term for a marking or device that is used to physically attach data to a physical object. Examples of data carriers include Bar Codes and RFID Tags. |
| Electronic Product Code (EPC) | Section 4 | <p>A universal identifier for any physical object. The EPC is designed so that every physical object of interest to information systems may be given an EPC that is globally unique and persistent through time.</p> <p>The primary representation of an EPC was previously in the form of a Pure Identity EPC URI (<i>q.v.</i>), which is a unique string that may be used in information systems, electronic messages, databases, and other contexts. A secondary representation, the EPC Binary Encoding (<i>q.v.</i>) is available for use in RFID Tags and other settings where a compact binary representation is required.</p> <p>Starting in TDS 2.0 and EPCIS 2.0 / CBV 2.0, there is now recognition that a GS1 Digital Link URI (or a constrained subset of these, specifically at instance-level granularity and without additional data attributes) is an equivalent way to denote a specific physical object within business applications and traceability data, with a number of advantages, such as ease of linking/redirection to multiple kinds of online information and services, making use of multiple link types and the resolver infrastructure for GS1 Digital Link. GS1 Digital Link URIs can also be used as identifiers within machine-interpretable Linked Data that expresses factual claims.</p> |
| EPC | Section 4 | See Electronic Product Code |
| EPC Bank (of a Gen 2 RFID Tag) | [UHFC1G2] | Bank 01 of a Gen 2 RFID Tag as specified in [UHFC1G2]. The EPC Bank holds the EPC Binary Encoding of an EPC, together with additional control information as specified in Section 7.11 . |
| EPC Binary Encoding | Section 13 | A compact encoding of an Electronic Product Code, together with a filter value (if the encoding scheme includes a filter value), into a binary bit string that is suitable for storage in RFID Tags, including the EPC Memory Bank of a Gen 2 RFID Tag. Owing to trade-offs between data capacity and the number of bits in the encoded value, more than one binary encoding scheme exists for certain EPC schemes. |

| Term | Defined Where | Meaning |
|---|------------------------------|---|
| EPC Binary Encoding Scheme | Section 13 | A particular format for the encoding of an Electronic Product Code, together with a Filter Value in some cases, into an EPC Binary Encoding. Each EPC Scheme has at least one corresponding EPC Binary Encoding Scheme. from a specified combination of data elements. Owing to trade-offs between data capacity and the number of bits in the encoded value, more than one binary encoding scheme exists for certain EPC schemes. An EPC Binary Encoding begins with an 8-bit header that identifies which binary encoding scheme is used for that binary encoding; this serves to identify how the remainder of the binary encoding is to be interpreted. |
| EPC Pure Identity URI | Section 6 | See Pure Identity EPC URI. |
| EPC Raw URI | Section 12 | A representation of the complete contents of the EPC Memory Bank of a Gen 2 RFID Tag, |
| EPC Scheme | Section 6 | A particular format for the construction of an Electronic Product Code from a specified combination of data elements. A Pure Identity EPC URI begins with the name of the EPC Scheme used for that URI, which both serves to ensure global uniqueness of the complete URI as well as identify how the remainder of the URI is to be interpreted. Each type of GS1 key has a corresponding EPC Scheme that allows for the construction of an EPC that corresponds to the value of a GS1 key, under certain conditions. Other EPC Schemes exist that allow for construction of EPCs not related to GS1 keys. |
| EPC Tag URI | Section 12 | 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. Because the EPC Tag URI represents the complete contents of the EPC Memory Bank, it includes control information in addition to the EPC, in contrast to the Pure Identity EPC URI. |
| Extended Tag Identification (XTID) | Section 16 | Information that may be included in the TID Bank of a Gen 2 RFID Tag in addition to the make and model information. The XTID may include a manufacturer-assigned unique serial number and may also include other information that describes the capabilities of the tag. |
| Filter Value | Section 10 | 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. |
| Gen 2 RFID Tag | Section 7.11 | An RFID Tag that conforms to one of the EPCglobal Gen 2 family of air interface protocols. This includes the UHF Class 1 Gen 2 Air Interface [UHFC1G2], and other standards currently under development within GS1. |
| GS1 Company Prefix | [GS1GS] | Part of the GS1 System identification number consisting of a GS1 Prefix and a Company Number, both of which are allocated by GS1 Member Organisations. |
| GS1 element string | [GS1GS] | The combination of a GS1 Application Identifier and GS1 Application Identifier Data Field. |
| GS1 key | [GS1GS] | A generic term for identification keys defined in the GS1 General Specifications [GS1GS], namely the GTIN, SSCC, GLN, GRAI, GIAI, GSRN, GDTI, GSIN, GINC, CPID, GCN and GMN. |
| Pure Identity EPC URI | Section 6 | 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. |
| Radio-Frequency Identification (RFID) Tag | | A data carrier that holds binary data, which may be affixed to a physical object, and which communicates the data to a interrogator ("reader") device through radio. |
| Reserved Bank (of a Gen 2 RFID Tag) | [UHFC1G2] | Bank 00 of a Gen 2 RFID Tag as specified in [UHFC1G2]. The Reserved Bank holds the access password and the kill password. |

| Term | Defined Where | Meaning |
|--|-------------------------|--|
| Tag Identification (TID) | [UHFC1G2] | Information that describes a Gen 2 RFID Tag itself, as opposed to describing the physical object to which the tag is affixed. The TID includes an indication of the make and model of the tag, and may also include Extended TID (XTID) information. |
| TID Bank (of a Gen 2 RFID Tag) | [UHFC1G2] | Bank 10 of a Gen 2 RFID Tag as specified in [UHFC1G2]. The TID Bank holds the TID and XTID (<i>q.v.</i>). |
| Uniform Resource Identifier (URI) | [RFC3986] | A compact sequence of characters that identifies an abstract or physical resource. A URI may be further classified as a Uniform Resource Name (URN) or a Uniform Resource Locator (URL), <i>q.v.</i> |
| Uniform Resource Locator (URL) | [RFC3986] | A Uniform Resource Identifier (URI) that, in addition to identifying a resource, provides a means of locating the resource by describing its primary access mechanism (e.g., its network "location"). |
| Uniform Resource Name (URN) | [RFC3986], [RFC2141] | A Uniform Resource Identifier (URI) that is part of the <code>urn</code> scheme as specified by [RFC2141]. Such URIs refer to a specific resource independent of its network location or other method of access, or which may not have a network location at all. The term URN may also refer to any other URI having similar properties. Because an Electronic Product Code is a unique identifier for a physical object that does not necessarily have a network location or other method of access, URNs are used to represent EPCs. |
| User Memory Bank (of a Gen 2 RFID Tag) | [UHFC1G2] | Bank 11 of a Gen 2 RFID Tag as specified in [UHFC1G2]. The User Memory may be used to hold additional business data elements beyond the EPC. |

C References

- 6025
- 6026 [ASN.1] CCITT, "Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)",
6027 CCITT Recommendation X.209, January 1988.
- 6028 [EPCAF] F. Armenio et al, "EPCglobal Architecture Framework," Version 1.7, May 2015,
6029 <https://www.gs1.org/id-keys-epcrfid-epcis/epc-rfid-architecture-framework/1-6>
- 6030 [GS1Arch] "The GS1 System Architecture," GS1 technical document,
6031 http://www.gs1.org/docs/gsmf/architecture/GS1_System_Architecture.pdf
- 6032 [GS1DL] GS1 Digital Link Standard: <https://www.gs1.org/standards/gs1-digital-link>
- 6033 [GS1GS] "GS1 General Specifications", GS1, [https://www.gs1.org/standards/barcodes-epcrfid-id-](https://www.gs1.org/standards/barcodes-epcrfid-id-keys/gs1-general-specifications)
6034 [keys/gs1-general-specifications](https://www.gs1.org/standards/barcodes-epcrfid-id-keys/gs1-general-specifications).
- 6035 [ISO15961] ISO/IEC 15961, "Information technology – Radio frequency identification (RFID) for
6036 item management – Data protocol: application interface".
- 6037 [ISO15962] ISO/IEC 15962, "Information technology – Radio frequency identification (RFID) for
6038 item management – Data protocol: data encoding rules and logical memory functions".
- 6039 [ISO15963] ISO/IEC 15963, "Information technology – Radio frequency identification for item
6040 management – Unique identification for RF tags"
- 6041 [ISO18000-63] ISO/IEC 18000-63, "Information technology – Radio frequency identification for
6042 item management – Part 63: Parameters for air interface communications at 860 MHz to 960 MHz
6043 Type C"
- 6044 [ISO20248] ISO/IEC 20248, "Information technology – Automatic identification and data capture
6045 techniques – Digital signature data structure schema".
- 6046 [ISO646] ISO/IEC 646, "Information technology – ISO 7-bit coded character set for information
6047 interchange"
- 6048 [ISO8859-6] ISO/IEC 8859-6, "Information technology – 8-bit single-byte coded graphic character
6049 sets – Part 6: Latin/Arabic alphabet"
- 6050 [ISODir2] ISO, "Rules for the structure and drafting of International Standards (ISO/IEC Directives,
6051 Part 2: 2001, 4th edition)," July 2002.
- 6052 [RFC2141] R. Moats, "URN Syntax," RFC2141, May 1997, <http://www.ietf.org/rfc/rfc2141>.
- 6053 [RFC3986] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifier (URI): Generic
6054 Syntax," RFC3986, January 2005, <http://www.ietf.org/rfc/rfc3986>.
- 6055 [RFC5234] D. Crocker, P. Overell, "Augmented BNF for Syntax Specifications: ABNF" RFC5234,
6056 January 2008, <http://www.ietf.org/rfc/rfc5234>.
- 6057 [RFC7405] P. Kyzivat, "Case-Sensitive String Support in ABNF" RFC7405, December 2014,
6058 <http://www.ietf.org/rfc/rfc7405>.
- 6059 [ONS] EPCglobal, "EPCglobal Object Naming Service (ONS), Version 1.0.1," EPCglobal Ratified
6060 Standard, May 2008, [http://www.epcglobalinc.org/standards/ons/ons_1_0_1-standard-](http://www.epcglobalinc.org/standards/ons/ons_1_0_1-standard-20080529.pdf)
6061 [20080529.pdf](http://www.epcglobalinc.org/standards/ons/ons_1_0_1-standard-20080529.pdf).
- 6062 [SPEC2000] Air Transport Association, "Spec 2000 E-Business Specification for Materials
6063 Management," May 2009, <http://www.spec2000.com>.
- 6064 [UHFC1G2] EPCglobal, "EPC™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID
6065 Protocol for Communications at 860 MHz – 960 MHz Version 1.2.0," EPCglobal Specification, May
6066 2008, http://www.epcglobalinc.org/standards/uhfc1g2/uhfc1g2_1_2_0-standard-20080511.pdf.
- 6067 [UID] "United States Department of Defense Guide to Uniquely Identifying Items" Version 3.0
6068 (December 2, 2014),
6069 [https://dodprocurementtoolbox.com/cms/sites/default/files/resources/DoD%20Guide%20to%20Uni-](https://dodprocurementtoolbox.com/cms/sites/default/files/resources/DoD%20Guide%20to%20Uniquely%20Identify%20Items%20v3.0.pdf)
6070 [quely%20Identify%20Items%20v3.0.pdf](https://dodprocurementtoolbox.com/cms/sites/default/files/resources/DoD%20Guide%20to%20Uniquely%20Identify%20Items%20v3.0.pdf).
- 6071 [USDOD] "United States Department of Defense Suppliers' Passive RFID Information Guide,"
6072 https://www.acq.osd.mil/log/LOG.AIT.html/DoD_Suppliers_Passive_RFID_Info_Guide_v15update.pdf

6073
6074
6075
6076
6077
6078
6079
6080
6081
6082
6083

D Extensible Bit Vectors

An Extensible Bit Vector (EBV) is a data structure with an extensible data range.

An EBV is an array of blocks. Each block contains a single extension bit followed by a specific number of data bits. If B is the total number of bits in one block, then a block contains B – 1 data bits. The notation EBV-*n* used in this specification indicates an EBV with a block size of *n*; e.g., EBV-8 denotes an EBV with B=8.

The data value represented by an EBV is simply the bit string formed by the data bits as read from left to right, ignoring all extension bits. The last block of an EBV has an extension bit of zero, and all blocks of an EBV preceding the last block (if any) have an extension bit of one.

The following table illustrates different values represented in EBV-6 format and EBV-8 format. Spaces are added to the EBVs for visual clarity.

| Value | EBV-6 | EBV-8 |
|--------------------|----------------------|----------------------------|
| 0 | 000000 | 00000000 |
| 1 | 000001 | 00000001 |
| 31 (2^5-1) | 011111 | 00011111 |
| 32 (2^5) | 100001 000000 | 00100000 |
| 33 (2^5+1) | 100001 000001 | 00100001 |
| 127 (2^7-1) | 100011 011111 | 01111111 |
| 128 (2^7) | 100100 000000 | 10000001 00000000 |
| 129 (2^7+1) | 100100 000001 | 10000001 00000001 |
| 16384 (2^{14}) | 110000 100000 000000 | 10000001 10000000 00000000 |

The Packed Objects specification in [I](#) makes use of EBV-3, EBV-6, and EBV-8.

6084

E (non-normative) Examples: EPC encoding and decoding

This section presents two detailed examples showing encoding and decoding between the Serialised Global Identification Number (SGTIN) and the EPC memory bank of a Gen 2 RFID tag, and summary examples showing various encodings of all EPC schemes.

As these are merely illustrative examples, in all cases the indicated normative sections of this specification should be consulted for the definitive rules for encoding and decoding. The diagrams and accompanying notes in this section are not intended to be a complete specification for encoding or decoding, but instead serve only to illustrate the highlights of how the normative encoding and decoding procedures function. The procedures for encoding other types of identifiers are different in significant ways, and the appropriate sections of this specification should be consulted.

E.1 Encoding a Serialised Global Trade Item Number (SGTIN) to SGTIN-96

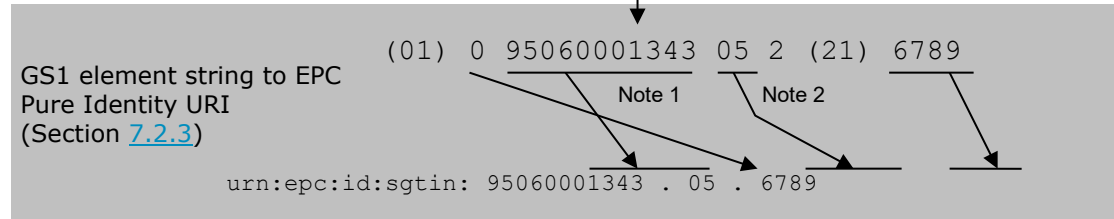
This example illustrates the encoding of a GS1 element string containing a Serialised Global Trade Item Number (SGTIN) into an EPC Gen 2 RFID tag using the SGTIN-96 EPC scheme, with intermediate steps including the EPC URI, the EPC Tag URI, and the EPC Binary Encoding.

In some applications, only a part of this illustration is relevant. For example, an application may only need to transform a GS1 element string into an EPC URI, in which case only the top of the illustration is needed.

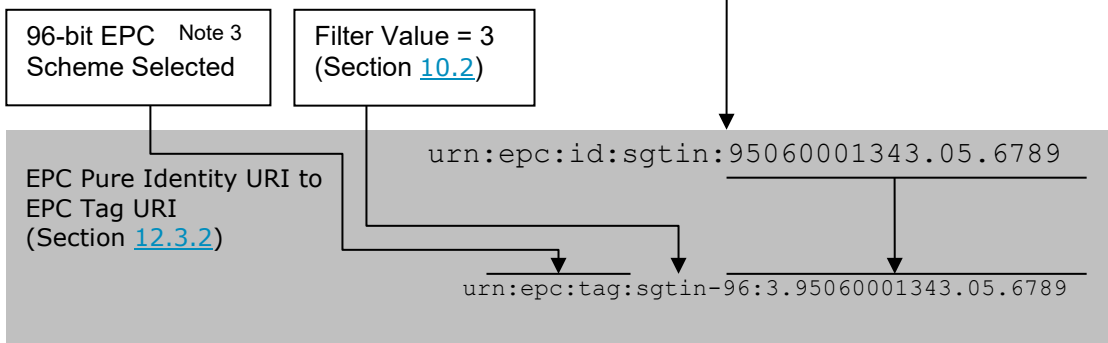
The illustration below makes reference to the following notes:

- **Note 1:** The step of converting a GS1 element string into the EPC Pure Identity URI requires that the number of digits in the GS1 Company Prefix be determined; e.g., by reference to an external table of company prefixes. In this example, the GS1 Company Prefix is shown to be seven digits.
- **Note 2:** The check digit in GTIN as it appears in the GS1 element string is not included in the EPC Pure Identity URI.
- **Note 3:** The SGTIN-96 EPC scheme may only be used if the Serial Number meets certain constraints. Specifically, the serial number must (a) consist only of digit characters; (b) not begin with a zero digit (unless the entire serial number is the single digit '0'); and (c) correspond to a decimal numeral whose numeric value that is less than 2^{38} (less than 274,877,906,944). For all other serial numbers, the SGTIN-198 EPC scheme must be used. Note that the EPC URI is identical regardless of whether SGTIN-96 or SGTIN-198 is used in the RFID Tag.
- **Note 4:** EPC Binary Encoding header values are defined in Section [14.2](#).
- **Note 5:** The number of bits in the GS1 Company Prefix and Indicator/Item Reference fields in the EPC Binary Encoding depends on the number of digits in the GS1 Company Prefix portion of the EPC URI, and this is indicated by a code in the Partition field of the EPC Binary Encoding. See [14.2](#). (for the SGTIN EPC only).
- **Note 6:** The Serial field of the EPC Binary Encoding for SGTIN-96 is 38 bits.

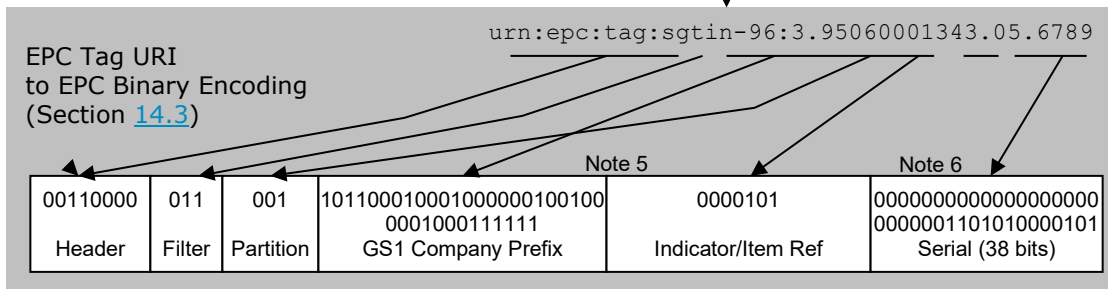
GS1 element string (01) 09506000134352 (21) 6789
 GS1 Digital Link URI <https://id.gs1.org/01/09506000134352/21/6789>



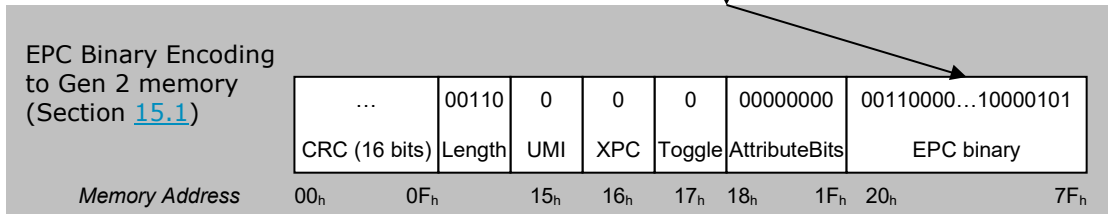
EPC Pure Identity URI urn:epc:id:sgtin:95060001343.05.6789



EPC Tag URI urn:epc:tag:sgtin-96:3.95060001343.05.6789



EPC Binary 0011000001100110110001000100000010010000010001111110000101000000000000
 00000000000000001101010000101



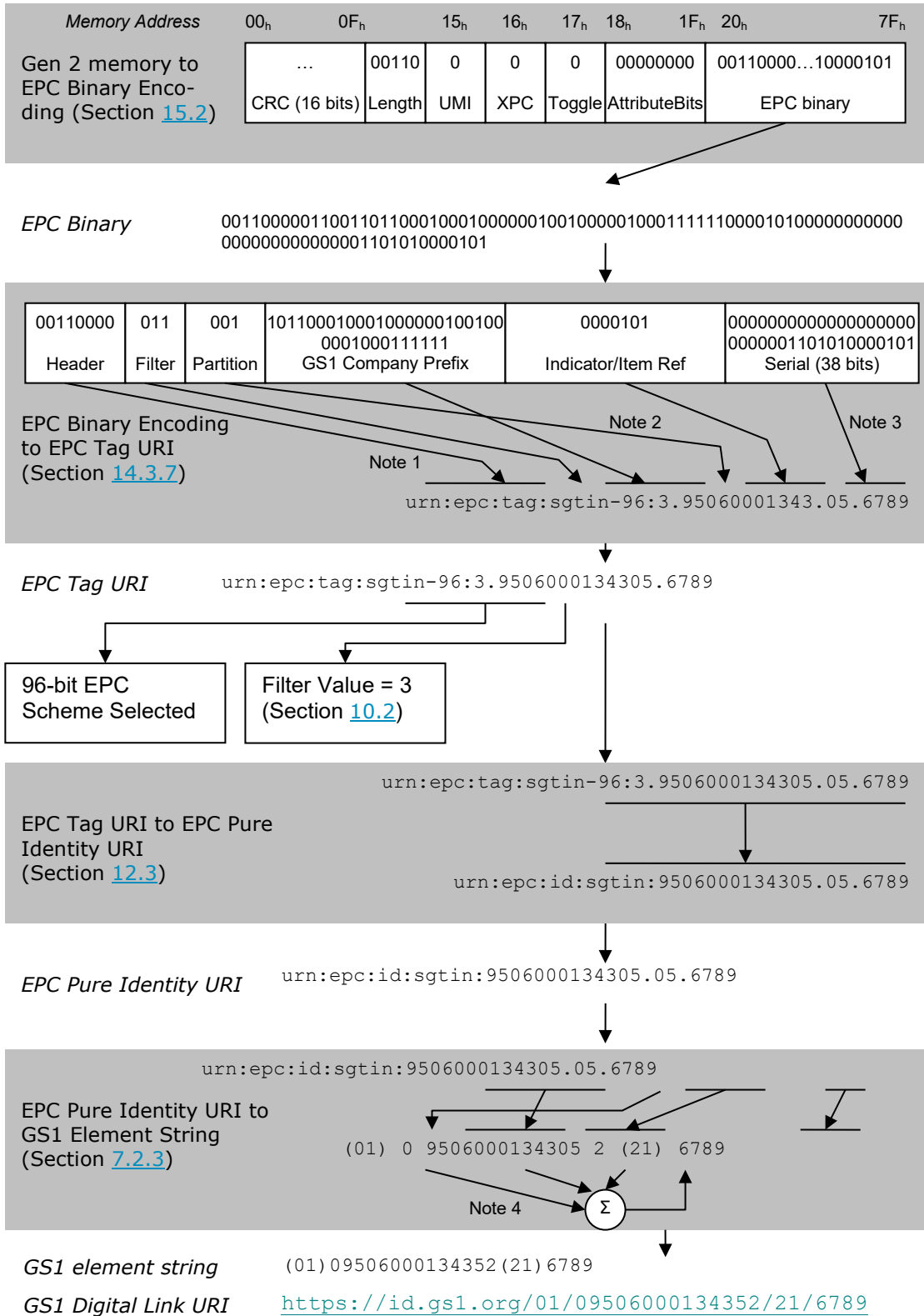
E.2 Decoding an SGTIN-96 to a Serialised Global Trade Item Number (SGTIN)

This example illustrates the decoding of an EPC Gen 2 RFID tag containing an SGTIN-96 EPC Binary Encoding into a GS1 element string containing a Serialised Global Trade Item Number (SGTIN), with intermediate steps including the EPC Binary Encoding, the EPC Tag URI, and the EPC URI.

In some applications, only a part of this illustration is relevant. For example, an application may only need to convert an EPC binary encoding to an EPC URI, in which case only the top of the illustration is needed.

The illustration below makes reference to the following notes:

- **Note 1:** The EPC Binary Encoding header indicates how to interpret the remainder of the binary data, and the EPC scheme name to be included in the EPC Tag URI. EPC Binary Encoding header values are defined in Section [14.2](#).
- **Note 2:** The Partition field of the EPC Binary Encoding contains a code that indicates the number of bits in the GS1 Company Prefix field and the Indicator/Item Reference field. The partition code also determines the number of decimal digits to be used for those fields in the EPC Tag URI (the decimal representation for those two fields is padded on the left with zero characters as necessary). See Section [14.2](#). (for the SGTIN EPC only).
- **Note 3:** For the SGTIN-96 EPC scheme, the Serial Number field is decoded by interpreting the bits as a binary integer and converting to a decimal numeral without leading zeros (unless all serial number bits are zero, which decodes as the string "0"). Serial numbers containing non-digit characters or that begin with leading zero characters may only be encoded in the SGTIN-198 EPC scheme.
- **Note 4:** The check digit in the GS1 element string is calculated from other digits in the EPC Pure Identity URI, as specified in Section [7.2.3](#).



6147 **E.3 Summary Examples of All EPC schemes**

| SGTIN-96 | |
|---------------------------|---|
| GS1 element string | (01)09506000134352(21)123456789 |
| GS1 Digital Link URI | https://id.gs1.org/01/09506000134352/21/123456789 |
| EPC URI | urn:epc:id:sgtin:95060001343.05.1234567896789 |
| EPC Tag URI | urn:epc:tag:sgtin-96:3.95060001343.05.123456789 |
| EPC Binary Encoding (hex) | 3066C4409047E140075BCD15 |

6148

| SGTIN-198 | |
|---------------------------|--|
| GS1 element string | (01)09506000134352(21)32a/b |
| GS1 Digital Link URI | https://id.gs1.org/01/09506000134352/21/32a%2Fb |
| EPC URI | urn:epc:id:sgtin:95060001343.05.32a%2Fb |
| EPC Tag URI | urn:epc:tag:sgtin-198:3.95060001343.05.32a%2Fb |
| EPC Binary Encoding (hex) | 3666C4409047E159B2C2BF10000000000000000000000000 |

6149

| SGTIN+ (assuming filter value 3 and no +AIDC data) | |
|--|--|
| GS1 element string | (01)79521141123453(21)32a/b |
| GS1 Digital Link URI | https://example.com/01/79521141123453/21/32a%2Fb |
| EPC Binary Encoding (hex) | F73795211411234538566CB0AFC4 |

6150

| DSGTIN+ (assuming filter value 3 and no +AIDC data) | |
|---|---|
| GS1 element string | (01)79521141123453(21)32a/b(17)220630 |
| GS1 Digital Link URI | https://example.com/01/79521141123453/21/32a%2Fb?17=220630 (https://example.com/01/79521141123453/21/32a%2Fb in EPCIS) |
| EPC Binary Encoding (hex) | FB342CDE795211411234538566CB0AFC4 |

6151

| SSCC-96 | |
|---------------------------|---|
| GS1 element string | (00)095201234567891235 |
| GS1 Digital Link URI | https://example.com/00/095201234567891235 |
| GCP length | 6 (partition value "6") |
| EPC URI | urn:epc:id:sscc:952012.03456789123 |
| Filter value | "All Others" (0) |
| EPC Tag URI | urn:epc:tag:sscc-96:0.952012.03456789123 |
| EPC Binary Encoding (hex) | 311BA1B300CE0A6A83000000 |

6152

| SSCC+ | |
|---------------------------|--|
| GS1 element string | (00)095201234567891235 |
| GS1 Digital Link URI | https://id.gs1.org/00/095201234567891235 |
| +Data appended to EPC? | no (0) |
| Filter value | "All Others" (0) |
| EPC Binary Encoding (hex) | F90095201234567891235 |

6153

| SGLN-96 | |
|---------------------------|--|
| GS1 element string | (414)9521141123454(254)5678 |
| GS1 Digital Link URI | https://example.com/414/9521141123454/254/5678 |
| EPC URI | urn:epc:id:sgln:9521141.12345.5678 |
| EPC Tag URI | urn:epc:tag:sgln-96:3.9521141.12345.5678 |
| EPC Binary Encoding (hex) | 3276451FD46072000000162E |

6154

| SGLN-195 | |
|---------------------------|---|
| GS1 element string | (414)9521141123454(254)32a/b |
| GS1 Digital Link URI | https://example.com/414/9521141123454/254/32a%2Fb |
| EPC URI | urn:epc:id:sgln:9521141.12345.32a%2Fb |
| EPC Tag URI | urn:epc:tag:sgln-195:3.9521141.12345.32a%2Fb |
| EPC Binary Encoding (hex) | 3976451FD46072CD9615F8800 |

6155

| SGLN+ | |
|---------------------------|---|
| GS1 element string | (414)9521141123454(254)32a/b |
| GS1 Digital Link URI | https://example.com/414/9521141123454/254/32a%2Fb |
| EPC Binary Encoding (hex) | F2395211411234548566CB0AFC4 |

6156

| GRAI-96 | |
|---------------------------|---|
| GS1 element string | (8003)095211411234545678 |
| GS1 Digital Link URI | https://example.com/8003/095211411234545678 |
| EPC URI | urn:epc:id:grai:9521141.12345.5678 |
| EPC Tag URI | urn:epc:tag:grai-96:3.9521141.12345.5678 |
| EPC Binary Encoding (hex) | 3376451FD40C0E400000162E |

6157

| GRAI-170 | |
|----------------------|--|
| GS1 element string | (8003)0952114112345432a/b |
| GS1 Digital Link URI | https://example.com/8003/0952114112345432a%2Fb |
| EPC URI | urn:epc:id:grai:9521141.12345.32a%2Fb |
| EPC Tag URI | urn:epc:tag:grai-170:3.9521141.12345.32a%2Fb |

6158

| GRAI-170 | |
|---------------------------|---|
| EPC Binary Encoding (hex) | 3776451FD40C0E59B2C2BF10000000000000000000000 |

6159

| GRAI+ | |
|---------------------------|--|
| GS1 element string | (8003)0952114112345432a/b |
| GS1 Digital Link URI | https://example.com/8003/0952114112345432a%2Fb |
| EPC Binary Encoding (hex) | F13095211411234548566CB0AFC4 |

6160

| GIAI-96 | |
|---------------------------|--------------------------------------|
| GS1 element string | (8004)95211415678 |
| GS1 Digital Link URI | https://example.com/8004/95211415678 |
| EPC URI | urn:epc:id:giai:9521141.5678 |
| EPC Tag URI | urn:epc:tag:giai-96:3.9521141.5678 |
| EPC Binary Encoding (hex) | 3476451FD400000000000162E |

6161

| GIAI-202 | |
|---------------------------|--|
| GS1 element string | (8004)952114132a/b |
| GS1 Digital Link URI | https://example.com/8004/952114132a%2Fb |
| EPC URI | urn:epc:id:giai:9521141.32a%2Fb |
| EPC Tag URI | urn:epc:tag:giai-202:3.9521141.32a%2Fb |
| EPC Binary Encoding (hex) | 3876451FD59B2C2BF100000000000000000000000000000000 |

6162

| GIAI+ | |
|---------------------------|---|
| GS1 element string | (8004)952114132a/b |
| GS1 Digital Link URI | https://example.com/8004/952114132a%2Fb |
| EPC Binary Encoding (hex) | FA3952114132E83C2BF10 |

6163

| GSRN-96 | |
|---------------------------|---|
| GS1 element string | (8018)952114112345678906 |
| GS1 Digital Link URI | https://example.com/8018/952114112345678906 |
| EPC URI | urn:epc:id:gsrc:9521141.1234567890 |
| EPC Tag URI | urn:epc:tag:gsrc-96:3.9521141.1234567890 |
| EPC Binary Encoding (hex) | 2D76451FD4499602D2000000 |

| GSRN+ | |
|---------------------------|---|
| GS1 element string | (8018)952114112345678906 |
| GS1 Digital Link URI | https://example.com/8018/952114112345678906 |
| EPC Binary Encoding (hex) | F43952114112345678906 |

6164

| GSRNP-96 | |
|---------------------------|---|
| GS1 element string | (8017)952114112345678906 |
| GS1 Digital Link URI | https://example.com/8017/952114112345678906 |
| EPC URI | urn:epc:id:gsrnp:9521141.1234567890 |
| EPC Tag URI | urn:epc:tag:gsrnp-96:3.9521141.1234567890 |
| EPC Binary Encoding (hex) | 2E76451FD4499602D2000000 |

6165

| GSRNP+ | |
|---------------------------|---|
| GS1 element string | (8017)952114112345678906 |
| GS1 Digital Link URI | https://example.com/8017/952114112345678906 |
| EPC Binary Encoding (hex) | F53952114112345678906 |

6166

| GDTI-96 | |
|---------------------------|---|
| GS1 element string | (253)95211411234545678 |
| GS1 Digital Link URI | https://example.com/253/95211411234545678 |
| EPC URI | urn:epc:id:gdti:9521141.12345.5678 |
| EPC Tag URI | urn:epc:tag:gdti-96:3.9521141.12345.5678 |
| EPC Binary Encoding (hex) | 2C76451FD46072000000162E |

6167

| GDTI-174 | |
|---------------------------|--|
| GS1 element string | (253)9521141987650ABCDefgh012345678 |
| GS1 Digital Link URI | https://example.com/253/9521141987650ABCDefgh012345678 |
| EPC URI | urn:epc:id:gdti:9521141.98765.ABCDefgh012345678 |
| EPC Tag URI | urn:epc:tag:gdti-174:3.9521141.98765.ABCDefgh012345678 |
| EPC Binary Encoding (hex) | 3E76451FD7039B061438997367D0C18B266D1AB66EE0 |

6168

| GDTI+ | |
|---------------------------|---|
| GS1 element string | (253)95211411234545678 |
| GS1 Digital Link URI | https://example.com/253/95211411234545678 |
| EPC Binary Encoding (hex) | F6395211411234540458B8 |

6169

| CPI-96 | |
|---------------------------|--|
| GS1 element string | (8010)952114198765(8011)12345 |
| GS1 Digital Link URI | https://example.com/8010/952114198765/8011/12345 |
| EPC URI | urn:epc:id:cpi:9521141.98765.12345 |
| EPC Tag URI | urn:epc:tag:cpi-96:3.9521141.98765.12345 |
| EPC Binary Encoding (hex) | 3C76451FD400C0E680003039 |

6170

| CPI-var | |
|---------------------------|---|
| GS1 element string | (8010)95211415PQ7/Z43(8011)12345 |
| GS1 Digital Link URI | https://example.com/8010/95211415PQ7%2FZ43/8011/12345 |
| EPC URI | urn:epc:id:cpi:9521141.5PQ7%2FZ43.12345 |
| EPC Tag URI | urn:epc:tag:cpi-var:3.9521141.5PQ7%2FZ43.12345 |
| EPC Binary Encoding (hex) | 3D76451FD75411DEF6B4CC00000003039000 |

6171

| CPI+ | |
|---------------------------|---|
| GS1 element string | (8010)95211415PQ7/Z43(8011)12345 |
| GS1 Digital Link URI | https://example.com/8010/95211415PQ7%2FZ43/8011/12345 |
| EPC Binary Encoding (hex) | F0395211415E87A145BAFB4D19A8C0E4 |

6172

| SGCN-96 | |
|---------------------------|--|
| GS1 element string | (255)952114167890904711 |
| GS1 Digital Link URI | https://example.com/255/952114167890904711 |
| EPC URI | urn:epc:id:sgcn:9521141.67890.04711 |
| EPC Tag URI | urn:epc:tag:sgcn-96:3.9521141.67890.04711 |
| EPC Binary Encoding (hex) | 3F76451FD612640000019907 |

6173

| SGCN+ | |
|---------------------------|--|
| GS1 element string | (255)952114167890904711 |
| GS1 Digital Link URI | https://example.com/255/952114167890904711 |
| EPC Binary Encoding (hex) | F839521141678909509338 |

6174

| GID-96 | |
|---------------------------|-------------------------------------|
| EPC URI | urn:epc:id:gid:952056.2718.1414 |
| EPC Tag URI | urn:epc:tag:gid-96:952056.2718.1414 |
| EPC Binary Encoding (hex) | 3500E86F8000A9E000000586 |

6175

| USDOD-96 | |
|---------------------------|-----------------------------------|
| EPC URI | urn:epc:id:usdod:CAGEY.5678 |
| EPC Tag URI | urn:epc:tag:usdod-96:3.CAGEY.5678 |
| EPC Binary Encoding (hex) | 2F320434147455900000162E |

6176

| ADI-var | |
|---------------------------|---|
| EPC URI | urn:epc:id:adi:35962.PQ7VZ4.M37GXB92 |
| EPC Tag URI | urn:epc:tag:adi-var:3.35962.PQ7VZ4.M37GXB92 |
| EPC Binary Encoding (hex) | 3B0E0CF5E76C9047759AD00373DC7602E7200 |



6177

| ITIP-110 | |
|---------------------------|--|
| GS1 element string | (8006)095211411234540102(21)981 |
| GS1 Digital Link URI | https://example.com/8006/095211411234540102/21/981 |
| EPC URI | urn:epc:id:itip:9521141.012345.01.02.981 |
| EPC Tag URI | urn:epc:tag:itip-110:3.9521141.012345.01.02.981 |
| EPC Binary Encoding (hex) | 4076451FD40C0E40820000000F54 |

6178

| ITIP-212 | |
|---------------------------|---|
| GS1 element string | (8006)095211411234540102(21)mw133 |
| GS1 Digital Link URI | https://example.com/8006/095211411234540102/21/mw133 |
| EPC URI | urn:epc:id:itip:9521141.012345.01.02.mw133 |
| EPC Tag URI | urn:epc:tag:itip-212:3.9521141.012345.01.02.mw133 |
| EPC Binary Encoding (hex) | 4176451FD40C0E4082DBDD8B3660000000000000000000000000000 |

6179

| ITIP+ | |
|---------------------------|---|
| GS1 element string | (8006)095211411234540102(21)rif981 |
| GS1 Digital Link URI | https://example.com/8006/095211411234540102/21/rif981 |
| EPC Binary Encoding (hex) | F3309521141123454010266AE27FDF35 |

6180

6181
6182
6183
6184
6185
6186
6187

F Packed objects ID Table for Data Format 9

This section provides the Packed Objects ID Table for Data Format 9, which defines Packed Objects ID values, OIDs, and format strings for GS1 Application Identifiers.

Section [F.1](#) is a non-normative listing of the content of the ID Table for Data Format 9, in a human readable, tabular format. Section [F.2](#) is the normative table, in machine readable, comma-separated-value format, as registered with ISO. As of TDS 2.1, **Section F.2 is supplemented with an external, normative artefact in CSV format.**

6188
6189
6190
6191

F.1 Tabular Format (non-normative)

This section is a non-normative listing of the content of the ID Table for Data Format 9, in a human readable, tabular format. See Section [F.2](#) for the normative, machine readable, comma-separated-value format, as registered with ISO.

| K-Text = GS1 AI ID Table for ISO/IEC 15961 Format 9 | | | | | | |
|---|---------|---------|----------|--|------------------------|--------------|
| K-Version = 1.00 | | | | | | |
| K-ISO15434=05 | | | | | | |
| K-Text = Primary Base Table | | | | | | |
| K-TableID = F9B0 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 90 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 00 | 1 | 0 | 00 | SSCC (Serial Shipping Container Code) | SSCC | 18n |
| 01 | 2 | 1 | 01 | Global Trade Item Number | GTIN | 14n |
| 02 + 37 | 3 | (2)(37) | (02)(37) | GTIN + Count of trade items contained in a logistic unit | CONTENT + COUNT | (14n)(1*8n) |
| 10 | 4 | 10 | 10 | Batch or lot number | BATCH/LOT | 1*20an |
| 11 | 5 | 11 | 11 | Production date (YYMMDD) | PROD DATE | 6n |
| 12 | 6 | 12 | 12 | Due date (YYMMDD) | DUE DATE | 6n |
| 13 | 7 | 13 | 13 | Packaging date (YYMMDD) | PACK DATE | 6n |
| 15 | 8 | 15 | 15 | Best before date (YYMMDD) | BEST BEFORE OR SELL BY | 6n |
| 17 | 9 | 17 | 17 | Expiration date (YYMMDD) | USE BY OR EXPIRY | 6n |
| 20 | 10 | 20 | 20 | Internal product variant | VARIANT | 2n |
| 21 | 11 | 21 | 21 | Serial number | SERIAL | 1*20an |
| 22 | 12 | 22 | 22 | Consumer product variant | CPV | 1*20an |
| 240 | 13 | 240 | 240 | Additional product identification assigned by the manufacturer | ADDITIONAL ID | 1*30an |

| K-Text = GS1 AI ID Table for ISO/IEC 15961 Format 9 | | | | | | |
|---|----|-------------------------|-------------------------|--|---------------------------|------------|
| 241 | 14 | 241 | 241 | Customer part number | CUST. PART NO. | 1*30an |
| 242 | 15 | 242 | 242 | Made-to-Order Variation Number | VARIATION NUMBER | 1*6n |
| 250 | 16 | 250 | 250 | Secondary serial number | SECONDARY SERIAL | 1*30an |
| 251 | 17 | 251 | 251 | Reference to source entity | REF. TO SOURCE | 1*30an |
| 253 | 18 | 253 | 253 | Global Document Type Identifier | DOC. ID | 13n 0*17an |
| 30 | 19 | 30 | 30 | Variable count of items (Variable Measure Trade Item) | VAR. COUNT | 1*8n |
| 310n 320n etc | 20 | K-Secondary = S00 | | Net weight, kilograms or pounds or troy oz (Variable Measure Trade Item) | | |
| 311n 321n etc | 21 | K-Secondary = S01 | | Length of first dimension (Variable Measure Trade Item) | | |
| 312n 324n etc | 22 | K-Secondary = S02 | | Width, diameter, or second dimension (Variable Measure Trade Item) | | |
| 313n 327n etc | 23 | K-Secondary = S03 | | Depth, thickness, height, or third dimension (Variable Measure Trade Item) | | |
| 314n 350n etc | 24 | K-Secondary = S04 | | Area (Variable Measure Trade Item) | | |
| 315n 316n etc | 25 | K-Secondary = S05 | | Net volume (Variable Measure Trade Item) | | |
| 330n or 340n | 26 | 330%x30-36 / 340%x30-36 | 330%x30-36 / 340%x30-36 | Logistic weight, kilograms or pounds | GROSS WEIGHT (kg) or (lb) | 6n / 6n |
| 331n, 341n, etc | 27 | K-Secondary = S09 | | Length or first dimension | | |
| 332n, 344n, etc | 28 | K-Secondary = S10 | | Width, diameter, or second dimension | | |
| 333n, 347n, etc | 29 | K-Secondary = S11 | | Depth, thickness, height, or third dimension | | |
| 334n 353n etc | 30 | K-Secondary = S07 | | Logistic Area | | |
| 335n 336n etc | 31 | K-Secondary = S06 | 335%x30-36 | Logistic volume | | |

| K-Text = GS1 AI ID Table for ISO/IEC 15961 Format 9 | | | | | | |
|---|----|-------------------------|-------------------------|---|---------------------------|---------------------|
| 337(***) | 32 | 337%x30-36 | 337%x30-36 | Kilograms per square metre | KG PER m^2 | 6n |
| 390n or 391n | 33 | 390%x30-39 / 391%x30-39 | 390%x30-39 / 391%x30-39 | Amount payable - single monetary area or with ISO currency code | AMOUNT | 1*15n / 4*18n |
| 392n or 393n | 34 | 392%x30-39 / 393%x30-39 | 392%x30-39 / 393%x30-39 | Amount payable for Variable Measure Trade Item - single monetary unit or ISO cc | PRICE | 1*15n / 4*18n |
| 400 | 35 | 400 | 400 | Customer's purchase order number | ORDER NUMBER | 1*30an |
| 401 | 36 | 401 | 401 | Global Identification Number for Consignment | GINC | 1*30an |
| 402 | 37 | 402 | 402 | Global Shipment Identification Number | GSIN | 17n |
| 403 | 38 | 403 | 403 | Routing code | ROUTE | 1*30an |
| 410 | 39 | 410 | 410 | Ship to - Deliver to Global Location Number | SHIP TO LOC | 13n |
| 411 | 40 | 411 | 420 | Bill to - Invoice to Global Location Number | BILL TO | 13n |
| 412 | 41 | 412 | 412 | Purchased from Global Location Number | PURCHASE FROM | 13n |
| 413 | 42 | 413 | 413 | Ship for - Deliver for - Forward to Global Location Number | SHIP FOR LOC | 13n |
| 414 and 254 | 43 | (414) [254] | (414) [254] | Identification of a physical location GLN, and optional Extension | LOC No + GLN EXTENSION | (13n) [1*20an] |
| 415 and 8020 | 44 | (415) (8020) | (415) (8020) | Global Location Number of the Invoicing Party and Payment Slip Reference Number | PAY + REF No | (13n) (1*25an) |
| 420 or 421 | 45 | (420/421) | (420/421) | Ship-to / Deliver-to postal code | SHIP TO POST | (1*20an / 3n 1*9an) |
| 422 | 46 | 422 | 422 | Country of origin of a trade item | ORIGIN | 3n |
| 423 | 47 | 423 | 423 | Country of initial processing | COUNTRY - INITIAL PROCESS | 3*15n |
| 424 | 48 | 424 | 424 | Country of processing | COUNTRY - INITIAL PROCESS | 3n |
| 425 | 49 | 425 | 425 | Country of disassembly | COUNTRY - DISASSEMBLY | 3n |

| K-Text = GS1 AI ID Table for ISO/IEC 15961 Format 9 | | | | | | |
|---|----|------|------|--|------------------------|------------|
| 426 | 50 | 426 | 426 | Country covering full process chain | COUNTRY - FULL PROCESS | 3n |
| 7001 | 51 | 7001 | 7001 | NATO stock number | NSN | 13n |
| 7002 | 52 | 7002 | 7002 | UN/ECE meat carcasses and cuts classification | MEAT CUT | 1*30an |
| 7003 | 53 | 7003 | 7003 | Expiration Date and Time | EXPIRY DATE/TIME | 10n |
| 7004 | 54 | 7004 | 7004 | Active Potency | ACTIVE POTENCY | 1*4n |
| 703s | 55 | 7030 | 7030 | Approval number of processor with ISO country code | PROCESSOR # s | 3n 1*27an |
| 703s | 56 | 7031 | 7031 | Approval number of processor with ISO country code | PROCESSOR # s | 3n 1*27an |
| 703s | 57 | 7032 | 7032 | Approval number of processor with ISO country code | PROCESSOR # s | 3n 1*27an |
| 703s | 58 | 7033 | 7033 | Approval number of processor with ISO country code | PROCESSOR # s | 3n 1*27an |
| 703s | 59 | 7034 | 7034 | Approval number of processor with ISO country code | PROCESSOR # s | 3n 1*27an |
| 703s | 60 | 7035 | 7035 | Approval number of processor with ISO country code | PROCESSOR # s | 3n 1*27an |
| 703s | 61 | 7036 | 7036 | Approval number of processor with ISO country code | PROCESSOR # s | 3n 1*27an |
| 703s | 62 | 7037 | 7037 | Approval number of processor with ISO country code | PROCESSOR # s | 3n 1*27an |
| 703s | 63 | 7038 | 7038 | Approval number of processor with ISO country code | PROCESSOR # s | 3n 1*27an |
| 703s | 64 | 7039 | 7039 | Approval number of processor with ISO country code | PROCESSOR # s | 3n 1*27an |
| 8001 | 65 | 8001 | 8001 | Roll products - width, length, core diameter, direction, splices | DIMENSIONS | 14n |
| 8002 | 66 | 8002 | 8002 | Electronic serial identifier for cellular mobile telephones | CMT No | 1*20an |
| 8003 | 67 | 8003 | 8003 | Global Returnable Asset Identifier | GRAI | 14n 0*16an |
| 8004 | 68 | 8004 | 8004 | Global Individual Asset Identifier | GIAI | 1*30an |
| 8005 | 69 | 8005 | 8005 | Price per unit of measure | PRICE PER UNIT | 6n |

| K-Text = GS1 AI ID Table for ISO/IEC 15961 Format 9 | | | | | | |
|---|----|-------------------|------|---|------------------|--------|
| 8006 | 70 | 8006 | 8006 | Identification of the component of a trade item | ITIP | 18n |
| 8007 | 71 | 8007 | 8007 | International Bank Account Number | IBAN | 1*34an |
| 8008 | 72 | 8008 | 8008 | Date and time of production | PROD TIME | 8*12n |
| 8018 | 73 | 8018 | 8018 | Global Service Relation Number - Recipient | GSRN - RECIPIENT | 18n |
| 8100 8101 etc | 74 | K-Secondary = S08 | | Coupon Codes | | |
| 90 | 75 | 90 | 90 | Information mutually agreed between trading partners (including FACT DIs) | INTERNAL | 1*30an |
| 91 | 76 | 91 | 91 | Company internal information | INTERNAL | 1*an |
| 92 | 77 | 92 | 92 | Company internal information | INTERNAL | 1*an |
| 93 | 78 | 93 | 93 | Company internal information | INTERNAL | 1*an |
| 94 | 79 | 94 | 94 | Company internal information | INTERNAL | 1*an |
| 95 | 80 | 95 | 95 | Company internal information | INTERNAL | 1*an |
| 96 | 81 | 96 | 96 | Company internal information | INTERNAL | 1*an |
| 97 | 82 | 97 | 97 | Company internal information | INTERNAL | 1*an |
| 98 | 83 | 98 | 98 | Company internal information | INTERNAL | 1*an |
| 99 | 84 | 99 | 99 | Company internal information | INTERNAL | 1*an |
| nnn | 85 | K-Secondary = S12 | | Additional AIs | | |
| K-TableEnd = F9B0 | | | | | | |

6192

| K-Text = Sec. IDT - Net weight, kilograms or pounds or troy oz (Variable Measure Trade Item) | | | | | | |
|--|---------|------------|------------|---|-----------------|--------------|
| K-TableID = F9S00 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 4 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 310(***) | 0 | 310%x30-35 | 310%x30-35 | Net weight, kilograms (Variable Measure Trade Item) | NET WEIGHT (kg) | 6n |

| K-Text = Sec. IDT - Net weight, kilograms or pounds or troy oz (Variable Measure Trade Item) | | | | | | |
|--|---|------------|------------|---|-----------------|----|
| 320(***) | 1 | 320%x30-35 | 320%x30-35 | Net weight, pounds (Variable Measure Trade Item) | NET WEIGHT (lb) | 6n |
| 356(***) | 2 | 356%x30-35 | 356%x30-35 | Net weight, troy ounces (Variable Measure Trade Item) | NET WEIGHT (t) | 6n |
| K-TableEnd = F9S00 | | | | | | |

6193

| K-Text = Sec. IDT - Length of first dimension (Variable Measure Trade Item) | | | | | | |
|---|---------|------------|------------|---|------------|--------------|
| K-TableID = F9S01 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 4 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 311(***) | 0 | 311%x30-35 | 311%x30-35 | Length of first dimension, metres (Variable Measure Trade Item) | LENGTH (m) | 6n |
| 321(***) | 1 | 321%x30-35 | 321%x30-35 | Length or first dimension, inches (Variable Measure Trade Item) | LENGTH (i) | 6n |
| 322(***) | 2 | 322%x30-35 | 322%x30-35 | Length or first dimension, feet (Variable Measure Trade Item) | LENGTH (f) | 6n |
| 323(***) | 3 | 323%x30-35 | 323%x30-35 | Length or first dimension, yards (Variable Measure Trade Item) | LENGTH (y) | 6n |
| K-TableEnd = F9S01 | | | | | | |

6194

| K-Text = Sec. IDT - Width, diameter, or second dimension (Variable Measure Trade Item) | | | | | | |
|--|---------|------------|------------|--|------------|--------------|
| K-TableID = F9S02 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 4 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 312(***) | 0 | 312%x30-35 | 312%x30-35 | Width, diameter, or second dimension, metres (Variable Measure Trade Item) | WIDTH (m) | 6n |

| K-Text = Sec. IDT - Width, diameter, or second dimension (Variable Measure Trade Item) | | | | | | |
|--|---|------------|------------|--|-----------|----|
| 324(***) | 1 | 324%x30-35 | 324%x30-35 | Width, diameter, or second dimension, inches (Variable Measure Trade Item) | WIDTH (i) | 6n |
| 325(***) | 2 | 325%x30-35 | 325%x30-35 | Width, diameter, or second dimension, (Variable Measure Trade Item) | WIDTH (f) | 6n |
| 326(***) | 3 | 326%x30-35 | 326%x30-35 | Width, diameter, or second dimension, yards (Variable Measure Trade Item) | WIDTH (y) | 6n |
| K-TableEnd = F9S02 | | | | | | |

6195

| K-Text = Sec. IDT - Depth, thickness, height, or third dimension (Variable Measure Trade Item) | | | | | | |
|--|---------|------------|------------|--|------------|--------------|
| K-TableID = F9S03 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 4 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 313(***) | 0 | 313%x30-35 | 313%x30-35 | Depth, thickness, height, or third dimension, metres (Variable Measure Trade Item) | HEIGHT (m) | 6n |
| 327(***) | 1 | 327%x30-35 | 327%x30-35 | Depth, thickness, height, or third dimension, inches (Variable Measure Trade Item) | HEIGHT (i) | 6n |
| 328(***) | 2 | 328%x30-35 | 328%x30-35 | Depth, thickness, height, or third dimension, feet (Variable Measure Trade Item) | HEIGHT (f) | 6n |
| 329(***) | 3 | 329%x30-35 | 329%x30-35 | Depth, thickness, height, or third dimension, yards (Variable Measure Trade Item) | HEIGHT (y) | 6n |
| K-TableEnd = F9S03 | | | | | | |

6196

| K-Text = Sec. IDT - Area (Variable Measure Trade Item) | | | | | | |
|--|---------|------------|------------|---|------------------------|--------------|
| K-TableID = F9S04 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 4 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 314(***) | 0 | 314%x30-35 | 314%x30-35 | Area, square metres (Variable Measure Trade Item) | AREA (m ²) | 6n |
| 350(***) | 1 | 350%x30-35 | 350%x30-35 | Area, square inches (Variable Measure Trade Item) | AREA (i ²) | 6n |
| 351(***) | 2 | 351%x30-35 | 351%x30-35 | Area, square feet (Variable Measure Trade Item) | AREA (f ²) | 6n |
| 352(***) | 3 | 352%x30-35 | 352%x30-35 | Area, square yards (Variable Measure Trade Item) | AREA (y ²) | 6n |
| K-TableEnd = F9S04 | | | | | | |

6197

| K-Text = Sec. IDT - Net volume (Variable Measure Trade Item) | | | | | | |
|--|---------|------------|------------|--|-------------------------------|--------------|
| K-TableID = F9S05 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 8 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 315(***) | 0 | 315%x30-35 | 315%x30-35 | Net volume, litres (Variable Measure Trade Item) | NET VOLUME (l) | 6n |
| 316(***) | 1 | 316%x30-35 | 316%x30-35 | Net volume, cubic metres (Variable Measure Trade Item) | NET VOLUME (m ³) | 6n |
| 357(***) | 2 | 357%x30-35 | 357%x30-35 | Net weight (or volume), ounces (Variable Measure Trade Item) | NET VOLUME (oz) | 6n |
| 360(***) | 3 | 360%x30-35 | 360%x30-35 | Net volume, quarts (Variable Measure Trade Item) | NET VOLUME (q) | 6n |
| 361(***) | 4 | 361%x30-35 | 361%x30-35 | Net volume, gallons U.S. (Variable Measure Trade Item) | NET VOLUME (g) | 6n |
| 364(***) | 5 | 364%x30-35 | 364%x30-35 | Net volume, cubic inches | VOLUME (i ³), log | 6n |

| K-Text = Sec. IDT - Net volume (Variable Measure Trade Item) | | | | | | |
|--|---|------------|------------|---|------------------|----|
| 365(***) | 6 | 365%x30-35 | 365%x30-35 | Net volume, cubic feet (Variable Measure Trade Item) | VOLUME (f3), log | 6n |
| 366(***) | 7 | 366%x30-35 | 366%x30-35 | Net volume, cubic yards (Variable Measure Trade Item) | VOLUME (y3), log | 6n |
| K-TableEnd = F9S05 | | | | | | |

6198

| K-Text = Sec. IDT - Logistic Volume | | | | | | |
|-------------------------------------|---------|------------|------------|-------------------------------|-------------------|--------------|
| K-TableID = F9S06 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 8 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 335(***) | 0 | 335%x30-35 | 335%x30-35 | Logistic volume, litres | VOLUME (l), log | 6n |
| 336(***) | 1 | 336%x30-35 | 336%x30-35 | Logistic volume, cubic meters | VOLUME (m^3), log | 6n |
| 362(***) | 2 | 362%x30-35 | 362%x30-35 | Logistic volume, quarts | VOLUME (q), log | 6n |
| 363(***) | 3 | 363%x30-35 | 363%x30-35 | Logistic volume, gallons | VOLUME (g), log | 6n |
| 367(***) | 4 | 367%x30-35 | 367%x30-35 | Logistic volume, cubic inches | VOLUME (q), log | 6n |
| 368(***) | 5 | 368%x30-35 | 368%x30-35 | Logistic volume, cubic feet | VOLUME (g), log | 6n |
| 369(***) | 6 | 369%x30-35 | 369%x30-35 | Logistic volume, cubic yards | VOLUME (i^3), log | 6n |
| K-TableEnd = F9S06 | | | | | | |

6199

| K-Text = Sec. IDT - Logistic Area | | | | | | |
|-----------------------------------|---------|------------|------------|---------------------|-----------------|--------------|
| K-TableID = F9S07 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 4 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 334(***) | 0 | 334%x30-35 | 334%x30-35 | Area, square metres | AREA (m^2), log | 6n |
| 353(***) | 1 | 353%x30-35 | 353%x30-35 | Area, square inches | AREA (i^2), log | 6n |
| 354(***) | 2 | 354%x30-35 | 354%x30-35 | Area, square feet | AREA (f^2), log | 6n |
| 355(***) | 3 | 355%x30-35 | 355%x30-35 | Area, square yards | AREA (y^2), log | 6n |
| K-TableEnd = F9S07 | | | | | | |

6200

| K-Text = Sec. IDT - Coupon Codes | | | | | | |
|----------------------------------|---------|------|----------|---|------------|--------------|
| K-TableID = F9S08 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 8 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 8100 | 0 | 8100 | 8100 | GS1-128 Coupon Extended Code - NSC + Offer Code ** DEPRECATED as of GS15i2 ** | - | 6n |
| 8101 | 1 | 8101 | 8101 | GS1-128 Coupon Extended Code - NSC + Offer Code + end of offer code ** DEPRECATED as of GS15i2 ** | - | 10n |
| 8102 | 2 | 8102 | 8102 | GS1-128 Coupon Extended Code - NSC ** DEPRECATED as of GS15i2 ** | - | 2n |
| 8110 | 3 | 8110 | 8110 | Coupon Code Identification for Use in North America | | 1*70an |
| 8111 | 4 | 8111 | 8111 | Loyalty points of a coupon | POINTS | 4n |
| K-TableEnd = F9S08 | | | | | | |

6201

| K-Text = Sec. IDT - Length or first dimension | | | | | | |
|---|---------|------------|------------|-----------------------------------|-----------------|--------------|
| K-TableID = F9S09 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 4 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 331(***) | 0 | 331%x30-35 | 331%x30-35 | Length or first dimension, metres | LENGTH (m), log | 6n |
| 341(***) | 1 | 341%x30-35 | 341%x30-35 | Length or first dimension, inches | LENGTH (i), log | 6n |
| 342(***) | 2 | 342%x30-35 | 342%x30-35 | Length or first dimension, feet | LENGTH (f), log | 6n |
| 343(***) | 3 | 343%x30-35 | 343%x30-35 | Length or first dimension, yards | LENGTH (y), log | 6n |
| K-TableEnd = F9S09 | | | | | | |

6202

| K-Text = Sec. IDT - Width, diameter, or second dimension | | | | | | |
|--|--|--|--|--|--|--|
| K-TableID = F9S10 | | | | | | |

| K-Text = Sec. IDT - Width, diameter, or second dimension | | | | | | |
|--|---------|------------|------------|--|----------------|--------------|
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 4 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 332(***) | 0 | 332%x30-35 | 332%x30-35 | Width, diameter, or second dimension, metres | WIDTH (m), log | 6n |
| 344(***) | 1 | 344%x30-35 | 344%x30-35 | Width, diameter, or second dimension | WIDTH (i), log | 6n |
| 345(***) | 2 | 345%x30-35 | 345%x30-35 | Width, diameter, or second dimension | WIDTH (f), log | 6n |
| 346(***) | 3 | 346%x30-35 | 346%x30-35 | Width, diameter, or second dimension | WIDTH (y), log | 6n |
| K-TableEnd = F9S10 | | | | | | |

6203

| K-Text = Sec. IDT - Depth, thickness, height, or third dimension | | | | | | |
|--|---------|------------|------------|--|-----------------|--------------|
| K-TableID = F9S11 | | | | | | |
| K-RootOID = urn:oid:1.0.15961.9 | | | | | | |
| K-IDsize = 4 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 333(***) | 0 | 333%x30-35 | 333%x30-35 | Depth, thickness, height, or third dimension, metres | HEIGHT (m), log | 6n |
| 347(***) | 1 | 347%x30-35 | 347%x30-35 | Depth, thickness, height, or third dimension | HEIGHT (i), log | 6n |
| 348(***) | 2 | 348%x30-35 | 348%x30-35 | Depth, thickness, height, or third dimension | HEIGHT (f), log | 6n |
| 349(***) | 3 | 349%x30-35 | 349%x30-35 | Depth, thickness, height, or third dimension | HEIGHT (y), log | 6n |
| K-TableEnd = F9S11 | | | | | | |

6204

| K-Text = Sec. IDT - Additional AIs |
|------------------------------------|
| K-TableID = F9S12 |
| K-RootOID = urn:oid:1.0.15961.9 |

| K-Text = Sec. IDT - Additional AIs | | | | | | |
|------------------------------------|---------|------------|------------|--|--------------------|--------------|
| K-IDsize = 128 | | | | | | |
| AI or AIs | IDvalue | OIDs | IDstring | Name | Data Title | FormatString |
| 243 | 0 | 243 | 243 | Packaging Component Number | PCN | 1*20an |
| 255 | 1 | 255 | 255 | Global Coupon Number | GCN | 13n 0*12n |
| 427 | 2 | 427 | 427 | Country Subdivision of Origin Code for a Trade Item | ORIGIN SUBDIVISION | 1*3an |
| 710 | 3 | 710 | 710 | National Healthcare Reimbursement Number - Germany (PZN) | NHRN PZN | 3n 1*27an |
| 711 | 4 | 711 | 711 | National Healthcare Reimbursement Number - France (CIP) | NHRN CIP | 3n 1*27an |
| 712 | 5 | 712 | 712 | National Healthcare Reimbursement Number - Spain (CN) | NHRN CN | 3n 1*27an |
| 713 | 6 | 713 | 713 | National Healthcare Reimbursement Number - Brazil (DRN) | NHRN DRN | 3n 1*27an |
| 8010 | 7 | 8010 | 8010 | Component / Part Identifier | CPID | 1*30an |
| 8011 | 8 | 8011 | 8011 | Component / Part Identifier Serial Number | CPID Serial | 1*12n |
| 8017 | 9 | 8017 | 8017 | Global Service Relation Number - Provider | GSRN - PROVIDER | 18n |
| 8019 | 10 | 8019 | 8019 | Service Relation Instance Number | SRIN | 1*10n |
| 8200 | 11 | 8200 | 8200 | Extended Packaging URL | PRODUCT URL | 1*70an |
| 16 | 12 | 16 | 16 | Sell by date (YYMMDD) | SELL BY | 6n |
| 394n | 13 | 394%x30-33 | 394%x30-33 | Percentage discount of a coupon | PCT OFF | 4n |
| 7005 | 14 | 7005 | 7005 | Catch area | CATCH AREA | 1*12an |
| 7006 | 15 | 7006 | 7006 | First freeze date | FIRST FREEZE DATE | 6n |
| 7007 | 16 | 7007 | 7007 | Harvest date | HARVEST DATE | 6*12an |

| K-Text = Sec. IDT - Additional AIs | | | | | | |
|------------------------------------|----|------------|------------|---|-------------------|----------|
| 7008 | 17 | 7008 | 7008 | Species for fishery purposes | ACQUATIC SPECIES | 1*3an |
| 7009 | 18 | 7009 | 7009 | Fishing gear type | FISHING GEAR TYPE | 1*10an |
| 7010 | 19 | 7010 | 7010 | Production method | PROD METHOD | 1*2an |
| 8012 | 20 | 8012 | 8012 | Software version | VERSION | 1*20an |
| 416 | 21 | 416 | 416 | GLN of the production or service location | PROD/SERV /LOC | 13n |
| 7020 | 22 | 7020 | 7020 | Refurbishment lot ID | REFURB LOT | 1*20an |
| 7021 | 23 | 7021 | 7021 | Functional status | FUNC STAT | 1*20an |
| 7022 | 24 | 7022 | 7022 | Revision status | REV STAT | 1*20an |
| 7023 | 25 | 7023 | 7023 | Global Individual Asset Identifier (GIAI) of an assembly | GIAI - ASSEMBLY | 1*30an |
| 235 | 26 | 235 | 235 | Third party controlled, serialised extension of GTIN | TPX | 1*28an |
| 417 | 27 | 417 | 417 | Global Location Number of Party | PARTY | 13n |
| 714 | 28 | 714 | 714 | National Healthcare Reimbursement Number - Portugal (AIM) | NHRN AIM | 1*an20 |
| 7040 | 29 | 7040 | 7040 | Unique Identification Code with Extensions (per EU 2018/574) | UIC | 1n 1*3an |
| 8013 | 30 | 8013 | 8013 | Global Model Number | GMN | 1*an30 |
| 8026 | 31 | 8026 | 8026 | Identification of pieces of a trade item (ITIP) contained in a logistics unit | ITIP CONTENT | 18n |
| 8112 | 32 | 8112 | 8112 | Paperless coupon code identification for use in North America | | 1*an70 |
| 7240 | 33 | 7240 | 7240 | Protocol ID | PROTOCOL | 1*20an |
| 395(***) | 34 | 395%x30-35 | 395%x30-35 | Amount Payable per unit of measure single monetary area (variable measure trade item) | PRICE/UoM | 6n |

| K-Text = Sec. IDT - Additional AIs | | | | | | |
|------------------------------------|----|------|------|---|-----------------|--------|
| 4300 | 35 | 4300 | 4300 | Ship-to / Deliver-to company name | SHIP TO COMP | 1*35an |
| 4301 | 36 | 4301 | 4301 | Ship-to / Deliver-to contact name: AI | SHIP TO NAME | 1*35an |
| 4302 | 37 | 4302 | 4302 | Ship-to / Deliver-to address line 1: AI | SHIP TO ADD1 | 1*70an |
| 4303 | 38 | 4303 | 4303 | Ship-to / Deliver-to address line 2: AI | SHIP TO ADD2 | 1*70an |
| 4304 | 39 | 4304 | 4304 | Ship-to / Deliver-to suburb | SHIP TO SUB | 1*70an |
| 4305 | 40 | 4305 | 4305 | Ship-to / Deliver-to locality | SHIP TO LOC | 1*70an |
| 4306 | 41 | 4306 | 4306 | Ship-to / Deliver-to region | SHIP TO REG | 1*70an |
| 4307 | 42 | 4307 | 4307 | Ship-to / Deliver-to country code | SHIP TO COUNTRY | 2an |
| 4308 | 43 | 4308 | 4308 | Ship-to / Deliver-to telephone number | SHIP TO PHONE | 1*30an |
| 4309 | 44 | 4309 | 4309 | Ship-to / Deliver-to GEO location | SHIP TO GEO | 20n |
| 4310 | 45 | 4310 | 4310 | Return-to company name | RTN TO COMP | 1*35an |
| 4311 | 46 | 4311 | 4311 | Return-to contact name | RTN TO NAME | 1*35an |
| 4312 | 47 | 4312 | 4312 | Return-to address line 1 | RTN TO ADD1 | 1*70an |
| 4313 | 48 | 4313 | 4313 | Return-to address line 2 | RTN TO ADD2 | 1*70an |
| 4314 | 49 | 4314 | 4314 | Return-to suburb | RTN TO SUB | 1*70an |
| 4315 | 50 | 4315 | 4315 | Return-to locality | RTN TO LOC | 1*70an |
| 4316 | 51 | 4316 | 4316 | Return-to region | RTN TO REG | 1*70an |
| 4317 | 52 | 4317 | 4317 | Return-to country code | RTN TO COUNTRY | 2an |
| 4318 | 53 | 4318 | 4318 | Return-to postal code | RTN TO POST | 1*20an |
| 4319 | 54 | 4319 | 4319 | Return-to telephone number | RTN TO PHONE | 1*30an |
| 4320 | 55 | 4320 | 4320 | Service code description | SRV DESCRIPTION | 1*35an |
| 4321 | 56 | 4321 | 4321 | Dangerous goods flag | DANGEROUS GOODS | 1n |
| 4322 | 57 | 4322 | 4322 | Authority to leave flag | AUTH LEAV | 1n |
| 4323 | 58 | 4323 | 4323 | Signature required flag | SIG REQUIRED | 1n |



| K-Text = Sec. IDT - Additional AIs | | | | | | |
|------------------------------------|----|------|------|---|-----------------|------------|
| 4324 | 59 | 4324 | 4324 | Not before delivery date/time | NBEF DEL DT | 10n |
| 4325 | 60 | 4325 | 4325 | Not after delivery date/time | NAFT DEL DT | 10n |
| 4326 | 61 | 4326 | 4326 | Release date | REL DATE | 6n |
| 715 | 62 | 715 | 715 | National Healthcare Reimbursement Number - United States of America NDC | NHRN NDC | 1*an20 |
| 723s | 63 | 7230 | 7230 | Certification reference | CERT # s | 2an 1*28an |
| 723s | 64 | 7231 | 7231 | Certification reference | CERT # s | 2an 1*28an |
| 723s | 65 | 7232 | 7232 | Certification reference | CERT # s | 2an 1*28an |
| 723s | 66 | 7233 | 7233 | Certification reference | CERT # s | 2an 1*28an |
| 723s | 67 | 7234 | 7234 | Certification reference | CERT # s | 2an 1*28an |
| 723s | 68 | 7235 | 7235 | Certification reference | CERT # s | 2an 1*28an |
| 723s | 69 | 7236 | 7236 | Certification reference | CERT # s | 2an 1*28an |
| 723s | 70 | 7237 | 7237 | Certification reference | CERT # s | 2an 1*28an |
| 723s | 71 | 7238 | 7238 | Certification reference | CERT # s | 2an 1*28an |
| 723s | 72 | 7239 | 7239 | Certification reference | CERT # s | 2an 1*28an |
| 7241 | 73 | 7241 | 7241 | AIDC media type | AIDC MEDIA TYPE | 2n |
| 7242 | 74 | 7242 | 7242 | Version Control Number (VCN) | VCN | 1*25an |
| 8030 | 75 | 8030 | 8030 | Digital Signature (DigSig) | DIGSIG | 1*90an |
| 7011 | 76 | 7011 | 7011 | Test by date | TEST BY DATE | 6n 0*4n |
| 4330 | 77 | 4330 | 4330 | Maximum temperature in Fahrenheit | MAX TEMP F | 6n 0*1an |
| 4331 | 78 | 4331 | 4331 | Maximum temperature in Celsius | MAX TEMP C | 6n 0*1an |
| 4332 | 79 | 4332 | 4332 | Minimum temperature in Fahrenheit | MIN TEMP F | 6n 0*1an |
| 4333 | 80 | 4333 | 4333 | Minimum temperature in Celsius | MIN TEMP C | 6n 0*1an |
| K-TableEnd = F9S12 | | | | | | |

F.2 Comma-Separated-Value (CSV) format

6205
6206
6207
6208
6209

This section is the Packed Objects ID Table for Data Format 9 (GS1 Application Identifiers) in machine readable, comma-separated-value format, as registered with ISO. See Section F.1 for a non-normative listing of the content of the ID Table for Data Format 9, in a human readable, tabular format.

6210
6211
6212
6213

In the comma-separated-value format, line breaks are significant. However, certain lines are too long to fit within the margins of this document. In the listing below, the symbol █ at the end of line indicates that the ID Table line is continued on the following line. Such a line shall be interpreted by concatenating the following line and omitting the █ symbol.

6214
6215
6216

Note that, as of TDS 2.1, the *Packed Objects ID Table for Data Format 9* in Section F.2 has been supplemented with an **external, normative artefact in CSV format**, which can be found online at <https://ref.gs1.org/standards/tds/artefacts>.

6217

6218

K-Text = GS1 AI ID Table for ISO/IEC 15961 Format 9,,,,,,

6219

K-Version = 1.00,,,,,,

6220

K-ISO15434=05,,,,,,

6221

K-Text = Primary Base Table,,,,,,

6222

K-TableID = F9B0,,,,,,

6223

K-RootOID = urn:oid:1.0.15961.9,,,,,,

6224

K-IDsize = 90,,,,,,

6225

AI or AIs, IDvalue, OIDs, IDstring, Name, Data Title, FormatString

6226

0,1,0,0,SSCC (Serial Shipping Container Code),SSCC,18n

6227

1,2,1,1,Global Trade Item Number,GTIN,14n

6228

02 + 37,3,(2)(37),(02)(37),GTIN + Count of trade items contained in a logistic █

6229

unit,CONTENT + COUNT,(14n)(1*8n)

6230

10,4,10,10,Batch or lot number,BATCH/LOT,1*20an

6231

11,5,11,11,Production date (YYMMDD),PROD DATE,6n

6232

12,6,12,12,Due date (YYMMDD),DUE DATE,6n

6233

13,7,13,13,Packaging date (YYMMDD),PACK DATE,6n

6234

15,8,15,15,Best before date (YYMMDD),BEST BEFORE OR SELL BY,6n

6235

17,9,17,17,Expiration date (YYMMDD),USE BY OR EXPIRY,6n

6236

20,10,20,20,Internal product variant,VARIANT,2n

6237

21,11,21,21,Serial number,SERIAL,1*20an

6238

22,12,22,22,Consumer product variant,CPV,1*20an

6239

240,13,240,240,Additional product identification assigned by the

6240

manufacturer,ADDITIONAL ID,1*30an

6241

241,14,241,241,Customer part number,CUST. PART NO.,1*30an

6242

242,15,242,242,Made-to-Order Variation Number,VARIATION NUMBER,1*6n

6243

250,16,250,250,Secondary serial number,SECONDARY SERIAL,1*30an

6244

251,17,251,251,Reference to source entity,REF. TO SOURCE,1*30an

6245

253,18,253,253,Global Document Type Identifier,DOC. ID,13n 0*17an

6246

30,19,30,30,Variable count,VAR. COUNT,1*8n

6247

310n 320n etc,20,K-Secondary = S00,, "Net weight, kilograms or pounds or troy oz █

6248

(Variable Measure Trade Item)",,

6249

311n 321n etc,21,K-Secondary = S01,, Length of first dimension (Variable Measure █

6250

Trade Item)",,

6251

312n 324n etc,22,K-Secondary = S02,, "Width, diameter, or second dimension (Variable █

6252

Measure Trade Item)",,

6253

313n 327n etc,23,K-Secondary = S03,, "Depth, thickness, height, or third dimension █

6254

(Variable Measure Trade Item)",,

6255

314n 350n etc,24,K-Secondary = S04,, Area (Variable Measure Trade Item)",,

6256

315n 316n etc,25,K-Secondary = S05,, Net volume (Variable Measure Trade Item)",,

6257

330n or 340n,26,330%x30-36 / 340%x30-36,330%x30-36 / 340%x30-36, "Logistic weight, █

6258

kilograms or pounds",GROSS WEIGHT (kg) or (lb),6n / 6n

6259

"331n, 341n, etc",27,K-Secondary = S09,, Length or first dimension",,

6260

"332n, 344n, etc",28,K-Secondary = S10,, "Width, diameter, or second dimension",,

6261

"333n, 347n, etc",29,K-Secondary = S11,, "Depth, thickness, height, or third █

6262

dimension)",,

6263

334n 353n etc,30,K-Secondary = S07,, Logistic Area",,

6264

335n 336n etc,31,K-Secondary = S06,335%x30-36,Logistic volume",,

6265

337(**),32,337%x30-36,337%x30-36,Kilograms per square metre,KG PER m^2,6n



6266 390n or 391n,33,390%x30-39 / 391%x30-39,390%x30-39 / 391%x30-39,Amount payable -
6267 single monetary area or with ISO currency code,AMOUNT,1*15n / 4*18n
6268 392n or 393n,34,392%x30-39 / 393%x30-39,392%x30-39 / 393%x30-39,Amount payable for
6269 Variable Measure Trade Item - single monetary unit or ISO cc, PRICE,1*15n / 4*18n
6270 400,35,400,400,Customer's purchase order number,ORDER NUMBER,1*30an
6271 401,36,401,401,Global Identification Number for Consignment,GINC,1*30an
6272 402,37,402,402,Global Shipment Identification Number,GSIN,17n
6273 403,38,403,403,Routing code,ROUTE,1*30an
6274 410,39,410,410,Ship to - Deliver to Global Location Number,SHIP TO LOC,13n
6275 411,40,411,411,Bill to - Invoice to Global Location Number,BILL TO,13n
6276 412,41,412,412,Purchased from Global Location Number,PURCHASE FROM,13n
6277 413,42,413,413,Ship for - Deliver for - Forward to Global Location Number,SHIP FOR
6278 LOC,13n
6279 414 and 254,43,(414) [254],(414) [254],"Identification of a physical location GLN,
6280 and optional Extension",LOC No + GLN EXTENSION,(13n) [1*20an]
6281 415 and 8020,44,(415) (8020),(415) (8020),Global Location Number of the Invoicing
6282 Party and Payment Slip Reference Number,PAY + REF No,(13n) (1*25an)
6283 420 or 421,45,(420/421),(420/421),Ship-to / Deliver-to postal code,SHIP TO
6284 POST,(1*20an / 3n 1*9an)
6285 422,46,422,422,Country of origin of a trade item,ORIGIN,3n
6286 423,47,423,423,Country of initial processing,COUNTRY - INITIAL PROCESS.,3*15n
6287 424,48,424,424,Country of processing,COUNTRY - PROCESS.,3n
6288 425,49,425,425,Country of disassembly,COUNTRY - DISASSEMBLY,3n
6289 426,50,426,426,Country covering full process chain,COUNTRY - FULL PROCESS,3n
6290 7001,51,7001,7001,NATO stock number,NSN,13n
6291 7002,52,7002,7002,UN/ECE meat carcasses and cuts classification,MEAT CUT,1*30an
6292 7003,53,7003,7003,Expiration Date and Time,EXPIRY DATE/TIME,10n
6293 7004,54,7004,7004,Active Potency,ACTIVE POTENCY,1*4n
6294 703s,55,7030,7030,Approval number of processor with ISO country code,PROCESSOR #
6295 s,3n 1*27an
6296 703s,56,7031,7031,Approval number of processor with ISO country code,PROCESSOR #
6297 s,3n 1*27an
6298 703s,57,7032,7032,Approval number of processor with ISO country code,PROCESSOR #
6299 s,3n 1*27an
6300 703s,58,7033,7033,Approval number of processor with ISO country code,PROCESSOR #
6301 s,3n 1*27an
6302 703s,59,7034,7034,Approval number of processor with ISO country code,PROCESSOR #
6303 s,3n 1*27an
6304 703s,60,7035,7035,Approval number of processor with ISO country code,PROCESSOR #
6305 s,3n 1*27an
6306 703s,61,7036,7036,Approval number of processor with ISO country code,PROCESSOR #
6307 s,3n 1*27an
6308 703s,62,7037,7037,Approval number of processor with ISO country code,PROCESSOR #
6309 s,3n 1*27an
6310 703s,63,7038,7038,Approval number of processor with ISO country code,PROCESSOR #
6311 s,3n 1*27an
6312 703s,64,7039,7039,Approval number of processor with ISO country code,PROCESSOR #
6313 s,3n 1*27an
6314 8001,65,8001,8001,"Roll products - width, length, core diameter, direction,
6315 splices",DIMENSIONS,14n
6316 8002,66,8002,8002,Electronic serial identifier for cellular mobile telephones,CMT
6317 No,1*20an
6318 8003,67,8003,8003,Global Returnable Asset Identifier,GRAI,14n 0*16an
6319 8004,68,8004,8004,Global Individual Asset Identifier,GIAI,1*30an
6320 8005,69,8005,8005,Price per unit of measure,PRICE PER UNIT,6n
6321 8006,70,8006,8006,Identification of the component of a trade item,GCTIN,18n
6322 8007,71,8007,8007,International Bank Account Number,IBAN,1*30an
6323 8008,72,8008,8008,Date and time of production,PROD TIME,8*12n
6324 8018,73,8018,8018,Global Service Relation Number - Recipient,GSRN - RECIPIENT,18n
6325 8100 8101 etc,74,K-Secondary = S08,,Coupon Codes,,
6326 90,75,90,90,Information mutually agreed between trading partners (including FACT
6327 DI),INTERNAL,1*30an
6328 91,76,91,91,Company internal information,INTERNAL,1*an
6329 92,77,92,92,Company internal information,INTERNAL,1*an
6330 93,78,93,93,Company internal information,INTERNAL,1*an
6331 94,79,94,94,Company internal information,INTERNAL,1*an


```

6332 95,80,95,95,Company internal information,INTERNAL,1*an
6333 96,81,96,96,Company internal information,INTERNAL,1*an
6334 97,82,97,97,Company internal information,INTERNAL,1*an
6335 98,83,98,98,Company internal information,INTERNAL,1*an
6336 99,84,99,99,Company internal information,INTERNAL,1*an
6337 nnn,85,K-Secondary = S12,,Additional AIs,,
6338 K-TableEnd = F9B0,,,,,,
6339
6340 "K-Text = Sec. IDT - Net weight, kilograms or pounds or troy oz (Variable Measure
6341 Trade Item)",,,,,,
6342 K-TableID = F9S00,,,,,,
6343 K-RootOID = urn:oid:1.0.15961.9,,,,,,
6344 K-IDsize = 4,,,,,,
6345 AI or AIs,IDvalue,OIDs,IDstring,Name,Data Title,FormatString
6346 310(***),0,310%x30-35,310%x30-35,"Net weight, kilograms (Variable Measure Trade
6347 Item)",NET WEIGHT (kg),6n
6348 320(***),1,320%x30-35,320%x30-35,"Net weight, pounds (Variable Measure Trade
6349 Item)",NET WEIGHT (lb),6n
6350 356(***),2,356%x30-35,356%x30-35,"Net weight, troy ounces (Variable Measure Trade
6351 Item)",NET WEIGHT (t),6n
6352 K-TableEnd = F9S00,,,,,,
6353
6354 K-Text = Sec. IDT - Length of first dimension (Variable Measure Trade Item),,,,,,
6355 K-TableID = F9S01,,,,,,
6356 K-RootOID = urn:oid:1.0.15961.9,,,,,,
6357 K-IDsize = 4,,,,,,
6358 AI or AIs,IDvalue,OIDs,IDstring,Name,Data Title,FormatString
6359 311(***),0,311%x30-35,311%x30-35,"Length of first dimension, metres (Variable
6360 Measure Trade Item)",LENGTH (m),6n
6361 321(***),1,321%x30-35,321%x30-35,"Length or first dimension, inches (Variable
6362 Measure Trade Item)",LENGTH (i),6n
6363 322(***),2,322%x30-35,322%x30-35,"Length or first dimension, feet (Variable Measure
6364 Trade Item)",LENGTH (f),6n
6365 323(***),3,323%x30-35,323%x30-35,"Length or first dimension, yards (Variable
6366 Measure Trade Item)",LENGTH (y),6n
6367 K-TableEnd = F9S01,,,,,,
6368
6369 "K-Text = Sec. IDT - Width, diameter, or second dimension (Variable Measure Trade
6370 Item)",,,,,,
6371 K-TableID = F9S02,,,,,,
6372 K-RootOID = urn:oid:1.0.15961.9,,,,,,
6373 K-IDsize = 4,,,,,,
6374 AI or AIs,IDvalue,OIDs,IDstring,Name,Data Title,FormatString
6375 312(***),0,312%x30-35,312%x30-35,"Width, diameter, or second dimension, metres
6376 (Variable Measure Trade Item)",WIDTH (m),6n
6377 324(***),1,324%x30-35,324%x30-35,"Width, diameter, or second dimension, inches
6378 (Variable Measure Trade Item)",WIDTH (i),6n
6379 325(***),2,325%x30-35,325%x30-35,"Width, diameter, or second dimension, (Variable
6380 Measure Trade Item)",WIDTH (f),6n
6381 326(***),3,326%x30-35,326%x30-35,"Width, diameter, or second dimension, yards
6382 (Variable Measure Trade Item)",WIDTH (y),6n
6383 K-TableEnd = F9S02,,,,,,
6384
6385 "K-Text = Sec. IDT - Depth, thickness, height, or third dimension (Variable Measure
6386 Trade Item)",,,,,,
6387 K-TableID = F9S03,,,,,,
6388 K-RootOID = urn:oid:1.0.15961.9,,,,,,
6389 K-IDsize = 4,,,,,,
6390 AI or AIs,IDvalue,OIDs,IDstring,Name,Data Title,FormatString
6391 313(***),0,313%x30-35,313%x30-35,"Depth, thickness, height, or third dimension,
6392 metres (Variable Measure Trade Item)",HEIGHT (m),6n
6393 327(***),1,327%x30-35,327%x30-35,"Depth, thickness, height, or third dimension,
6394 inches (Variable Measure Trade Item)",HEIGHT (i),6n
6395 328(***),2,328%x30-35,328%x30-35,"Depth, thickness, height, or third dimension,
6396 feet (Variable Measure Trade Item)",HEIGHT (f),6n

```

```

6397      329(***) ,3,329%x30-35,329%x30-35,"Depth, thickness, height, or third dimension,
6398      yards (Variable Measure Trade Item)",HEIGHT (y),6n
6399      K-TableEnd = F9S03,,,,,,,,
6400
6401      K-Text = Sec. IDT - Area (Variable Measure Trade Item),,,,,,,,,
6402      K-TableID = F9S04,,,,,,,,
6403      K-RootOID = urn:oid:1.0.15961.9,,,,,,,,
6404      K-IDsize = 4,,,,,,,,
6405      AI or AIs, IDvalue, OIDs, IDstring, Name, Data Title, FormatString
6406      314(***) ,0,314%x30-35,314%x30-35,"Area, square metres (Variable Measure Trade
6407      Item)",AREA (m^2),6n
6408      350(***) ,1,350%x30-35,350%x30-35,"Area, square inches (Variable Measure Trade
6409      Item)",AREA (i^2),6n
6410      351(***) ,2,351%x30-35,351%x30-35,"Area, square feet (Variable Measure Trade
6411      Item)",AREA (f^2),6n
6412      352(***) ,3,352%x30-35,352%x30-35,"Area, square yards (Variable Measure Trade
6413      Item)",AREA (y^2),6n
6414      K-TableEnd = F9S04,,,,,,,,
6415
6416      K-Text = Sec. IDT - Net volume (Variable Measure Trade Item),,,,,,,,,
6417      K-TableID = F9S05,,,,,,,,
6418      K-RootOID = urn:oid:1.0.15961.9,,,,,,,,
6419      K-IDsize = 8,,,,,,,,
6420      AI or AIs, IDvalue, OIDs, IDstring, Name, Data Title, FormatString
6421      315(***) ,0,315%x30-35,315%x30-35,"Net volume, litres (Variable Measure Trade
6422      Item)",NET VOLUME (l),6n
6423      316(***) ,1,316%x30-35,316%x30-35,"Net volume, cubic metres (Variable Measure Trade
6424      Item)",NET VOLUME (m^3),6n
6425      357(***) ,2,357%x30-35,357%x30-35,"Net weight (or volume), ounces (Variable Measure
6426      Trade Item)",NET VOLUME (oz),6n
6427      360(***) ,3,360%x30-35,360%x30-35,"Net volume, quarts (Variable Measure Trade
6428      Item)",NET VOLUME (q),6n
6429      361(***) ,4,361%x30-35,361%x30-35,"Net volume, gallons U.S. (Variable Measure Trade
6430      Item)",NET VOLUME (g),6n
6431      364(***) ,5,364%x30-35,364%x30-35,"Net volume, cubic inches","VOLUME (i^3), log",6n
6432      365(***) ,6,365%x30-35,365%x30-35,"Net volume, cubic feet (Variable Measure Trade
6433      Item)","VOLUME (f^3), log",6n
6434      366(***) ,7,366%x30-35,366%x30-35,"Net volume, cubic yards (Variable Measure Trade
6435      Item)","VOLUME (y^3), log",6n
6436      K-TableEnd = F9S05,,,,,,,,
6437
6438      K-Text = Sec. IDT - Logistic Volume,,,,,,,,
6439      K-TableID = F9S06,,,,,,,,
6440      K-RootOID = urn:oid:1.0.15961.9,,,,,,,,
6441      K-IDsize = 8,,,,,,,,
6442      AI or AIs, IDvalue, OIDs, IDstring, Name, Data Title, FormatString
6443      335(***) ,0,335%x30-35,335%x30-35,"Logistic volume, litres","VOLUME (l), log",6n
6444      336(***) ,1,336%x30-35,336%x30-35,"Logistic volume, cubic meters","VOLUME (m^3),
6445      log",6n
6446      362(***) ,2,362%x30-35,362%x30-35,"Logistic volume, quarts","VOLUME (q), log",6n
6447      363(***) ,3,363%x30-35,363%x30-35,"Logistic volume, gallons","VOLUME (g), log",6n
6448      367(***) ,4,367%x30-35,367%x30-35,"Logistic volume, cubic inches","VOLUME (q),
6449      log",6n
6450      368(***) ,5,368%x30-35,368%x30-35,"Logistic volume, cubic feet","VOLUME (g), log",6n
6451      369(***) ,6,369%x30-35,369%x30-35,"Logistic volume, cubic yards","VOLUME (i^3),
6452      log",6n
6453      K-TableEnd = F9S06,,,,,,,,
6454
6455      K-Text = Sec. IDT - Logistic Area,,,,,,,,
6456      K-TableID = F9S07,,,,,,,,
6457      K-RootOID = urn:oid:1.0.15961.9,,,,,,,,
6458      K-IDsize = 4,,,,,,,,
6459      AI or AIs, IDvalue, OIDs, IDstring, Name, Data Title, FormatString
6460      334(***) ,0,334%x30-35,334%x30-35,"Area, square metres","AREA (m^2), log",6n
6461      353(***) ,1,353%x30-35,353%x30-35,"Area, square inches","AREA (i^2), log",6n
6462      354(***) ,2,354%x30-35,354%x30-35,"Area, square feet","AREA (f^2), log",6n

```

6463 355(***),3,355%x30-35,355%x30-35,"Area, square yards","AREA (y^2), log",6n
6464 K-TableEnd = F9S07,,,,,,,,
6465
6466 K-Text = Sec. IDT - Coupon Codes,,,,,,,,
6467 K-TableID = F9S08,,,,,,,,
6468 K-RootOID = urn:oid:1.0.15961.9,,,,,,,,
6469 K-IDsize = 8,,,,,,,,
6470 AI or AIs,IDvalue,OIDs,IDstring,Name,Data Title,FormatString
6471 8100,0,8100,8100,GS1-128 Coupon Extended Code - NSC + Offer Code ** DEPRECATED as of
6472 GS1GS15i2 **,-,6n
6473 8101,1,8101,8101,GS1-128 Coupon Extended Code - NSC + Offer Code + end of offer
6474 code ** DEPRECATED as of GS1GS15i2 **,-,10n
6475 8102,2,8102,8102,GS1-128 Coupon Extended Code - NSC ** DEPRECATED as of GS1GS15i2
6476 **,-,2n
6477 8110,3,8110,8110,Coupon Code Identification for Use in North America,,1*70an
6478 8111,22,8111,8111,Loyalty points of a coupon,POINTS,4n
6479 K-TableEnd = F9S08,,,,,,,,
6480
6481 K-Text = Sec. IDT - Length or first dimension,,,,,,,,
6482 K-TableID = F9S09,,,,,,,,
6483 K-RootOID = urn:oid:1.0.15961.9,,,,,,,,
6484 K-IDsize = 4,,,,,,,,
6485 AI or AIs,IDvalue,OIDs,IDstring,Name,Data Title,FormatString
6486 331(***),0,331%x30-35,331%x30-35,"Length or first dimension, metres","LENGTH (m),
6487 log",6n
6488 341(***),1,341%x30-35,341%x30-35,"Length or first dimension, inches","LENGTH (i),
6489 log",6n
6490 342(***),2,342%x30-35,342%x30-35,"Length or first dimension, feet","LENGTH (f),
6491 log",6n
6492 343(***),3,343%x30-35,343%x30-35,"Length or first dimension, yards","LENGTH (y),
6493 log",6n
6494 K-TableEnd = F9S09,,,,,,,,
6495
6496 "K-Text = Sec. IDT - Width, diameter, or second dimension",,,,,,,,,
6497 K-TableID = F9S10,,,,,,,,
6498 K-RootOID = urn:oid:1.0.15961.9,,,,,,,,
6499 K-IDsize = 4,,,,,,,,
6500 AI or AIs,IDvalue,OIDs,IDstring,Name,Data Title,FormatString
6501 332(***),0,332%x30-35,332%x30-35,"Width, diameter, or second dimension,
6502 metres","WIDTH (m), log",6n
6503 344(***),1,344%x30-35,344%x30-35,"Width, diameter, or second dimension","WIDTH
6504 (i), log",6n
6505 345(***),2,345%x30-35,345%x30-35,"Width, diameter, or second dimension","WIDTH
6506 (f), log",6n
6507 346(***),3,346%x30-35,346%x30-35,"Width, diameter, or second dimension","WIDTH
6508 (y), log",6n
6509 K-TableEnd = F9S10,,,,,,,,
6510
6511 "K-Text = Sec. IDT - Depth, thickness, height, or third dimension",,,,,,,,,
6512 K-TableID = F9S11,,,,,,,,
6513 K-RootOID = urn:oid:1.0.15961.9,,,,,,,,
6514 K-IDsize = 4,,,,,,,,
6515 AI or AIs,IDvalue,OIDs,IDstring,Name,Data Title,FormatString
6516 333(***),0,333%x30-35,333%x30-35,"Depth, thickness, height, or third dimension,
6517 metres","HEIGHT (m), log",6n
6518 347(***),1,347%x30-35,347%x30-35,"Depth, thickness, height, or third
6519 dimension","HEIGHT (i), log",6n
6520 348(***),2,348%x30-35,348%x30-35,"Depth, thickness, height, or third
6521 dimension","HEIGHT (f), log",6n
6522 349(***),3,349%x30-35,349%x30-35,"Depth, thickness, height, or third
6523 dimension","HEIGHT (y), log",6n
6524 K-TableEnd = F9S11,,,,,,,,
6525
6526 K-Text = Sec. IDT - Additional AIs,,,,,,,,
6527 K-TableID = F9S12,,,,,,,,
6528 K-RootOID = urn:oid:1.0.15961.9,,,,,,,,



6529 K-IDsize = 128,,,,,
6530 AI or AIs, IDvalue, OIDs, IDstring, Name, Data Title, FormatString
6531 243,0,243,243, Packaging Component Number, PCN, 1*20an
6532 255,1,255,255, Global Coupon Number, GCN, 13n 0*12n
6533 427,2,427,427, Country Subdivision of Origin Code for a Trade Item, ORIGIN
6534 SUBDIVISION, 1*3an
6535 710,3,710,710, National Healthcare Reimbursement Number - Germany (PZN), NHRN PZN, 3n
6536 1*27an
6537 711,4,711,711, National Healthcare Reimbursement Number - France (CIP), NHRN CIP, 3n
6538 1*27an
6539 712,5,712,712, National Healthcare Reimbursement Number - Spain (CN), NHRN CN, 3n
6540 1*27an
6541 713,6,713,713, National Healthcare Reimbursement Number - Brazil (DRN), NHRN DRN, 3n
6542 1*27an
6543 8010,7,8010,8010, Component / Part Identifier, CPID, 1*30an
6544 8011,8,8011,8011, Component / Part Identifier Serial Number, CPID Serial, 1*12n
6545 8017,9,8017,8017, Global Service Relation Number - Provider, GSRN - PROVIDER, 18n
6546 8019,10,8019,8019, Service Relation Instance Number, SRIN, 1*10n
6547 8200,11,8200,8200, Extended Packaging URL, PRODUCT URL, 1*70an
6548 16,12,16,16, Sell by date (YYMMDD), SELL BY, 6n
6549 394n,13,394%x30-39,394%x30-39, Percentage discount of a coupon, PCT OFF, 4n
6550 7005,14,7005,7005, Catch area, CATCH AREA, 1*12an
6551 7006,15,7006,7006, First freeze date, FIRST FREEZE DATE, 6n
6552 7007,16,7007,7007, Harvest date, HARVEST DATE, 6*12an
6553 7008,17,7008,7008, Species for fishery purposes, ACQUATIC SPECIES, 1*3an
6554 7009,18,7009,7009, Fishing gear type, FISHING GEAR TYPE, 1*10an
6555 7010,19,7010,7010, Production method, PROD METHOD, 1*2an
6556 8012,20,8012,8012, Software version, VERSION, 1*20an
6557 416,21,416,416, GLN of the production or service location, PROD/SERV/LOC, 13n
6558 7020,22,7020,7020, Refurbishment lot ID, REFURB LOT, 1*20an
6559 7021,23,7021,7021, Functional status, FUNC STAT, 1*20an
6560 7022,24,7022,7022, Revision status, REV STAT, 1*20an
6561 7023,25,7023,7023, Global Individual Assset Identifier (GIAI) of an Assembly, GIAI-
6562 ASSEMBLY, 1*30an
6563 235,26,235,235, "Third party controlled, serialised extension of GTIN", TPX, 1*28n
6564 417,27,417,417, Global Location Number of Party, PGLN, 13n
6565 714,28,714,714, National Healthcare Reimbursement Number - Portugal (AIM), NHRH
6566 AIM, 1*an20
6567 7040,29,7040,7040, Unique Identification Code with Extensions (per EU 2018/574), UIC,
6568 1n 1*3an
6569 8013,30,8013,8013, Global Model Number, GMN, 1*an30
6570 8026,31,8026,8026, Identification of pieces of a trade item (ITIP) contained in a
6571 logistics unit, ITIP CONTENT, 18n
6572 8112,32,8112,8112, Paperless coupon code identification for use in North
6573 America,, 1*an70
6574 7240,33,7240,7240, Protocol ID, PROTOCOL, 1*20an
6575 395(**), 34,395%x30-35,395%x30-35, Amount Payable per unit of measure single
6576 monetary area (variable measure trade item), PRICE/UoM, 6n
6577 4300,35,4300,4300, Ship-to / Deliver-to company name, SHIP TO COMP, 1*35an
6578 4301,36,4301,4301, Ship-to / Deliver-to contact name, SHIP TO NAME, 1*35an
6579 4302,37,4302,4302, Ship-to / Deliver-to address line 1, SHIP TO ADD1, 1*70an
6580 4303,38,4303,4303, Ship-to / Deliver-to address line 2, SHIP TO ADD2, 1*70an
6581 4304,39,4304,4304, Ship-to / Deliver-to suburb, SHIP TO SUB, 1*70an
6582 4305,40,4305,4305, Ship-to / Deliver-to locality, SHIP TO LOC, 1*70an
6583 4306,41,4306,4306, Ship-to / Deliver-to region, SHIP TO REG, 1*70an
6584 4307,42,4307,4307, Ship-to / Deliver-to country code, SHIP TO COUNTRY, 2an
6585 4308,43,4308,4308, Ship-to / Deliver-to telephone number, SHIP TO PHONE, 1*30an
6586 4309,44,4309,4309, Ship-to / Deliver-to GEO location, SHIP TO GEO, 20n
6587 4310,45,4310,4310, Return-to company name, RTN TO COMP, 1*35an
6588 4311,46,4311,4311, Return-to contact name, RTN TO NAME, 1*35an
6589 4312,47,4312,4312, Return-to address line 1, RTN TO ADD1, 1*70an
6590 4313,48,4313,4313, Return-to address line 2, RTN TO ADD2, 1*70an
6591 4314,49,4314,4314, Return-to suburb, RTN TO SUB, 1*70an
6592 4315,50,4315,4315, Return-to locality, RTN TO LOC, 1*70an
6593 4316,51,4316,4316, Return-to region, RTN TO REG, 1*70an
6594 4317,52,4317,4317, Return-to country code, RTN TO COUNTRY, 2an



6595 4318,53,4318,4318,Return-to postal code,RTN TO POST,1*20an
6596 4319,54,4319,4319,Return-to telephone number,RTN TO PHONE,1*30an
6597 4320,55,4320,4320,Service code,SRV,1*35an
6598 4321,56,4321,4321,Dangerous goods flag,DANGEROUS GOODS,1n
6599 4322,57,4322,4322,Authority to leave flag,AUTH LEAV,1n
6600 4323,58,4323,4323,Signature required flag,SIG REQUIRED,1n
6601 4324,59,4324,4324,Not before delivery date/time,NBEF DEL DT,10n
6602 4325,60,4325,4325,Not after delivery date/time,NAFT DEL DT,10n
6603 4326,61,4326,4326,Release date,REL DATE,6n
6604 715,62,715,715,National Healthcare Reimbursement Number - United States of America
6605 (NDC),NHRN NDC,1*an20
6606 723s,63,7230,7230,Certification reference,CERT # s,2an 1*28an
6607 723s,64,7231,7231,Certification reference,CERT # s,2an 1*28an
6608 723s,65,7232,7232,Certification reference,CERT # s,2an 1*28an
6609 723s,66,7233,7233,Certification reference,CERT # s,2an 1*28an
6610 723s,67,7234,7234,Certification reference,CERT # s,2an 1*28an
6611 723s,68,7235,7235,Certification reference,CERT # s,2an 1*28an
6612 723s,69,7236,7236,Certification reference,CERT # s,2an 1*28an
6613 723s,70,7237,7237,Certification reference,CERT # s,2an 1*28an
6614 723s,71,7238,7238,Certification reference,CERT # s,2an 1*28an
6615 723s,72,7239,7239,Certification reference,CERT # s,2an 1*28an
6616 7241,73,7241,7241,AIDC Media Type,AIDC MEDIA TYPE,2an
6617 7242,74,7242,7242,Version Control Number (VCN),VCN,1*25an
6618 8030,75,7239,8030,Digital Signature (DigSig),DIGSIG,1*90an
6619 7011,76,7011,7011,Test by date,TEST BY DATE,6n 0*4n
6620 4330,77,4330,4330,Maximum temperature in Fahrenheit,MAX TEMP F,6n 0*1an
6621 4331,78,4331,4331,Maximum temperature in Celsius,MAX TEMP C,6n 0*1an
6622 4332,79,4332,4332,Minimum temperature in Farenheit,MIN TEMP F,6n 0*1an
6623 4333,80,4333,4333,Minimum temperature in Celsius,MIN TEMP C,6n 0*1an
6624 K-TableEnd = F9S12,,,,,,

6625
6626
6627
6628
6629
6630
6631
6632
6633
6634
6635
6636
6637
6638
6639

G 6-Bit Alphanumeric Character Set

The following table specifies the characters that are used in the Component / Part Reference in CPI EPCs and in the original part number and serial number in ADI EPCs. A subset of these characters are also used for the CAGE/DoDAAC code in ADI EPCs. The columns are as follows:

- **Graphic symbol:** The printed representation of the character as used in human-readable forms.
- **Name:** The common name for the character
- **Binary Value:** A Binary numeral that gives the 6-bit binary value for the character as used in EPC binary encodings. This binary value is always equal to the least significant six bits of the ISO/IEC 646 [ISO646] (ASCII) code for the character.
- **URI Form:** The representation of the character within Pure Identity EPC URI and EPC Tag URI forms. This is either a single character whose ASCII code's least significant six bits is equal to the value in the "binary value" column, or an escape triplet consisting of a percent character followed by two characters giving the hexadecimal value for the character.

Table I.3.1-1 Characters Permitted in 6-bit Alphanumeric Fields

| Graphic symbol | Name | Binary value | URI Form | Graphic symbol | Name | Binary value | URI Form |
|----------------|--------------------|--------------|----------|----------------|------------------|--------------|----------|
| # | Pound/ Number Sign | 100011 | %23 | H | Capital H | 001000 | H |
| - | Hyphen/ Minus Sign | 101101 | - | I | Capital I | 001001 | I |
| / | Forward Slash | 101111 | %2F | J | Capital J | 001010 | J |
| 0 | Zero Digit | 110000 | 0 | K | Capital K | 001011 | K |
| 1 | One Digit | 110001 | 1 | L | Capital L | 001100 | L |
| 2 | Two Digit | 110010 | 2 | M | Capital M | 001101 | M |
| 3 | Three Digit | 110011 | 3 | N | Capital N | 001110 | N |
| 4 | Four Digit | 110100 | 4 | O | Capital O | 001111 | O |
| 5 | Five Digit | 110101 | 5 | P | Capital P | 010000 | P |
| 6 | Six Digit | 110110 | 6 | Q | Capital Q | 010001 | Q |
| 7 | Seven Digit | 110111 | 7 | R | Capital R | 010010 | R |
| 8 | Eight Digit | 111000 | 8 | S | Capital S | 010011 | S |
| 9 | Nine Digit | 111001 | 9 | T | Capital T | 010100 | T |
| A | Capital A | 000001 | A | U | Capital U | 010101 | U |
| B | Capital B | 000010 | B | V | Capital V | 010110 | V |
| C | Capital C | 000011 | C | W | Capital W | 010111 | W |
| D | Capital D | 000100 | D | X | Capital X | 011000 | X |
| E | Capital E | 000101 | E | Y | Capital Y | 011001 | Y |
| F | Capital F | 000110 | F | Z | Capital Letter Z | 011010 | Z |
| G | Capital G | 000111 | G | | | | |

6640
6641
6642

H (Intentionally Omitted)

[This annex is omitted so that Annexes I through M, which specify Packed Objects, have the same annex letters as the corresponding annexes of ISO/IEC 15962, 2nd Edition.]

6643 I Packed Objects structure

6644 I.1 Overview

6645 The Packed Objects format provides for efficient encoding and access of user data. The Packed
 6646 Objects format offers increased encoding efficiency compared to the No-Directory and Directory
 6647 Access-Methods partly by utilising sophisticated compaction methods, partly by defining an inherent
 6648 directory structure at the front of each Packed Object (before any of its data is encoded) that
 6649 supports random access while reducing the fixed overhead of some prior methods, and partly by
 6650 utilising data-system-specific information (such as the GS1 definitions of fixed-length Application
 6651 Identifiers).

6652 I.2 Overview of Packed Objects documentation

6653 The formal description of Packed Objects is presented in this Annex and Annexes J, K, L, and M, as
 6654 follows:

- 6655 ■ The overall structure of Packed Objects is described in Section [I.3](#).
- 6656 ■ The individual sections of a Packed Object are described in Sections [I.4](#) through [I.9](#).
- 6657 ■ The structure and features of ID Tables (utilised by Packed Objects to represent various data
 6658 system identifiers) are described in Annex [J](#).
- 6659 ■ The numerical bases and character sets used in Packed Objects are described in Annex [K](#).
- 6660 ■ An encoding algorithm and worked example are described in Annex [L](#).
- 6661 ■ The decoding algorithm for Packed Objects is described in Annex [M](#).

6662 In addition, note that all descriptions of specific ID Tables for use with Packed Objects are registered
 6663 separately, under the procedures of ISO/IEC 15961-2 as is the complete formal description of the
 6664 machine-readable format for registered ID Tables.

6665 I.3 High-Level Packed Objects format design

6666 I.3.1 Overview

6667 The Packed Objects memory format consists of a sequence in memory of one or more "Packed
 6668 Objects" data structures. Each Packed Object may contain either encoded data or directory
 6669 information, but not both. The first Packed Object in memory is preceded by a DSFID. The DSFID
 6670 indicates use of Packed Objects as the memory's Access Method, and indicates the registered Data
 6671 Format that is the default format for every Packed Object in that memory. Every Packed Object may
 6672 be optionally preceded or followed by padding patterns (if needed for alignment on word or block
 6673 boundaries). In addition, at most one Packed Object in memory may optionally be preceded by a
 6674 pointer to a Directory Packed Object (this pointer may itself be optionally followed by padding). This
 6675 series of Packed Objects is terminated by optional padding followed by one or more zero-valued
 6676 octets aligned on byte boundaries. See [Figure I.3.1-1](#), which shows this sequence when appearing in
 6677 an RFID tag.

6678 ✔ **Note:** Because the data structures within an encoded Packed Object are bit-aligned rather
 6679 than byte-aligned, this Annex uses the term 'octet' instead of 'byte' except in case where an
 6680 eight-bit quantity must be aligned on a byte boundary.

6681 **Figure I.3.1-1** Overall Memory structure when using Packed Objects

| | | | | | | | | |
|-------|---|------------------------|---|-------------------------------------|-----|------------------------------|---|------------------|
| DSFID | Optional Pointer* And/Or Padding | First Packed Object | Optional Pointer* And/Or Padding | Optional Second Packed Object | ... | Optional Packed Object | Optional Pointer* And/Or Padding | Zero Octet(s) |
|-------|---|------------------------|---|-------------------------------------|-----|------------------------------|---|------------------|

6682 *Note: the Optional Pointer to a Directory Packed Object may appear at most only once in memory

6683
6684
6685
6686
6687

Every Packed Object represents a sequence of one or more data system Identifiers, each specified by reference to an entry within a Base ID Table from a registered data format. The entry is referenced by its relative position within the Base Table; this relative position or Base Table index is referred to throughout this specification as an "ID Value." There are two different Packed Objects methods available for representing a sequence of Identifiers by reference to their ID Values:

6688
6689

- An ID List Packed Object (IDLPO) encodes a series of ID Values as a list, whose length depends on the number of data items being represented;

6690
6691
6692
6693

- An ID Map Packed Object (IDMPO) instead encodes a fixed-length bit array, whose length depends on the total number of entries defined in the registered Base Table. Each bit in the array is '1' if the corresponding table entry is represented by the Packed Object, and is '0' otherwise.

6694
6695
6696
6697
6698
6699
6700
6701
6702

An ID List is the default Packed Objects format, because it uses fewer bits than an ID Map, if the list contains only a small percentage of the data system's defined ID Values. However, if the Packed Object includes more than about one-quarter of the defined entries, then an ID Map requires fewer bits. For example, if a data system has sixteen entries, then each ID Value (table index) is a four bit quantity, and a list of four ID Values takes as many bits as would the complete ID Map. An ID Map's fixed-length characteristic makes it especially suitable for use in a Directory Packed Object, which lists all of the Identifiers in all of the Packed Objects in memory (see Section 1.9). The overall structure of a Packed Object is the same, whether an IDLPO or an IDMPO, as shown in Figure I 3-2 and as described in the next subsection.

6703

Figure I.3.1-2 Packed object structure

| | | | | |
|-----------------------------|---|--|--------------------------------------|-----------------------------|
| Optional Format Flags | Object Info Section (IDLPO or IDMPO) | Secondary ID Section (if needed) | Aux Format Section (if needed) | Data Section (if needed) |
|-----------------------------|---|--|--------------------------------------|-----------------------------|

6704
6705
6706
6707
6708

Packed objects may be made "editable", by adding an optional Addendum subsection to the end of the Object Info section, which includes a pointer to an "Addendum Packed Object" where additions and/or deletions have been made. One or more such "chains" of editable "parent" and "child" Packed Objects may be present within the overall sequence of Packed Objects in memory, but no more than one chain of Directory Packed Objects may be present.

6709 **I.3.2 Descriptions of each section of a Packed Object's structure**

6710
6711
6712
6713
6714

Each Packed Object consists of several bit-aligned sections (that is, no pad bits between sections are used), carried in a variable number of octets. All required and optional Packed Objects formats are encompassed by the following ordered list of Packed Objects sections. Following this list, each Packed Objects section is introduced, and later sections of this Annex describe each Packed Objects section in detail.

6715
6716
6717
6718

- **Format Flags:** A Packed Object may optionally begin with the pattern '0000' which is reserved to introduce one or more Format Flags, as described in 1.4.2. These flags may indicate use of the non-default ID Map format. If the Format Flags are not present, then the Packed Object defaults to the ID List format.

6719
6720
6721

- Certain flag patterns indicate an inter-Object pattern (Directory Pointer or Padding)
- Other flag patterns indicate the Packed Object's type (Map or. List), and may indicated the presence of an optional Addendum subsection for editing.

6722
6723

- **Object Info:** All Packed Objects contain an Object Info Section which includes Object Length Information and ID Value Information:

6724
6725
6726

- Object Length Information includes an ObjectLength field (indicating the overall length of the Packed Object in octets) followed by Pad Indicator bit, so that the number of significant bits in the Packed Object can be determined.

6727
6728
6729

- ID Value Information indicates which Identifiers are present and in what order, and (if an IDLPO) also includes a leading NumberOfIDs field, indicating how many ID Values are encoded in the ID List.

6730
6731

The Object Info section is encoded in one of the following formats, as shown in [Figure I.3.2-1](#) and [Figure I.3.2-2](#).

6732
6733
6734
6735
6736
6737

- ID List (IDLPO) Object Info format:
 - Object Length (EBV-6) plus Pad Indicator bit
 - A single ID List or an ID Lists Section (depending on Format Flags)
- ID Map (IDMPO) Object Info format:
 - One or more ID Map sections
 - Object Length (EBV-6) plus Pad Indicator bit

For either of these Object Info formats, an Optional Addendum subsection may be present at the end of the Object Info section.

6740
6741
6742
6743
6744
6745
6746
6747
6748
6749
6750
6751
6752
6753
6754

- **Secondary ID Bits:** A Packed Object may include a Secondary ID section, if needed to encode additional bits that are defined for some classes of IDs (these bits complete the definition of the ID).
- **Aux Format Bits:** A Data Packed Object may include an Aux Format Section, which if present encodes one or more bits that are defined to support data compression, but do not contribute to defining the ID.
- **Data Section:** A Data Packed Object includes a Data Section, representing the compressed data associated with each of the identifiers listed within the Packed Object. This section is omitted in a Directory Packed Object, and in a Packed Object that uses No-directory compaction (see [I.7.1](#)). Depending on the declaration of data format in the relevant ID table, the Data section will contain either or both of two subsections:
 - **Known-Length Numerics subsection:** this subsection compacts and concatenates all of the non-empty data strings that are known a priori to be numeric.
 - **AlphaNumeric subsection:** this subsection concatenates and compacts all of the non-empty data strings that are not a priori known to be all-numeric.

Figure I.3.2-1 IDLPO Object Info Structure

| Object Info, in a Default ID List PO | | | | or | Object Info, in a Non-default ID List PO | | |
|--------------------------------------|---------------|---------|-------------------|----|--|--------------------------------------|-------------------|
| Object Length | Number Of IDs | ID List | Optional Addendum | | Object Length | ID Lists Section (one or more lists) | Optional Addendum |

Figure I.3.2-2 IDMPO Object Info Structure

| Object Info, in an ID Map PO | | |
|-----------------------------------|---------------|-------------------|
| ID Map Section (one or more maps) | Object Length | Optional Addendum |

6757
6758
6759
6760
6761
6762
6763
6764
6765
6766

I.4 Format Flags section

The default layout of memory, under the Packed Objects access method, consists of a leading DSFID, immediately followed by an ID List Packed Object (at the next byte boundary), then optionally additional ID List Packed Objects (each beginning at the next byte boundary), and terminated by a zero-valued octet at the next byte boundary (indicating that no additional Packed Objects are encoded). This section defines the valid Format Flags patterns that may appear at the expected start of a Packed Object to override the default layout if desired (for example, by changing the Packed Object's format, or by inserting padding patterns to align the next Packed Object on a word or block boundary). The set of defined patterns are shown below.

Table I.3.2-1 Format Flag

| Bit Pattern | Description | Additional Info | See Section |
|-------------|-------------------------------|---------------------------------|-----------------------|
| 0000 0000 | Termination Pattern | No more Packed Objects follow | I.4.1 |
| LLLLLL xx | First octet of an IDLPO | For any LLLLLL > 3 | I.5 |
| 0000 | Format Flags starting pattern | (if the full EBV-6 is non-zero) | I.4.2 |

| Bit Pattern | Description | Additional Info | See Section |
|-------------------------------------|---|--|-----------------------|
| 0000 10NA | IDLPO with: N = 1: non-default Info A = 1: Addendum Present | If N = 1: allows multiple ID tables If A = 1: Addendum ptr(s) at end of Object Info section | I.4.3 |
| 0000 01xx | Inter-PO pattern | A Directory Pointer, or padding | I.4.4 |
| 0000 0100 | Signifies a padding octet | No padding length indicator follows | I.4.4 |
| 0000 0101 | Signifies run-length padding | An EBV-8 padding length follows | I.4.4 |
| 0000 0110 | RFU | | I.4.4 |
| 0000 0111 | Directory pointer | Followed by EBV-8 pattern | I.4.4 |
| 0000 11xx | ID Map Packed Object | | I.4.2 |
| 0000 0001 0000 0010 0000 0011 | [Invalid] | Invalid pattern | |

6767 **I.4.1 Data terminating flag pattern**

6768 A pattern of eight or more '0' bits at the expected start of a Packed Object denotes that no more
6769 Packed Objects are present in the remainder of memory.


6770 NOTE: Six successive '0' bits at the expect start of a Packed Object would (if interpreted as a Packed
6771 Object) indicate an ID List Packed Object of length zero.

6772 **I.4.2 Format flag section starting bit patterns**

6773 A non-zero EBV-6 with a leading pattern of "0000" is used as a Format Flags section Indication
6774 Pattern. The additional bits following an initial '0000' format Flag Indicating Pattern are defined as
6775 follows:

- 6776 ■ A following two-bit pattern of '10' (creating an initial pattern of '000010') indicates an IDLPO
6777 with at least one non-default optional feature (see [I.4.3](#))
- 6778 ■ A following two-bit pattern of '11' indicates an IDMPPO, which is a Packed Object using an ID Map
6779 format instead of ID List-format The ID Map section (see [I.9](#)) immediately follows this two-bit
6780 pattern.
- 6781 ■ A following two-bit pattern of '01' signifies an External pattern (Padding pattern or Pointer) prior
6782 to the start of the next Packed Object (see [I.4.4](#))

6783 A leading EBV-6 Object Length of less than four is invalid as a Packed Objects length.

6784  **Note:** The shortest possible Packed Object is an IDLPO, for a data system using four bits per
6785 ID Value, encoding a single ID Value. This Packed Object has a total of 14 fixed bits.
6786 Therefore, a two-octet Packed Object would only contain two data bits, and is invalid. A three-
6787 octet Packed Object would be able to encode a single data item up to three digits long. In
6788 order to preserve "3" as an invalid length in this scenario, the Packed Objects encoder shall
6789 encode a leading Format Flags section (with all options set to zero, if desired) in order to
6790 increase the object length to four.

6791 **I.4.3 IDLPO Format Flags**

6792 The appearance of '000010' at the expected start of a Packed Object is followed by two additional
6793 bits, to form a complete IDLPO Format Flags section of "000010NA", where:

- 6794 ■ If the first additional bit 'N' is '1', then a non-default format is employed for the IDLPO Object
6795 Info section. Whereas the default IDLPO format allows for only a single ID List (utilising the
6796 registration's default Base ID Table), the optional non-default IDLPO Object Info format

6797 supports a sequence of one or more ID Lists, and each such list begins with identifying
6798 information as to which registered table it represents (see [I.5.1](#)).

6799 ■ If the second additional bit 'A' is '1', then an Addendum subsection is present at the end of the
6800 Object Info section (see [I.5.6](#)).

6801 **I.4.4 Patterns for use between Packed Objects**

6802 The appearance of '000001' at the expected start of a Packed Object is used to indicate either
6803 padding or a directory pointer, as follows:

- 6804 ■ A following two-bit pattern of '11' indicates that a Directory Packed Object Pointer follows the
6805 pattern. The pointer is one or more octets in length, in EBV-8 format. This pointer may be Null
6806 (a value of zero), but if non-zero, indicates the number of octets from the start of the pointer to
6807 the start of a Directory Packed Object (which if editable, shall be the first in its "chain"). For
6808 example, if the Format Flags byte for a Directory Pointer is encoded at byte offset 1, the Pointer
6809 itself occupies bytes beginning at offset 2, and the Directory starts at byte offset 9, then the Dir
6810 Ptr encodes the value "7" in EBV-8 format. A Directory Packed Object Pointer may appear before
6811 the first Packed Object in memory, or at any other position where a Packed Object may begin,
6812 but may only appear once in a given data carrier memory, and (if non-null) must be at a lower
6813 address than the Directory it points to. The first octet after this pointer may be padding (as
6814 defined immediately below), a new set of Format Flag patterns, or the start of an ID List Packed
6815 Object.
- 6816 ■ A following two-bit pattern of '00' indicates that the full eight-bit pattern of '00000100' serves
6817 as a padding byte, so that the next Packed Object may begin on a desired word or block
6818 boundary. This pattern may repeat as necessary to achieve the desired alignment.
- 6819 ■ A following two-bit pattern of '01' as a run-length padding indicator, and shall be immediately
6820 followed by an EBV-8 indicating the number of octets from the start of the EBV-8 itself to the
6821 start of the next Packed Object (for example, if the next Packed Object follows immediately, the
6822 EBV-8 has a value of one). This mechanism eliminates the need to write many words of memory
6823 in order to pad out a large memory block.
- 6824 ■ A following two-bit pattern of '10' is Reserved.

6825 **I.5 Object Info section**

6826 Each Packed Object's Object Info section contains both Length Information (the size of the Packed
6827 Object, in bits and in octets), and ID Values Information. A Packed Object encodes representations
6828 of one or more data system Identifiers and (if a Data Packed Object) also encodes their associated
6829 data elements (AI strings, DI strings, etc). The ID Values information encodes a complete listing of
6830 all the Identifiers (AIs, DIs, etc) encoded in the Packed Object, or (in a Directory Packed Object) all
6831 the Identifiers encoded anywhere in memory.

6832 To conserve encoded and transmitted bits, data system Identifiers (each typically represented in
6833 data systems by either two, three, or four ASCII characters) is represented within a Packed Object
6834 by an ID Value, representing an index denoting an entry in a registered Base Table of ID Values. A
6835 single ID Value may represent a single Object Identifier, or may represent a commonly-used
6836 sequence of Object Identifiers. In some cases, the ID Value represents a "class" of related Object
6837 Identifiers, or an Object Identifier sequence in which one or more Object Identifiers are optionally
6838 encoded; in these cases, Secondary ID Bits (see [I.6](#)) are encoded in order to specify which selection
6839 or option was chosen when the Packed Object was encoded. A "fully-qualified ID Value" (FQIDV) is
6840 an ID Value, plus a particular choice of associated Secondary ID bits (if any are invoked by the ID
6841 Value's table entry). Only one instance of a particular fully-qualified ID Value may appear in a data
6842 carrier's Data Packed Objects, but a particular ID Value may appear more than once, if each time it
6843 is "qualified" by different Secondary ID Bits. If an ID Value does appear more than once, all
6844 occurrences shall be in a single Packed Object (or within a single "chain" of a Packed Object plus its
6845 Addenda).

6846 There are two methods defined for encoding ID Values: an ID List Packed Object uses a variable-
6847 length list of ID Value bit fields, whereas an ID Map Packed Object uses a fixed-length bit array.
6848 Unless a Packed Object's format is modified by an initial Format Flags pattern, the Packed Object's
6849 format defaults to that of an ID List Packed Object (IDLPO), containing a single ID List, whose ID

6850 Values correspond to the default Base ID Table of the registered Data Format. Optional Format Flags
 6851 can change the format of the ID Section to either an IDMPO format, or to an IDLPO format encoding
 6852 an ID Lists section (which supports multiple ID Tables, including non-default data systems).

6853 Although the ordering of information within the Object Info section varies with the chosen format
 6854 (see [I.5.1](#)), the Object Info section of every Packed Object shall provide Length information as
 6855 defined in [I.5.2](#), and ID Values information (see [I.5.3](#)) as defined in [I.5.4](#), or [I.5.5](#). The Object Info
 6856 section (of either an IDLPO or an IDMPO) may conclude with an optional Addendum subsection (see
 6857 [I.5.6](#)).

6858 **I.5.1 Object Info formats**

6859 **IDLPO default Object Info format**

6860 The default IDLPO Object Info format is used for a Packed Object either without a leading Format
 6861 Flags section, or with a Format Flags section indicating an IDLPO with a possible Addendum and a
 6862 default Object Info section. The default IDLPO Object Info section contains a single ID List
 6863 (optionally followed by an Addendum subsection if so indicated by the Format Flags). The format of
 6864 the default IDLPO Object Info section is shown in the table below.

6865 **Table I.5.1-1** Default IDLPO Object Info format

| Field Name: | Length Information | NumberOfIDs | ID Listing | Addendum subsection |
|-------------|--|--|--|--|
| Usage: | The number of octets in this Object, plus a last-octet pad indicator | number of ID Values in this Object (minus one) | A single list of ID Values; value size depends on registered Data Format | Optional pointer(s) to other Objects containing Edit information |
| Structure: | Variable: see I.5.2 | Variable:EBV-3 | See I.5.4 | See I.5.6 |

6866 In a IDLPO’s Object Info section, the NumberOfIDs field is an EBV-3 Extensible Bit Vector, consisting
 6867 of one or more repetitions of an Extension Bit followed by 2 value bits. This EBV-3 encodes one less
 6868 than the number of ID Values on the associated ID Listing. For example, an EBV-3 of ‘101 000’
 6869 indicates $(4 + 0 + 1) = 5$ IDs values. The Length Information is as described in [I.5.2](#) for all Packed
 6870 Objects. The next fields are an ID Listing (see [I.5.4](#)) and an optional Addendum subsection (see
 6871 [I.5.6](#)).

6872 **IDLPO non-default Object Info format**

6873 Leading Format Flags may modify the Object Info structure of an IDLPO, so that it may contain
 6874 more than one ID Listing, in an ID Lists section (which also allows non-default ID tables to be
 6875 employed). The non-default IDLPO Object Info structure is shown in the table below.

6876

Table I.5.1-2 Non-Default IDLPO Object Info format

| Field Name: | Length Info | ID Lists Section, first List | | | Optional Additional ID List(s) | Null App Indicator (single zero bit) | Addendum Subsection |
|-------------|--|--|---|---|--|--------------------------------------|--|
| | | Application Indicator | Number of IDs | ID Listing | | | |
| Usage: | The number of octets in this Object, plus a last-octet pad indicator | Indicates the selected ID Table and the size of each entry | Number Of ID Values on the list (minus one) | Listing of ID Values, then one F/R Use bit | Zero or more repeated lists, each for a different ID Table | | Optional pointer(s) to other Objects containing Edit information |
| Structure: | see I.5.2 | see I.5.3 | See I.5.1 | See I.5.4 and I.5.3 | References in previous columns | See I.5.3 | See I.5.6 |

6877

IDMPO Object Info format

6878

Leading Format Flags may define the Object Info structure to be an IDMPO, in which the Length Information (and optional Addendum subsection) follow an ID Map section (see [I.5.5](#)). This arrangement ensures that the ID Map is in a fixed location for a given application, of benefit when used as a Directory. The IDMPO Object Info structure is shown in the table below.

6879

6880

6881

6882

Table I.5.1-3 IDMPO Object Info format

| Field Name: | ID Map section | Length Information | Addendum |
|-------------|--|--|--|
| Usage: | One or more ID Map structures, each using a different ID Table | The number of octets in this Object, plus a last-octet pad indicator | Optional pointer(s) to other Objects containing Edit information |
| Structure: | see I.5.3 | See I.5.2 | See I.5.6 |

6883

I.5.2 Length Information

6884

The format of the Length information, always present in the Object Info section of any Packed Object, is shown in the table below.

6885

6886

Table I.5.2-1 Packed Object Length information

| Field Name: | ObjectLength | Pad Indicator |
|-------------|--|--|
| Usage: | The number of 8-bit bytes in this Object This includes the 1st byte of this Packed Object, including its IDLPO/IDMPO format flags if present. It excludes patterns for use between Packed Objects, as specified in I.4.4 | If '1': the Object's last byte contains at least 1 pad |
| Structure: | Variable: EBV-6 | Fixed: 1 bit |

6887

The first field, ObjectLength, is an EBV-6 Extensible Bit Vector, consisting of one or more repetitions of an Extension Bit and 5 value bits. An EBV-6 of '000100' (value of 4) indicates a four-byte Packed Object, An EBV-6 of '100001 000000' (value of 32) indicates a 32-byte Object, and so on.

6888

6889

6890

The Pad Indicator bit immediately follows the end of the EBV-6 ObjectLength. This bit is set to '0' if there are no padding bits in the last byte of the Packed Object. If set to '1', then bitwise padding begins with the least-significant or rightmost '1' bit of the last byte, and the padding consists of this rightmost '1' bit, plus any '0' bits to the right of that bit. This method effectively uses a *single* bit to indicate a *three*-bit quantity (i.e., the number of trailing pad bits). When a receiving system wants to determine the total number of bits (rather than bytes) in a Packed Object, it would examine the ObjectLength field of the Packed Object (to determine the number of bytes) and multiply the result by eight, and (if the Pad Indicator bit is set) examine the last byte of the Packed Object and decrement the bit count by (1 plus the number of '0' bits following the rightmost '1' bit of that final byte).

6891

6892

6893

6894

6895

6896

6897

6898

6899

6900 **I.5.3 General description of ID values**

6901 A registered data format defines (at a minimum) a Primary Base ID Table (a detailed specification
 6902 for registered ID tables may be found in Annex J). This base table defines the data system
 6903 Identifier(s) represented by each row of the table, any Secondary ID Bits or Aux Format bits
 6904 invoked by each table entry, and various implicit rules (taken from a predefined rule set) that
 6905 decoding systems shall use when interpreting data encoded according to each entry. When a data
 6906 item is encoded in a Packed Object, its associated table entry is identified by the entry's relative
 6907 position in the Base Table. This table position or index is the ID Value that is represented in Packed
 6908 Objects.

6909 A Base Table containing a given number of entries inherently specifies the number of bits needed to
 6910 encode a table index (i.e., an ID Value) in an ID List Packed Object (as the Log (base 2) of the
 6911 number of entries). Since current and future data system ID Tables will vary in unpredictable ways
 6912 in terms of their numbers of table entries, there is a need to pre-define an ID Value Size mechanism
 6913 that allows for future extensibility to accommodate new tables, while minimising decoder complexity
 6914 and minimising the need to upgrade decoding software (other than the addition of new tables).
 6915 Therefore, regardless of the exact number of Base Table entries defined, each Base Table definition
 6916 shall utilise one of the predefined sizes for ID Value encodings defined in Table I 5-5 (any unused
 6917 entries shall be labelled as reserved, as provided in Annex J). The ID Size Bit pattern is encoded in a
 6918 Packed Object only when it uses a non-default Base ID Table. Some entries in the table indicate a
 6919 size that is not an integral power of two. When encoding (into an IDLPO) ID Values from tables that
 6920 utilise such sizes, each pair of ID Values is encoded by multiplying the earlier ID of the pair by the
 6921 base specified in the fourth column of Table I-5-5 and adding the later ID of the pair, and encoding
 6922 the result in the number of bits specified in the fourth column. If there is a trailing single ID Value
 6923 for this ID Table, it is encoded in the number of bits specified in the third column of the table below.

6924 **Table I.5.3-1** Defined ID Value sizes

| ID Size Bit pattern | Maximum number of Table Entries | Number of Bits per single or trailing ID Value, and how encoded | Number of Bits per pair of ID Values, and how encoded |
|---------------------|---------------------------------|---|---|
| 000 | Up to 16 | 4, as 1 Base 16 value | 8, as 2 Base 16 values |
| 001 | Up to 22 | 5, as 1 Base 22 value | 9, as 2 Base 22 values |
| 010 | Up to 32 | 5, as 1 Base 32 value | 10, as 2 Base 32 values |
| 011 | Up to 45 | 6, as 1 Base 45 value | 11, as 2 Base 45 values |
| 100 | Up to 64 | 6, as 1 Base 64 value | 12, as 2 Base 64 values |
| 101 | Up to 90 | 7, as 1 Base 90 value | 13, as 2 Base 90 values |
| 110 | Up to 128 | 7, as 1 Base 128 value | 14, as 2 Base 128 values |
| 1110 | Up to 256 | 8, as 1 Base 256 value | 16, as 2 Base 256 values |
| 111100 | Up to 512 | 9, as 1 Base 512 value | 18, as 2 Base 512 values |
| 111101 | Up to 1024 | 10, as 1 Base 1024 value | 20, as 2 Base 1024 values |
| 111110 | Up to 2048 | 11, as 1 Base 2048 value | 22, as 2 Base 2048 values |
| 111111 | Up to 4096 | 12, as 1 Base 4096 value | 24, as 2 Base 4096 values |

6925 **Application indicator subsection**

6926 An Application Indicator subsection can be utilised to indicate use of ID Values from a default or
 6927 non-default ID Table. This subsection is required in every IDMPO, but is only required in an IDLPO
 6928 that uses the non-default format supporting multiple ID Lists.

6929 An Application Indicator consists of the following components:

- 6930 ■ A single AppIndicatorPresent bit, which if '0' means that no additional ID List or Map follows.
6931 Note that this bit is always omitted for the first List or Map in an Object Info section. When this
6932 bit is present and '0', then none of the following bit fields are encoded.
- 6933 ■ A single ExternalReg bit that, if '1', indicates use of an ID Table from a registration other than
6934 the memory's default. If '1', this bit is immediately followed by a 9-bit representation of a Data
6935 Format registered under ISO/IEC 15961.
- 6936 ■ An ID Size pattern which denotes a table size (and therefore an ID Map bit length, when used in
6937 an IDMPO), which shall be one of the patterns defined by [Table I.5.2-1](#). The table size indicated
6938 in this field must be less than or equal to the table size indicated in the selected ID table. The
6939 purpose of this field is so that the decoder can parse past the ID List or ID Map, even if the ID
6940 Table is not available to the decoder.
- 6941 ■ A three-bit ID Subset pattern. The registered data format's Primary Base ID Table, if used by
6942 the current Packed Object, shall always be indicated by an encoded ID Subset pattern of '000'.
6943 However, up to seven Alternate Base Tables may also be defined in the registration (with
6944 varying ID Sizes), and a choice from among these can be indicated by the encoded Subset
6945 pattern. This feature can be useful to define smaller sector-specific or application-specific
6946 subsets of a full data system, thus substantially reducing the size of the encoded ID Map.

6947 **Full/Restricted Use bits**

6948 When contemplating the use of new ID Table registrations, or registrations for external data
6949 systems, application designers may utilise a "restricted use" encoding option that adds some
6950 overhead to a Packed Object but in exchange results in a format that can be fully decoded by
6951 receiving systems not in possession of the new or external ID table. With the exception of a IDLPO
6952 using the default Object Info format, one Full/Restricted Use bit is encoded immediately after each
6953 ID table is represented in the ID Map section or ID Lists section of a Data or Directory Packed
6954 Object. In a Directory Packed Object, this bit shall always be set to '0' and its value ignored. If an
6955 encoder wishes to utilise the "restricted use" option in an IDLPO, it shall preface the IDLPO with a
6956 Format Flags section invoking the non-default Object Info format.

6957 If a "Full/Restricted Use" bit is '0' then the encoding of data strings from the corresponding
6958 registered ID Table makes full use of the ID table's IDstring and FormatString information. If the bit
6959 is '1', then this signifies that some encoding overhead was added to the Secondary ID section and
6960 (in the case of Packed-Object compaction) the Aux Format section, so that a decoder without access
6961 to the table can nonetheless output OIDs and data from the Packed Object according to the scheme
6962 specified in [J.4.1](#). Specifically, a Full/Restricted Use bit set to '1' indicates that:

- 6963 ■ for each encoded ID Value, the encoder added an EBV-3 indicator to the Secondary ID section,
6964 to indicate how many Secondary ID bits were invoked by that ID Value. If the EBV-3 is nonzero,
6965 then the Secondary ID bits (as indicated by the table entry) immediately follow, followed in turn
6966 by another EBV-3, until the entire list of ID Values has been represented.
- 6967 ■ the encoder did not take advantage of the information from the referenced table's FormatString
6968 column. Instead, corresponding to each ID Value, the encoder inserted an EBV-3 into the Aux
6969 Format section, indicating the number of discrete data string lengths invoked by the ID Value
6970 (which could be more than one due to combinations and/or optional components), followed by
6971 the indicated number of string lengths, each length encoded as though there were no
6972 FormatString in the ID table. All data items were encoded in the A/N subsection of the Data
6973 section.

6974 **I.5.4 ID Values representation in an ID Value-list Packed Object**

6975 Each ID Value is represented within an IDLPO on a list of bit fields; the number of bit fields on the
6976 list is determined from the NumberOfIDs field (see Section [1.5.6](#)). Each ID Value bit field's length is
6977 in the range of four to eleven bits, depending on the size of the Base Table index it represents. In
6978 the optional non-default format for an IDLPO's Object Info section, a single Packed Object may
6979 contain multiple ID List subsections, each referencing a different ID Table. In this non-default
6980 format, each ID List subsection consists of an Application Indicator subsection (which terminates the
6981 ID Lists, if it begins with a '0' bit), followed by an EBV-3 NumberOfIDs, an ID List, and a
6982 Full/Restricted Use flag.

6983 **I.5.5 ID Values representation in an ID Map Packed Object**

6984 Encoding an ID Map can be more efficient than encoding a list of ID Values, when representing a
 6985 relatively large number of ID Values (constituting more than about 10 percent of a large Base
 6986 Table's entries, or about 25 percent of a small Base Table's entries). When encoded in an ID Map,
 6987 each ID Value is represented by its relative position within the map (for example, the first ID Map
 6988 bit represents ID Value "0", the third bit represents ID Value "2", and the last bit represents ID
 6989 Value 'n' (corresponding to the last entry of a Base Table with (n+1) entries). The value of each bit
 6990 within an ID Map indicates whether the corresponding ID Value is present (if the bit is '1') or absent
 6991 (if '0'). An ID Map is always encoded as part of an ID Map Section structure (see [I.9.1](#)).

6992 **I.5.6 Optional Addendum subsection of the Object Info section**

6993 The Packed Object Addendum feature supports basic editing operations, specifically the ability to
 6994 add, delete, or replace individual data items in a previously-written Packed Object, without a need
 6995 to rewrite the entire Packed Object. A Packed Object that does not contain an Addendum subsection
 6996 cannot be edited in this fashion, and must be completely rewritten if changes are required.

6997 An Addendum subsection consists of a Reverse Links bit, followed by a Child bit, followed by either
 6998 one or two EBV-6 links. Links from a Data Packed Object shall only go to other Data Packed Objects
 6999 as addenda; links from a Directory Packed Object shall only go to other Directory Packed Objects as
 7000 addenda. The standard Packed Object structure rules apply, with some restrictions that are
 7001 described in [I.5.6](#).

7002 The Reverse Links bit shall be set identically in every Packed Object of the same "chain." The
 7003 Reverse Links bit is defined as follows:

- 7004 ■ If the Reverse Links bit is '0', then each child in this chain of Packed Objects is at a higher
 7005 memory location than its parent. The link to a Child is encoded as the number of octets (plus
 7006 one) that are in between the last octet of the current Packed Object and the first octet of the
 7007 Child. The link to the parent is encoded as the number of octets (plus one) that are in between
 7008 the first octet of the parent Packed Object and the first octet of the current Packed Object.
- 7009 ■ If the Reverse Links bit is '1', then each child in this chain of Packed Objects is at a lower
 7010 memory location than its parent. The link to a Child is encoded as the number of octets (plus
 7011 one) that are in between the first octet of the current Packed Object and the first octet of the
 7012 Child. The link to the parent is encoded as the number of octets (plus one) that are in between
 7013 the last octet of the current Packed Object and the first octet of the parent.

7014 The Child bit is defined as follows:


- 7015 ■ If the Child bit is a '0', then this Packed Object is an editable "Parentless" Packed Object (i.e.,
 7016 the first of a chain), and in this case the Child bit is immediately followed by a single EBV-6 link
 7017 to the first "child" Packed Object that contains editing addenda for the parent.
- 7018 ■ If the Child bit is a '1', then this Packed Object is an editable "child" of an edited "parent," and
 7019 the bit is immediately followed by one EBV-6 link to the "parent" and a second EBV-6 line to the
 7020 next "child" Packed Object that contains editing addenda for the parent.

7021 A link value of zero is a Null pointer (no child exists), and in a Packed Object whose Child bit is '0',
 7022 this indicates that the Packed Object is editable, but has not yet been edited. A link to the Parent is
 7023 provided, so that a Directory may indicate the presence and location of an ID Value in an Addendum
 7024 Packed Object, while still providing an interrogator with the ability to efficiently locate the other ID
 7025 Values that are logically associated with the original "parent" Packed Object. A link value of zero is
 7026 invalid as a pointer towards a Parent.

7027 In order to allow room for a sufficiently-large link, when the future location of the next "child" is
 7028 unknown at the time the parent is encoded, it is permissible to use the "redundant" form of the
 7029 EBV-6 (for example using "100000 000000" to represent a link value of zero).

7030 **Addendum "EditingOP" list (only in ID List Packed Objects)**

7031 In an IDLPO only, each Addendum section of a "child" ID List Packed Object contains a set of
 7032 "EditingOp" bits encoded immediately after its last EBV-6 link. The number of such bits is

- 7033 determined from the number of entries on the Addendum Packed Object's ID list. For each ID Value
7034 on this list, the corresponding EditingOp bit or bits are defined as follows:
- 7035 ■ '1' means that the corresponding Fully-Qualified ID Value (FQIDV) is Replaced. A Replace
7036 operation has the effect that the data originally associated with the FQIDV matching the FQIDV
7037 in this Addendum Packed Object shall be ignored, and logically replaced by the Aux Format bits
7038 and data encoded in this Addendum Packed Object)
 - 7039 ■ '00' means that the corresponding FQIDV is Deleted but not replaced. In this case, neither the
7040 Aux Format bits nor the data associated with this ID Value are encoded in the Addendum Packed
7041 Object.
 - 7042 ■ '01' means that the corresponding FQIDV is Added (either this FQIDV was not previously
7043 encoded, or it was previously deleted without replacement). In this case, the associated Aux
7044 Format Bits and data shall be encoded in the Addendum Packed Object.
- 7045  **Note:** If an application requests several "edit" operations at once (including some Delete or
7046 Replace operations as well as Adds) then implementations can achieve more efficient
7047 encoding if the Adds share the Addendum overhead, rather than being implemented in a new
7048 Packed Object.

7049 **Packed Objects containing an addendum subsection**

7050 A Packed Object containing an Addendum subsection is otherwise identical in structure to other
7051 Packed Objects. However, the following observations apply:

- 7052 ■ A "parentless" Packed Object (the first in a chain) may be either an ID List Packed Object or an
7053 ID Map Packed Object (and a parentless IDMPO may be either a Data or Directory IDMPO).
7054 When a "parentless" PO is a directory, only directory IDMPOs may be used as addenda. A
7055 Directory IDMPO's Map bits shall be updated to correctly reflect the end state of the chain of
7056 additions and deletions to the memory bank; an Addendum to the Directory is not utilised to
7057 perform this maintenance (a Directory Addendum may only add new structural components, as
7058 described later in this section). In contrast, when the edited parentless object is an ID List
7059 Packed Object or ID Map Packed Object, its ID List or ID Map cannot be updated to reflect the
7060 end state of the aggregate Object (parents plus children).
- 7061 ■ Although a "child" may be either an ID List or an ID Map Packed Object, only an IDLPO can
7062 indicate deletions or changes to the current set of fully-qualified ID Values and associated data
7063 that is embodied in the chain.
 - 7064 □ When a child is an IDMPO, it shall only be utilised to add (not delete or modify) structural
7065 information, and shall not be used to modify existing information. In a Directory chain, a
7066 child IDMPO may add new ID tables, or may add a new AuxMap section or subsections, or
7067 may extend an existing PO Index Table or ObjectOffsets list. In a Data chain, an IDMPO
7068 shall not be used as an Addendum, except to add new ID Tables.
 - 7069 □ When a child is an IDLPO, its ID list (followed by "EditingOp" bits) lists only those FQIDVs
7070 that have been deleted, added, or replaced, relative to the cumulative ID list from the prior
7071 Objects linked to it.

7072 **I.6 Secondary ID Bits section**

7073 The Packed Objects design requirements include a requirement that all of the data system
7074 Identifiers (AI's, DI's, etc.) encoded in a Packed Object's can be fully recognised without expanding
7075 the compressed data, even though some ID Values provide only a partially-qualified Identifier. As a
7076 result, if any of the ID Values invoke Secondary ID bits, the Object Info section shall be followed by
7077 a Secondary ID Bits section. Examples include a four-bit field to identify the third digit of a group of
7078 related Logistics AIs.

7079 Secondary ID bits can be invoked for several reasons, as needed in order to fully specify Identifiers.
7080 For example, a single ID Table entry's ID Value may specify a choice between two similar identifiers
7081 (requiring one encoded bit to select one of the two IDs at the time of encoding), or may specify a
7082 combination of required and optional identifiers (requiring one encoded bit to enable or disable each
7083 option). The available mechanisms are described in Annex J. All resulting Secondary ID bit fields are

7084
7085
7086
7087

concatenated in this Secondary ID Bits section, in the same order as the ID Values that invoked them were listed within the Packed Object. Note that the Secondary ID Bits section is identically defined, whether the Packed Object is an IDLPO or an IDMPO, but is not present in a Directory IDMPO.

7088

I.7 Aux Format section

7089
7090
7091

The Aux Format section of a Data Packed Object encodes auxiliary information for the decoding process. A Directory Packed Object does not contain an Aux Format section. In a Data Packed Object, the Aux Format section begins with "Compact-Parameter" bits as defined in the table below.

7092

Table I.5.6-1 Compact-Parameter bit patterns

| Bit Pattern | Compaction method used in this Packed Object | Reference |
|-------------|---|---------------------------|
| '1' | "Packed-Object" compaction | See I.7.2 |
| '000' | "Application-Defined", as defined for the No-Directory access method | See I.7.1 |
| '001' | "Compact", as defined for the No-Directory access method | See I.7.1 |
| '010' | "UTF-8", as defined for the No-Directory access method | See I.7.1 |
| '011bbbb' | ('bbbb' shall be in the range of 4..14): reserved for future definition | See I.7.1 |

7093
7094
7095

If the Compact-Parameter bit pattern is '1', then the remainder of the Aux Format section is encoded as described in [I.7.2](#); otherwise, the remainder of the Aux Format section is encoded. See [I.7.1](#) as described in [I.7.1](#).

7096

I.7.1 Support for No-Directory compaction methods

7097
7098
7099
7100

If any of the No-Directory compaction methods were selected by the Compact-Parameter bits, then the Compact-Parameter bits are followed by a byte-alignment padding pattern consisting of zero or more '0' bits followed by a single '1' bit, so that the next bit after the '1' is aligned as the most-significant bit of the next byte.

7101
7102
7103
7104
7105

This next byte is defined as the first octet of a "No-Directory Data section", which is used in place of the Data section described in I.8. The data strings of this Packed Object are encoded in the order indicated by the Object Info section of the Packed Object, compacted exactly as described in Annex D of [ISO15962] (Encoding rules for No-Directory Access-Method), with the following two exceptions:

7106
7107

- The Object-Identifier is not encoded in the "No-Directory Data section", because it has already been encoded into the Object Info and Secondary ID sections.

7108
7109

- The Precursor is modified in that only the three Compaction Type Code bits are significant, and the other bits in the Precursor are set to '0'.

7110
7111

Therefore, each of the data strings invoked by the ID Table entry are separately encoded in a modified data set structure as:

7112

<modified precursor> <length of compacted object> <compacted object octets>

7113
7114
7115

The <compacted object octets> are determined and encoded as described in D.1.1 and D.1.2 of [ISO15962] and the <length of compacted object> is determined and encoded as described in D.2 of [ISO15962].

7116
7117

Following the last data set, a terminating precursor value of zero shall not be encoded (the decoding system recognises the end of the data using the encoded ObjectLength of the Packed Object).

7118

I.7.2 Support for the packed-object compaction method

7119
7120
7121
7122

If the Packed-Object compaction method was selected by the Compact-Parameter bits, then the Compact-Parameter bits are followed by zero or more Aux Format bits, as may be invoked by the ID Table entries used in this Packed Object. The Aux Format bits are then immediately followed by a Data section that uses the Packed-Object compaction method described in I.8.

7123 An ID Table entry that was designed for use with the Packed-Object compaction method can call for
 7124 various types of auxiliary information beyond the complete indication of the ID itself (such as bit
 7125 fields to indicate a variable data length, to aid the data compaction process). All such bit fields are
 7126 concatenated in this portion, in the order called for by the ID List or Map. Note that the Aux Format
 7127 section is identically defined, whether the Packed Object is an IDLPO or an IDMPO.

7128 An ID Table entry invokes Aux Format length bits for all entries that are not specified as fixed-length
 7129 in the table (however, these length bits are not actually encoded if they correspond to the last data
 7130 item encoded in the A/N subsection of a Packed Object). This information allows the decoding
 7131 system to parse the decoded data into strings of the appropriate lengths. An encoded Aux Format
 7132 length entry utilises a variable number of bits, determined from the specified range between the
 7133 shortest and longest data strings allowed for the data item, as follows:

- 7134 ■ If a maximum length is specified, and the specified range (defined as the maximum length
 7135 minus the minimum length) is less than eight, or greater than 44, then lengths in this range are
 7136 encoded in the fewest number of bits that can express lengths within that range, and an
 7137 encoded value of zero represents the minimum length specified in the format string. For
 7138 example, if the range is specified as from three to six characters, then lengths are encoded
 7139 using two bits, and '00' represents a length of three.
- 7140 ■ Otherwise (including the case of an unspecified maximum length), the value (actual length –
 7141 specified minimum) is encoded in a variable number of bits, as follows:
- 7142 ■ Values from 0 to 14 (representing lengths from 1 to 15, if the specified minimum length is one
 7143 character, for example) are encoded in four bits
- 7144 ■ Values from 15 to 29 are encoded in eight bits (a prefix of '1111' followed by four bits
 7145 representing values from 15 ('0000') to 29 ('1110'))
- 7146 ■ Values from 30 to 44 are encoded in twelve bits (a prefix of '1111 1111' followed by four bits
 7147 representing values from 30 ('0000') to 44 ('1110'))
- 7148 ■ Values greater than 44 are encoded as a twelve-bit prefix of all '1's, followed by an EBV-6
 7149 indication of (value – 44).

7150 **Notes:**

- 7151 ■ if a range is specified with identical upper and lower bounds (i.e., a range of zero), this is
 7152 treated as a fixed length, not a variable length, and no Aux Format bits are invoked.
- 7153 ■ If a range is unspecified, or has unspecified upper or lower bounds, then this is treated as a
 7154 default lower bound of one, and/or an unlimited upper bound.

7155 **I.8 Data section**

7156 A Data section is always present in a Packed Object, except in the case of a Directory Packed Object
 7157 or Directory Addendum Packed Object (which encode no data elements), the case of a Data
 7158 Addendum Packed Object containing only Delete operations, and the case of a Packed Object that
 7159 uses No-directory compaction (see [I.7.1](#)). When a Data section is present, it follows the Object Info
 7160 section (and the Secondary ID and Aux Format sections, if present). Depending on the
 7161 characteristics of the encoded IDs and data strings, the Data section may include one or both of two
 7162 subsections in the following order: a Known-Length Numerics subsection, and an AlphaNumerics
 7163 subsection. The following paragraphs provide detailed descriptions of each of these Data Section
 7164 subsections. If all of the subsections of the Data section are utilised in a Packed Object, then the
 7165 layout of the Data section is as shown in the table below.

7166 **Table I.7.2-1** Maximum Structure of a Packed Objects Data section

| Known-Length Numeric subsection | | | | AlphaNumeric subsection | | | | | | | |
|----------------------------------|----------------------------------|-----|-----------------------|---------------------------|------------------------------------|------------------------------------|-------------|-------------------------|--------------------------------|----------------------|-------------------|
| | | | | A/N Header Bits | | | | Binary Data Segments | | | |
| 1 st KLN Binary | 2 nd KLN Binary | ... | Last KLN Binary | Non-Num Base Bit(s) | Prefix Bit, Prefix Run(s) | Suffix Bit, Suffix Run(s) | Char Map | Ext'd. Num Binary | Ext'd Non- Num Binary | Base 10 Binary | Non-Num Binary |
| | | | | | | | | | | | |

7167 **I.8.1 Known-length-Numerics subsection of the data section**

7168 For always-numeric data strings, the ID table may indicate a fixed number of digits (this fixed-
 7169 length information is not encoded in the Packed Object) and/or a variable number of digits (in which
 7170 case the string's length was encoded in the Aux Format section, as described above). When a single
 7171 data item is specified in the FormatString column (see [J.2.3](#)) as containing a fixed-length numeric
 7172 string followed by a variable-length string, the numeric string is encoded in the Known-length-
 7173 numerics subsection and the alphanumeric string in the Alphanumeric subsection.

7174 The summation of fixed-length information (derived directly from the ID table) plus variable-length
 7175 information (derived from encoded bits as just described) results in a "known-length entry" for each
 7176 of the always-numeric strings encoded in the current Packed Object. Each all-numeric data string in
 7177 a Packed Object (if described as all-numeric in the ID Table) is encoded by converting the digit
 7178 string into a single Binary number (up to 160 bits, representing a binary value between 0 and $(10^{48}-$
 7179 $1)$). Figure K-1 in Annex [K](#) shows the number of bits required to represent a given number of digits.
 7180 If an all-numeric string contains more than 48 digits, then the first 48 are encoded as one 160-bit
 7181 group, followed by the next group of up to 48 digits, and so on. Finally, the Binary values for each
 7182 all-numeric data string in the Object are themselves concatenated to form the Known-length-
 7183 Numerics subsection.

7184 **I.8.2 Alphanumeric subsection of the data section**

7185 The Alphanumeric (A/N) subsection, if present, encodes all of the Packed Object's data from any
 7186 data strings that were not already encoded in the Known-length Numerics subsection. If there are
 7187 no alphanumeric characters to encode, the entire A/N subsection is omitted. The Alphanumeric
 7188 subsection can encode any mix of digits and non-digit ASCII characters, or eight-bit data. The digit
 7189 characters within this data are encoded separately, at an average efficiency of 4.322 bits per digit or
 7190 better, depending on the character sequence. The non-digit characters are independently encoded
 7191 at an average efficiency that varies between 5.91 bits per character or better (all uppercase letters),
 7192 to a worst-case limit of 9 bits per character (if the character mix requires Base 256 encoding of non-
 7193 numeric characters).

7194 An Alphanumeric subsection consists of a series of A/N Header bits (see I.8.2.1), followed by from
 7195 one to four Binary segments (each segment representing data encoded in a single numerical Base,
 7196 such as Base 10 or Base 30, see I.8.2.4), padded if necessary to complete the final byte (see I
 7197 8.2.5).

7198 **A/N Header Bits**

7199 The A/N Header Bits are defined as follows:

- 7200 ■ One or two Non-Numeric Base bits, as follows:
 - 7201 □ '0' indicates that Base 30 was chosen for the non-numeric Base;
 - 7202 □ '10' indicates that Base 74 was chosen for the non-numeric Base;
 - 7203 □ '11' indicates that Base 256 was chosen for the non-numeric Base
- 7204 ■ Either a single '0' bit (indicating that no Character Map Prefix is encoded), or a '1' bit followed
 7205 by one or more "Runs" of six Prefix bits as defined in I.8.2.3.
- 7206 ■ Either a single '0' bit (indicating that no Character Map Suffix is encoded), or a '1' bit followed
 7207 by one or more "Runs" of six Suffix bits as defined in I.8.2.3.
- 7208 ■ A variable-length "Character Map" bit pattern (see I.8.2.2), representing the base of each of the
 7209 data characters, if any, that were not accounted for by a Prefix or Suffix.

7210 **Dual-base Character-map encoding**

7211 Compaction of the ordered list of alphanumeric data strings (excluding those data strings already
 7212 encoded in the Known-Length Numerics subsection) is achieved by first concatenating the data
 7213 characters into a single data string (the individual string lengths have already been recorded in the
 7214 Aux Format section). Each of the data characters is classified as either Base 10 (for numeric digits),
 7215 Base 30 non-numeric (primarily uppercase A-Z), Base 74 non-numeric (which includes both
 7216 uppercase and lowercase alphas, and other ASCII characters), or Base 256 characters. These
 7217 character sets are fully defined in Annex K. All characters from the Base 74 set are also accessible

7218 from Base 30 via the use of an extra "shift" value (as are most of the lower 128 characters in the
7219 Base 256 set). Depending on the relative percentage of "native" Base 30 values vs. other values in
7220 the data string, one of those bases is selected as the more efficient choice for a non-numeric base.

7221 Next, the precise sequence of numeric and non-numeric characters is recorded and encoded, using
7222 a variable-length bit pattern, called a "character map," where each '0' represents a Base 10 value
7223 (encoding a digit) and each '1' represents a value for a non-numeric character (in the selected
7224 base). Note that, (for example) if Base 30 encoding was selected, each data character (other than
7225 uppercase letters and the space character) needs to be represented by a pair of base-30 values, and
7226 thus each such data character is represented by a *pair* of '1' bits in the character map.

7227 **Prefix and Suffix Run-Length encoding**

7228 For improved efficiency in cases where the concatenated sequence includes runs of six or more
7229 values from the same base, provision is made for optional run-length representations of one or
7230 more Prefix or Suffix "Runs" (single-base character sequences), which can replace the first and/or
7231 last portions of the character map. The encoder shall not create a Run that separates a Shift value
7232 from its next (shifted) value, and thus a Run always represents an integral number of source
7233 characters.

7234 An optional Prefix Representation, if present, consists of one or more occurrences of a Prefix Run.
7235 Each Prefix Run consists of one Run Position bit, followed by two Basis Bits, then followed by three
7236 Run Length bits, defined as follows:

- 7237 ■ The Run Position bit, if '0', indicates that at least one more Prefix Run is encoded following this
7238 one (representing another set of source characters to the right of the current set). The Run
7239 Position bit, if '1', indicates that the current Prefix Run is the last (rightmost) Prefix Run of the
7240 A/N subsection.
- 7241 ■ The first basis bit indicates a choice of numeric vs. non-numeric base, and the second basis bit,
7242 if '1', indicates that the chosen base is extended to include characters from the "opposite" base.
7243 Thus, '00' indicates a run-length-encoded sequence of base 10 values; '01' indicates a sequence
7244 that is primarily (but not entirely) digits, encoded in Base 13; '10' indicates a sequence a
7245 sequence of values from the non-numeric base that was selected earlier in the A/N header, and
7246 '11' indicates a sequence of values primarily from that non-numeric base, but extended to
7247 include digit characters as well. Note an exception: if the non-numeric base that was selected in
7248 the A/N header is Base 256, then the "extended" version is defined to be Base 40.
- 7249 ■ The 3-bit Run Length value assumes a minimum useable run of six same-base characters, and
7250 the length value is further divided by 2. Thus, the possible 3-bit Run Length values of 0, 1, 2, ...
7251 7 indicate a Run of 6, 8, 10, ... 20 characters from the same base. Note that a trailing "odd"
7252 character value at the end of a same-base sequence must be represented by adding a bit to the
7253 Character Map.

7254 An optional Suffix Representation, if present, is a series of one or more Suffix Runs, each identical in
7255 format to the Prefix Run just described. Consistent with that description, note that the Run Position
7256 bit, if '1', indicates that the current Suffix Run is the last (rightmost) Suffix Run of the A/N
7257 subsection, and thus any preceding Suffix Runs represented source characters to the left of this final
7258 Suffix Run.

7259 **Encoding into Binary Segments**

7260 Immediately after the last bit of the Character Map, up to four binary numbers are encoded, each
7261 representing all of the characters that were encoded in a single base system. First, a base-13 bit
7262 sequence is encoded (if one or more Prefix or Suffix Runs called for base-13 encoding). If present,
7263 this bit sequence directly represents the binary number resulting from encoding the combined
7264 sequence of all Prefix and Suffix characters (in that order) classified as Base 13 (ignoring any
7265 intervening characters not thus classified) as a single value, or in other words, applying a base 13 to
7266 Binary conversion. The number of bits to encode in this sequence is directly determined from the
7267 number of base-13 values being represented, as called for by the sum of the Prefix and Suffix Run
7268 lengths for base 13 sequences. The number of bits, for a given number of Base 13 values, is
7269 determined from the Figure in Annex K. Next, an Extended-NonNumeric Base segment (either Base-
7270 40 or Base 84) is similarly encoded (if any Prefix or Suffix Runs called for Extended-NonNumeric
7271 encoding).

7272 Next, a Base-10 Binary segment is encoded that directly represents the binary number resulting
 7273 from encoding the sequence of the digits in the Prefix and/or character map and/or Suffix (ignoring
 7274 any intervening non-digit characters) as a single value, or in other words, applying a base 10 to
 7275 Binary conversion. The number of bits to encode in this sequence is directly determined from the
 7276 number of digits being represented, as shown in Annex [K](#).

7277 Immediately after the last bit of the Base-10 bit sequence (if any), a non-numeric (Base 30, Base
 7278 74, or Base 256) bit sequence is encoded (if the character map indicates at least one non-numeric
 7279 character). This bit sequence represents the binary number resulting from a base-30 to Binary
 7280 conversion (or a Base-74 to Binary conversion, or a direct transfer of Base-256 values) of the
 7281 sequence of non-digit characters in the data (ignoring any intervening digits). Again, the number of
 7282 encoded bits is directly determined from the number of non-numeric values being represented, as
 7283 shown in Annex [K](#). Note that if Base 256 was selected as the non-Numeric base, then the encoder is
 7284 free to classify and encode each digit either as Base 10 or as Base 256 (Base 10 will be more
 7285 efficient, unless outweighed by the ability to take advantage of a long Prefix or Suffix).

7286 Note that an Alphanumeric subsection ends with several variable-length bit fields (the character
 7287 map, and one or more Binary sections (representing the numeric and non-numeric Binary values).
 7288 Note further that none of the lengths of these three variable-length bit fields are explicitly encoded
 7289 (although one or two Extended-Base Binary segments may also be present, these have known
 7290 lengths, determined from Prefix and/or Suffix runs). In order to determine the boundaries between
 7291 these three variable-length fields, the decoder needs to implement a procedure, using knowledge of
 7292 the remaining number of daIa bits, in order to correctly parse the Alphanumeric subsection. An
 7293 example of such a procedure is described in Annex [M](#).

7294 **Padding the last Byte**

7295 The last (least-significant) bit of the final Binary segment is also the last significant bit of the Packed
 7296 Object. If there are any remaining bit positions in the last byte to be filled with pad bits, then the
 7297 most significant pad bit shall be set to '1', and any remaining less-significant pad bits shall be set to
 7298 '0'. The decoder can determine the total number of non-pad bits in a Packed Object by examining
 7299 the Length Section of the Packed Object (and if the Pad Indicator bit of that section is '1', by also
 7300 examining the last byte of the Packed Object).

7301 **I.9 ID Map and Directory encoding options**

7302 An ID Map can be more efficient than a list of ID Values, when encoding a relatively large number of
 7303 ID Values. Additionally, an ID Map representation is advantageous for use in a Directory Packed
 7304 Object. The ID Map itself (the first major subsection of every ID Map section) is structured
 7305 identically whether in a Data or Directory IDMPO, but a Directory IDMPO's ID Map section contains
 7306 additional optional subsections. The structure of an ID Map section, containing one or more ID
 7307 Maps, is described in the section below, explained in terms of its usage in a Data IDMPO;
 7308 subsequent sections explain the added structural elements in a Directory IDMPO.

7309 **I.9.1 ID Map Section structure**

7310 An IDMPO represents ID Values using a structure called an ID Map section, containing one or more
 7311 ID Maps. Each ID Value encoded in a Data IDMPO is represented as a '1' bit within an ID Map bit
 7312 field, whose fixed length is equal to the number of entries in the corresponding Base Table.
 7313 Conversely, each '0' in the ID Map Field indicates the absence of the corresponding ID Value. Since
 7314 the total number of '1' bits within the ID Map Field equals the number of ID Values being
 7315 represented, no explicit NumberOfIDs field is encoded. In order to implement the range of
 7316 functionality made possible by this representation, the ID Map Section contains elements other than
 7317 the ID Map itself. If present, the optional ID Map Section immediately follows the leading pattern
 7318 indicating an IDMPO (as was described in [I.4.2](#)), and contains the following elements in the order
 7319 listed below:

- 7320 ■ An Application Indicator subsection (see [I.5.3](#))
- 7321 ■ an ID Map bit field (whose length is determined from the ID Size in the Application Indicator)
- 7322 ■ a Full/Restricted Use bit (see [I.5.3](#))
- 7323 ■ (the above sequence forms an ID Map, which may optionally repeat multiple times)

- 7324 ■ a Data/Directory indicator bit,
- 7325 ■ an optional AuxMap section (never present in a Data IDMPO), and
- 7326 ■ Closing Flag(s), consisting of an "Addendum Flag" bit. If '1', then an Addendum subsection is
- 7327 present at the end of the Object Info section (after the Object Length Information).

7328 These elements, shown in the table below as a maximum structure (every element is present), are

7329 described in each of the next subsections.

7330 **Table I.9.1-1** ID Map section

| First ID Map | | Optional additional ID Map(s) | | Null App Indicator (single zero bit) | Data/Directory Indicator Bit | (If directory) Optional AuxMap Section | Closing Flag Bit(s) |
|---------------------------|---|-------------------------------|----------------------------------|--------------------------------------|------------------------------|--|---------------------|
| App Indicator | ID Map Bit Field (ends with F/R bit) | App Indicator | ID Map Field (ends with F/R bit) | | | | |
| See I.5.3 | See I.9.1 and I.5.3 | As previous | As previous | See I.5.3 | | See I.9.2 | Addendum Flag Bit |

7331 When an ID Map section is encoded, it is always followed by an Object Length and Pad Indicator,

7332 and optionally followed by an Addendum subsection (all as have been previously defined), and then

7333 may be followed by any of the other sections defined for Packed Objects, except that a Directory

7334 IDMPO shall not include a Data section.

7335 **ID Map and ID Map bit field**

7336 An ID Map usually consists of an Application Indicator followed by an ID Map bit field, ending with a

7337 Full/Restricted Use bit. An ID Map bit field consists of a single "MapPresent" flag bit, then (if

7338 MapPresent is '1') a number of bits equal to the length determined from the ID Size pattern within

7339 the Application Indicator, plus one (the Full/Restricted Use bit). The ID Map bit field indicates the

7340 presence/absence of encoded data items corresponding to entries in a specific registered Primary or

7341 Alternate Base Table. The choice of base table is indicated by the encoded combination of DSFID

7342 and Application Indicator pattern that precedes the ID Map bit field. The MSB of the ID Map bit field

7343 corresponds to ID Value 0 in the base table, the next bit corresponds to ID Value 1, and so on.

7344 In a Data Packed Object's ID Map bit field, each '1' bit indicates that this Packed Object contains an

7345 encoded occurrence of the data item corresponding to an entry in the registered Base Table

7346 associated with this ID Map. Note that the valid encoded entry may be found either in the first

7347 ("parentless") Packed Object of the chain (the one containing the ID Map) or in an Addendum IDLPO

7348 of that chain. Note further that one or more data entries may be encoded in an IDMPO, but marked

7349 "invalid" (by a Delete entry in an Addendum IDLPO).

7350 An ID Map shall not correspond to a Secondary ID Table instead of a Base ID Table. Note that data

7351 items encoded in a "parentless" Data IDMPO shall appear in the same relative order in which they

7352 are listed in the associated Base Table. However, additional "out of order" data items may be added

7353 to an existing data IDMPO by appending an Addendum IDLPO to the Object.

7354 An ID Map cannot indicate a specific number of instances (greater than one) of the same ID Value,

7355 and this would seemingly imply that only one data instance using a given ID Value can be encoded

7356 in a Data IDMPO. However, the ID Map method needs to support the case where more two or more

7357 encoded data items are from the same identifier "class" (and thus share the same ID Value). The

7358 following mechanisms address this need:

- 7359 ■ Another data item of the same class can be encoded in an Addendum IDLPO of the IDMPO.
- 7360 Multiple occurrences of the same ID Value can appear on an ID List, each associated with
- 7361 different encoded values of the Secondary ID bits.
- 7362 ■ A series of two or more encoded instances of the same "class" can be efficiently indicated by a
- 7363 single instance of an ID Value (or equivalently by a single ID Map bit), if the corresponding Base
- 7364 Table entry defines a "Repeat" Bit (see [1.2.2](#)).

7365 An ID Map section may contain multiple ID Maps; a null Application Indicator section (with its
7366 AppIndicatorPresent bit set to '0') terminates the list of ID Maps.

7367 **Data/Directory and AuxMap indicator bits**

7368 A Data/Directory indicator bit is always encoded immediately following the last ID Map. By
7369 definition, a Data IDMPO has its Data/Directory bit set to '0', and a Directory IDMPO has its
7370 Data/Directory bit set to '1'. If the Data/Directory bit is set to '1', it is immediately followed by an
7371 AuxMap indicator bit which, if '1', indicates that an optional AuxMap section immediately follows.

7372 Closing Flags bit(s)

7373 The ID Map section ends with a single Closing Flag:

- 7374 ■ The final bit of the Closing Flags is an Addendum Flag Bit which, if '1', indicates that there is an
7375 optional Addendum subsection encoded at the end of the Object Info section of the Packed
7376 Object. If present, the Addendum subsection is as described in Section [I.5.6](#).

7377 **I.9.2 Directory Packed Objects**

7378 A "Directory Packed Object" is an IDMPO whose Directory bit is set to '1'. Its only inherent
7379 difference from a Data IDMPO is that it does not contain any encoded data items. However,
7380 additional mechanisms and usage considerations apply only to a Directory Packed Object, and these
7381 are described in the following subsections.

7382 **ID Maps in a Directory IDMPO**

7383 Although the structure of an ID Map is identical whether in a Data or Directory IDMPO, the
7384 semantics of the structure are somewhat different. In a Directory Packed Object's ID Map bit field,
7385 each '1' bit indicates that a Data Packed Object in the same data carrier memory bank contains a
7386 valid data item associated with the corresponding entry in the specified Base Table for this ID Map.
7387 Optionally, a Directory Packed Object may further indicate *which* Packed Object contains each data
7388 item (see the description of the optional AuxMap section below).

7389 Note that, in contrast to a Data IDMPO, there is no required correlation between the order of bits in
7390 a Directory's ID Map and the order in which these data items are subsequently encoded in memory
7391 within a sequence of Data Packed Objects.

7392 **Optional AuxMap Section (Directory IDMPOs only)**

7393 An AuxMap Section optionally allows a Directory IDMPO's ID Map to indicate not only
7394 presence/absence of all the data items in this memory bank of the tag, but also which Packed
7395 Object encodes each data item. If the AuxMap indicator bit is '1', then an AuxMap section shall be
7396 encoded immediately after this bit. If encoded, the AuxMap section shall contain one PO Index Field
7397 for each of the ID Maps that precede this section. After the last PO Index Field, the AuxMap Section
7398 may optionally encode an ObjectOffsets list, where each ObjectOffset generally indicates the
7399 number of bytes from the start of the previous Packed Object to the start of the next Packed Object.
7400 This AuxMap structure is shown (for an example IDMPO with two ID Maps) in the table below.

7401 **Table I.9.2-1** Optional AuxMap section structure

| PO Index Field for first ID Map | | PO Index Field for second ID Map | | Object Offsets Present bit | Optional ObjectOffsets subsection | | | | |
|---------------------------------|---------------|----------------------------------|---------------|----------------------------|-----------------------------------|-----------------------|-----------------------|-----|-----------------------|
| POindex Length | POindex Table | POindex Length | POindex Table | | Object Offsets Multiplier | Object1 offset (EBV6) | Object2 offset (EBV6) | ... | ObjectN offset (EBV6) |

7402 Each PO Index Field has the following structure and semantics:

- 7403 ■ A three-bit POindexLength field, indicating the number of index bits encoded for each entry in
7404 the PO Index Table that immediately follows this field (unless the POindex length is '000', which
7405 means that no PO Index Table follows).
- 7406 ■ A PO Index Table, consisting of an array of bits, one bit (or group of bits, depending on the
7407 POindexLength) for every bit in the corresponding ID Map of this directory Packed Object. A PO

- 7408 Index Table entry (i.e., a "PO Index") indicates (by relative order) which Packed Object contains
 7409 the data item indicated by the corresponding '1' bit in the ID Map. If an ID Map bit is '0', the
 7410 corresponding PO Index Table entry is present but its contents are ignored.
- 7411 ■ Every Packed Object is assigned an index value in sequence, without regard as to whether it is a
 7412 "parentless" Packed Object or a "child" of another Packed Object, or whether it is a Data or
 7413 Directory Packed Object.
 - 7414 ■ If the PO Index is within the first PO Index Table (for the associated ID Map) of the Directory
 7415 "chain", then:
 - 7416 □ a PO Index of zero refers to the first Packed Object in memory,
 - 7417 □ a value of one refers to the next Packed Object in memory, and so on
 - 7418 □ a value of m , where m is the largest value that can be encoded in the PO Index (given the
 7419 number of bits per index that was set in the POindexLength), indicates a Packed Object
 7420 whose relative index (position in memory) is m or higher. This definition allows Packed
 7421 Objects higher than m to be indexed in an Addendum Directory Packed Object, as described
 7422 immediately below. If no Addendum exists, then the precise position is either m or some
 7423 indeterminate position greater than m .
 - 7424 ■ If the PO Index is not within the first PO Index Table of the directory chain for the associated ID
 7425 Map (i.e., it is in an Addendum IDMPO), then:
 - 7426 □ a PO Index of zero indicates that a prior PO Index Table of the chain provided the index
 7427 information,
 - 7428 □ a PO Index of n ($n > 0$) refers to the n th Packed Object above the highest index value
 7429 available in the immediate parent directory PO; e.g., if the maximum index value in the
 7430 immediate parent directory PO refers to PO number "3 or greater," then a PO index of 1 in
 7431 this addendum refers to PO number 4.
 - 7432 □ A PO Index of m (as defined above) similarly indicates a Packed Object whose position is the
 7433 m th position, or higher, than the limit of the previous table in the chain.
 - 7434 ■ If the valid instance of an ID Value is in an Addendum Packed Object, an implementation may
 7435 choose to set a PO Index to point directly to that Addendum, or may instead continue to point to
 7436 the Packed Object in the chain that originally contained the ID Value.
 7437 NOTE: The first approach sometimes leads to faster searching; the second sometimes leads to
 7438 faster directory updates.
- 7439 After the last PO Index Field, the AuxMap section ends with (at minimum) a single "ObjectOffsets
 7440 Present" bit. A '0' value of this bit indicates that no ObjectOffsets subsection is encoded. If instead
 7441 this bit is a '1', it is immediately followed by an ObjectOffsets subsection, which holds a list of EBV-6
 7442 "offsets" (the number of octets between the start of a Packed Object and the start of the next
 7443 Packed Object). If present, the ObjectOffsets subsection consists of an ObjectOffsetsMultiplier
 7444 followed by an Object Offsets list, defined as follows:
- 7445 ■ An EBV-6 ObjectOffsetsMultiplier, whose value, when multiplied by 6, sets the total number of
 7446 bits reserved for the entire ObjectOffsets list. The value of this multiplier should be selected to
 7447 ideally result in sufficient storage to hold the offsets for the maximum number of Packed Objects
 7448 that can be indexed by this Directory Packed Object's PO Index Table (given the value in the
 7449 POindexLength field, and given some estimated average size for those Packed Objects).
 - 7450 ■ a fixed-sized field containing a list of EBV-6 ObjectOffsets. The size of this field is exactly the
 7451 number of bits as calculated from the ObjectOffsetsMultiplier. The first ObjectOffset represents
 7452 the start of the second Packed Object in memory, relative to the first octet of memory (there
 7453 would be little benefit in reserving extra space to store the offset of the first Packed Object).
 7454 Each succeeding ObjectOffset indicates the start of the next Packed Object (relative to the
 7455 previous ObjectOffset on the list), and the final ObjectOffset on the list points to the all-zero
 7456 termination pattern where the next Packed Object may be written. An invalid offset of zero
 7457 (EBV-6 pattern "000000") shall be used to terminate the ObjectOffset list. If the reserved
 7458 storage space is fully occupied, it need not include this terminating pattern.
 - 7459 ■ In applications where the average Packed Object Length is difficult to predict, the reserved
 7460 ObjectOffset storage space may sometimes prove to be insufficient. In this case, an Addendum
 7461 Packed Object can be appended to the Directory Packed Object. This Addendum Directory

7462 Packed Object may contain null subsections for all but its ObjectOffsets subsection. Alternately,
 7463 if it is anticipated that the capacity of the PO Index Table will also eventually be exceeded, then
 7464 the Addendum Packed Object may also contain one or more non-null PO Index fields. Note that
 7465 in a given instance of an AuxMap section, either a PO Index Table or an ObjectOffsets
 7466 subsection may be the first to exceed its capacity. Therefore, the first position referenced by an
 7467 ObjectOffsets list in an Addendum Packed Object need not coincide with the first position
 7468 referenced by the PO Index Table of that same Addendum. Specifically, in an Addendum Packed
 7469 Object, the first ObjectOffset listed is an offset referenced to the last ObjectOffset on the list of
 7470 the "parent" Directory Packed Object.

7471 **Usage as a Presence/Absence Directory**

7472 In many applications, an Interrogator may choose to read the entire contents of any data carrier
 7473 containing one or more "target" data items of interest. In such applications, the positional
 7474 information of those data items within the memory is not needed during the initial reading
 7475 operations; only a presence/absence indication is needed at this processing stage. An ID Map can
 7476 form a particularly efficient Presence/Absence directory for denoting the contents of a data carrier in
 7477 such applications. A full directory structure encodes the offset or address (memory location) of
 7478 every data element within the data carrier, which requires the writing of a large number of bits
 7479 (typically 32 bits or more per data item). Inevitably, such an approach also requires reading a large
 7480 number of bits over the air, just to determine whether an identifier of interest is present on a
 7481 particular tag. In contrast, when only presence/absence information is needed, using an ID Map
 7482 conveys the same information using only one bit per data item defined in the data system. The
 7483 entire ID Map can be typically represented in 128 bits or less, and stays the same size as more data
 7484 items are written to the tag.

7485 A "Presence/Absence Directory" Packed Object is defined as a Directory IDMPO that does not
 7486 contain a PO Index, and therefore provides no encoded information as to where individual data
 7487 items reside within the data carrier. A Presence/Absence Directory can be converted to an "Indexed
 7488 Directory" Packed Object (see I.9.2.4) by adding a PO Index in an Addendum Packed Object, as a
 7489 "child" of the Presence/Absence Packed Object.

7490 **Usage as an Indexed Directory**

7491 In many applications involving large memories, an Interrogator may choose to read a Directory
 7492 section covering the entire memory's contents, and then issue subsequent Reads to fetch the
 7493 "target" data items of interest. In such applications, the positional information of those data items
 7494 within the memory is important, but if many data items are added to a large memory over time, the
 7495 directory itself can grow to an undesirable size.

7496 An ID Map, used in conjunction with an AuxMap containing a PO Index, can form a particularly-
 7497 efficient "Indexed Directory" for denoting the contents of an RFID tag, and their approximate
 7498 locations as well. Unlike a full tag directory structure, which encodes the offset or address (memory
 7499 location) of every data element within the data carrier, an Indexed Directory encodes a small
 7500 relative position or index indicating which Packed Object contains each data element. An application
 7501 designer may choose to also encode the locations of each Packed Object in an optional ObjectOffsets
 7502 subsection as described above, so that a decoding system, upon reading the Indexed Directory
 7503 alone, can calculate the start addresses of all Packed Objects in memory.

7504 The utility of an ID Map used in this way is enhanced by the rule of most data systems that a given
 7505 identifier may only appear once within a single data carrier. This rule, when an Indexed Directory is
 7506 utilised with Packed Object encoding of the data in subsequent objects, can provide nearly-complete
 7507 random access to reading data using relatively few directory bits. As an example, an ID Map
 7508 directory (one bit per defined ID) can be associated with an additional AuxMap "PO Index" array
 7509 (using, for example, three bits per defined ID). Using this arrangement, an interrogator would read
 7510 the Directory Packed Object, and examine its ID Map to determine if the desired data item were
 7511 present on the tag. If so, it would examine the 3 "PO Index" bits corresponding to that data item, to
 7512 determine which of the first 8 Packed Objects on the tag contain the desired data item. If an
 7513 optional ObjectOffsets subsection was encoded, then the Interrogator can calculate the starting
 7514 address of the desired Packed Object directly; otherwise, the interrogator may perform successive
 7515 read operations in order to fetch the desired Packed Object.

7516 **J Packed Objects ID tables**

7517 **J.1 Packed Objects data format registration file structure**

7518 A Packed Objects registered Data Format file consists of a series of "Keyword lines" and one or more
 7519 ID Tables. Blank lines may occur anywhere within a Data Format File, and are ignored. Also, any
 7520 line may end with extra blank columns, which are also ignored.

- 7521 ■ A Keyword line consists of a Keyword (which always starts with "K-") followed by an equals sign
 7522 and a character string, which assigns a value to that Keyword. Zero or more space characters
 7523 may be present on either side of the equals sign. Some Keyword lines shall appear only once, at
 7524 the top of the registration file, and others may appear multiple times, once for each ID Table in
 7525 the file.
- 7526 ■ An ID Table lists a series of ID Values (as defined in [I.5.3](#)). Each row of an ID Table contains a
 7527 single ID Value (in a required "IDvalue" column), and additional columns may associate Object
 7528 IDs (OIDs), ID strings, Format strings, and other information with that ID Value. A registration
 7529 file always includes a single "Primary" Base ID Table, zero or more "Alternate" Base ID Tables,
 7530 and may also include one or more Secondary ID Tables (that are referenced by one or more
 7531 Base ID Table entries).

7532 To illustrate the file format, a hypothetical data system registration is shown in Figure J-1. In this
 7533 hypothetical data system, each ID Value is associated with one or more OIDs and corresponding ID
 7534 strings. The following subsections explain the syntax shown in the Figure.

7535 **Figure I.9.2-1** Hypothetical Data Format registration file

| K-Text = Hypothetical Data Format 100 | | | | |
|--|----------------------|----------------------|---|----------------------|
| K-Version = 1.0 | | | | |
| K-TableID = F100B0 | | | | |
| K-RootOID = urn:oid:1.0.12345.100 | | | | |
| K-IDsize = 16 | | | | |
| IDvalue | OIDs | IDstring | Explanation | FormatString |
| 0 | 99 | 1Z | Legacy ID "1Z" corresponds to OID 99, is assigned IDval 0 | 14n |
| 1 | 9%x30-33 | 7%x42-45 | An OID in the range 90..93, Corresponding to ID 7B..7E | 1*8an |
| 2 | (10)(20)(25)(37) | (A)(B)(C)(D) | a commonly-used set of IDs | (1n)(2n)(3n)(4n) |
| 3 | 26/27 | 1A/2B | Either 1A or 2B is encoded, but not both | 10n / 20n |
| 4 | (30) [31] | (2A) [3B] | 2A is always encoded, optionally followed by 3B | (11n) [1*20n] |
| 5 | (40/41/42) (53) [55] | (4A/4B/4C) (5D) [5E] | One of A/B/C is encoded, then D, and optionally E | (1n/2n/3n) (4n) [5n] |

| | | | | |
|----------------------------|----------------------|------------------------|---|----------------------|
| 6 | (60/61/(64)[66]) | (6A /6B / (6C [6D]) | Selections, one of which includes an Option | (1n / 2n / (3n)[4n]) |
| K-TableEnd = F100B0 | | | | |

7536 **J.1.1 File Header section**

7537 Keyword lines in the File Header (the first portion of every registration file) may occur in any order,
7538 and are as follows:

- 7539 ■ **(Mandatory) K-Version = nn.nn**, which the registering body assigns, to ensure that any
7540 future revisions to their registration are clearly labelled.
- 7541 ■ **(Optional) K-Interpretation = string**, where the "string" argument shall be one of the
7542 following: "ISO-646", "UTF-8", "ECI-nnnnnn" (where nnnnnn is a registered six-digit ECI
7543 number), ISO-8859-nn, or "UNSPECIFIED". The Default interpretation is "UNSPECIFIED". This
7544 keyword line allows non-default interpretations to be placed on the octets of data strings that
7545 are decoded from Packed Objects.
- 7546 ■ **(Optional) K-ISO15434=nn**, where "nn" represents a Format Indicator (a two-digit numeric
7547 identifier) as defined in ISO/IEC 15434. This keyword line allows receiving systems to optionally
7548 represent a decoded Packed Object as a fully-compliant ISO/IEC 15434 message. There is no
7549 default value for this keyword line.
- 7550 ■ **(Optional) K-AppPunc = nn**, where nn represents (in decimal) the octet value of an ASCII
7551 character that is commonly used for punctuation in this application. If this keyword line is not
7552 present, the default Application Punctuation character is the hyphen.

7553 In addition, h may be included using the optional Keyword assignment line "K-text = string", and
7554 may appear zero or more times within a File Header or Table Header, but not in an ID Table body.

7555 **J.1.2 Table Header section**

7556 One or more Table Header sections (each introducing an ID Table) follow the File Header section.
7557 Each Table Header begins with a K-TableID keyword line, followed by a series of additional required
7558 and optional Keyword lines (which may occur in any order), as follows:

- 7559 ■ **(Mandatory) K-TableID = FnnXnn**, where **Fnn** represents the ISO-assigned Data Format
7560 number (where 'nn' represents one or more decimal digits), and Xnn (where 'X' is either 'B' or
7561 'S') is a registrant-assigned Table ID for each ID Table in the file. The first ID Table shall always
7562 be the Primary Base ID Table of the registration, with a Table ID of "B0". As many as seven
7563 additional "Alternate" Base ID Tables may be included, with higher sequential "Bnn" Table IDs.
7564 Secondary ID Tables may be included, with sequential Table IDs of the form "Snn".
- 7565 ■ **(Mandatory) K-IDsize = nn**. For a base ID table, the value **nn** shall be one of the values
7566 from the "Maximum number of Table Entries" column of Table I 5-5. For a secondary ID table,
7567 the value **nn** shall be a power of two (even if not present in Table I 5-5).
- 7568 ■ **(Optional) K-RootOID = urn:oid:i.j.k.ff** where:
 - 7569 □ **I, j, and k** are the leading arcs of the OID (as many arcs as required) and
 - 7570 □ **ff** is the last arc of the Root OID (typically, the registered Data Format number)

7571 If the K-RootOID keyword is not present, then the default Root OID is:

- 7572 □ **urn:oid:1.0.15961.ff**, where "ff" is the registered Data Format number
- 7573 ■ **Other optional Keyword lines:** in order to override the file-level defaults (to set different
7574 values for a particular table), a Table Header may invoke one or more of the Optional Keyword
7575 lines listed in for the File Header section.

7576 The end of the Table Header section is the first non-blank line that does not begin with a Keyword.
7577 This first non-blank line shall list the titles for every column in the ID Table that immediately follows
7578 this line; column titles are case-sensitive.

7579 An Alternate Base ID Table, if present, is identical in format to the Primary Base ID Table (but
7580 usually represents a smaller choice of identifiers, targeted for a specific application).

7581 A Secondary ID Table can be invoked by a keyword in a Base Table's **OIDs** column. A Secondary ID
 7582 Table is equivalent to a single Selection list (see [1.3](#)) for a single ID Value of a Base ID Table (except
 7583 that a Secondary table uses K-Idsize to explicitly define the number of Secondary ID bits per ID);
 7584 the IDvalue column of a Secondary table lists the value of the corresponding Secondary ID bits
 7585 pattern for each row in the Secondary Table. An **OIDs** entry in a Secondary ID Table shall not itself
 7586 contain a Selection list nor invoke another Secondary ID Table.

7587 **J.1.3 ID Table section**

7588 Each ID table consists of a series of one or more rows, each row including a mandatory "IDvalue"
 7589 column, several defined Optional columns (such as "OIDs", "IDstring", and "FormatString"), and any
 7590 number of Informative columns (such as the "Explanation" column in the hypothetical example
 7591 shown above).

7592 Each ID Table ends with a required Keyword line of the form:

- 7593 ■ **K-TableEnd = FnnXnn**, where **FnnXnn** shall match the preceding **K-TableID** keyword line
 7594 that introduced the table.

7595 The syntax and requirements of all Mandatory and Optional columns shall be as described J.2.

7596 **J.2 Mandatory and optional ID table columns**

7597 Each ID Table in a Packed Objects registration shall include an IDvalue column, and may include
 7598 other columns that are defined in this specification as Optional, and/or Informative columns (whose
 7599 column heading is not defined in this specification).

7600 **J.2.1 IDvalue column (Mandatory)**

7601 Each ID Table in a Packed Objects registration shall include an IDvalue column. The ID Values on
 7602 successive rows shall increase monotonically. However, the table may terminate before reaching the
 7603 full number of rows indicated by the Keyword line containing **K-IDsize**. In this case, a receiving
 7604 system will assume that all remaining ID Values are reserved for future assignment (as if the OIDs
 7605 column contained the keyword "K-RFA"). If a registered Base ID Table does not include the optional
 7606 OIDs column described below, then the IDvalue shall be used as the last arc of the OID.

7607 **J.2.2 OIDs and IDstring columns (Optional)**

7608 A Packed Objects registration always assigns a final OID arc to each identifier (either a number
 7609 assigned in the "OIDs" column as will be described below, or if that column is absent, the IDvalue is
 7610 assigned as the default final arc). The OIDs column is required rather than optional, if a single
 7611 IDvalue is intended to represent either a combination of OIDs or a choice between OIDs (one or
 7612 more Secondary ID bits are invoked by any entry that presents a choice of OIDs).

7613 A Packed Objects registration may include an IDstring column, which if present assigns an ASCII-
 7614 string name for each OID. If no name is provided, systems must refer to the identifier by its OID
 7615 (see [1.3](#)). However, many registrations will be based on data systems that do have an ASCII
 7616 representation for each defined Identifier, and receiving systems may optionally output a
 7617 representation based on those strings. If so, the ID Table may contain a column indicating the
 7618 IDstring that corresponds to each OID. An empty IDstring cell means that there is no corresponding
 7619 ASCII string associated with the OID. A non-empty IDstring shall provide a name for every OID
 7620 invoked by the OIDs column of that row (or a single name, if no OIDs column is present). Therefore,
 7621 the sequence of combination and selection operations in an IDstring shall exactly match those in the
 7622 row's OIDs column.

7623 A non-empty **OIDs** cell may contain either a keyword, an ASCII string representing (in decimal) a
 7624 single OID value, or a compound string (in ABNF notation) that defines a choice and/or a
 7625 combination of OIDs. The detailed syntax for compound OID strings in this column (which also

7626
7627

applies to the IDstring column) is as defined in section 1.3. Instead of containing a simple or compound OID representation, an OIDs entry may contain one of the following Keywords:

7628
7629
7630

- **K-Verbatim = OIDddBnn**, where "dd" represents the chosen penultimate arc of the OID, and "Bnn" indicates one of the Base 10, Base 40, or Base 74 encoding tables. This entry invokes a number of Secondary ID bits that serve two purposes:

7631
7632
7633
7634
7635

- They encode an ASCII identifier "name" that might not have existed at the time the table was registered. The name is encoded in the Secondary ID bits section as a series of Base-n values representing the ASCII characters of the name, preceded by a four-bit field indicating the number of Base-n values that follow (zero is permissible, in order to support RFA entries as described below).

7636
7637
7638

- The cumulative value of these Secondary ID bits, considered as a single unsigned binary integer and converted to decimal, is the final "arc" of the OID for this "verbatim-encoded" identifier.

7639
7640
7641
7642
7643

- **K-Secondary = Snn**, where "Snn" represents the Table ID of a Secondary ID Table in the same registration file. This is equivalent to a Base ID Table row OID entry that contains a single Selection list (with no other components at the top level), but instead of listing these components in the Base ID Table, each component is listed as a separate row in the Secondary ID Table, where each may be assigned a unique OID, ID string, and FormatString.

7644
7645
7646

- **K-Proprietary=OIDddPnn**, where nn represents a fixed number of Secondary ID bits that encode an optional Enterprise Identifier indicating who wrote the proprietary data (an entry of **K-Proprietary=OIDddPO** indicates an "anonymous" proprietary data item).

7647
7648
7649
7650
7651
7652
7653

- **K-RFA = OIDddBnn**, where "Bnn" is as defined above for Verbatim encoding, except that "B0" is a valid assignment (meaning that no Secondary ID bits are invoked). This keyword represents a Reserved for Future Assignment entry, with an option for Verbatim encoding of the Identifier "name" once a name is assigned by the entity who registered this Data Format. Encoders may use this entry, with a four-bit "verbatim" length of zero, until an Identifier "name" is assigned. A specific FormatString may be assigned to K-RFA entries, or the default a/n encoding may be utilised.

7654
7655
7656
7657
7658
7659

Finally, any OIDs entry may end with a single "R" character (preceded by one or more space characters), to indicate that a "Repeat" bit shall be encoded as the last Secondary ID bit invoked by the entry. If '1', this bit indicates that another instance of this class of identifier is also encoded (that is, this bit acts as if a repeat of the ID Value were encoded on an ID list). If '0', then this bit is followed by another series of Secondary ID bits, to represent the particulars of this additional instance of the ID Value.

7660
7661

An IDstring column shall not contain any of the above-listed Keyword entries, and an IDstring entry shall be empty when the corresponding OIDs entry contains a Keyword.

7662 J.2.3 FormatString column (Optional)

7663
7664
7665
7666
7667
7668
7669
7670

An ID Table may optionally define the data characteristics of the data associated with a particular identifier, in order to facilitate data compaction. If present, the FormatString entry specifies whether a data item is all-numeric or alphanumeric (i.e., may contain characters other than the decimal digits), and specifies either a fixed length or a variable length. If no FormatString entry is present, then the default data characteristic is alphanumeric. If no FormatString entry is present, or if the entry does not specify a length, then any length ≥ 1 is permitted. Unless a single fixed length is specified, the length of each encoded data item is encoded in the Aux Format section of the Packed Object, as specified in 1.7.

7671
7672
7673

If a given IDstring entry defines more than a single identifier, then the corresponding FormatString column shall show a format string for each such identifier, using the same sequence of punctuation characters (disregarding concatenation) as was used in the corresponding IDstring.

7674

The format string for a single identifier shall be one of the following:

7675
7676

- A length qualifier followed by "n" (for always-numeric data);
- A length qualifier followed by "an" (for data that may contain non-digits); or

- 7677
- 7678
- A fixed-length qualifier, followed by "n", followed by one or more space characters, followed by a variable-length qualifier, followed by "an".

7679 A length qualifier shall be either null (that is, no qualifier present, indicating that any length ≥ 1 is legal), a single decimal number (indicating a fixed length) or a length range of the form "i*j", where "I" represents the minimum allowed length of the data item, "j" represents the maximum allowed length, and $i \leq j$. In the latter case, if "j" is omitted, it means the maximum length is unlimited.

7683 Data corresponding to an "n" in the FormatString are encoded in the KLN subsection; data corresponding to an "an" in the FormatString are encoded in the A/N subsection.

7685 When a given instance of the data item is encoded in a Packed Object, its length is encoded in the Aux Format section as specified in [1.7.2](#). The minimum value of the range is not itself encoded, but is specified in the ID Table's FormatString column.

7688 **Example:**

7689 A FormatString entry of "3*6n" indicates an all-numeric data item whose length is always between three and six digits inclusive. A given length is encoded in two bits, where '00' would indicate a string of digits whose length is "3", and '11' would indicate a string length of six digits.

7692 **J.2.4 Interp column (Optional)**

7693 Some registrations may wish to specify information needed for output representations of the Packed Object's contents, other than the default OID representation of the arcs of each encoded identifier. If this information is invariant for a particular table, the registration file may include keyword lines as previously defined. If the interpretation varies from row to row within a table, then an Interp column may be added to the ID Table. This column entry, if present, may contain one or more of the following keyword assignments (separated by semicolons), as were previously defined (see J.1.1 and J.1.2):

- 7700
- K-RootOID = urn:oid:i.j.k.l...
 - 7701 ■ K-Interpretation = string
 - 7702 ■ K-ISO15434=nn

7703 If used, these override (for a particular Identifier) the default file-level values and/or those specified in the Table Header section.

7705 **J.3 Syntax of OIDs, IDstring, and FormatString Columns**

7706 In a given ID Table entry, the OIDs, IDString, and FormatString column may indicate one or more mechanisms described in this section. [J.3.1](#) specifies the semantics of the mechanisms, and [J.3.2](#) specifies the formal grammar for the ID Table columns.

7709 **J.3.1 Semantics for OIDs, IDString, and FormatString Columns**

7710 In the descriptions below, the word "Identifier" means either an OID final arc (in the context of the OIDs column) or an IDString name (in the context of the IDstring column). If both columns are present, only the OIDs column actually invokes Secondary ID bits.

- 7713
- 7714 ■ A **Single component** resolving to a single Identifier, in which case no additional Secondary ID bits are invoked.
 - 7715 ■ (For OIDs and IDString columns only) A single component resolving to one of a series of closely-related Identifiers, where the Identifier's string representation varies only at one or more character positions. This is indicated using the **Concatenation** operator '%' to introduce a range of ASCII characters at a specified position. For example, an OID whose final arc is defined as "391n", where the fourth digit 'n' can be any digit from '0' to '6' (ASCII characters 30_{hex} to 36_{hex} inclusive) is represented by the component **391%x30-39** (note that no spaces are allowed). A Concatenation invokes the minimum number of Secondary ID digits needed to indicate the specified range. When both an OIDs column and an IDstring column are populated for a given row, both shall contain the same number of concatenations, with the same ranges (so that the numbers and values of Secondary ID bits invoked are consistent). However, the minimum value listed for the two ranges can differ, so that (for example) the OID's digit can

range from 0 to 3, while the corresponding IDstring character can range from "B" to "E" if so desired. Note that the use of Concatenation inherently constrains the relationship between OID and IDString, and so Concatenation may not be useable under all circumstances (the Selection operation described below usually provides an alternative).

- A **Combination** of two or more identifier components in an ordered sequence, indicated by surrounding each component of the sequence with parentheses. For example, an IDstring entry **(A)(%x30-37B)(2C)** indicates that the associated ID Value represents a sequence of the following three identifiers:

- Identifier "A", then

- An identifier within the range "0B" to "7B" (invoking three Secondary ID bits to represent the choice of leading character), then

- Identifier "2C"

Note that a Combination does not itself invoke any Secondary ID bits (unless one or more of its components do).

- An **Optional** component is indicated by surrounding the component in brackets, which may be viewed as a "conditional combination." For example the entry (A) [B][C][D] indicates that the ID Value represents identifier A, optionally followed by B, C, and/or D. A list of Options invokes one Secondary ID bit for each component in brackets, wherein a '1' indicates that the optional component was encoded.

- A **Selection** between several mutually-exclusive components is indicated by separating the components by forward slash characters. For example, the IDstring entry **(A/B/C/(D)(E))** indicates that the fully-qualified ID Value represents a single choice from a list of four choices (the fourth of which is a Combination). A Selection invokes the minimum number of Secondary ID bits needed to indicate a choice from a list of the specified number of components.

In general, a "compound" OIDs or IDstring entry may contain any or all of the above operations. However, to ensure that a single left-to-right parsing of an OIDs entry results in a deterministic set of Secondary ID bits (which are encoded in the same left-to-right order in which they are invoked by the OIDs entry), the following restrictions are applied:

- A given Identifier may only appear once in an OIDs entry. For example, the entry (A)(B/A) is invalid

- A OIDs entry may contain at most a single Selection list

- There is no restriction on the number of Combinations (because they invoke no Secondary ID bits)

- There is no restriction on the total number of Concatenations in an OIDs entry, but no single Component may contain more than two Concatenation operators.

- An Optional component may be a component of a Selection list, but an Optional component may not be a compound component, and therefore shall not include a Selection list nor a Combination nor Concatenation.

- A OIDs or IDstring entry may not include the characters `'`, `'`, `'[`, `']`, `'%`, `'-`, or `'/'`, unless used as an Operator as described above. If one of these characters is part of a defined data system Identifier "name", then it shall be represented as a single literal Concatenated character.

J.3.2 Formal Grammar for OIDs, IDString, and FormatString Columns

In each ID Table entry, the contents of the OIDs, IDString, and FormatString columns shall conform to the following grammar for `Expr`, unless the column is empty or (in the case of the OIDs column) it contains a keyword as specified in 1.2.2. All three columns share the same grammar, except that the syntax for `COMPONENT` is different for each column as specified below. In a given ID Table Entry, the contents of the OIDs, IDString, and FormatString column (except if empty) shall have identical parse trees according to this grammar, except that the `COMPONENTS` may be different. Space characters are permitted (and ignored) anywhere in an `Expr`, except that in the interior of a `COMPONENT` spaces are only permitted where explicitly specified below.

```
Expr = SelectionExpr / "(" SelectionExpr ")" / SelectionSubexpr
```

7777 SelectionExpr = SelectionSubexpr 1*("/" SelectionSubexpr)
 7778
 7779 SelectionSubexpr = COMPONENT / ComboExpr
 7780
 7781 ComboExpr = 1*ComboSubexpr
 7782
 7783 ComboSubexpr = "(" COMPONENT ")" / "[" COMPONENT "]"
 7784

7785 For the OIDs column, COMPONENT shall conform to the following grammar:

7786 COMPONENT_OIDs = 1*(COMPONENT_OIDs_Char / Concat)
 7787
 7788 COMPONENT_OIDs_Char = 1*(%x30-39) ; 0-9
 7789

7790 For the IDString column, COMPONENT shall conform to the following grammar:

7791 COMPONENT_IDString = UnquotedIDString / QuotedIDString
 7792
 7793 UnquotedIDString = 1*(UnquotedIDStringChar / Concat)
 7794
 7795 UnquotedIDStringChar = %x30-39 / %x41-5A / %x61-7A / "_" ; 0-9 A-Z a-z _
 7796
 7797 QuotedIDString = QUOTE 1*QuotedIDStringConstituent QUOTE
 7798
 7799 QuotedIDStringConstituent = " " / "!" / "#".~" / (QUOTE QUOTE)
 7800
 7801 QUOTE = %x22 ; ASCII double quote
 7802
 7803 QUOTE refers to ASCII character 34 (decimal), the double quote character.

7804 When the QuotedIDString form for COMPONENT_IDString is used, the beginning and ending
 7805 QUOTE characters shall *not* be considered part of the IDString. Between the beginning and ending
 7806 QUOTE, all ASCII characters in the range 32 (decimal) through 126 (decimal), inclusive, are allowed,
 7807 except that two QUOTE characters in a row shall denote a single double-quote character to be
 7808 included in the IDString.

7809 In the QuotedIDString form, a % character does not denote the concatenation operator, but
 7810 instead is just a percent character included literally in the IDString. To use the concatenation
 7811 operator, the UnquotedIDString form must be used. In that case, a degenerate concatenation
 7812 operator (where the start character equals the end character) may be used to include a character
 7813 into the IDString that is not one of the characters listed for UnquotedIDStringChar.

7814 For the FormatString column, COMPONENT shall conform to the following grammar:

7815 COMPONENT_FormatString = 0*1Range ("an" / "n")
 7816 / FixedRange "n" 1*" " VarRange "an"
 7817
 7818 Range = FixedRange / VarRange
 7819
 7820 FixedRange = Number
 7821
 7822 VarRange = Number "*" 0*1(Number)
 7823
 7824 Number = 1*(%x30-39) ; 0-9

7825 The syntax for COMPONENT for the OIDs and IDString columns make reference to Concat, whose
 7826 syntax is specified as follows:

7827 Concat = "%" "x" HexChar "-" HexChar
 7828 HexChar = (%x30-39 / %x41-46) ; 0-9 A-F

7827 The hex value following the hyphen shall be greater than or equal to the hex value preceding the
 7828 hyphen. In the OIDs column, each hex value shall be in the range 30_{hex} to 39_{hex}, inclusive. In the
 7829 IDString column, each hex value shall be in the range 20_{hex} to 7E_{hex}, inclusive.

7830 J.4 OID input/output representation

7831 The default method for representing the contents of a Packed Object to a receiving system is as a
 7832 series of name/value pairs, where the name is an OID, and the value is the decoded data string
 7833 associated with that OID. Unless otherwise specified by a **K-RootOID** keyword line, the default root
 7834 OID is **urn:oid:1.0.15961.ff**, where **ff** is the Data Format encoded in the DSFID. The final arc of
 7835 the OID is (by default) the IDvalue, but this is typically overridden by an entry in the OIDs column.
 7836 Note that an encoded Application Indicator (see [I.5.3](#)) may change **ff** from the value indicated by
 7837 the DSFID.

7838 If supported by information in the ID Table's IDstring column, a receiving system may translate the
 7839 OID output into various alternative formats, based on the IDString representation of the OIDs. One
 7840 such format, as described in ISO/IEC 15434, requires as additional information a two-digit Format
 7841 identifier; a table registration may provide this information using the **K-ISO15434** keyword as
 7842 described above.

7843 The combination of the K-RootOID keyword and the OIDs column provides the registering entity an
 7844 ability to assign OIDs to data system identifiers without regard to how they are actually encoded,
 7845 and therefore the same OID assignment can apply regardless of the access method.

7846 J.4.1 "ID Value OID" output representation

7847 If the receiving system does not have access to the relevant ID Table (possibly because it is newly-
 7848 registered), the Packed Objects decoder will not have sufficient information to convert the IDvalue
 7849 (plus Secondary ID bits) to the intended OID. In order to ease the introduction of new or external
 7850 tables, encoders have an option to follow "restricted use" rules (see [I.5.3](#)).

7851 When a receiving system has decoded a Packed Object encoded following "restricted use" rules, but
 7852 does not have access to the indicated ID Table, it shall construct an "ID Value OID" in the following
 7853 format:

7854 **urn:oid:1.0.15961.300.ff.bb.idval.secbits**

7855 where **1.0.15961.300** is a Root OID with a reserved Data Format of "300" that is never encoded in
 7856 a DSFID, but is used to distinguish an "ID Value OID" from a true OID (as would have been used if
 7857 the ID Table were available). The reserved value of 300 is followed by the encoded table's Data
 7858 Format (**ff**) (which may be different from the DSFID's default), the table ID (**bb**) (always '0', unless
 7859 otherwise indicated via an encoded Application Indicator), the encoded ID value, and the decimal
 7860 representation of the invoked Secondary ID bits. This process creates a unique OID for each unique
 7861 fully-qualified ID Value. For example, using the hypothetical ID Table shown in Annex [L](#) (but
 7862 assuming, for illustration purposes, that the table's specified Root OID is **urn:oid:1.0.12345.9**,
 7863 then an "AMOUNT" ID with a fourth digit of '2' has a true OID of:

7864 **urn:oid:1.0.12345.9.3912**

7865 **and an "ID Value OID" of**

7866 **urn:oid:1.0.15961.300.9.0.51.2**

7867 When a single ID Value represents multiple component identifiers via combinations or optional
 7868 components, their multiple OIDs and data strings shall be represented separately, each using the
 7869 same "ID Value OID" (up through and including the Secondary ID bits arc), but adding as a final arc
 7870 the component number (starting with "1" for the first component decoded under that IDvalue).

7871 If the decoding system encounters a Packed Object that references an ID Table that is unavailable
 7872 to the decoder, but the encoder chose not to set the "Restricted Use" bit in the Application Indicator,
 7873 then the decoder shall either discard the Packed Object, or relay the entire Packed Object to the
 7874 receiving system as a single undecoded binary entity, a sequence of octets of the length specified in
 7875 the ObjectLength field of the Packed Object. The OID for an undecoded Packed Object shall be
 7876 **urn:oid:1.0.15961.301.ff.n**, where "301" is a Data Format reserved to indicate an undecoded
 7877 Packed Object, "ff" shall be the Data Format encoded in the DSFID at the start of memory, and an

7878
7879

optional final arc 'n' may be incremented sequentially to distinguish between multiple undecoded Packed Objects in the same data carrier memory.

K Packed Objects encoding tables

Packed Objects primarily utilise two encoding bases:

- Base 10, which encodes each of the digits '0' through '9' in one Base 10 value
- Base 30, which encodes the capital letters and selectable punctuation in one Base-30 value, and encodes punctuation and control characters from the remainder of the ASCII character set in two base-30 values (using a Shift mechanism)

For situations where a high percentage of the input data's non-numeric characters would require pairs of base-30 values, two alternative bases, Base 74 and Base 256, are also defined:

- The values in the Base 74 set correspond to the invariant subset of ISO/IEC 646 [ISO646] (which includes the GS1 character set), but with the digits eliminated, and with the addition of GS and <space> (GS is supported for uses other than as a data delimiter).
- The values in the Base 256 set may convey octets with no graphical-character interpretation, or "extended ASCII values" as defined in ISO/IEC 8859-6 [ISO8859-6], or UTF-8 (the interpretation may be set in the registered ID Table for an application). The characters '0' through '9' (ASCII values 48 through 57) are supported, and an encoder may therefore encode the digits either by using a prefix or suffix (in Base 256) or by using a character map (in Base 10). Note that in GS1 data, FNC1 is represented by ASCII <GS> (octet value 29_{dec}).

Finally, there are situations where compaction efficiency can be enhanced by run-length encoding of base indicators, rather than by character map bits, when a long run of characters can be classified into a single base. To facilitate that classification, additional "extension" bases are added, only for use in Prefix and Suffix Runs.

- In order to support run-length encoding of a primarily-numeric string with a few interspersed letters, a Base 13 is defined, per Table B-2
- Two of these extension bases (Base 40 and Base 84) are simply defined, in that they extend the corresponding non-numeric bases (Base 30 and Base 74, respectively) to also include the ten decimal digits. The additional entries, for characters '0' through '9', are added as the next ten sequential values (values 30 through 39 for Base 40, and values 74 through 83 for Base 84).
- The "extended" version of Base 256 is defined as Base 40. This allows an encoder the option of encoding a few ASCII control or upper-ASCII characters in Base 256, while using a Prefix and/or Suffix to more efficiently encode the remaining non-numeric characters.

The number of bits required to encode various numbers of Base 10, Base 16, Base 30, Base 40, Base 74, and Base 84 characters are shown in Figure B-1. In all cases, a limit is placed on the size of a single input group, selected so as to output a group no larger than 20 octets.

7913

Figure J.4.1-1 Required number of bits for a given number of Base 'N' values

```

7914 /* Base10 encoding accepts up to 48 input values per group: */
7915 static const unsigned char bitsForNumBase10[] = {
7916 /* 0 - 9 */ 0, 4, 7, 10, 14, 17, 20, 24, 27, 30,
7917 /* 10 - 19 */ 34, 37, 40, 44, 47, 50, 54, 57, 60, 64,
7918 /* 20 - 29 */ 67, 70, 74, 77, 80, 84, 87, 90, 94, 97,
7919 /* 30 - 39 */ 100, 103, 107, 110, 113, 117, 120, 123, 127, 130,
7920 /* 40 - 48 */ 133, 137, 140, 143, 147, 150, 153, 157, 160};
7921
7922 /* Base13 encoding accepts up to 43 input values per group: */
7923 static const unsigned char bitsForNumBase13[] = {
7924 /* 0 - 9 */ 0, 4, 8, 12, 15, 19, 23, 26, 30, 34,
7925 /* 10 - 19 */ 38, 41, 45, 49, 52, 56, 60, 63, 67, 71,
7926 /* 20 - 29 */ 75, 78, 82, 86, 89, 93, 97, 100, 104, 108,
7927 /* 30 - 39 */ 112, 115, 119, 123, 126, 130, 134, 137, 141, 145,
7928 /* 40 - 43 */ 149, 152, 156, 160 };
7929
7930 /* Base30 encoding accepts up to 32 input values per group: */
7931 static const unsigned char bitsForNumBase30[] = {
7932 /* 0 - 9 */ 0, 5, 10, 15, 20, 25, 30, 35, 40, 45,
7933 /* 10 - 19 */ 50, 54, 59, 64, 69, 74, 79, 84, 89, 94,
7934 /* 20 - 29 */ 99, 104, 108, 113, 118, 123, 128, 133, 138, 143,
7935 /* 30 - 32 */ 148, 153, 158};
7936
7937 /* Base40 encoding accepts up to 30 input values per group: */
7938 static const unsigned char bitsForNumBase40[] = {
7939 /* 0 - 9 */ 0, 6, 11, 16, 22, 27, 32, 38, 43, 48,
7940 /* 10 - 19 */ 54, 59, 64, 70, 75, 80, 86, 91, 96, 102,
7941 /* 20 - 29 */ 107, 112, 118, 123, 128, 134, 139, 144, 150, 155,
7942 /* 30 */ 160 };
7943
7944 /* Base74 encoding accepts up to 25 input values per group: */
7945 static const unsigned char bitsForNumBase74[] = {
7946 /* 0 - 9 */ 0, 7, 13, 19, 25, 32, 38, 44, 50, 56,
7947 /* 10 - 19 */ 63, 69, 75, 81, 87, 94, 100, 106, 112, 118,
7948 /* 20 - 25 */ 125, 131, 137, 143, 150, 156 };
7949
7950 /* Base84 encoding accepts up to 25 input values per group: */
7951 static const unsigned char bitsForNumBase84[] = {
7952 /* 0 - 9 */ 0, 7, 13, 20, 26, 32, 39, 45, 52, 58,
7953 /* 10 - 19 */ 64, 71, 77, 84, 90, 96, 103, 109, 116, 122,
7954 /* 20 - 25 */ 128, 135, 141, 148, 154, 160 };

```

7955

Table J.4.1-1 Base 30 Character set

| Val | Basic set | | Shift 1 set | | Shift 2 set | |
|-----|---------------------|---------|-------------|---------|-------------|---------|
| | Char | Decimal | Char | Decimal | Char | Decimal |
| 0 | A-Punc ¹ | N/A | NUL | 0 | space | 32 |
| 1 | A | 65 | SOH | 1 | ! | 33 |
| 2 | B | 66 | STX | 2 | " | 34 |
| 3 | C | 67 | ETX | 3 | # | 35 |
| 4 | D | 68 | EOT | 4 | \$ | 36 |
| 5 | E | 69 | ENQ | 5 | % | 37 |
| 6 | F | 70 | ACK | 6 | & | 38 |
| 7 | G | 71 | BEL | 7 | ` | 39 |
| 8 | H | 72 | BS | 8 | (| 40 |
| 9 | I | 73 | HT | 9 |) | 41 |
| 10 | J | 74 | LF | 10 | * | 42 |

| Val | Basic set | | Shift 1 set | | Shift 2 set | |
|-----|---------------------|-----|-------------|-----|-------------|-----|
| | | | | | | |
| 11 | K | 75 | VT | 11 | + | 43 |
| 12 | L | 76 | FF | 12 | , | 44 |
| 13 | M | 77 | CR | 13 | - | 45 |
| 14 | N | 78 | SO | 14 | . | 46 |
| 15 | O | 79 | SI | 15 | / | 47 |
| 16 | P | 80 | DLE | 16 | : | 58 |
| 17 | Q | 81 | ETB | 23 | ; | 59 |
| 18 | R | 82 | ESC | 27 | < | 60 |
| 19 | S | 83 | FS | 28 | = | 61 |
| 20 | T | 84 | GS | 29 | > | 62 |
| 21 | U | 85 | RS | 30 | ? | 63 |
| 22 | V | 86 | US | 31 | @ | 64 |
| 23 | W | 87 | invalid | N/A | \ | 92 |
| 24 | X | 88 | invalid | N/A | ^ | 94 |
| 25 | Y | 89 | invalid | N/A | _ | 95 |
| 26 | Z | 90 | [| 91 | ` | 96 |
| 27 | Shift 1 | N/A |] | 93 | | 124 |
| 28 | Shift 2 | N/A | { | 123 | ~ | 126 |
| 29 | P-Punc ² | N/A | } | 125 | invalid | N/A |

7956
7957

Note 1: **Application-Specified Punctuation** character (Value 0 of the Basic set) is defined by default as the ASCII hyphen character (45_{dec}), but may be redefined by a registered Data Format

7958
7959
7960
7961
7962
7963
7964

Note 2: **Programmable Punctuation** character (Value 29 of the Basic set): the first appearance of P-Punc in the alphanumeric data for a Packed Object, whether that first appearance is compacted into the Base 30 segment or the Base 40 segment, acts as a <Shift 2>, and also "programs" the character to be represented by second and subsequent appearances of P-Punc (in either segment) for the remainder of the alphanumeric data in that Packed Object. The Base 30 or Base 40 value immediately following that first appearance is interpreted using the Shift 2 column (Punctuation), and assigned to subsequent instances of P-Punc for the Packed Object.

7965

Table J.4.1-2 Base 13 Character set

| Value | Basic set | | Shift 1 set | | Shift 2 set | | Shift 3 set | |
|-------|-----------|---------|-------------|---------|-------------|---------|-------------|---------|
| | Char | Decimal | Char | Decimal | Char | Decimal | Char | Decimal |
| 0 | 0 | 48 | A | 65 | N | 78 | space | 32 |
| 1 | 1 | 49 | B | 66 | O | 79 | \$ | 36 |
| 2 | 2 | 50 | C | 67 | P | 80 | % | 37 |
| 3 | 3 | 51 | D | 68 | Q | 81 | & | 38 |
| 4 | 4 | 52 | E | 69 | R | 82 | * | 42 |
| 5 | 5 | 53 | F | 70 | S | 83 | + | 43 |
| 6 | 6 | 54 | G | 71 | T | 84 | , | 44 |
| 7 | 7 | 55 | H | 72 | U | 85 | - | 45 |
| 8 | 8 | 56 | I | 73 | V | 86 | . | 46 |
| 9 | 9 | 57 | J | 74 | W | 87 | / | 47 |
| 10 | Shift1 | N/A | K | 75 | X | 88 | ? | 63 |
| 11 | Shift2 | N/A | L | 76 | Y | 89 | _ | 95 |
| 12 | Shift3 | N/A | M | 77 | Z | 90 | <GS> | 29 |

7966

Table J.4.1-3 Base 40 Character set

| Val | Basic set | | Shift 1 set | | Shift 2 set | |
|-----|---------------|---------|-------------|---------|-------------|---------|
| | Char | Decimal | Char | Decimal | Char | Decimal |
| 0 | See Table K-1 | | | | | |
| ... | ... | | | | | |
| 29 | See Table K-1 | | | | | |
| 30 | 0 | 48 | | | | |
| 31 | 1 | 49 | | | | |
| 32 | 2 | 50 | | | | |
| 33 | 3 | 51 | | | | |
| 34 | 4 | 52 | | | | |
| 35 | 5 | 53 | | | | |
| 36 | 6 | 54 | | | | |
| 37 | 7 | 55 | | | | |
| 38 | 8 | 56 | | | | |
| 39 | 9 | 57 | | | | |

7967

Table J.4.1-4 Character Set

| Val | Char | Decimal | Val | Char | Decimal | Val | Char | Decimal |
|-----|------|---------|-----|------|---------|-----|------|---------|
| 0 | GS | 29 | 25 | F | 70 | 50 | d | 100 |
| 1 | ! | 33 | 26 | G | 71 | 51 | e | 101 |
| 2 | " | 34 | 27 | H | 72 | 52 | f | 102 |
| 3 | % | 37 | 28 | I | 73 | 53 | g | 103 |
| 4 | & | 38 | 29 | J | 74 | 54 | h | 104 |
| 5 | ' | 39 | 30 | K | 75 | 55 | i | 105 |

| Val | Char | Decimal | Val | Char | Decimal | Val | Char | Decimal |
|-----|------|---------|-----|------|---------|-----|-------|---------|
| 6 | (| 40 | 31 | L | 76 | 56 | j | 106 |
| 7 |) | 41 | 32 | M | 77 | 57 | k | 107 |
| 8 | * | 42 | 33 | N | 78 | 58 | l | 108 |
| 9 | + | 43 | 34 | O | 79 | 59 | m | 109 |
| 10 | , | 44 | 35 | P | 80 | 60 | n | 110 |
| 11 | - | 45 | 36 | Q | 81 | 61 | o | 111 |
| 12 | . | 46 | 37 | R | 82 | 62 | p | 112 |
| 13 | / | 47 | 38 | S | 83 | 63 | q | 113 |
| 14 | : | 58 | 39 | T | 84 | 64 | r | 114 |
| 15 | ; | 59 | 40 | U | 85 | 65 | s | 115 |
| 16 | < | 60 | 41 | V | 86 | 66 | t | 116 |
| 17 | = | 61 | 42 | W | 87 | 67 | u | 117 |
| 18 | > | 62 | 43 | X | 88 | 68 | v | 118 |
| 19 | ? | 63 | 44 | Y | 89 | 69 | w | 119 |
| 20 | A | 65 | 45 | Z | 90 | 70 | x | 120 |
| 21 | B | 66 | 46 | _ | 95 | 71 | y | 121 |
| 22 | C | 67 | 47 | a | 97 | 72 | z | 122 |
| 23 | D | 68 | 48 | b | 98 | 73 | Space | 32 |
| 24 | E | 69 | 49 | c | 99 | | | |

7968

Table J.4.1-5 Base 84 Character Set

| Val | Char | Decimal | Val | Char | Decimal | Val | Char | Decimal |
|------|---------------|---------|-----|------|---------|-----|------|---------|
| 0 | FNC1 | N/A | 25 | F | | 50 | d | |
| 1-73 | See Table K-4 | | | | | | | |
| 74 | 0 | 48 | 78 | 4 | 52 | 82 | 8 | 56 |
| 75 | 1 | 49 | 79 | 5 | 53 | 83 | 9 | 57 |
| 76 | 2 | 50 | 80 | 6 | 54 | | | |
| 77 | 3 | 51 | 81 | 7 | 55 | | | |

L Encoding Packed Objects (non-normative)

In order to illustrate a number of the techniques that can be invoked when encoding a Packed Object, the following sample input data consists of data elements from a hypothetical data system. This data represents:

- An Expiration date (OID 7) of October 31, 2006, represented as a six-digit number 061031.
- An Amount Payable (OID 3n) of 1234.56 Euros, represented as a digit string 978123456 ("978" is the ISO Country Code indicating that the amount payable is in Euros). As shown in Table L-1, this data element is all-numeric, with at least 4 digits and at most 18 digits. In this example, the OID "3n" will be "32", where the "2" in the data element name indicates the decimal point is located two digits from the right.
- A Lot Number (OID 1) of 1A23B456CD

The application will present the above input to the encoder as a list of OID/Value pairs. The resulting input data, represented below as a single data string (wherein each OID final arc is shown in parentheses) is:

(7)061031(32)978123456(1)1A23B456CD

The example uses a hypothetical ID Table. In this hypothetical table, each ID Value is a seven-bit index into the Base ID Table; the entries relevant to this example are shown in Table L-1.

Encoding is performed in the following steps:

- Three data elements are to be encoded, using Table L-1.
- As shown in the table's IDstring column, the combination of OID 7 and OID 1 is efficiently supported (because it is commonly seen in applications), and thus the encoder re-orders the input so that 7 and 1 are adjacent and in the order indicated in the OIDs column:
- (7)061031(1)1A23B456CD(32)978123456
- Now, this OID pair can be assigned a single ID Value of 125 (decimal). The FormatString column for this entry shows that the encoded data will always consist of a fixed-length 6-digit string, followed by a variable-length alphanumeric string.
- Also as shown in Table L-1, OID 3n has an ID Value of 51 (decimal). The OIDs column for this entry shows that the OID is formed by concatenating "3" with a suffix consisting of a single character in the range 30_{hex} to 39_{hex} (i.e., a decimal digit). Since that is a range of ten possibilities, a four-bit number will need to be encoded in the Secondary ID section to indicate which suffix character was chosen. The FormatString column for this entry shows that its data is variable-length numeric; the variable length information will require four bits to be encoded in the Aux Format section.
- Since only a small percentage of the 128-entry ID Table is utilised in this Packed Object, the encoder chooses an ID List format, rather than an ID Map format. As this is the default format, no Format Flags section is required.
- This results in the following Object Info section:
 - EBV-6 (ObjectLength): the value is TBD at this stage of the encoding process
 - Pad Indicator bit: TBD at this stage
 - EBV-3 (numberOfIDs) of 001 (meaning two ID Values will follow)
 - An ID List, including:
 - First ID Value: 125 (dec) in 7 bits, representing OID 7 followed by OID 1
 - Second ID Value: 51 (decimal) in 7 bits, representing OID 3n
- A Secondary ID section is encoded as '0010', indicating the trailing '2' of the 3n OID. It so happens this '2' means that two digits follow the implied decimal point, but that information is not needed in order to encode or decode the Packed Object.
- Next, an Aux Format section is encoded. An initial '1' bit is encoded, invoking the Packed-Object compaction method. Of the three OIDs, only OID (3n) requires encoded Aux Format

8017
8018

information: a four-bit pattern of '0101' (representing "six" variable-length digits – as "one" is the first allowed choice, a pattern of "0101" denotes "six").

8019
8020
8021
8022
8023

- Next, the encoder encodes the first data item, for OID 7, which is defined as a fixed-length six-digit data item. The six digits of the source data string are "061031", which are converted to a sequence of six Base-10 values by subtracting 30_{hex} from each character of the string (the resulting values are denoted as values v₅ through v₀ in the formula below). These are then converted to a single Binary value, using the following formula:

8024

$$\square \quad 10^5 * v_5 + 10^4 * v_4 + 10^3 * v_3 + 10^2 * v_2 + 10^1 * v_1 + 10^0 * v_0$$

8025
8026
8027

According to Figure K-1, a six-digit number is always encoded into 20 bits (regardless of any leading zero's in the input), resulting in a Binary string of:
"0000 11101110 01100111"

8028
8029
8030

- The next data item is for OID 1, but since the table indicates that this OID's data is alphanumeric, encoding into the Packed Object is deferred until after all of the known-length numeric data is encoded.

8031
8032
8033
8034
8035
8036

- Next, the encoder finds that OID 3n is defined by Table L-1 as all-numeric, whose length of 9 (in this example) was encoded as (9 – 4 = 5) into four bits within the Aux Format subsection. Thus, a Known-Length-Numeric subsection is encoded for this data item, consisting of a binary value bit-pattern encoding 9 digits. Using Figure K-1 in Annex K, the encoder determines that 30 bits need to be encoded in order to represent a 9-digit number as a binary value. In this example, the binary value equivalent of "978123456" is the 30-bit binary sequence:

8037

"111010010011001111101011000000"

8038
8039

- At this point, encoding of the Known-Length Numeric subsection of the Data Section is complete.

8040
8041
8042
8043

Note that, so far, the total number of encoded bits is (3 + 6 + 1 + 7 + 7 + 4 + 5 + 20 + 30) or 83 bits, representing the IDLPO Length Section (assuming that a single EBV-6 vector remains sufficient to encode the Packed Object's length), two 7-bit ID Values, the Secondary ID and Aux Format sections, and two Known-Length-Numeric compacted binary fields.

8044
8045
8046
8047
8048
8049
8050
8051
8052

At this stage, only one non-numeric data string (for OID 1) remains to be encoded in the Alphanumeric subsection. The 10-character source data string is "1A23B456CD". This string contains no characters requiring a base-30 Shift out of the basic Base-30 character set, and so Base-30 is selected for the non-numeric base (and so the first bit of the Alphanumeric subsection is set to '0' accordingly). The data string has no substrings with six or more successive characters from the same base, and so the next two bits are set to '00' (indicating that neither a Prefix nor a Suffix is run-length encoded). Thus, a full 10-bit Character Map needs to be encoded next. Its specific bit pattern is '0100100011', indicating the specific sequence of digits and non-digits in the source data string "1A23B456CD".

8053
8054
8055
8056
8057
8058

Up to this point, the Alphanumeric subsection contains the 13-bit sequence '0 00 0100100011'. From Annex K, it can be determined that lengths of the two final bit sequences (encoding the Base-10 and Base-30 components of the source data string) are 20 bits (for the six digits) and 20 bits (for the four uppercase letters using Base 30). The six digits of the source data string "1A23B456CD" are "123456", which encodes to a 20-bit sequence of:
"00011110001001000000"

8059

which is appended to the end of the 13-bit sequence cited at the start of this paragraph.

8060
8061
8062

The four non-digits of the source data string are "ABCD", which are converted (using Table K-1) to a sequence of four Base-30 values 1, 2, 3, and 4 (denoted as values v₃ through v₀ in the formula below). These are then converted to a single Binary value, using the following formula:

8063

$$\square \quad 30^3 * v_3 + 30^2 * v_2 + 30^1 * v_1 + 30^0 * v_0$$

8064
8065
8066
8067
8068

In this example, the formula calculates as (27000 * 1 + 900 * 2 + 30 * 3 + 1 * 4) which is equal to 070DE (hexadecimal) encoded as the 20-bit sequence "00000111000011011110" which is appended to the end of the previous 20-bit sequence. Thus, the AlphaNumeric section contains a total of (13 + 20 + 20) or 53 bits, appended immediately after the previous 83 bits, for a grand total of 136 significant bits in the Packed Object.

8069 The final encoding step is to calculate the full length of the Packed Object (to encode the EBV-6
 8070 within the Length Section) and to pad-out the last byte (if necessary). Dividing 136 by eight shows
 8071 that a total of 17 bytes are required to hold the Packed Object, and that no pad bits are required in
 8072 the last byte. Thus, the EBV-6 portion of the Length Section is "010001", where this EBV-6 value
 8073 indicates 17 bytes in the Object. Following that, the Pad Indicator bit is set to '0' indicating that no
 8074 padding bits are present in the last data byte.

8075 The complete encoding process may be summarised as follows:

8076 Original input: (7)061031(32)978123456(1)1A23B456CD

8077 Re-ordered as: (7)061031(1)1A23B456CD(32)978123456

8078
 8079 FORMAT FLAGS SECTION: (empty)

8080 OBJECT INFO SECTION:

8081 ebvObjectLen: 010001

8082 paddingPresent: 0

8083 ebvNumIDs: 001

8084 IDvals: 1111101 0110011

8085 SECONDARY ID SECTION:

8086 IDbits: 0010

8087 AUX FORMAT SECTION:

8088 auxFormatbits: 1 0101

8089 DATA SECTION:

8090 KLn timeric: 0000 11101110 01100111 111010 01001100 11111010 11000000

8091 ANheader: 0

8092 ANprefix: 0

8093 ANsuffix: 0

8094 ANmap: 01 00100011

8095 ANdigitVal: 0001 11100010 01000000

8096 ANnonDigitsVal: 0000 01110000 11011110

8097 Padding: none

8098 Total Bits in Packed Object: 136; when byte aligned: 136

8099 Output as: 44 7E B3 2A 87 73 3F 49 9F 58 01 23 1E 24 00 70 DE

8100 Table L-1 shows the relevant subset of a hypothetical ID Table for a hypothetical ISO-registered
 8101 Data Format 99.

8102 **Table J.4.1-1** hypothetical Base ID Table, for the example in Annex L

| K-Version = 1.0 | | | |
|----------------------------------|------|------------|--------------|
| K-TableID = F99B0 | | | |
| K-RootOID = urn:oid:1.0.15961.99 | | | |
| K-IDsize = 128 | | | |
| IDvalue | OIDs | Data Title | FormatString |
| 3 | 1 | BATCH/LOT | 1*20an |

| | | | |
|--------------------|----------|--------------------|---------------|
| K-Version = 1.0 | | | |
| 8 | 7 | USE BY OR EXPIRY | 6n |
| 51 | 3%x30-39 | AMOUNT | 4*18n |
| 125 | (7) (1) | EXPIRY + BATCH/LOT | (6n) (1*20an) |
| | | | |
| K-TableEnd = F99B0 | | | |

8103 M Decoding Packed Objects (non-normative)

8104 M.1 Overview

8105 The decode process begins by decoding the first byte of the memory as a DSFID. If the leading two
 8106 bits indicate the Packed Objects access method, then the remainder of this Annex applies. From the
 8107 remainder of the DSFID octet or octets, determine the Data Format, which shall be applied as the
 8108 default Data Format for all of the Packed Objects in this memory. From the Data Format, determine
 8109 the default ID Table which shall be used to process the ID Values in each Packed Object.

8110 Typically, the decoder takes a first pass through the initial ID Values list, as described earlier, in
 8111 order to complete the list of identifiers. If the decoder finds any identifiers of interest in a Packed
 8112 Object (or if it has been asked to report back all the data strings from a tag's memory), then it will
 8113 need to record the implied fixed lengths (from the ID table) and the encoded variable lengths (from
 8114 the Aux Format subsection), in order to parse the Packed Object's compressed data. The decoder,
 8115 when recording any variable-length bit patterns, must first convert them to variable string lengths
 8116 per the table (for example, a three-bit pattern may indicate a variable string length in the range of
 8117 two to nine).

8118 Starting at the first byte-aligned position after the end of the DSFID, parse the remaining memory
 8119 contents until the end of encoded data, repeating the remainder of this section until a Terminating
 8120 Pattern is reached.

8121 Determine from the leading bit pattern (see [I.4](#)) which one of the following conditions applies:

- 8122 1. there are no further Packed Objects in Memory (if the leading 8-bit pattern is all zeroes, this
 8123 indicates the Terminating Pattern)
- 8124 2. one or more Padding bytes are present. If padding is present, skip the padding bytes, which are
 8125 as described in Annex [I](#), and examine the first non-pad byte.
- 8126 3. a Directory Pointer is encoded. If present, record the offset indicated by the following bytes, and
 8127 then continue examining from the next byte in memory
- 8128 4. a Format Flags section is present, in which case process this section according to the format
 8129 described in Annex [I](#)
- 8130 5. a default-format Packed Object begins at this location

8131 If the Packed Object had a Format Flags section, then this section may indicate that the Packed
 8132 Object is of the ID Map format, otherwise it is of the ID List format. According to the indicated
 8133 format, parse the Object Information section to determine the Object Length and ID information
 8134 contained in the Packed Object. See Annex [I](#) for the details of the two formats. Regardless of the
 8135 format, this step results in a known Object length (in bits) and an ordered list of the ID Values
 8136 encoded in the Packed Object. From the governing ID Table, determine the list of characteristics for
 8137 each ID (such as the presence and number of Secondary ID bits).

8138 Parse the Secondary ID section of the Object, based on the number of Secondary ID bits invoked by
 8139 each ID Value in sequence. From this information, create a list of the fully-qualified ID Values
 8140 (FQIDVs) that are encoded in the Packed Object.

8141 Parse the Aux Format section of the Object, based on the number of Aux Format bits invoked by
 8142 each FQIDV in sequence.

8143 Parse the Data section of the Packed Object:

- 8144 1. If one or more of the FQIDVs indicate all-numeric data, then the Packed Object's Data section
 8145 contains a Known-Length Numeric subsection, wherein the digit strings of these all-numeric
 8146 items have been encoded as a series of binary quantities. Using the known length of each of
 8147 these all-numeric data items, parse the correct numbers of bits for each data item, and convert
 8148 each set of bits to a string of decimal digits.
- 8149 2. If (after parsing the preceding sections) one or more of the FQIDVs indicate alphanumeric data,
 8150 then the Packed Object's Data section contains an AlphaNumeric subsection, wherein the
 8151 character strings of these alphanumeric items have been concatenated and encoded into the
 8152 structure defined in Annex [I](#). Decode this data using the "Decoding Alphanumeric data"
 8153 procedure outlined below.

- 8154
- 8155
- 8156
- 8157
- 8158
- 8159
- 8160
3. For each FQIDV in the decoded sequence:
 4. convert the FQIDV to an OID, by appending the OID string defined in the registered format's ID Table to the root OID string defined in that ID Table (or to the default Root OID, if none is defined in the table)
 5. Complete the OID/Value pair by parsing out the next sequence of decoded characters. The length of this sequence is determined directly from the ID Table (if the FQIDV is specified as fixed length) or from a corresponding entry encoded within the Aux Format section.

8161 **M.2 Decoding alphanumeric data**

8162 Within the Alphanumeric subsection of a Packed Object, the total number of data characters is not
 8163 encoded, nor is the bit length of the character map, nor are the bit lengths of the succeeding Binary
 8164 sections (representing the numeric and non-numeric Binary values). As a result, the decoder must
 8165 follow a specific procedure in order to correctly parse the AlphaNumeric section.

8166 When decoding the A/N subsection using this procedure, the decoder will first count the number of
 8167 non-bitmapped values in each base (as indicated by the various Prefix and Suffix Runs), and (from
 8168 that count) will determine the number of bits required to encode these numbers of values in these
 8169 bases. The procedure can then calculate, from the remaining number of bits, the number of
 8170 explicitly-encoded character map bits. After separately decoding the various binary fields (one field
 8171 for each base that was used), the decoder "re-interleaves" the decoded ASCII characters in the
 8172 correct order.

8173 The A/N subsection decoding procedure is as follows:

- 8174
- 8175
- 8176
- 8177
- 8178
- 8179
- 8180
- 8181
- 8182
- 8183
- 8184
- 8185
- 8186
- 8187
- 8188
- 8189
- 8190
- 8191
- 8192
- 8193
- 8194
- 8195
- 8196
- 8197
- 8198
- 8199
- 8200
- Determine the total number of non-pad bits in the Packed Object, as described in section [1.8.2](#)
 - Keep a count of the total number of bits parsed thus far, as each of the subsections prior to the Alphanumeric subsection is processed
 - Parse the initial Header bits of the Alphanumeric subsection, up to but not including the Character Map, and add this number to previous value of TotalBitsParsed.
 - Initialise a DigitsCount to the total number of base-10 values indicated by the Prefix and Suffix (which may be zero)
 - Initialise an ExtDigitsCount to the total number of base-13 values indicated by the Prefix and Suffix (which may be zero)
 - Initialise a NonDigitsCount to the total number of base-30, base 74, or base-256 values indicated by the Prefix and Suffix (which may be zero)
 - Initialise an ExtNonDigitsCount to the total number of base-40 or base 84 values indicated by the Prefix and Suffix (which may be zero)
 - Calculate Extended-base Bit Counts: Using the tables in Annex [K](#), calculate two numbers:
 - ExtDigitBits, the number of bits required to encode the number of base-13 values indicated by ExtDigitsCount, and
 - ExtNonDigitBits, the number of bits required to encode the number of base-40 (or base-84) values indicated by ExtNonDigitsCount
 - Add ExtDigitBits and ExtNonDigitBits to TotalBitsParsed
 - Create a PrefixCharacterMap bit string, a sequence of zero or more quad-base character-map pairs, as indicated by the Prefix bits just parsed. Use quad-base bit pairs defined as follows:
 - '00' indicates a base 10 value;
 - '01' indicates a character encoded in Base 13;
 - '10' indicates the non-numeric base that was selected earlier in the A/N header, and
 - '11' indicates the Extended version of the non-numeric base that was selected earlier
 - Create a SuffixCharacterMap bit string, a sequence of zero or more quad-base character-map pairs, as indicated by the Suffix bits just parsed.

- 8201
8202
- Initialise the FinalCharacterMap bit string and the MainCharacterMap bit string to an empty string
- 8203
- **Calculate running Bit Counts:** Using the tables in Annex B, calculate two numbers:
 - DigitBits, the number of bits required to encode the number of base-10 values currently indicated by DigitsCount, and
 - NonDigitBits, the number of bits required to encode the number of base-30 (or base 74 or base-256) values currently indicated by NonDigitsCount
- 8206
8207
- set AlnumBits equal to the sum of DigitBits plus NonDigitBits
- 8208
- if the sum of TotalBitsParsed and AlnumBits equals the total number of non-pad bits in the Packed Object, then no more bits remain to be parsed from the character map, and so the remaining bit patterns, representing Binary values, are ready to be converted back to extended base values and/or base 10/base 30/base 74/base-256 values (skip to the **Final Decoding** steps below). Otherwise, get the next encoded bit from the encoded Character map, convert the bit to a quad-base bit-pair by converting each '0' to '00' and each '1' to '10', append the pair to the end of the MainCharacterMap bit string, and:
 - If the encoded map bit was '0', increment DigitsCount,
 - Else if '1', increment NonDigitsCount
 - Loop back to the **Calculate running Bit Counts** step above and continue
- 8216
8217
8218
- **Final decoding steps:** once the encoded Character Map bits have been fully parsed:
 - Fetch the next set of zero or more bits, whose length is indicated by ExtDigitBits. Convert this number of bits from Binary values to a series of base 13 values, and store the resulting array of values as ExtDigitVals.
 - Fetch the next set of zero or more bits, whose length is indicated by ExtNonDigitBits. Convert this number of bits from Binary values to a series of base 40 or base 84 values (depending on the selection indicated in the A/N Header), and store the resulting array of values as ExtNonDigitVals.
 - Fetch the next set of bits, whose length is indicated by DigitBits. Convert this number of bits from Binary values to a series of base 10 values, and store the resulting array of values as DigitVals.
 - Fetch the final set of bits, whose length is indicated by NonDigitBits. Convert this number of bits from Binary values to a series of base 30 or base 74 or base 256 values (depending on the value of the first bits of the Alphanumeric subsection), and store the resulting array of values as NonDigitVals.
 - Create the FinalCharacterMap bit string by copying to it, in this order, the previously-created PrefixCharacterMap bit string, then the MainCharacterMap string, and finally append the previously-created SuffixCharacterMap bit string to the end of the FinalCharacterMap string.
 - Create an interleaved character string, representing the concatenated data strings from all of the non-numeric data strings of the Packed Object, by parsing through the FinalCharacterMap, and:
 - For each '00' bit-pair encountered in the FinalCharacterMap, copy the next value from DigitVals to InterleavedString (add 48 to each value to convert to ASCII);
 - For each '01' bit-pair encountered in the FinalCharacterMap, fetch the next value from ExtDigitVals, and use Table K-2 to convert that value to ASCII (or, if the value is a Base 13 shift, then increment past the next '01' pair in the FinalCharacterMap, and use that Base 13 shift value plus the next Base 13 value from ExtDigitVals to convert the pair of values to ASCII). Store the result to InterleavedString;
 - For each '10' bit-pair encountered in the FinalCharacterMap, get the next character from NonDigitVals, convert its base value to an ASCII value using Annex K, and store the resulting ASCII value into InterleavedString. Fetch and process an additional Base 30 value for every Base 30 Shift values encountered, to create and store a single ASCII character.
- 8219
- 8220
8221
8222
- 8223
8224
8225
8226
- 8227
8228
8229
- 8230
8231
8232
8233
- 8234
8235
8236
- 8237
8238
8239
- 8240
8241
- 8242
8243
8244
8245
8246
- 8247
8248
8249
8250

8251
8252
8253

- For each '11' bit-pair encountered in the FinalCharacterMap, get the next character from ExtNonDigitVals, convert its base value to an ASCII value using Annex [K](#), and store the resulting ASCII value into InterleavedString, processing any Shifts as previously described.

8254
8255
8256
8257
8258

Once the full FinalCharacterMap has been parsed, the InterleavedString is completely populated. Starting from the first AlphaNumeric entry on the ID list, copy characters from the InterleavedString to each such entry, ending each copy operation after the number of characters indicated by the corresponding Aux Format length bits, or at the end of the InterleavedString, whichever comes first.