



EPCglobal Tag Data Standards Version 1.3

Ratified Specification

March 8, 2006

Copyright Notice

© 2006, EPCglobal Inc.

All rights reserved. Unauthorized reproduction, modification, and/or use of this document is not permitted. Requests for permission to reproduce should be addressed to epcglobal@epcglobalinc.org.

Disclaimer

EPCglobal Inc.TM is providing this document as a service to interested industries. This document was developed through a consensus process of interested parties. Although efforts have been to assure that the document is correct, reliable, and technically accurate, EPCglobal Inc. makes NO WARRANTY, EXPRESS OR IMPLIED, THAT THIS DOCUMENT IS CORRECT, WILL NOT REQUIRE MODIFICATION AS EXPERIENCE AND TECHNOLOGICAL ADVANCES DICTATE, OR WILL BE SUITABLE FOR ANY PURPOSE OR WORKABLE IN ANY APPLICATION, OR OTHERWISE. Use of this document is with the understanding that EPCglobal Inc. has no liability for any claim to the contrary, or for any damage or loss of any kind or nature.

24
25

26
27
28
29

28
29

30

31

32

33

Abstract

This document defines the EPC Tag Data Standards version 1.3. It applies to RFID tags conforming to “EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz-960MHz Version 1.0.9” (“Gen2 Specification”). Such tags will be referred to as “Gen 2 Tags” in the remainder of this document. These standards define completely that portion of EPC tag data that is standardized, including how that data is encoded on the EPC tag itself (i.e. the EPC Tag Encodings), as well as how it is encoded for use in the information systems layers of the EPC Systems Network (i.e. the EPC URI or Uniform Resource Identifier Encodings).

The EPC Tag Encodings include a Header field followed by one or more Value Fields. The Header field defines the overall length and format of the Values Fields. The Value Fields contain a unique EPC Identifier and a required Filter Value when the latter is judged to be important to encode on the tag itself.

The EPC URI Encodings provide the means for applications software to process EPC Tag Encodings either literally (i.e. at the bit level) or at various levels of semantic abstraction that is independent of the tag variations. This document defines four categories of URI:

1. URIs for pure identities, sometimes called “canonical forms.” These contain only the unique information that identifies a specific physical object, location or organization, and are independent of tag encodings.
2. URIs that represent specific tag encodings. These are used in software applications where the encoding scheme is relevant, as when commanding software to write a tag.
3. URIs that represent patterns, or sets of EPCs. These are used when instructing software how to filter tag data.
4. URIs that represent raw tag information, generally used only for error reporting purposes.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at EPCglobal. This document is the Ratified Specification named Tag Data Standards Version 1.3 as ratified by the EPCglobal Board of Governors on March 8, 2006. This version contains ratified content but is not finalized for web publication, that should occur in April of 2006. Comments on this document should be sent to epcinfo@epcglobalinc.org.

Changes from Previous Versions

This Tag Data Standards Version 1.3 is aimed for use in Gen 2 Tags, whereas the previous Version 1.1, was aimed for use in UHF Class 1 Generation 1 tags. Version 1.3 maintains compatibility with version 1.1 in the identity level. In other words, this version will continue to support the EAN.UCC system and DoD identity types.

However, in Version 1.3, there are significant changes to prior versions, including:

1. The deprecation of 64 bit encodings.
2. The elimination of tiered header rules.
3. The encoding of EPC to fit the structure of Gen 2 Tags
4. The addition of the Extension Component to the SGLN
5. Addition of SGTIN-198, SGLN-195, GRAI-170, GIAI-202 and corresponding changes in URI expression for alpha-numeric serial number encoding.

Table of Contents

1	Introduction	8
2	Identity Concepts.....	9
2.1	Pure Identities	10
2.1.1	General Types	10
2.1.2	EAN.UCC System Identity Types	11
2.1.2.1	Serialized Global Trade Item Number (SGTIN).....	12
2.1.2.2	Serial Shipping Container Code (SSCC)	13
2.1.2.3	Serialized Global Location Number (SGLN).....	14
2.1.2.4	Global Returnable Asset Identifier (GRAI)	16
2.1.2.5	Global Individual Asset Identifier (GIAI).....	16
2.1.3	DoD Identity Type	17
3	EPC Tag Bit-level Encodings	17
3.1	Headers	18
3.2	Use of EPCs on UHF Class 1 Generation 2 Tags.....	20
3.2.1	EPC Memory Contents	20
3.2.2	The Length Bits.....	22
3.3	Notational Conventions	22
3.4	General Identifier (GID-96).....	23
3.4.1.1	GID-96 Encoding Procedure	24
3.4.1.2	GID-96 Decoding Procedure.....	25
3.5	Serialized Global Trade Item Number (SGTIN)	25
3.5.1	SGTIN-96	25
3.5.1.1	SGTIN-96 Encoding Procedure	27
3.5.1.2	SGTIN-96 Decoding Procedure	28
3.5.2	SGTIN-198	29
3.5.2.1	SGTIN-198 Encoding Procedure	30
3.5.2.2	SGTIN-198 Decoding Procedure	31
3.6	Serial Shipping Container Code (SSCC).....	32
3.6.1	SSCC-96	32
3.6.1.1	SSCC-96 Encoding Procedure	34

114	3.6.1.2	SSCC-96 Decoding Procedure	34
115	3.7	Serialized Global Location Number (SGLN)	35
116	3.7.1	SGLN-96	36
117	3.7.1.1	SGLN-96 Encoding Procedure	38
118	3.7.1.2	SGLN-96 Decoding Procedure	38
119	3.7.2	SGLN-195	39
120	3.7.2.1	SGLN-195 Encoding Procedure	40
121	3.7.2.2	SGLN-195 Decoding Procedure	41
122	3.8	Global Returnable Asset Identifier (GRAI)	42
123	3.8.1	GRAI-96	42
124	3.8.1.1	GRAI-96 Encoding Procedure	44
125	3.8.1.2	GRAI-96 Decoding Procedure	45
126	3.8.2	GRAI-170	46
127	3.8.2.1	GRAI-170 Encoding Procedure	47
128	3.8.2.2	GRAI-170 Decoding Procedure	47
129	3.9	Global Individual Asset Identifier (GIAI)	48
130	3.9.1	GIAI-96	48
131	3.9.1.1	GIAI-96 Encoding Procedure	50
132	3.9.1.2	GIAI-96 Decoding Procedure	51
133	3.9.2	GIAI-202	51
134	3.9.2.1	GIAI-202 Encoding Procedure	53
135	3.9.2.2	GIAI-202 Decoding Procedure	54
136	3.10	DoD Tag Data Constructs	54
137	3.10.1	DoD-96	54
138	4	URI Representation	55
139	4.1	URI Forms for Pure Identities	56
140	4.2	URI Forms for Related Data Types	58
141	4.2.1	URIs for EPC Tags	58
142	4.2.2	URIs for Raw Bit Strings Arising From Invalid Tags	59
143	4.2.2.1	Use of the Raw URI with Gen 2 Tags	60
144	4.2.2.2	The Length Field of a Raw URI when using Gen 2 Tags (non-normative)	60
145	4.2.3	URIs for EPC Patterns	61
146	4.2.4	URIs for EPC Pure Identity Patterns	62

147	4.3	Syntax	62
148	4.3.1	Common Grammar Elements	62
149	4.3.2	EPCGID-URI	63
150	4.3.3	SGTIN-URI	63
151	4.3.4	SSCC-URI	64
152	4.3.5	SGLN-URI	64
153	4.3.6	GRAI-URI	64
154	4.3.7	GIAI-URI	64
155	4.3.8	EPC Tag URI	65
156	4.3.9	Raw Tag URI	66
157	4.3.10	EPC Pattern URI	66
158	4.3.11	EPC Identity Pattern URI	67
159	4.3.12	DoD Construct URI	67
160	4.3.13	Summary (non-normative)	68
161	5	Translation between EPC-URI and Other EPC Representations	71
162	5.1	Bit string into EPC-URI (pure identity)	71
163	5.2	Bit String into Tag or Raw URI	73
164	5.3	Gen 2 Tag EPC Memory into EPC-URI (pure identity)	76
165	5.4	Gen 2 Tag EPC Memory into Tag or Raw URI	76
166	5.5	URI into Bit String	77
167	5.6	URI into Gen 2 Tag EPC Memory	80
168	6	Semantics of EPC Pattern URIs	81
169	7	Background Information (non-normative)	82
170	8	References	82
171	9	Appendix A: Encoding Scheme Summary Tables (non-normative)	83
172	10	Appendix B: TDS 1.3 EAN.UCC Identities Bit Allocation and Required Physical Tag	
173		Bit Length for Encoding (non-normative)	88
174	11	Appendix C: Example of a Specific Trade Item <SGTIN> (non-normative)	90
175	12	Appendix D: Decimal values of powers of 2 Table (non-normative)	93
176	13	Appendix E: List of Abbreviations	94
177	14	Appendix F: General EAN.UCC Specifications (non-normative)	95
178	15	Appendix G: EAN.UCC Alphanumeric Character Set	96
179			

1 Introduction

The Electronic Product Code™ (EPC™) is an identification scheme for universally identifying physical objects via Radio Frequency Identification (RFID) tags and other means. The standardized EPC Tag Encodings consists of an EPC (or EPC Identifier) that uniquely identifies an individual object, as well as a Filter Value when judged to be necessary to enable effective and efficient reading of the EPC tags.

The EPC Identifier is a meta-coding scheme designed to support the needs of various industries by accommodating both existing coding schemes where possible and defining new schemes where necessary. The various coding schemes are referred to as Domain Identifiers, to indicate that they provide object identification within certain domains such as a particular industry or group of industries. As such, the Electronic Product Code represents a family of coding schemes (or “namespaces”) and a means to make them unique across all possible EPC-compliant tags. These concepts are depicted in the chart below.

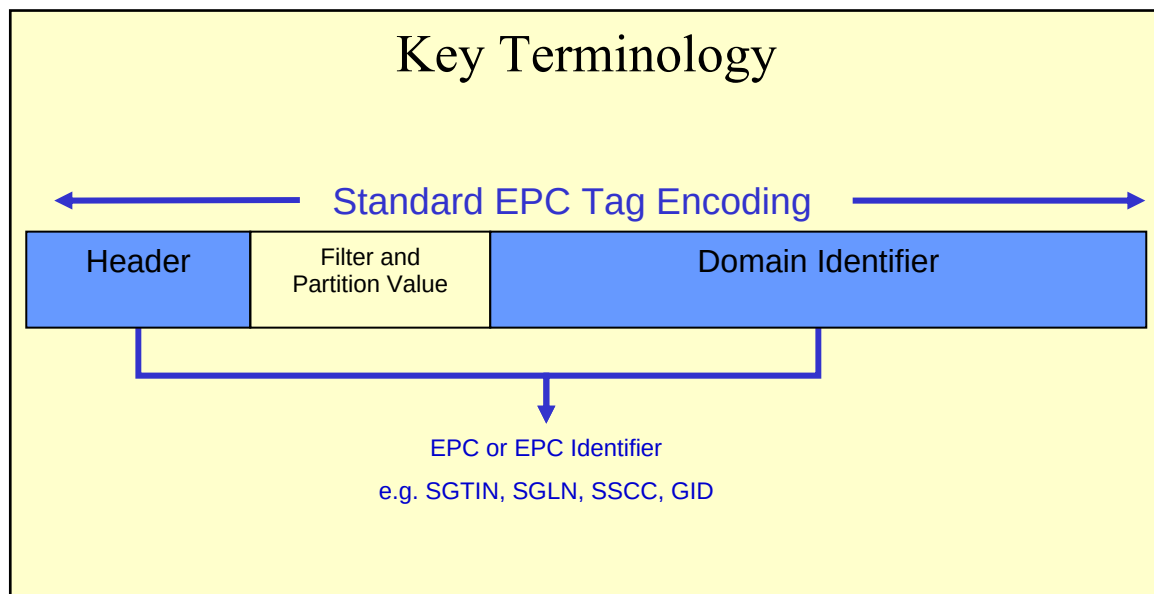


Figure A. EPC Terminology

In this version of the EPC – EPC Version 1.3 – the specific coding schemes include a General Identifier (GID), a serialized version of the EAN.UCC Global Trade Item Number (GTIN®), the EAN.UCC Serial Shipping Container Code (SSCC®), the EAN.UCC Global Location Number (GLN®), the EAN.UCC Global Returnable Asset Identifier (GRAI®), the EAN.UCC Global Individual Asset Identifier (GIAI®) and the DOD Construct.

In the following sections, we will describe the structure and organization of the EPC and provide illustrations to show its recommended use.

The EPCglobal Tag Data Standard V1.3 has been approved by GS1 with the restrictions outlined in the General EAN.UCC Specifications Section 3.7, which is excerpted into Tag Data Standard Appendix F.

The latest version of this specification can be [obtained](#) from EPCglobal.

2 Identity Concepts

To better understand the overall framework of the EPC Tag Data Standards, it's helpful to distinguish between three levels of identification (See Figure B). Although this specification addresses the pure identity and encoding layers in detail, all three layers are described below to explain the layer concepts and the context for the encoding layer.

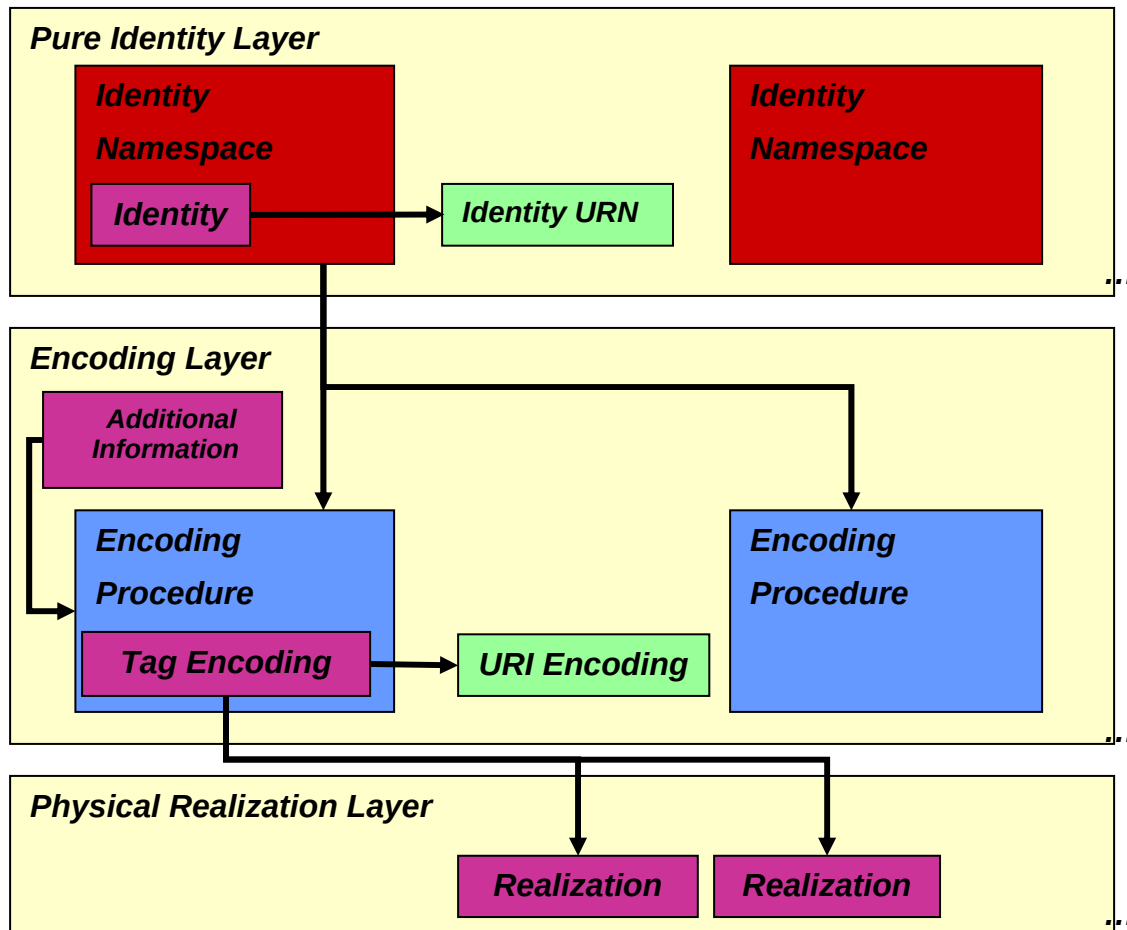


Figure B. Defined Identity Namespaces, Encodings, and Realizations.

- Pure identity -- the identity associated with a specific physical or logical entity, independent of any particular encoding vehicle such as an RF tag, bar code or database field. As such, a pure identity is an abstract name or number used to identify an entity. A pure identity consists of the information required to uniquely identify a specific entity, and no more.
- Identity URI -- a representation of a pure identity as a Uniform Resource Identifier (URI). A URI is a character string representation that is commonly used to exchange identity data between software components of a larger system.
- Encoding -- a pure identity, together with additional information such as filter value, rendered into a specific syntax (typically consisting of value fields of specific sizes). A

given pure identity may have a number of possible encodings, such as a Barcode Encoding, various Tag Encodings, and various URI Encodings. Encodings may also incorporate additional data besides the identity (such as the Filter Value used in some encodings), in which case the encoding scheme specifies what additional data it can hold.

- Physical Realization of an Encoding -- an encoding rendered in a concrete implementation suitable for a particular machine-readable form, such as a specific kind of RF tag or specific database field. A given encoding may have a number of possible physical realizations.

For example, the Serial Shipping Container Code (SSCC) format as defined by the EAN.UCC System is an example of a pure identity. An SSCC encoded into the EPC-SSCC 96-bit format is an example of an encoding. That 96-bit encoding, written onto a UHF Class 1 RF Tag, is an example of a physical realization.

A particular encoding scheme may implicitly impose constraints on the range of identities that may be represented using that encoding. In general, each encoding scheme specifies what constraints it imposes on the range of identities it can represent.

Conversely, a particular encoding scheme may accommodate values that are not valid with respect to the underlying pure identity type, thereby requiring an explicit constraint. For example, the EPC-SSCC 96-bit encoding provides 24 bits to encode a 7-digit company prefix. In a 24-bit field, it is possible to encode the decimal number 10,000,001, which is longer than 7 decimal digits. Therefore, this does not represent a valid SSCC, and is forbidden. In general, each encoding scheme specifies what limits it imposes on the value that may appear in any given encoded field.

2.1 Pure Identities

This section defines the pure identity types for which this document specifies encoding schemes.

2.1.1 General Types

This version of the EPC Tag Data Standards defines one general identity type. The *General Identifier (GID-96)* is independent of any known, existing specifications or identity schemes. The General Identifier is composed of three fields - the *General Manager Number*, *Object Class* and *Serial Number*. Encodings of the GID include a fourth field, the header, to guarantee uniqueness in the EPC namespace.

The *General Manager Number* identifies an organizational entity (essentially a company, manager or other organization) that is responsible for maintaining the numbers in subsequent fields – Object Class and Serial Number. EPCglobal assigns the General Manager Number to an entity, and ensures that each General Manager Number is unique.

The *Object Class* is used by an EPC managing entity to identify a class or “type” of thing. These object class numbers, of course, must be unique within each General Manager Number domain. Examples of Object Classes could include case Stock Keeping Units of

consumer-packaged goods or different structures in a highway system, like road signs, lighting poles, and bridges, where the managing entity is a County.

Finally, the *Serial Number* code, or serial number, is unique within each object class. In other words, the managing entity is responsible for assigning unique, non-repeating serial numbers for every instance within each object class.

2.1.2 EAN.UCC System Identity Types

This version of the EPC Tag Data Standards defines five EPC identity types derived from the EAN.UCC System family of product codes, each described in the subsections below.

The rules regarding the usage of the EAN.UCC codes can be found in the General Specifications of EAN.UCC. This document only explains the incorporation of these numbers in EPC tags.

EAN.UCC System codes have a common structure, consisting of a fixed number of decimal digits that encode the identity, plus one additional “check digit” which is computed algorithmically from the other digits. Within the non-check digits, there is an implicit division into two fields: a Company Prefix assigned by GS1 to a managing entity, and the remaining digits, which are assigned by the managing entity. (The digits apart from the Company Prefix are called by a different name by each of the EAN.UCC System codes.) The number of decimal digits in the Company Prefix varies from 6 to 12 depending on the particular Company Prefix assigned. The number of remaining digits therefore varies inversely so that the total number of digits is fixed for a particular EAN.UCC System code type.

The GS1 recommendations for the encoding of EAN.UCC System identities into bar codes, as well as for their use within associated data processing software, stipulate that the digits comprising a EAN.UCC System code should always be processed together as a unit, and not parsed into individual fields. This recommendation, however, is not appropriate within the EPC Network, as the ability to divide a code into the part assigned to the managing entity (the Company Prefix in EAN.UCC System types) versus the part that is managed by the managing entity (the remainder) is essential to the proper functioning of the Object Name Service (ONS). In addition, the ability to distinguish the Company Prefix is believed to be useful in filtering or otherwise securing access to EPC-derived data. Hence, the EPC Tag Encodings for EAN.UCC code types specified herein deviate from the aforementioned recommendations in the following ways:

- EPC Tag Encodings carry an explicit division between the Company Prefix and the remaining digits, with each individually encoded into binary. Hence, converting from the traditional decimal representation of an EAN.UCC System code and an EPC Tag Encoding requires independent knowledge of the length of the Company Prefix.
- EPC Tag Encodings do not include the check digit. Hence, converting from an EPC Tag Encoding to a traditional decimal representation of a code requires that the check digit be recalculated from the other digits.

2.1.2.1 Serialized Global Trade Item Number (SGTIN)

The Serialized Global Trade Item Number is a new identity type based on the EAN.UCC Global Trade Item Number (GTIN) code defined in the General EAN.UCC Specifications. A GTIN by itself does not fit the definition of an EPC pure identity, because it does not uniquely identify a single physical object. Instead, a GTIN identifies a particular class of object, such as a particular kind of product or SKU.

All representations of SGTIN support the full 14-digit GTIN format. This means that the zero indicator-digit and leading zero in the Company Prefix for UCC-12, and the zero indicator-digit for EAN.UCC-13, can be encoded and interpreted accurately from an EPC Tag Encoding. EAN.UCC-8 is not currently supported in EPC, but would be supported in full 14-digit GTIN format as well.

To create a unique identifier for individual objects, the GTIN is augmented with a serial number, which the managing entity is responsible for assigning uniquely to individual object classes. The combination of GTIN and a unique serial number is called a Serialized GTIN (SGTIN).

The SGTIN consists of the following information elements:

- The *Company Prefix*, assigned by GS1 to a managing entity. The Company Prefix is the same as the Company Prefix digits within an EAN.UCC GTIN decimal code.
- The *Item Reference*, assigned by the managing entity to a particular object class. The Item Reference for the purposes of EPC Tag Encoding is derived from the GTIN by concatenating the Indicator Digit of the GTIN and the Item Reference digits, and treating the result as a single integer.
- The *Serial Number*, assigned by the managing entity to an individual object. The serial number is not part of the GTIN code, but is formally a part of the SGTIN.

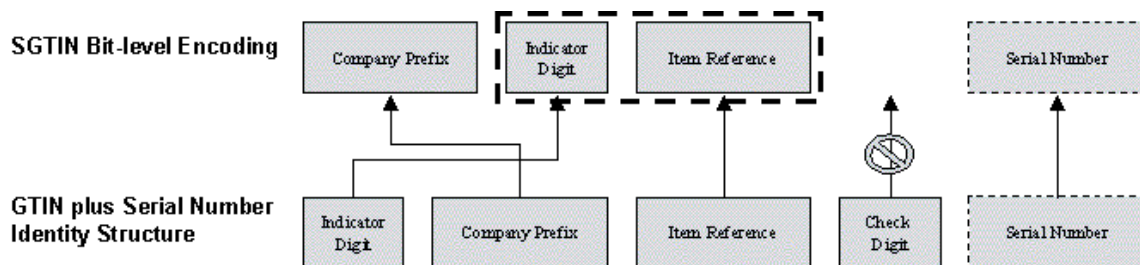


Figure C. How the parts of the decimal SGTIN are extracted, rearranged, and augmented for encoding.

The SGTIN is not explicitly defined in the EAN.UCC General Specifications. However, it may be considered equivalent to a EAN.UCC-128 bar code that contains both a GTIN (Application Identifier 01) and a serial number (Application Identifier 21). Serial numbers in AI 21 consist of one to twenty characters, where each character can be a digit, uppercase or lowercase letter, or one of a number of allowed punctuation characters. The complete set of

characters allowed is illustrated in Appendix G. The complete AI 21 syntax is supported by the pure identity URI syntax specified in Section 4.3.1.

When representing serial numbers in 96-bit tags, however, only a subset of the serial numbers allowed in the General EAN.UCC Specifications for Application Identifier 21 are permitted. Specifically, the permitted serial numbers are those consisting of one or more digits with no leading zeros, and whose value when considered as an integer fits within the range restrictions of the 96-bit tag encodings.

While these limitations exist for 96-bit tag encodings, future tag encodings allow a wider range of serial numbers. Therefore, application authors and database designers should take the EAN.UCC specifications for Application Identifier 21 into account in order to accommodate further expansions of the Tag Data Standard.

For the requirement of using longer serial number, or alphabet and other non numeric codings allowed in Application Identifier 21, this version of specification introduces a longer bit encoding format SGTIN-198.

[illegible]

2.1.2.2 Serial Shipping Container Code (SSCC)

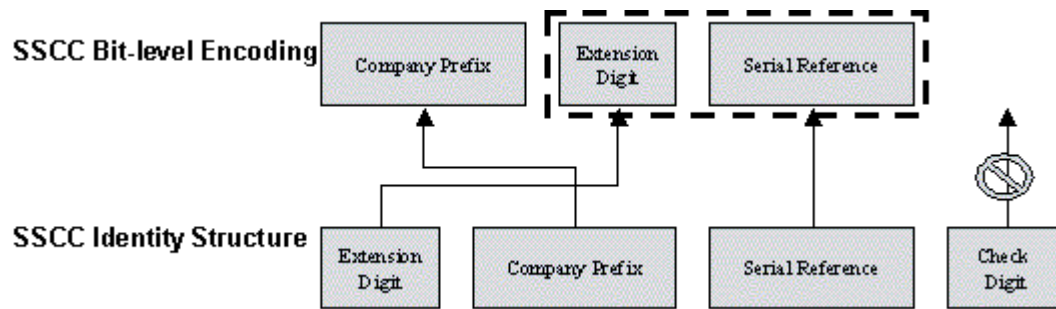
The Serial Shipping Container Code (SSCC) is defined by the General EAN.UCC Specifications. Unlike the GTIN, the SSCC is already intended for assignment to individual objects and therefore does not require any additional fields to serve as an EPC pure identity.

Note (Non-Normative): Many applications of SSCC have historically included the Application Identifier (00) in the SSCC identifier field when stored in a database. This is not a standard requirement, but a widespread practice. The Application Identifier is a sort of header used in bar code applications, and can be inferred directly from EPC headers representing SSCC. In other words, an SSCC EPC can be interpreted as needed to include the (00) as part of the SSCC identifier or not.

The SSCC consists of the following information elements:

- The *Company Prefix*, assigned by GS1 to a managing entity. The Company Prefix is the same as the Company Prefix digits within an EAN.UCC SSCC decimal code.
- The *Serial Reference*, assigned uniquely by the managing entity to a specific shipping unit. The Serial Reference for the purposes of EPC Tag Encoding is derived from the SSCC by concatenating the Extension Digit of the SSCC and the Serial Reference digits, and treating the result as a single integer.

377



378

379 **Figure D.** How the parts of the decimal SSCC are extracted and rearranged for encoding.

380 2.1.2.3 Serialized Global Location Number (SGLN)

381 The Global Location Number (GLN) is defined by the General EAN.UCC Specifications as
382 an identifier of physical or legal entities.

383 A GLN can represent either a discrete, unique physical location such as a dock door or a
384 warehouse slot, or an aggregate physical location such as an entire warehouse. In addition, a
385 GLN can represent a logical entity such as an “organization” that performs a business
386 function such as placing an order.

387 Within the GS1 system, high capacity data carriers use Application Identifiers (AI) to
388 distinguish data elements encoded within a single data carrier. The GLN can be associated
389 with many AI’s including physical location, ship to location, invoice to location etc.

390 Recognizing these variables, the EPC SGLN (serialized GLN) represents only the physical
391 location sub-type of GLN AI (414). The serial component is represented by the GLN
392 Extension AI (254). Rules regarding the allocation of a SGLN can be found within the
393 EAN.UCC General Specifications.

394 The SGLN consists of the following information elements:

- 395 • The *Company Prefix*, assigned by GS1 to a managing entity. The Company Prefix is the
396 same as the Company Prefix digits within an EAN.UCC GLN decimal code.
- 397 • The *Location Reference*, assigned uniquely by the managing entity to an aggregate or
398 specific physical location.
- 399 • The *GLN Extension*, assigned by the managing entity to an individual unique location.
 - 400 ➤ The use of the GLN Extension is intended for internal purposes. For communication
401 between trading partners a GLN will be used. The rules defining the use of the
402 SGLN are described in Section 3.7.

403 .

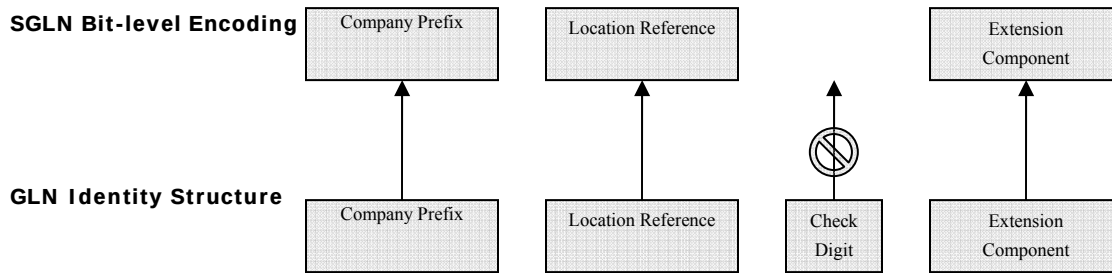


Figure E. How the parts of the decimal SGLN are extracted and rearranged for encoding

The SGLN is not explicitly defined in the EAN.UCC General Specifications. However, it may be considered equivalent to a EAN.UCC-128 bar code that contains both a GLN (Application Identifier 414) and an Extension Component (Application Identifier 254). Extension Components in AI 254 consist of one to twenty characters, where each character can be a digit, uppercase or lowercase letter, or one of a number of allowed punctuation characters. The complete set of characters allowed is illustrated in Appendix G. The complete AI 254 syntax is supported by the pure identity URI syntax specified in Section 4.3.1.

When representing Extension Components in 96-bit tags, however, only a subset of the Extension Component allowed in the General EAN.UCC Specifications for Application Identifier 254 is permitted. Specifically, the permitted Extension Component are those consisting of one or more digits characters, with no leading zeros, and whose value when considered as an integer fits within the range restrictions of the 96-bit tag encodings.

While these limitations exist for 96-bit tag encodings, future tag encodings allow a wider range of Extension Component. Therefore, application authors and database designers should take the EAN.UCC specifications for Application Identifier 254 into account in order to accommodate further expansions of the Tag Data Standard.

For the requirement of using a longer Extension Component, or alphabet and other non numeric codings allowed in Application Identifier 254, this version of specification introduces a longer bit encoding format SGLN-195.

Explanation (non-normative): The restrictions are necessary for 96-bit tags in order for the Extension Component to fit within the small number of bits available in earlier Class 1 Generation 1 tags. The Extension Component range is restricted to numeric values and an alphanumeric Extension Component is disallowed. Leading zeros are forbidden so that the Extension Component can be considered as a decimal integer when encoding the integer value in binary. By considering it to be a decimal integer, "00034", "034", or "34" (for example) can't be distinguished as different integer values. In order to insure that every encoded value can be decoded uniquely, Extension Components can't have leading zeros. Then, when the bits 0000000000000000000010010 occurs on the tag, the Extension Component as "34" (not "034" or "00034") is decoded.

2.1.2.4 Global Returnable Asset Identifier (GRAI)

The Global Returnable Asset Identifier (GRAI) is defined by the General EAN.UCC Specifications. Unlike the GTIN, the GRAI is already intended for assignment to individual objects and therefore does not require any additional fields to serve as an EPC pure identity.

The GRAI consists of the following information elements:

- The *Company Prefix*, assigned by GS1 to a managing entity. The Company Prefix is the same as the Company Prefix digits within an EAN.UCC GRAI decimal code.
- The *Asset Type*, assigned by the managing entity to a particular class of asset.
- The *Serial Number*, assigned by the managing entity to an individual object. The GRAI-96 representation is only capable of representing a subset of Serial Numbers allowed in the General EAN.UCC Specifications. Specifically, only those Serial Numbers consisting of one or more digits, with no leading zeros, are permitted [see Appendix F for details].
For the requirement of using longer serial number, or alphabet and other non numeric codings allowed in Application Identifier 8003, this version of specification introduces longer bit encoding format GRAI-170.

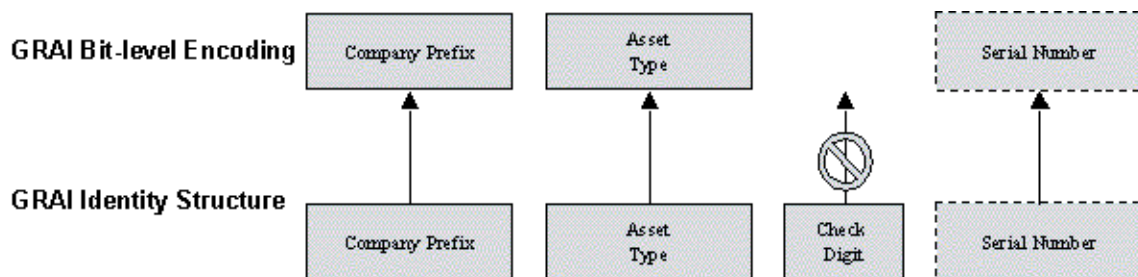


Figure F. How the parts of the decimal GRAI are extracted and rearranged for encoding.

2.1.2.5 Global Individual Asset Identifier (GIAI)

The Global Individual Asset Identifier (GIAI) is defined by the General EAN.UCC Specifications. Unlike the GTIN, the GIAI is already intended for assignment to individual objects and therefore does not require any additional fields to serve as an EPC pure identity.

The GIAI consists of the following information elements:

- The *Company Prefix*, assigned by GS1 to a managing entity. The Company Prefix is the same as the Company Prefix digits within an EAN.UCC GIAI decimal code.
- The *Individual Asset Reference*, assigned uniquely by the managing entity to a specific asset. The GIAI-96 representation is only capable of representing a subset of Individual Asset References allowed in the General EAN.UCC Specifications. Specifically, only those Individual Asset References consisting of one or more digits, with no leading zeros, are permitted.
For the requirement of using longer serial number, or alphabet and other non numeric

codings allowed in Application Identifier 8004, this version of specification introduces the longer bit encoding format GIAI-202.

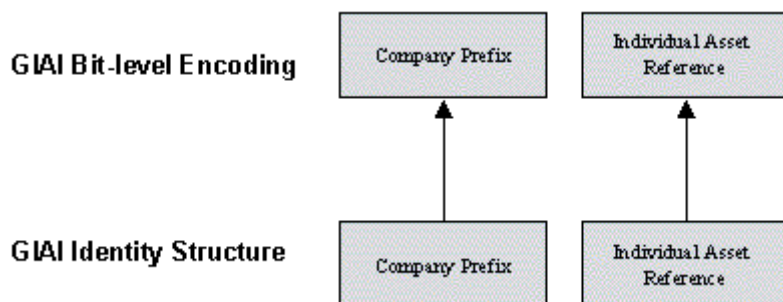


Figure G. How the parts of the decimal GIAI are extracted and rearranged for encoding.

2.1.3 DoD Identity Type

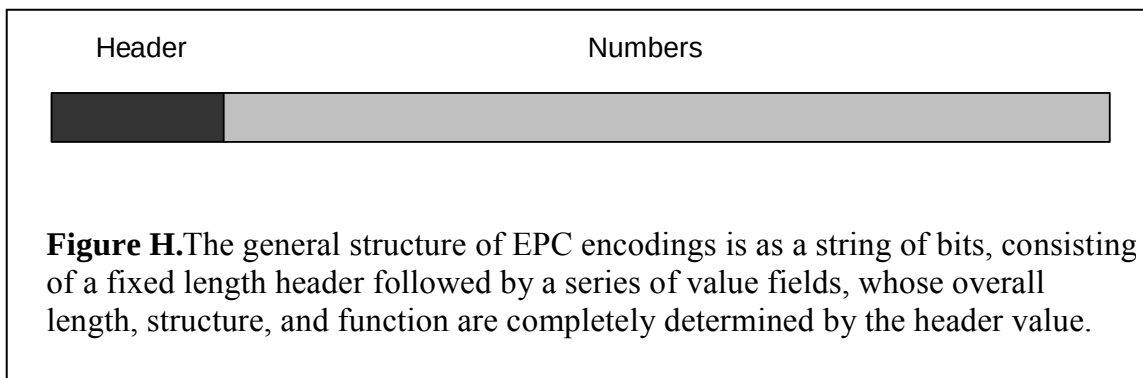
The DoD Construct identifier is defined by the United States Department of Defense.

This tag data construct may be used to encode 96-bit Class 1 tags for shipping goods to the United States Department of Defense by a supplier who has already been assigned a CAGE (Commercial and Government Entity) code.

At the time of this writing, the details of what information to encode into these fields is explained in a document titled "United States Department of Defense Supplier's Passive RFID Information Guide" that can be obtained at the United States Department of Defense's web site (<http://www.dodrfid.org/suppliarguide.htm>).

3 EPC Tag Bit-level Encodings

The general structure of EPC Tag Encodings on a tag is as a string of bits (i.e., a binary representation), consisting of a fixed length (8-bits) header followed by a series of numeric fields (Figure H) whose overall length, structure, and function are completely determined by the header value. For future expansion purpose, a header value of 11111111 is defined, to indicate that longer header beyond 8-bits is used.



3.1 Headers

As previously stated, the Header defines the overall length, identity type, and structure of the EPC Tag Encoding. Headers defined in this version of the Tag Data Standard are eight bits in length. The value of 11111111 in the header bits, however, is reserved for future expansion of header space, so that more than 256 headers may be accommodated by using longer headers. Therefore, the present specification provides for up to 255 8-bit headers, plus a currently undetermined number of longer headers.

Back-compatibility note (non-normative) In a prior version of the Tag Data Standard, the header was of variable length, using a tiered approach in which a zero value in each tier indicated that the header was drawn from the next longer tier. For the encodings defined in the earlier specification, headers were either 2 bits or 8 bits. Given that a zero value is reserved to indicate a header in the next longer tier, the 2-bit header had 3 possible values (01, 10, and 11, not 00), and the 8-bit header had 63 possible values (recognizing 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 Tag Encodings.

In this version of the Tag Data Standard, 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. To facilitate an orderly transition, the portions of header space formerly occupied by 64-bit encodings are reserved in this version of the Tag Data Standard, with the intention that they be reclaimed after a “sunset date” has passed. After the “sunset date,” tags containing 64-bit EPCs with 2-bit headers and tags with 64-bit headers starting with 00001 will no longer be properly interpreted.

Eleven encoding schemes have been defined in this version of the EPC Tag Data Standard, as shown in Table 1 below. The table also indicates header values that are currently unassigned, as well as header values that have been reserved to allow for an orderly “sunset” of 64-bit encodings defined in a prior version of the EPC Tag Data Standard. These will not be available for assignment until after the “sunset date” has passed.

Header Value (binary)	Header Value (hex)	Encoding Length (bits)	Encoding Scheme
0000 0000	00	NA	Unprogrammed Tag
<u>0000 0001</u>	<u>01</u>	NA	Reserved for Future Use
<u>0000 001x</u>	<u>02,03</u>	NA	Reserved for Future Use
<u>0000 01xx</u>	<u>04,05</u>	NA	Reserved for Future Use
	<u>06,07</u>	NA	Reserved for Future Use
0000 1000	08	64	Reserved until 64bit Sunset <SSCC-64>
0000 1001	09	64	Reserved until 64bit Sunset <SGLN-64>
0000 1010	0A	64	Reserved until 64bit Sunset <GRAI-64>
0000 1011	0B	64	Reserved until 64bit Sunset <GIAI-64>

Header Value (binary)	Header Value (hex)	Encoding Length (bits)	Encoding Scheme
<u>0000 1100</u> to <u>0000 1111</u>	0C to 0F		<u>Reserved until 64 bit Sunset</u> <u>Due to 64 bit encoding rule in Gen 1</u>
<u>0001 0000</u> to <u>0010 1110</u>	<u>10</u> to <u>2E</u>	NA NA	<u>Reserved for Future Use</u>
0010 1111	2F	96	DoD-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	<u>36</u>	<u>198</u>	<u>SGTIN-198</u>
0011 0111	<u>37</u>	<u>170</u>	<u>GRAI-170</u>
0011 1000	<u>38</u>	<u>202</u>	<u>GIAI-202</u>
0011 1001	<u>39</u>	<u>195</u>	<u>SGLN-195</u>
<u>0011 1010</u> to <u>0011 1111</u>	<u>3A</u> to <u>3F</u>		<u>Reserved for future Header values</u>
0100 0000 to 0111 1111	40 to 7F		<u>Reserved until 64 bit Sunset</u>
1000 0000 to 1011 1111	80 to BF	<u>64</u>	Reserved until 64 bit Sunset <SGTIN-64> (64 header values)
<u>1100 0000</u> to <u>1100 1101</u>	<u>C0</u> to <u>CD</u>		Reserved until 64 bit Sunset

Header Value (binary)	Header Value (hex)	Encoding Length (bits)	Encoding Scheme
1100 1110	CE	64	Reserved until 64 bit Sunset <DoD-64>
<u>1100 1111</u> to <u>1111 1110</u>	CF to FE		<u>Reserved until 64 bit Sunset</u>
1111 1111	FF	NA	Reserved for future headers longer than 8 bits

Table 1. Electronic Product Code Headers

3.2 Use of EPCs on UHF Class 1 Generation 2 Tags

This section defines how the Electronic Product Code is encoded onto RFID tags conforming to the Gen 2 Specification.

In the Gen 2 Specification, the tag memory is separated into four distinct banks, each of which may comprise one or more memory words, where each word is 16 bits long. These memory banks are described as “Reserved”, “EPC”, “TID” and “User”. The “Reserved” memory bank contains kill and access passwords, the “EPC” memory bank contains data used for identifying the object to which the tag is or will be attached, the “TID” memory bank contains data that can be used by the reader to identify the tag’s capability, and “User” memory bank is intended to contain user-specific data.

This version of the Tag Data Standards specifies normatively how Electronic Product Codes (EPC) are encoded in the EPC memory bank of Gen 2 Tags. It is anticipated that EPCs may also be used in the User memory bank, but such use is not addressed in this version of the specification. Normative descriptions for encoding of the Reserved and User memory bank will be addressed in future versions of this specification. For encodings of the TID memory bank refer to the Gen 2 Specification.

3.2.1 EPC Memory Contents

The EPC memory bank of a Gen 2 Tag holds an EPC, plus additional control information. The complete contents of the EPC memory bank consist of:

- *CRC-16 (16 bits)* Bits that represent the error check code and are auto-calculated by the Tag. (For further details of the CRC, refer to UHF Class 1 Generation 2 Tag Protocol specification Section 6.3.2.1.3)
- *Protocol-Control (PC) (16 bits total)* which is subdivided into:
 - *Length (5 bits)* Represents the number of 16-bit words comprising the PC field and the EPC field (below). See discussion below for the encoding of this field.
 - *Reserved for Future Use (RFU) (2 bits)* Always zero in the current version of the UHF Class 1 Generation 2 Tag Protocol Specification.

- *Numbering System Identifier (NSI) (9 bits total)* which is further subdivided into:
 - *Toggle bit (1 bit)* Boolean flag indicating whether the next 8 bits of the NSI represents reserved memory or an ISO 15961 Application Family Identifier (AFI). If set to “zero” indicates that the NSI contains reserved memory, if set to “one” indicates that the NSI contains an ISO AFI.
 - *Reserved/AFI (8 bits)* Based on the value of the Toggle Bit above, these 8 bits are either Reserved and must all be set to “zero”, or contain an AFI whose value is defined under the authority of ISO.
- *EPC (variable length)* When the Toggle Bit is set to “zero”, an EPC Tag Encoding as defined in the remaining sections of this chapter is contained here. When the Toggle Bit is set to “one”, these bits are part of a non-EPC coding scheme identified by the AFI field (see above) whose interpretation is outside the scope of this specification.
- *Zero fill (variable length)* If there is any additional memory beyond EPC Tag Encoding required to meet the 16 bit word boundaries specified in Gen 2 Specification, it is filled with zeros. An implementation shall not put any data into EPC memory following the EPC Tag Encoding and any required zero fill (15 bits or less); if it does, it is not in compliance with the specification and risks the possibility of incompatibility with a future version of the spec.

The following figure depicts the complete contents of the EPC bank of a Gen 2 Tag, including the EPC and the surrounding control information, when an EPC is encoded into the EPC bank:

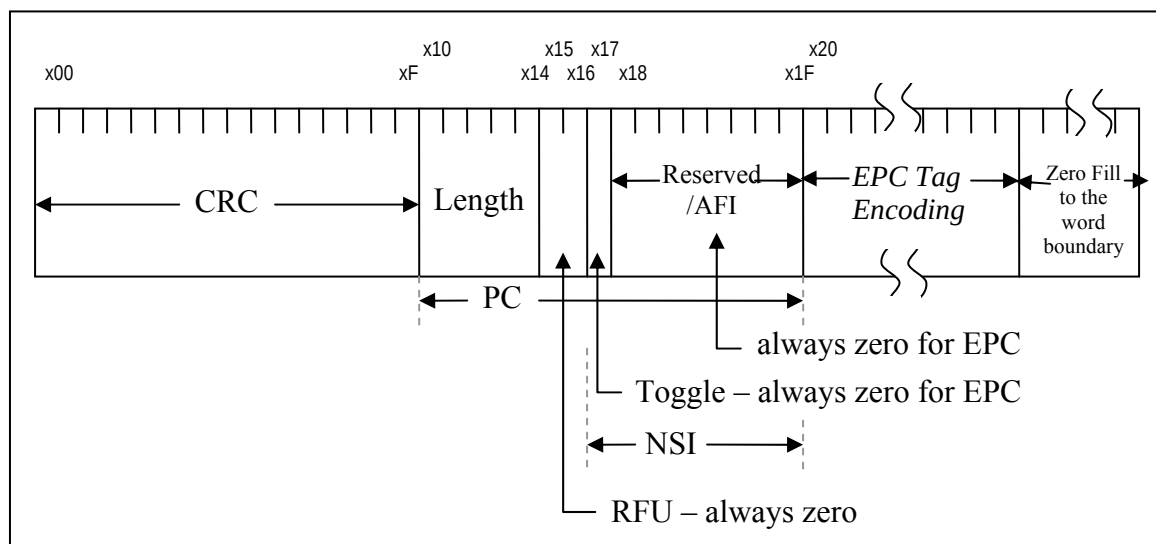


Figure I. Complete contents of EPC memory bank of a Gen 2 Tag.

Except for the 16 bit CRC it is the responsibility of the application or process communicating with the reader to provide all the bits to encode in the EPC memory bank.

573 The complete contents of the EPC are defined by the remaining subsections within this
 574 chapter.

575 **3.2.2 The Length Bits**

576 The length field is used to let a reader know how much of the EPC memory is occupied with
 577 valid data. The value of the length field is the number of 16-bit segments occupied with
 578 valid data, not including the CRC, minus one. For example, if set to '000000', the length
 579 field indicates that valid data extends through bit x1F, if set to '00001', the length field
 580 indicates that valid data extends through bit x2F, and so on.

581 When a Gen 2 Tag contains an EPC Tag Encoding in the EPC bank, the length field is
 582 normally set to the smallest number that would contain the particular kind of EPC Tag
 583 Encoding in use. Specifically, if the EPC bank contains an N-bit EPC Tag Encoding, then
 584 the length field is normally set to N/16, rounded up to the nearest integer. For example, with
 585 a 96-bit EPC Tag Encoding, the length field is normally set to 6 (00110 in binary).

586 It is important to note that the length of the EPC Tag Encoding is indicated by the EPC
 587 header, not by the length field in the PC bits. This is necessarily so, because the length field
 588 indicates only the nearest multiple of 16 bits, but the actual amount of EPC memory
 589 consumed by the EPC Tag Encoding does not necessarily fall on a multiple-of-16-bit
 590 boundary.

591 Moreover, there are applications in which the length field may be set to a different value than
 592 the one determined by the formula above. For example, there may be applications in which
 593 the EPC is not written to the EPC bank in one operation, but where a prefix of the EPC is
 594 written in one operation (perhaps excluding the serial number) and subsequently the
 595 remainder of the EPC is written. In such an application, a length field smaller than the
 596 normal value might be used to indicate that the EPC is incompletely written.

597 **3.3 Notational Conventions**

598 In the remainder of this section, EPC Tag Encoding schemes are depicted using the
 599 following notation (See Table 2).

	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
SGTIN-96	8	3	3	20-40	24-4	38
	0011 0000 (Binary value)	(Refer to Table 5 for values)	(Refer to Table 6 for values)	999,999 – 999,999,999,999 (Max. decimal range*)	9,999,999 – 9 (Max. decimal range*)	274,877,906 ,943 (Max. decimal value)

600 *Max. decimal value range of Item Reference field varies with the length of the Company Prefix

601 **Table 2.** Example of Notation Conventions.

The first column of the table gives the formal name for the encoding. The remaining columns specify the layout of each field within the encoding. The field in the leftmost column occupies the most significant bits of the encoding (this is always the header field), and the field in the rightmost column occupies the least significant bits. Each field is a non-negative integer, encoded into binary using a specified number of bits. Any unused bits (i.e., bits not required by a defined field) are explicitly indicated in the table, so that the columns in the table are concatenated with no gaps to form the complete binary encoding.

Reading down each column, the table gives the formal name of the field, the number of bits used to encode the field's value, and the value or range of values for the field. The value may represent one of the following:

- The value of a binary number indicated by (*Binary value*), as is the case for the Header field in the example table above
- The maximum decimal value indicated by (*Max. decimal value*) of a fixed length field. This is calculated as $2^n - 1$, where n = the fixed number of bits in the field.
- A range of maximum decimal values indicated by (*Max. decimal range*). This range is calculated using the normative rules expressed in the related encoding procedure section
- A reference to a table that provides the valid values defined for the field..

In some cases, the number of possible values in one field depends on the specific value assigned to another field. In such cases, a range of maximum decimal values is shown. In the example above, the maximum decimal value for the Item Reference field depends on the length of the Company Prefix field; hence the maximum decimal value is shown as a range. Where a field must contain a specific value (as in the Header field), the last row of the table specifies the specific value rather than the number of possible values.

Some encodings have fields that are of variable length. The accompanying text specifies how the field boundaries are determined in those cases.

Following an overview of each encoding scheme are a detailed encoding procedure and decoding procedure. The encoding and decoding procedure provide the normative specification for how each type of encoding is to be formed and interpreted.

3.4 General Identifier (GID-96)

The *General Identifier* is defined for a 96-bit EPC, and is independent of any existing identity specification or convention. In addition to the header which guarantees uniqueness in the EPC namespace, the *General Identifier* is composed of three fields - the *General Manager Number*, *Object Class* and *Serial Number*., as shown in Table 3.

	Header	General Manager	Object Class	Serial Number
--	--------	-----------------	--------------	---------------

		Number		
GID-96	8	28	24	36
	0011 0101 (Binary value)	268,435,455 (Max. decimal value)	16,777,215 (Max. decimal value)	68,719,476,735 (Max. decimal value)

Table 3. The General Identifier (GID-96) includes three fields in addition to the header – the *General Manager Number*, *Object class* and *Serial Number* numbers.

- The *Header* is 8-bits, with a binary value of 0011 0101.
- The *General Manager Number* identifies essentially a company, manager or organization; that is an entity responsible for maintaining the numbers in subsequent fields – Object Class and Serial Number. EPCglobal assigns the General Manager Number to an entity, and ensures that each General Manager Number is unique.

Note (non-normative): Currently, GS1 is only allocating an integer value in the range from 95,100,000 to 95,199,999 for this number.

- The *Object Class* is used by an EPC managing entity to identify a class or “type” of thing. These object class numbers, of course, must be unique within each General Manager Number domain. Examples of Object Classes could include case Stock Keeping Units of consumer-packaged goods and component parts in an assembly.
- The *Serial Number* code, or serial number, is unique within each object class. In other words, the managing entity is responsible for assigning unique – non-repeating serial numbers for every instance within each object class code.

3.4.1.1 GID-96 Encoding Procedure

The following procedure creates a GID-96 encoding.

Given:

- A General Manager Number M where $0 \leq M < 2^{28}$
- An Object Class C where $0 \leq C < 2^{24}$
- A Serial Number S where $0 \leq S < 2^{36}$

Procedure:

1. Construct the General Manager Number by considering digits $d_1d_2\dots d_8$ to be a decimal integer, M . If the value is outside the range specified above, stop: this GID cannot be encoded as a valid GID-96
2. If the Object class and/or the Serial Number are provided with a value outside the acceptable range specified above, stop: this GID cannot be encoded as a valid GID-96

3. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header 00110101, General Manager Number M (28 bits), Object Class C (24 bits), Serial Number S (36 bits).

3.4.1.2 GID-96 Decoding Procedure

Given:

- A GID-96 as a 96-bit string $00110101b_{87}b_{86}\dots b_0$ (where the first eight bits 00110101 are the header)

Yields:

- A General Manager Number
- An Object Class
- A Serial Number

Procedure:

- Bits $b_{87}b_{86}\dots b_{60}$, considered as an unsigned integer, are the General Manager Number.
- Bits $b_{59}b_{58}\dots b_{36}$, considered as an unsigned integer, are the Object Class.
- Bits $b_{35}b_{34}\dots b_0$, considered as an unsigned integer, are the Serial Number.

3.5 Serialized Global Trade Item Number (SGTIN)

The EPC Tag Encoding scheme for SGTIN permits the direct embedding of EAN.UCC System standard GTIN and Serial Number codes on EPC tags. In all cases, the check digit is not encoded.

3.5.1 SGTIN-96

In addition to a Header, the SGTIN-96 is composed of five fields: the *Filter Value*, *Partition*, *Company Prefix*, *Item Reference*, and *Serial Number*, as shown in Table 4.

	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
SGTIN-96	8	3	3	20-40	24-4	38
	0011 0000 (Binary value)	(Refer to Table 5 for values)	(Refer to Table 6 for values)	999,999 – 999,999,999,999 (Max. decimal range*)	9,999,999 – 9 (Max. decimal range*)	274,877,906,943 (Max. decimal value)

*Max. decimal value range of Company Prefix and Item Reference fields vary according to the contents of the Partition field.

Table 4. The EPC SGTIN-96 bit allocation, header, and maximum decimal values.

- *Header* is 8-bits, with a binary value of 0011 0000.
- *Filter Value* is not part of the SGTIN pure identity, but is additional data that is used for fast filtering and pre-selection of basic logistics types. The normative specifications for Filter Values are specified in Table 5.
The value of 000 means “All Others”. That is, a filter value of 000 means that the object to which the tag is affixed does not match any of the logistic types defined as other filter values in this specification. It should be noted that tags conforming to earlier versions of this specification, in which 000 was the only value approved for use, will have filter value equal to 000, but following the ratification of this standard, the filter value should be set to match the object to which the tag is affixed, and use 000 only if the filter value for such object does not exist in the specification.
A Standard Trade Item grouping represents all levels of packaging for logistical units. The Single Shipping / Consumer Trade item type should be used when the individual item is also the logistical unit (e.g. Large screen television, Bicycle).

Type	Binary Value
All Others	000
Retail Consumer Trade Item	001
Standard Trade Item Grouping	010
Single Shipping/ Consumer Trade Item	011
Reserved	100
Reserved	101
Reserved	110
Reserved	111

Table 5. SGTIN Filter Values .

- *Partition* is an indication of where the subsequent Company Prefix and Item Reference numbers are divided. This organization matches the structure in the EAN.UCC GTIN in which the Company Prefix added to the Item Reference number (prefixed by the single Indicator Digit) totals 13 digits, yet the Company Prefix may vary from 6 to 12 digits and the concatenation of single Indicator Digit and Item Reference from 7 to 1 digit(s). The available values of *Partition* and the corresponding sizes of the *Company Prefix* and *Item Reference* fields are defined in Table 6.
- *Company Prefix* contains a literal embedding of the EAN.UCC Company Prefix.

- *Item Reference* contains a literal embedding of the GTIN Item Reference number. The Indicator Digit is combined with the Item Reference field in the following manner: Leading zeros on the item reference are significant. Put the Indicator Digit in the leftmost position available within the field. *For instance, 00235 is different than 235. With the indicator digit of 1, the combination with 00235 is 100235.* The resulting combination is treated as a single integer, and encoded into binary to form the *Item Reference* field.
- *Serial Number* contains a serial number. The SGTIN-96 encoding is only capable of representing integer-valued serial numbers with limited range. The EAN.UCC specifications permit a broader range of serial numbers. The EAN.UCC-128 barcode symbology provides for a 20-character alphanumeric serial number to be associated with a GTIN using Application Identifier (AI) 21 [EAN.UCCGS]. It is possible to convert between the serial numbers in the SGTIN-96 tag encoding and the serial numbers in AI 21 barcodes under certain conditions. Specifically, such interconversion is possible when the alphanumeric serial number in AI 21 happens to consist only of digits with no leading zeros, and whose value when interpreted as an integer falls within the range limitations of the SGTIN-96 tag encoding. These considerations are reflected in the encoding and decoding procedures below.

Partition Value (P)	Company Prefix		Indicator Digit and Item Reference	
	Bits (M)	Digits (L)	Bits (N)	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

Table 6. SGTIN Partitions.

3.5.1.1 SGTIN-96 Encoding Procedure

The following procedure creates an SGTIN-96 encoding.

Given:

- An EAN.UCC GTIN-14 consisting of digits $d_1d_2\dots d_{14}$
- The length L of the Company Prefix portion of the GTIN

743 • A Serial Number S where $0 \leq S < 2^{38}$, or an EAN.UCC-128 Application Identifier 21
744 consisting of characters $s_1s_2 \dots s_K$.

745 • A Filter Value F where $0 \leq F < 8$

746 Procedure:

747 1. Look up the length L of the Company Prefix in the “Company Prefix Digits” column of
748 the Partition Table (Table 6) to determine the Partition Value, P , the number of bits M in the
749 Company Prefix field, and the number of bits N in the Item Reference and Indicator Digit
750 field. If L is not found in any row of Table 6, stop: this GTIN cannot be encoded in an
751 SGTIN-96.

752 2. Construct the Company Prefix by concatenating digits $d_2d_3 \dots d_{(L+1)}$ and considering the
753 result to be a decimal integer, C .

754 3. Construct the Indicator Digit and Item Reference by concatenating digits
755 $d_1d_{(L+2)}d_{(L+3)} \dots d_{13}$ and considering the result to be a decimal integer, I .

756 4. When the Serial Number is provided directly as an integer S where $0 \leq S < 2^{38}$, proceed to
757 Step 5. Otherwise, when the Serial Number is provided as an EAN.UCC-128 Application
758 Identifier 21 consisting of characters $s_1s_2 \dots s_K$, construct the Serial Number by concatenating
759 digits $s_1s_2 \dots s_K$. If any of these characters is not a digit, stop: this Serial Number cannot be
760 encoded in the SGTIN-96 encoding. Also, if $K > 1$ and $s_1 = 0$, stop: this Serial Number
761 cannot be encoded in the SGTIN-96 encoding (because leading zeros are not permitted
762 except in the case where the Serial Number consists of a single zero digit). Otherwise,
763 consider the result to be a decimal integer, S . If $S \geq 2^{38}$, stop: this Serial Number cannot be
764 encoded in the SGTIN-96 encoding.

765 5. Construct the final encoding by concatenating the following bit fields, from most
766 significant to least significant: Header 00110000 (8 bits), Filter Value F (3 bits), Partition
767 Value P from Step 1 (3 bits), Company Prefix C from Step 2 (M bits), Item Reference from
768 Step 3 (N bits), Serial Number S from Step 4 (38 bits). Note that $M+N = 44$ bits for all P .

769 3.5.1.2 SGTIN-96 Decoding Procedure

770 Given:

771 • An SGTIN-96 as a 96-bit bit string $00110000b_{87}b_{86} \dots b_0$ (where the first eight bits
772 00110000 are the header)

773 Yields:

774 • An EAN.UCC GTIN-14

775 • A Serial Number

776 • A Filter Value

777 Procedure:

778 1. Bits $b_{87}b_{86}b_{85}$, considered as an unsigned integer, are the Filter Value.

779 2. Extract the Partition Value P by considering bits $b_{84}b_{83}b_{82}$ as an unsigned integer. If
780 $P = 7$, stop: this bit string cannot be decoded as an SGTIN-96.

3. Look up the Partition Value P in Table 6 to obtain the number of bits M in the Company Prefix and the number of digits L in the Company Prefix.
4. Extract the Company Prefix C by considering bits $b_{81}b_{80}\dots b_{(82-M)}$ as an unsigned integer. If this integer is greater than or equal to 10^L , stop: the input bit string is not a legal SGTIN-96 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\dots p_L$, adding leading zeros as necessary to make up L digits in total.
5. Extract the Item Reference and Indicator by considering bits $b_{(81-M)}b_{(80-M)}\dots b_{38}$ as an unsigned integer. If this integer is greater than or equal to $10^{(13-L)}$, stop: the input bit string is not a legal SGTIN-96 encoding. Otherwise, convert this integer to a $(13-L)$ -digit decimal number $i_1i_2\dots i_{(13-L)}$, adding leading zeros as necessary to make $(13-L)$ digits.
6. Construct a 13-digit number $d_1d_2\dots d_{13}$ where $d_1 = i_1$ from Step 5, $d_2d_3\dots d_{(L+1)} = p_1p_2\dots p_L$ from Step 4, and $d_{(L+2)}d_{(L+3)}\dots d_{13} = i_2i_3\dots i_{(13-L)}$ from Step 5.
7. Calculate the check digit $d_{14} = (-3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13}) - (d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12})) \bmod 10$.
8. The EAN.UCC GTIN-14 is the concatenation of digits from Steps 6 and 7: $d_1d_2\dots d_{14}$.
9. Bits $b_{37}b_{36}\dots b_0$, considered as an unsigned integer, are the Serial Number.
10. (Optional) If it is desired to represent the serial number as a EAN.UCC-128 Application Identifier 21, convert the integer from Step 9 to a decimal string with no leading zeros. If the integer in Step 9 is zero, convert it to a string consisting of the single character "0".

3.5.2 SGTIN-198

In addition to a Header, the SGTIN-198 is composed of five fields: the *Filter Value*, *Partition*, *Company Prefix*, *Item Reference*, and *Serial Number*, as shown in Table 7.

	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
SGTIN-198	8	3	3	20-40	24-4	140
	0011 0110 (Binary value)	(Refer to Table 5 for values)	(Refer to Table 6 for values)	999,999 – 999,999,999,999 (Max. decimal range*)	9,999,999 – 9 (Max. decimal range*)	Up to 20 alphanumeric characters

*Max. decimal value range of Company Prefix and Item Reference fields vary according to the contents of the Partition field.

Table 7. The EPC SGTIN-198 bit allocation, header, and maximum decimal values.

- *Header* is 8-bits, with a binary value of 0011 0110.

- *Filter Value* is not part of the GTIN or EPC identifier, but is used for fast filtering and pre-selection of basic logistics types. The Filter Values for 96-bit, and 198-bit GTIN are the same. See Table 5.
- *Partition* is an indication of where the subsequent Company Prefix and Item Reference numbers are divided. This organization matches the structure in the EAN.UCC GTIN in which the Company Prefix added to the Item Reference number (prefixed by the single Indicator Digit) totals 13 digits, yet the Company Prefix may vary from 6 to 12 digits and the Item Reference (including the single Indicator Digit) from 7 to 1 digit(s). The available values of *Partition* and the corresponding sizes of the *Company Prefix* and *Item Reference* fields are defined in Table 6.
- *Company Prefix* contains a literal embedding of the EAN.UCC Company Prefix.
- *Item Reference* contains a literal embedding of the GTIN Item Reference number. The Indicator Digit is combined with the Item Reference field in the following manner: Leading zeros on the item reference are significant. Put the Indicator Digit in the leftmost position available within the field. *For instance, 00235 is different than 235. With the indicator digit of 1, the combination with 00235 is 100235.* The resulting combination is treated as a single integer, and encoded into binary to form the *Item Reference* field.
- *Serial Number* contains a serial number. The SGTIN-198 encoding is capable of representing alphanumeric serial numbers of up to 20 characters, permitting the full range of serial numbers available in the EAN.UCC-128 barcode symbology using Application Identifier (AI) 21 [EAN.UCCGS]. See Appendix G for permitted values.

3.5.2.1 SGTIN-198 Encoding Procedure

The following procedure creates an SGTIN-198 encoding.

Given:

- An EAN.UCC GTIN-14 consisting of digits $d_1d_2\dots d_{14}$
- The length L of the Company Prefix portion of the GTIN
- An EAN.UCC-128 Application Identifier 21 consisting of characters $s_1s_2\dots s_K$, where $K \leq 20$.
- A Filter Value F where $0 \leq F < 8$

Procedure:

1. Look up the length L of the Company Prefix in the “Company Prefix Digits” column of the Partition Table (Table 6) to determine the Partition Value, P , the number of bits M in the Company Prefix field, and the number of bits N in the Item Reference and Indicator Digit field. If L is not found in any row of Table 6, stop: this GTIN cannot be encoded in an SGTIN-198.
2. Construct the Company Prefix by concatenating digits $d_2d_3\dots d_{(L+1)}$ and considering the result to be a decimal integer, C .

- 846 3. Construct the Indicator Digit and Item Reference by concatenating digits
 847 $d_1d_{(L+2)}d_{(L+3)}\dots d_{13}$ and considering the result to be a decimal integer, I .
- 848 4. Check that each of the characters $s_1s_2\dots s_K$ is one of the 82 characters listed in the table
 849 in Appendix G. If this is not the case, stop: this character string cannot be encoded as an
 850 SGTIN-198. Otherwise construct the Serial Number by concatenating the 7-bit code, as
 851 given in Appendix G, for each of the characters $s_1s_2\dots s_K$, yielding $7K$ bits total. If $K < 20$,
 852 concatenate additional zero bits to the right to make a total of 140 bits.
- 853 5. Construct the final encoding by concatenating the following bit fields, from most
 854 significant to least significant: Header 00110110 (8 bits), Filter Value F (3 bits), Partition
 855 Value P from Step 1 (3 bits), Company Prefix C from Step 2 (M bits), Item Reference from
 856 Step 3 (N bits), Serial Number from Step 4 (140 bits). Note that $M+N = 44$ bits for all P .

857 3.5.2.2 SGTIN-198 Decoding Procedure

858 Given:

- 859 • An SGTIN-198 as a 198-bit bit string $00110110b_{189}b_{188}\dots b_0$ (where the first eight bits
 860 00110110 are the header)

861 Yields:

- 862 • An EAN.UCC GTIN-14
 863 • A Serial Number
 864 • A Filter Value

865 Procedure:

- 866 1. Bits $b_{189}b_{188}b_{187}$, considered as an unsigned integer, are the Filter Value.
- 867 2. Extract the Partition Value P by considering bits $b_{186}b_{185}b_{184}$ as an unsigned integer. If
 868 $P = 7$, stop: this bit string cannot be decoded as an SGTIN-198.
- 869 3. Look up the Partition Value P in Table 6 to obtain the number of bits M in the Company
 870 Prefix and the number of digits L in the Company Prefix.
- 871 4. Extract the Company Prefix C by considering bits $b_{183}b_{182}\dots b_{(184-M)}$ as an unsigned
 872 integer. If this integer is greater than or equal to 10^L , stop: the input bit string is not a legal
 873 SGTIN-198 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\dots p_L$,
 874 adding leading zeros as necessary to make up L digits in total.
- 875 5. Extract the Item Reference and Indicator by considering bits $b_{(183-M)}b_{(182-M)}\dots b_{140}$ as an
 876 unsigned integer. If this integer is greater than or equal to $10^{(13-L)}$, stop: the input bit string
 877 is not a legal SGTIN-198 encoding. Otherwise, convert this integer to a $(13-L)$ -digit decimal
 878 number $i_1i_2\dots i_{(13-L)}$, adding leading zeros as necessary to make $(13-L)$ digits.
- 879 6. Construct a 13-digit number $d_1d_2\dots d_{13}$ where $d_1 = i_1$ from Step 5, $d_2d_3\dots d_{(L+1)} = p_1p_2\dots p_L$
 880 from Step 4, and $d_{(L+2)}d_{(L+3)}\dots d_{13} = i_2i_3\dots i_{(13-L)}$ from Step 5.
- 881 7. Calculate the check digit $d_{14} = (-3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13}) - (d_2 + d_4 + d_6 + d_8 +$
 882 $d_{10} + d_{12})) \bmod 10$.
- 883 8. The EAN.UCC GTIN-14 is the concatenation of digits from Steps 6 and 7: $d_1d_2\dots d_{14}$.

9. Divide the remaining bits $b_{139}b_{138}...b_0$ into 7-bit segments. The result should consist of K non-zero segments followed by 20-K zero segments. If this is not the case, stop: this bit string cannot be decoded as an SGTIN-198. Otherwise, look up each of the non-zero 7-bit segments in Appendix G to obtain a corresponding character. If any of the non-zero 7-bit segments has a value that is not in Appendix G, stop: this bit string cannot be decoded as an SGTIN-198. Otherwise, the K characters so obtained, considered as a character string, is the value of the EAN.UCC AI 21.

10. The EAN.UCC SGTIN-198 is the concatenation of the digits from Steps 6 and 7 and the characters from Step 9. : $d_1d_2...d_{14}s_1s_2...s_K$

3.6 Serial Shipping Container Code (SSCC)

The EPC Tag Encoding scheme for SSCC permits the direct embedding of EAN.UCC System standard SSCC codes on EPC tags. In all cases, the check digit is not encoded.

3.6.1 SSCC-96

In addition to a Header, the EPC SSCC-96 is composed of four fields: the *Filter Value*, *Partition*, *Company Prefix*, and *Serial Reference*, as shown in Table 8.

	Header	Filter Value	Partition	Company Prefix	Serial Reference	Unallocated
SSCC-96	8	3	3	20-40	38-18	24
	0011 0001 (Binary value)	(Refer to Table 9 for values)	(Refer to Table 10 for values)	999,999 – 999,999,999,999 (Max. decimal range*)	99,999,999,999 – 99,999 (Max. decimal range*)	[Not Used]

*Max. decimal value range of Company Prefix and Serial Reference fields vary according to the contents of the Partition field.

Table 8. The EPC 96-bit SSCC bit allocation, header, and maximum decimal values.

- *Header* is 8-bits, with a binary value of 0011 0001.
- *Filter Value* is not part of the SSCC or EPC identifier, but is used for fast filtering and pre-selection of basic logistics types. The normative specifications for Filter Values are specified in Table 9.
The value of 000 means “All Others”. That is, a filter value of 000 means that the object to which the tag is affixed does not match any of the logistic types defined as

other filter values in the specification. It should be noted that tags conforming to earlier versions of this specification, in which 000 was the only value approved for use, will have filter value equal to 000, but following the ratification of this standard, the filter value should be set to match the object to which the tag is affixed, and use 000 only if the filter value for such object does not exist in the specification.

Type	Binary Value
All Others	000
Undefined	001
Logistical / Shipping Unit	010
Reserved	011
Reserved	100
Reserved	101
Reserved	110
Reserved	111

Table 9. SSCC Filter Values

- The *Partition* is an indication of where the subsequent Company Prefix and Serial Reference numbers are divided. This organization matches the structure in the EAN.UCC SSCC in which the Company Prefix added to the Serial Reference number (prefixed by the single Extension Digit) totals 17 digits, yet the Company Prefix may vary from 6 to 12 digits and the Serial Reference from 11 to 5 digits. Table 10 shows allowed values of the partition value and the corresponding lengths of the company prefix and serial reference.

Partition Value (P)	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

Table 10. SSCC-96 Partitions.

- *Company Prefix* contains a literal embedding of the Company Prefix.
- *Serial Reference* is a unique number for each instance, comprised of the Extension Digit and the Serial Reference. The Extension Digit is combined with the Serial Reference field in the following manner: Leading zeros on the Serial Reference are significant. Put the Extension Digit in the leftmost position available within the field. *For instance, 000042235 is different than 42235. With the extension digit of 1, the combination with 000042235 is 1000042235.* The resulting combination is treated as a single integer, and encoded into binary to form the Serial Reference field. To avoid unmanageably large and out-of-specification serial references, they should not exceed the capacity specified in EAN.UCC specifications, which are (inclusive of extension digit) 9,999 for company prefixes of 12 digits up to 9,999,999,999 for company prefixes of 6 digits.
- *Unallocated* is not used. This field must contain zeros to conform with this version of the specification.

3.6.1.1 SSCC-96 Encoding Procedure

The following procedure creates an SSCC-96 encoding.

Given:

- An EAN.UCC SSCC consisting of digits $d_1d_2\dots d_{18}$
- The length L of the Company Prefix portion of the SSCC
- A Filter Value F where $0 \leq F < 8$

Procedure:

1. Look up the length L of the Company Prefix in the “Company Prefix Digits” column of the Partition Table (Table 10) to determine the Partition Value, P , the number of bits M in the Company Prefix field, and the number of bits N in the Extension Digit and the Serial Reference. If L is not found in any row of Table 10, stop: this SSCC cannot be encoded in an SSCC-96.
2. Construct the Company Prefix by concatenating digits $d_2d_3\dots d_{(L+1)}$ and considering the result to be a decimal integer, C .
3. Construct the Extension Digit and the Serial Reference by concatenating digits $d_1d_{(L+2)}d_{(L+3)}\dots d_{17}$ and considering the result to be a decimal integer, S .
4. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header 00110001 (8 bits), Filter Value F (3 bits), Partition Value P from Step 1 (3 bits), Company Prefix C from Step 2 (M bits), Serial Reference S from Step 3 (N bits), and 24 zero bits. Note that $M+N = 58$ bits for all P .

3.6.1.2 SSCC-96 Decoding Procedure

Given:

961 • An SSCC-96 as a 96-bit bit string $00110001b_{87}b_{86}\dots b_0$ (where the first eight bits
 962 00110001 are the header)

963 Yields:

964 • An EAN.UCC SSCC

965 • A Filter Value

966 Procedure:

967 1. Bits $b_{87}b_{86}b_{85}$, considered as an unsigned integer, are the Filter Value.

968 2. Extract the Partition Value P by considering bits $b_{84}b_{83}b_{82}$ as an unsigned integer. If
 969 $P = 7$, stop: this bit string cannot be decoded as an SSCC-96.

970 3. Look up the Partition Value P in Table 10 to obtain the number of bits M in the Company
 971 Prefix and the number of digits L in the Company Prefix.

972 4. Extract the Company Prefix C by considering bits $b_{81}b_{80}\dots b_{(82-M)}$ as an unsigned integer.
 973 If this integer is greater than or equal to 10^L , stop: the input bit string is not a legal SSCC-96
 974 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\dots p_L$, adding leading
 975 zeros as necessary to make up L digits in total.

976 5. Extract the Serial Reference by considering bits $b_{(81-M)}b_{(80-M)}\dots b_{24}$ as an unsigned integer.
 977 If this integer is greater than or equal to $10^{(17-L)}$, stop: the input bit string is not a legal
 978 SSCC-96 encoding. Otherwise, convert this integer to a $(17-L)$ -digit decimal number
 979 $i_1i_2\dots i_{(17-L)}$, adding leading zeros as necessary to make $(17-L)$ digits.

980 6. Construct a 17-digit number $d_1d_2\dots d_{17}$ where $d_1 = s_1$ from Step 5, $d_2d_3\dots d_{(L+1)} = p_1p_2\dots p_L$
 981 from Step 4, and $d_{(L+2)}d_{(L+3)}\dots d_{17} = i_2i_3\dots i_{(17-L)}$ from Step 5.

982 7. Calculate the check digit $d_{18} = (-3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13} + d_{15} + d_{17}) - (d_2 + d_4$
 983 $+ d_6 + d_8 + d_{10} + d_{12} + d_{14} + d_{16})) \bmod 10$.

984 8. The EAN.UCC SSCC is the concatenation of digits from Steps 6 and 7: $d_1d_2\dots d_{18}$.

985 **3.7 Serialized Global Location Number (SGLN)**

986 The EPC Tag Encoding scheme for GLN permits the direct embedding of the EAN.UCC
 987 System standard GLN on EPC tags. EAN.UCC has defined the GLN as AI (414) and has
 988 defined a GLN Extension Component as AI (254). The AI (254) uses the Set of Characters
 989 defined in Appendix G.

990 The use of the GLN Extension Component is intended for internal company purposes. For
 991 communication between trading partners a GLN will be used. Trading partners can only use
 992 the GLN Extension through mutual agreement but would have to establish an “out of band”
 993 exchange of master data describing the extensions. If the GLN only encoding is used, then
 994 the *Extension Component* shall be set to a fixed value of binary “0” for SGLN-96 and to
 995 binary 0110000 followed by 133 binary “0” bits for SGLN-195 encoding as described in the
 996 following SGLN procedures. In all cases the check digit is not encoded.

3.7.1 SGLN-96

In addition to a Header, the SGLN-96 is composed of five fields: the *Filter Value*, *Partition*, *Company Prefix*, *Location Reference*, and *Extension Component*, as shown in Table 11.

	Header	Filter Value	Partition	Company Prefix	Location Reference	Extension Component
SGLN-96	8	3	3	20-40	21-1	41
	0011 0010 (Binary value)	(Refer to Table 12 for values)	(Refer to Table 13 for values)	999,999 – 999,999,999,999 (Max. decimal range*)	999,999 – 0 (Max. decimal range*)	999,999,999,999(Max Decimal Value allowed) Minimum Decimal value=1 Reserved=0 All bits shall be set to 0 when an Extension Component is not encoded signifying GLN only.

*Max. decimal value range of Company Prefix and Location Reference fields vary according to contents of the Partition field.

Table 11. The EPC SGLN-96 bit allocation, header, and maximum decimal values.

- *Header* is 8-bits, with a binary value of 0011 0010.
- *Filter Value* is not part of the GLN or EPC identifier, but is used for fast filtering and pre-selection of basic location types. The Filter Values for an SGLN is shown in Table 12 below.

Type	Binary Value
All Others	000
Physical Location	001
Reserved	010
Reserved	011
Reserved	100
Reserved	101
Reserved	110
Reserved	111

Table 12. SGLN Filter Values .

1008

1009 • *Partition* is an indication of where the subsequent Company Prefix and Location
 1010 Reference numbers are divided. This organization matches the structure in the
 1011 EAN.UCC GLN in which the Company Prefix added to the Location Reference
 1012 number totals 12 digits, yet the Company Prefix may vary from 6 to 12 digits and the
 1013 Location Reference number from 6 to 0 digit(s). The available values of *Partition* and
 1014 the corresponding sizes of the *Company Prefix* and *Location Reference* fields are
 1015 defined in Table 13.

1016 • *Company Prefix* contains a literal embedding of the EAN.UCC Company Prefix.

1017 • *Location Reference*, if present, encodes the GLN Location Reference number.

1018 • *Extension Component* contains a serial number. If an *Extension Component* is not used
 1019 this value shall be set to a binary value of 0 0000 0000 0000 0000 0000 0000 0000
 1020 0000 0000 0000. The SGLN-96 encoding is only capable of representing integer-
 1021 valued Extension Components with limited range. The EAN.UCC specifications
 1022 permit a broader range of Extension Components. The EAN.UCC-128 barcode
 1023 symbology provides for a 20-character alphanumeric Extension Component to be
 1024 associated with a GLN using Application Identifier (AI) 254 [EAN.UCCGS]. It is
 1025 possible to convert between the Extension Component in the SGLN-96 tag encoding
 1026 and the Extension Component in AI 254 barcodes under certain conditions.
 1027 Specifically, such interconversion is possible when the alphanumeric Extension
 1028 Component in AI 254 happens to consist only of digits, with no leading zeros, and
 1029 whose value when interpreted as an integer falls within the range limitations of the
 1030 SGLN-96 tag encoding. These considerations are reflected in the encoding and
 1031 decoding procedures below.

1032

Partition Value (P)	Company Prefix		Location 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
5	24	7	17	5
6	20	6	21	6

1033

Table 13. SGLN Partitions.

3.7.1.1 SGLN-96 Encoding Procedure

The following procedure creates an SGLN-96 encoding.

Given:

- An EAN.UCC GLN consisting of digits $d_1d_2\dots d_{13}$
- The length L of the Company Prefix portion of the GLN
- An Extension Component S where $0 \leq S < 2^{40}$, or an EAN.UCC-128 Application Identifier 254 consisting of characters $s_1s_2\dots s_K$. When the Extension Component S is 0, the Encoding will be considered as a GLN only.

- A Filter Value F where $0 \leq F < 8$

Procedure:

1. Look up the length L of the Company Prefix in the “Company Prefix Digits” column of the Partition Table (Table 13) to determine the Partition Value, P , the number of bits M in the Company Prefix field, and the number of bits N in the Location Reference field. If L is not found in any row of Table 13, stop: this GLN cannot be encoded in an SGLN-96.
2. Construct the Company Prefix by concatenating digits $d_1d_2\dots d_L$ and considering the result to be a decimal integer, C .
3. If $L < 12$ construct the Location Reference by concatenating digits $d_{(L+1)}d_{(L+2)}\dots d_{12}$ and considering the result to be a decimal integer, I . If $L = 12$ set b_{41} to 0 since there is no Location Reference digit.
4. When the Extension Component is provided directly as an integer S where $0 \leq S < 2^{40}$, proceed to Step 5. Otherwise, when the Extension Component is provided as an EAN.UCC-128 Application Identifier 254 consisting of characters $s_1s_2\dots s_K$, construct the Extension Component by concatenating characters $s_1s_2\dots s_K$. If any of these characters is not a digit, stop: this Extension Component cannot be encoded in the SGLN-96 encoding. Also, if $K > 1$ and $s_1 = 0$, stop: this Extension Component cannot be encoded in the SGLN-96 encoding (because leading zeros are not permitted except in the case where the Extension Component consists of a single zero digit). Otherwise, consider the result to be a decimal integer, S . If $S \geq 2^{40}$, stop: this Extension Component cannot be encoded in the SGLN-96 encoding.
5. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header 00110010 (8 bits), Filter Value F (3 bits), Partition Value P from Step 1 (3 bits), Company Prefix C from Step 2 (M bits), Location Reference I from Step 3 (N bits), Extension Component S from Step 4 (41 bits). Note that $M+N = 41$ bits for all P .

3.7.1.2 SGLN-96 Decoding Procedure

Given:

- An SGLN-96 as a 96-bit bit string $00110010b_{87}b_{86}\dots b_0$ (where the first eight bits 00110010 are the header)

1072 Yields:

1073 • An EAN.UCC GLN

1074 • An Extension Component

1075 • A Filter Value

1076 Procedure:

1077 1. Bits $b_{87}b_{86}b_{85}$, considered as an unsigned integer, are the Filter Value.

1078 2. Extract the Partition Value P by considering bits $b_{84}b_{83}b_{82}$ as an unsigned integer. If

1079 $P = 7$, stop: this bit string cannot be decoded as an SGLN-96.

1080 3. Look up the Partition Value P in Table 13 to obtain the number of bits M in the Company

1081 Prefix and the number of digits L in the Company Prefix.

1082 4. Extract the Company Prefix C by considering bits $b_{81}b_{80}\dots b_{(82-M)}$ as an unsigned integer.

1083 If this integer is greater than or equal to 10^L , stop: the input bit string is not a legal SGLN-96

1084 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\dots p_L$, adding leading

1085 zeros as necessary to make up L digits in total.

1086 5. If $L < 12$ extract the Location Reference by considering bits $b_{(81-M)}b_{(80-M)}\dots b_{41}$ as an

1087 unsigned integer. If this integer is greater than or equal to $10^{(12-L)}$, stop: the input bit string

1088 is not a legal SGLN-96 encoding. Otherwise, convert this integer to a $(12-L)$ -digit decimal

1089 number $i_1i_2\dots i_{(12-L)}$, adding leading zeros as necessary to make $(12-L)$ digits.

1090 6. Construct a 12-digit number $d_1d_2\dots d_{12}$ where $d_1d_2\dots d_L = p_1p_2\dots p_L$ from Step 4, and if $L <$

1091 12 $d_{(L+1)}d_{(L+2)}\dots d_{12} = i_1i_2\dots i_{(12-L)}$ from Step 5.

1092 7. Calculate the check digit $d_{13} = (-3(d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12}) - (d_1 + d_3 + d_5 + d_7 + d_9 +$

1093 $d_{11})) \bmod 10$.

1094 8. The EAN.UCC GLN is the concatenation of digits from Steps 6 and 7: $d_1d_2\dots d_{13}$.

1095 9. Bits $b_{40}b_{39}\dots b_0$, considered as an unsigned integer, are the *Extension Component*.

1096 10. (Optional) If it is desired to represent the Extension Component as a EAN.UCC-128

1097 Application Identifier 254, convert the integer from Step 9 to a decimal string with no

1098 leading zeros. If the integer in Step 9 is zero, convert it to a string consisting of the single

1099 character "0".

1100 **3.7.2 SGLN-195**

1101 In addition to a Header, the SGLN-195 is composed of five fields: the *Filter Value*, *Partition*,

1102 *Company Prefix*, *Location Reference*, and *Extension Component*, as shown in Table 14.

	Header	Filter Value	Partition	Company Prefix	Location Reference	Extension Component
SGLN-195	8	3	3	20-40	21-1	140
	0011 1001 (Binary value)	(Refer to Table 12 for values)	(Refer to Table 13 for values)	999,999 – 999,999,999 (Max. decimal range*)	999,999 – 0 (Max. decimal range*)	Up to 20 alphanumeric characters If the Extension Component is not used this value must be set to 0110000 followed by 133 binary 0 bits.

*Max. decimal value range of Company Prefix and Location Reference fields vary according to contents of the Partition field.

Table 14. The EPC SGLN-195 bit allocation, header, and maximum decimal values.

- *Header* is 8-bits, with a binary value of 0011 1001.
- *Filter Value* is not part of the GLN or EPC identifier, but is used for fast filtering and pre-selection of basic location types. The Filter Values for an SGLN is shown in Table 12.
- *Partition* is an indication of where the subsequent Company Prefix and Location Reference numbers are divided. This organization matches the structure in the EAN.UCC GLN in which the Company Prefix added to the Location Reference number totals 12 digits, yet the Company Prefix may vary from 6 to 12 digits and the Location Reference number from 6 to 0 digit(s). The available values of *Partition* and the corresponding sizes of the *Company Prefix* and *Location Reference* fields are defined in Table 13.
- *Company Prefix* contains a literal embedding of the EAN.UCC Company Prefix.
- *Location Reference*, if present, encodes the GLN Location Reference number.
- *ExtensionComponent* contains a serial number. If an *Extension Component* is not used signifying a GLN only, then this value shall be set to binary 0110000 followed by 133 binary “0” bits. SGLN.-195 encoding is capable of representing alphanumeric Extension Component of up to 20 characters, permitting the full range of Extension Component available in the EAN.UCC-128 barcode symbology using Application Identifier (AI) 254 [EAN.UCCGS]. See Appendix G for permitted values.

3.7.2.1 SGLN-195 Encoding Procedure

The following procedure creates an SGLN-195 encoding.

Given:

- An EAN.UCC GLN consisting of digits $d_1d_2\dots d_{13}$
- The length L of the Company Prefix portion of the GLN

1130 • An EAN.UCC-128 Application Identifier 254 consisting of characters $s_1s_2\dots s_K$, where K
1131 ≤ 20 .. If the Application Identifier 254 consists of a single character 0 where $K=1$, this
1132 Encoding is considered to be a GLN only.

1133 • A Filter Value F where $0 \leq F < 8$

1134 Procedure:

- 1135 1. Look up the length L of the Company Prefix in the “Company Prefix Digits” column of
1136 the Partition Table (Table 13) to determine the Partition Value, P , the number of bits M in
1137 the Company Prefix field, and the number of bits N in the Location Reference field. If L is
1138 not found in any row of Table 13, stop: this GLN cannot be encoded in an SGLN-195.
- 1139 2. Construct the Company Prefix by concatenating digits $d_1d_2\dots d_L$ and considering the result
1140 to be a decimal integer, C .
- 1141 3. If $L < 12$ construct the Location Reference by concatenating digits $d_{(L+1)}d_{(L+2)}\dots d_{12}$ and
1142 considering the result to be a decimal integer, I . If $L = 12$ set b_{140} to 0 since there is no
1143 Location Reference digit.
- 1144 4. Check that each of the characters $s_1s_2\dots s_K$ is one of the 82 characters listed in the table
1145 in Appendix G. If this is not the case, stop: this character string cannot be encoded as an
1146 SGLN-195. Otherwise construct the Extension Component by concatenating the 7-bit code,
1147 as given in Appendix G, for each of the characters $s_1s_2\dots s_K$, yielding $7K$ bits total. If $K < 20$,
1148 concatenate additional zero bits to the right to make a total of 140 bits.
- 1149 5. Construct the final encoding by concatenating the following bit fields, from most
1150 significant to least significant: Header 00111001 (8 bits), Filter Value F (3 bits), Partition
1151 Value P from Step 1 (3 bits), Company Prefix C from Step 2 (M bits), Location Reference I
1152 from Step 3 (N bits), Extension Component S from Step 4 (140 bits). Note that $M+N =$
1153 41 bits for all P .

1154 3.7.2.2 SGLN-195 Decoding Procedure

1155 Given:

1156 • An SGLN-195 as a 195-bit bit string $00111001b_{186}b_{185}\dots b_0$ (where the first eight bits
1157 00111001 are the header)

1158 Yields:

- 1159 • An EAN.UCC GLN
- 1160 • An Extension Component
- 1161 • A Filter Value

1162 Procedure:

- 1163 1. Bits $b_{186}b_{185}b_{184}$, considered as an unsigned integer, are the Filter Value.
- 1164 2. Extract the Partition Value P by considering bits $b_{183}b_{182}b_{181}$ as an unsigned integer. If
1165 $P = 7$, stop: this bit string cannot be decoded as an SGLN-195.
- 1166 3. Look up the Partition Value P in Table 13 to obtain the number of bits M in the Company
1167 Prefix and the number of digits L in the Company Prefix.

1168 4. Extract the Company Prefix C by considering bits $b_{180}b_{179}\dots b_{(181-M)}$ as an unsigned
1169 integer. If this integer is greater than or equal to 10^L , stop: the input bit string is not a legal
1170 SGLN-195 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\dots p_L$,
1171 adding leading zeros as necessary to make up L digits in total.

1172 5. When $L < 12$ extract the Location Reference by considering bits $b_{(180-M)}b_{(179-M)}\dots b_{140}$ as
1173 an unsigned integer. If this integer is greater than or equal to $10^{(12-L)}$, stop: the input bit
1174 string is not a legal SGLN-195 encoding. Otherwise, convert this integer to a $(12-L)$ -digit
1175 decimal number $i_1i_2\dots i_{(12-L)}$, adding leading zeros as necessary to make $(12-L)$ digits.

1176 6. Construct a 12-digit number $d_1d_2\dots d_{12}$ where $d_1d_2\dots d_L = p_1p_2\dots p_L$ from Step 4, and if $L <$
1177 12 $d_{(L+1)}d_{(L+2)}\dots d_{12} = i_2i_3\dots i_{(12-L)}$ from Step 5.

1178 7. Calculate the check digit $d_{13} = (-3(d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12}) - (d_1 + d_3 + d_5 + d_7 + d_9 +$
1179 $d_{11})) \bmod 10$.

1180 8. The EAN.UCC GLN is the concatenation of digits from Steps 6 and 7: $d_1d_2\dots d_{13}$.

1181 9. Divide the remaining bits $b_{139}b_{138}\dots b_0$ into 7-bit segments. The result should consist of K
1182 non-zero binary segments followed by $20-K$ binary zero segments. If this is not the case,
1183 stop: this bit string cannot be decoded as an SGLN-195. Otherwise, look up each of the
1184 non-zero 7-bit segments in Appendix G to obtain a corresponding character. If any of the
1185 non-zero 7-bit segments has a value that is not in Appendix G, stop: this bit string cannot be
1186 decoded as an SGLN-195. If $K=1$ and $s_1=0$, then this indicates a GLN only with no
1187 *Extension Component*. Otherwise, the K characters so obtained, considered as a character
1188 string $s_1s_2\dots s_K$, is the value of the EAN.UCC AI 254.

1189 10. The EAN.UCC SGLN-195 is the concatenation of the digits from Steps 6 and 7 and the
1190 characters from Step 9. : $d_1d_2\dots d_{13} s_1s_2\dots s_K$

1191

1192 **3.8 Global Returnable Asset Identifier (GRAI)**

1193 The EPC Tag Encoding scheme for GRAI permits the direct embedding of EAN.UCC
1194 System standard GRAI on EPC tags. In all cases, the check digit is not encoded.

1195 **3.8.1 GRAI-96**

1196 In addition to a Header, the GRAI-96 is composed of five fields: the *Filter Value*, *Partition*,
1197 *Company Prefix*, *Asset Type*, and *Serial Number*, as shown in Table 15.

	Header	Filter Value	Partition	Company Prefix	Asset Type	Serial Number
GRAI-96	8	3	3	20-40	24-4	38
	0011 0011 (Binary value)	(Refer to Table 16 for values)	(Refer to Table 17 for values)	999,999 – 999,999,999,999 (Max. decimal range*)	999,999 – 0 (Max. decimal range*)	274,877,906,943 (Max. decimal value)

*Max. decimal value range of Company Prefix and Asset Type fields vary according to contents of the Partition field.

Table 15. The EPC GRAI-96 bit allocation, header, and maximum decimal values.

- *Header* is 8-bits, with a binary value of 0011 0011.
- *Filter Value* is not part of the GRAI or EPC identifier, but is used for fast filtering and pre-selection of basic asset types. The Filter Values for 96-bit and 170-bit GRAI are the same. See Table 16.

Type	Binary Value
All Others	000
Reserved	001
Reserved	010
Reserved	011
Reserved	100
Reserved	101
Reserved	110
Reserved	111

Table 16. GRAI Filter Values

- *Partition* is an indication of where the subsequent Company Prefix and Asset Type numbers are divided. This organization matches the structure in the EAN.UCC GRAI in which the Company Prefix added to the Asset Type number totals 12 digits, yet the Company Prefix may vary from 6 to 12 digits and the Asset Type from 6 to 0 digit(s). The available values of *Partition* and the corresponding sizes of the *Company Prefix* and *Asset Type* fields are defined in Table 17.

Partition Value (<i>P</i>)	Company Prefix		Asset Type	
	Bits (<i>M</i>)	Digits (<i>L</i>)	Bits (<i>N</i>)	Digits
0	40	12	4	0
1	37	11	7	1
2	34	10	10	2
3	30	9	14	3
4	27	8	17	4
5	24	7	20	5
6	20	6	24	6

Table 17. GRAI Partitions.

- *Company Prefix* contains a literal embedding of the EAN.UCC Company Prefix.
- *Asset Type*, if present, encodes the GRAI Asset Type number.
- *Serial Number* contains a serial number. The 96-bit tag encodings are only capable of representing a subset of Serial Numbers allowed in the General EAN.UCC Specifications. The capacity of this mandatory serial number is less than the maximum EAN.UCC System specification for serial number, no leading zeros are permitted, and only numbers are permitted.

3.8.1.1 GRAI-96 Encoding Procedure

The following procedure creates a GRAI-96 encoding.

Given:

- An EAN.UCC GRAI consisting of digits $0d_2d_3\dots d_K$, where $15 \leq K \leq 30$.
- The length L of the Company Prefix portion of the GRAI
- A Filter Value F where $0 \leq F < 8$

Procedure:

1. Look up the length L of the Company Prefix in the “Company Prefix Digits” column of the Partition Table (Table 17) to determine the Partition Value, P , the number of bits M in the Company Prefix field, and the number of bits N in Asset Type field. If L is not found in any row of Table 17, stop: this GRAI cannot be encoded in a GRAI-96.
2. Construct the Company Prefix by concatenating digits $d_2d_3\dots d_{(L+1)}$ and considering the result to be a decimal integer, C .

1234 3. If $L < 12$ construct the Asset Type by concatenating digits $d_{(L+2)}d_{(L+3)}\dots d_{13}$ and
 1235 considering the result to be a decimal integer, I . Otherwise set bits $b_{41}, b_{40}, b_{39}, b_{38}$ to 0000.

1236 4. Construct the Serial Number by concatenating digits $d_{15}d_{16}\dots d_K$. If any of these
 1237 characters is not a digit, stop: this GRAI cannot be encoded in the GRAI-96 encoding.
 1238 Otherwise, consider the result to be a decimal integer, S . If $S \geq 2^{38}$, stop: this GRAI cannot
 1239 be encoded in the GRAI-96 encoding. Also, if $K > 15$ and $d_{15} = 0$, stop: this GRAI cannot be
 1240 encoded in the GRAI-96 encoding (because leading zeros are not permitted except in the
 1241 case where the Serial Number consists of a single zero digit).

1242 5. Construct the final encoding by concatenating the following bit fields, from most
 1243 significant to least significant: Header 00110011 (8 bits), Filter Value F (3 bits), Partition
 1244 Value P from Step 1 (3 bits), Company Prefix C from Step 2 (M bits), Asset Type I from
 1245 Step 3 (N bits), Serial Number S from Step 4 (38 bits). Note that $M+N = 44$ bits for all P .

1246 3.8.1.2 GRAI-96 Decoding Procedure

1247 Given:

- 1248 • An GRAI-96 as a 96-bit bit string $00110011b_{87}b_{86}\dots b_0$ (where the first eight bits
 1249 00110011 are the header)

1250 Yields:

- 1251 • An EAN.UCC GRAI
- 1252 • A Filter Value

1253 Procedure:

- 1254 1. Bits $b_{87}b_{86}b_{85}$, considered as an unsigned integer, are the Filter Value.
- 1255 2. Extract the Partition Value P by considering bits $b_{84}b_{83}b_{82}$ as an unsigned integer. If
 1256 $P = 7$, stop: this bit string cannot be decoded as a GRAI-96.
- 1257 3. Look up the Partition Value P in Table 17 to obtain the number of bits M in the Company
 1258 Prefix and the number of digits L in the Company Prefix.
- 1259 4. Extract the Company Prefix C by considering bits $b_{81}b_{80}\dots b_{(82-M)}$ as an unsigned integer.
 1260 If this integer is greater than or equal to 10^L , stop: the input bit string is not a legal GRAI-96
 1261 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\dots p_L$, adding leading
 1262 zeros as necessary to make up L digits in total.
- 1263 5. If $L < 12$ extract the Asset Type by considering bits $b_{(81-M)}b_{(80-M)}\dots b_{38}$ as an unsigned
 1264 integer. If this integer is greater than or equal to $10^{(12-L)}$, stop: the input bit string is not a
 1265 legal GRAI-96 encoding. Otherwise, convert this integer to a $(12-L)$ -digit decimal number
 1266 $i_1i_2\dots i_{(12-L)}$, adding leading zeros as necessary to make $(12-L)$ digits.
- 1267 6. Construct a 13-digit number $0d_2d_3\dots d_{13}$ where $d_2d_3\dots d_{(L+1)} = p_1p_2\dots p_L$ from Step 4, and
 1268 $d_{(L+2)}d_{(L+3)}\dots d_{13} = i_1i_2\dots i_{(12-L)}$ from Step 5.
- 1269 7. Calculate the check digit $d_{14} = (-3(d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13}) - (d_2 + d_4 + d_6 + d_8 + d_{10}$
 1270 $+ d_{12})) \bmod 10$.

8. Extract the Serial Number by considering bits $b_{37}b_{36}...b_0$ as an unsigned integer. Convert this integer to a decimal number $d_{15}d_{16}...d_K$, with no leading zeros (exception: if the integer is equal to zero, convert it to a single zero digit).

9. The EAN.UCC GRAI is the concatenation of a single zero digit and the digits from Steps 6, 7, and 8: $0d_2d_3...d_K$.

3.8.2 GRAI-170

In addition to a Header, the GRAI-170 is composed of five fields: the *Filter Value*, *Partition*, *Company Prefix*, *Asset Type*, and *Serial Number*, as shown in Table 18.

	Header	Filter Value	Partition	Company Prefix	Asset Type	Serial Number
GRAI-170	8	3	3	20-40	24-4	112
	0011 0111 (Binary value)	(Refer to Table 16 for values)	(Refer to Table 17 for values)	999,999 – 999,999,999,999 (Max. decimal range*)	999,999 – 0 (Max. decimal range*)	Up to 16 alphanumeric characters

*Max. decimal value range of Company Prefix and Asset Type fields vary according to contents of the Partition field.

Table 18. The EPC GRAI-170 bit allocation, header, and maximum decimal values.

- *Header* is 8-bits, with a binary value of 0011 0111
- *Filter Value* is not part of the GRAI or EPC identifier, but is used for fast filtering and pre-selection of basic asset types. The Filter Values for 96-bit and 170-bit GRAI are the same. See Table 16. This specification anticipates that valuable Filter Values will be determined once there has been time to consider the possible use cases.
- *Partition* is an indication of where the subsequent Company Prefix and Asset Type numbers are divided. This organization matches the structure in the EAN.UCC GRAI in which the Company Prefix added to the Asset Type number totals 12 digits, yet the Company Prefix may vary from 6 to 12 digits and the Asset Type from 6 to 0 digit(s). The available values of *Partition* and the corresponding sizes of the *Company Prefix* and *Asset Type* fields for 96-bit and 170-bit GRAI are defined in Table 17.
- *Company Prefix* contains a literal embedding of the EAN.UCC Company Prefix.
- *Asset Type*, if present, encodes the GRAI Asset Type number.
- *Serial Number* contains a mandatory alphanumeric serial number. The GRAI-170 encoding is capable of representing alphanumeric serial numbers of up to 16 characters,

1297 permitting the full range of serial numbers available in the EAN.UCC-128 barcode
1298 symbology using Application Identifier (AI) 8003 [EAN.UCCGS].

1299 **3.8.2.1 GRAI-170 Encoding Procedure**

1300 The following procedure creates a GRAI-170 encoding.

1301 Given:

- 1302 • An EAN.UCC GRAI consisting of digits $0d_2d_3\dots d_{14}$, and a variable length alphanumeric
- 1303 serial number $s_{15}s_{16}\dots s_K$ where $15 \leq K \leq 30$.
- 1304 • The length L of the Company Prefix portion of the GRAI
- 1305 • A Filter Value F where $0 \leq F < 8$

1306 Procedure:

- 1307 1. Look up the length L of the Company Prefix in the “Company Prefix Digits” column of
- 1308 the Partition Table (Table 17) to determine the Partition Value, P , the number of bits M in
- 1309 the Company Prefix field, and the number of bits N in Asset Type field. If L is not found in
- 1310 any row of Table 17, stop: this GRAI cannot be encoded in a GRAI-96.
- 1311 2. Construct the Company Prefix by concatenating digits $d_2d_3\dots d_{(L+1)}$ and considering the
- 1312 result to be a decimal integer, C .
- 1313 3. If $L < 12$ construct the Asset Type by concatenating digits $d_{(L+2)}d_{(L+3)}\dots d_{13}$ and
- 1314 considering the result to be a decimal integer, I . Otherwise set bits $b_{115}, b_{114}, b_{113}, b_{112}$ to 0000.
- 1315 4. Check that each of the characters $s_{15}s_{16}\dots s_K$ is one of the 82 characters listed in the table
- 1316 in Appendix G. If this is not the case, stop: this character string cannot be encoded as an
- 1317 GRAI-170. Otherwise construct the Serial Number by concatenating the 7-bit code, as given
- 1318 in Appendix G, for each of the characters $s_{15}s_{16}\dots s_K$, yielding $7*(K-14)$ bits total. If $K < 30$,
- 1319 concatenate additional zero bits to the right to make a total of 112 bits.
- 1320 5. Construct the final encoding by concatenating the following bit fields, from most
- 1321 significant to least significant: Header 00110111 (8 bits), Filter Value F (3 bits), Partition
- 1322 Value P from Step 1 (3 bits), Company Prefix C from Step 2 (M bits), Asset Type I from
- 1323 Step 3 (N bits), Serial Number S from Step 4 (112 bits). Note that $M+N = 44$ bits for all P .

1324 **3.8.2.2 GRAI-170 Decoding Procedure**

1325 Given:

- 1326 • An GRAI-170 as a 170-bit bit string $00110111b_{162}b_{161}\dots b_0$ (where the first eight bits
- 1327 00110111 are the header)

1328 Yields:

- 1329 • An EAN.UCC GRAI
- 1330 • A Filter Value

1331 Procedure:

- 1332 1. Bits $b_{162}b_{161}b_{160}$, considered as an unsigned integer, are the Filter Value.

1333 2. Extract the Partition Value P by considering bits $b_{159}b_{158}b_{157}$ as an unsigned integer. If
1334 $P = 7$, stop: this bit string cannot be decoded as a GRAI-170.

1335 3. Look up the Partition Value P in Table 17 to obtain the number of bits M in the Company
1336 Prefix and the number of digits L in the Company Prefix.

1337 4. Extract the Company Prefix C by considering bits $b_{156}b_{155}\dots b_{(157-M)}$ as an unsigned
1338 integer. If this integer is greater than or equal to 10^L , stop: the input bit string is not a legal
1339 GRAI-170 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\dots p_L$,
1340 adding leading zeros as necessary to make up L digits in total.

1341 5. If $L < 12$ extract the Asset Type by considering bits $b_{(156-M)}b_{(155-M)}\dots b_{112}$ as an unsigned
1342 integer. If this integer is greater than or equal to $10^{(12-L)}$, stop: the input bit string is not a
1343 legal GRAI-170 encoding. Otherwise, convert this integer to a $(12-L)$ -digit decimal number
1344 $i_1i_2\dots i_{(12-L)}$, adding leading zeros as necessary to make $(12-L)$ digits.

1345 6. Construct a 13-digit number $0d_2d_3\dots d_{13}$ where $d_2d_3\dots d_{(L+1)} = p_1p_2\dots p_L$ from Step 4, and if
1346 $L < 12$ $d_{(L+2)}d_{(L+3)}\dots d_{13} = i_1i_2\dots i_{(12-L)}$ from Step 5.

1347 7. Calculate the check digit $d_{14} = (-3(d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13}) - (d_2 + d_4 + d_6 + d_8 + d_{10}$
1348 $+ d_{12})) \bmod 10$.

1349 8. Divide the remaining bits $b_{111}b_{110}\dots b_0$ into 7-bit segments. This string should consist of
1350 K non-zero segments followed by $16-K$ zero segments. If this is not the case, stop: this bit
1351 string cannot be decoded as an GRAI-170. Otherwise, look up each of the non-zero 7-bit
1352 segments in Appendix G to obtain a corresponding character. If any of the non-zero 7-bit
1353 segments has a value that is not in Appendix G, stop: this bit string cannot be decoded as an
1354 GRAI-170. Otherwise, the first K characters considered as a character string is the serial
1355 number $s_{15}s_{16}\dots s_K$.

1356 9. The EAN.UCC GRAI is the concatenation of a single zero digit, the digits from Steps 6
1357 and 7 and the characters from Step 8. : $0d_2d_3\dots d_{14}s_{15}s_{16}\dots s_K$

1358

1359 3.9 Global Individual Asset Identifier (GIAI)

1360 The EPC Tag Encoding scheme for GIAI permits the direct embedding of EAN.UCC System
1361 standard GIAI codes on EPC tags.

1362 3.9.1 GIAI-96

1363 In addition to a Header, the EPC GIAI-96 is composed of four fields: the *Filter Value*,
1364 *Partition*, *Company Prefix*, and *Individual Asset Reference*, as shown in Table 19.

1365

	Header	Filter Value	Partition	Company Prefix	Individual Asset Reference
GIAI-96	8	3	3	20-40	62-42
	0011 0100 (Binary value)	(Refer to Table 20 for values)	(Refer to Table 21 for values)	999,999 – 999,999,999,999 (Max. decimal range*)	4,611,686,018,427,387,903 – 4,398,046,511,103 (Max. decimal range*)

*Max. decimal value range of Company Prefix and Individual Asset Reference fields vary according to contents of the Partition field.

Table 19. The EPC 96-bit GIAI bit allocation, header, and maximum decimal values.

- *Header* is 8-bits, with a binary value of 0011 0100.
- *Filter Value* is not part of the GIAI or EPC identifier, but is used for fast filtering and pre-selection of basic asset types. The Filter Values for 96-bit and 202-bit GIAI are the same. See Table 20.

Type	Binary Value
All Others	000
Reserved	001
Reserved	010
Reserved	011
Reserved	100
Reserved	101
Reserved	110
Reserved	111

Table 20. GIAI Filter Values

- The *Partition* is an indication of where the subsequent Company Prefix and Individual Asset Reference numbers are divided. This organization matches the structure in the EAN.UCC GIAI in which the Company Prefix may vary from 6 to 12 digits. The available values of *Partition* and the corresponding sizes of the *Company Prefix* and *Asset Reference* fields are defined in Table 21.

Partition Value (<i>P</i>)	Company Prefix		Individual Asset Reference	
	Bits (<i>M</i>)	Digits (<i>L</i>)	Bits (<i>N</i>)	Digits
0	40	12	42	12
1	37	11	45	13
2	34	10	48	14
3	30	9	52	15
4	27	8	55	16
5	24	7	58	17
6	20	6	62	18

Table 21. GIAI-96 Partitions.

- *Company Prefix* contains a literal embedding of the Company Prefix.
- *Individual Asset Reference* is a mandatory unique number for each instance. The EPC representation is only capable of representing a subset of asset references allowed in the General EAN.UCC Specifications. The capacity of this asset reference is less than the maximum EAN.UCC System specification for asset references, no leading zeros are permitted, and only numbers are permitted.

3.9.1.1 GIAI-96 Encoding Procedure

The following procedure creates a GIAI-96 encoding.

Given:

- An EAN.UCC GIAI consisting of digits $d_1d_2\dots d_K$, where $K \leq 30$.
- The length L of the Company Prefix portion of the GIAI
- A Filter Value F where $0 \leq F < 8$

Procedure:

1. Look up the length L of the Company Prefix in the “Company Prefix Digits” column of the Partition Table (Table 21) to determine the Partition Value, P , the number of bits M in the Company Prefix field, and the number of bits N in the Individual Asset Reference field. If L is not found in any row of Table 21, stop: this GIAI cannot be encoded in a GIAI-96.
2. Construct the Company Prefix by concatenating digits $d_1d_2\dots d_L$ and considering the result to be a decimal integer, C .
3. Construct the Individual Asset Reference by concatenating digits $d_{(L+1)}d_{(L+2)}\dots d_K$. If any of these characters is not a digit, stop: this GIAI cannot be encoded in the GIAI-96 encoding. Otherwise, consider the result to be a decimal integer, S . If $S \geq 2^N$, stop: this GIAI cannot be encoded in the GIAI-96 encoding. Also, if $K > L+1$ and $d_{(L+1)} = 0$, stop: this GIAI cannot be

1404 encoded in the GIAI-96 encoding (because leading zeros are not permitted except in the case
1405 where the Individual Asset Reference consists of a single zero digit).

1406 4. Construct the final encoding by concatenating the following bit fields, from most
1407 significant to least significant: Header 00110100 (8 bits), Filter Value F (3 bits), Partition
1408 Value P from Step 2 (3 bits), Company Prefix C from Step 3 (M bits), Individual Asset
1409 Number S from Step 4 (N bits). Note that $M+N = 82$ bits for all P .

1410 **3.9.1.2 GIAI-96 Decoding Procedure**

1411 Given:

- 1412 • A GIAI-96 as a 96-bit bit string $00110100b_{87}b_{86}\dots b_0$ (where the first eight bits
1413 00110100 are the header)

1414 Yields:

- 1415 • An EAN.UCC GIAI
- 1416 • A Filter Value

1417 Procedure:

- 1418 1. Bits $b_{87}b_{86}b_{85}$, considered as an unsigned integer, are the Filter Value.
- 1419 2. Extract the Partition Value P by considering bits $b_{84}b_{83}b_{82}$ as an unsigned integer. If
1420 $P = 7$, stop: this bit string cannot be decoded as a GIAI-96.
- 1421 3. Look up the Partition Value P in Table 21 to obtain the number of bits M in the Company
1422 Prefix and the number of digits L in the Company Prefix.
- 1423 4. Extract the Company Prefix C by considering bits $b_{81}b_{80}\dots b_{(82-M)}$ as an unsigned integer.
1424 If this integer is greater than or equal to 10^L , stop: the input bit string is not a legal GIAI-96
1425 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\dots p_L$, adding leading
1426 zeros as necessary to make up L digits in total.
- 1427 5. Extract the Individual Asset Reference by considering bits $b_{(81-M)}b_{(80-M)}\dots b_0$ as an
1428 unsigned integer. If this integer is greater than or equal to $10^{(30-L)}$, stop: the input bit string
1429 is not a legal GIAI-96 encoding. Otherwise, convert this integer to a decimal number
1430 $s_1s_2\dots s_J$, with no leading zeros (exception: if the integer is equal to zero, convert it to a single
1431 zero digit).
- 1432 6. Construct a K -digit number $d_1d_2\dots d_K$ where $d_1d_2\dots d_L = p_1p_2\dots p_L$ from Step 4, and
1433 $d_{(L+1)}d_{(L+2)}\dots d_K = s_1s_2\dots s_J$ from Step 5. This K -digit number, where $K \leq 30$, is the
1434 EAN.UCC GIAI.

1435 **3.9.2 GIAI-202**

1436 In addition to a Header, the EPC GIAI-202 is composed of four fields: the *Filter Value*,
1437 *Partition*, *Company Prefix*, and *Individual Asset Reference*, as shown in Table 22.

1438

	Header	Filter Value	Partition	Company Prefix	Individual Asset Reference
GIAI-202	8	3	3	20-40	168-126
	0011 1000 (Binary value)	(Refer to Table 20 for values)	(Refer to Table 21 for values)	999,999 – 999,999,999,999 (Max. decimal range*)	Up to 24 alphanumeric characters

1439

1440

1441

*Max. decimal value range of Company Prefix and Individual Asset Reference fields vary according to contents of the Partition field.

1442

Table 22. The EPC 202-bit GIAI bit allocation, header, and maximum decimal values.

1443

- *Header* is 8-bits, with a binary value of 0011 1000.

1444

1445

1446

- *Filter Value* is not part of the GIAI or EPC identifier, but is used for fast filtering and pre-selection of basic asset types. The Filter Values for 96-bit and 202-bit GIAI are the same. See Table 20.

1447

1448

1449

1450

- The *Partition* is an indication of the size of the subsequent Company Prefix. This organization matches the structure in the EAN.UCC GIAI in which the Company Prefix may vary from 6 to 12 digits. The available values of *Partition* and the corresponding size of the *Company Prefix* field is defined in Table 23.

1451

Partition Value (P)	Company Prefix		Individual Asset Reference	
	Bits (M)	Digits (L)	Bits (N)	Characters
0	40	12	126	18
1	37	11	133	19
2	34	10	140	20
3	30	9	147	21
4	27	8	154	22
5	24	7	161	23
6	20	6	168	24

1452

1453

Table 23. GIAI-202 Partitions.

1454

- *Company Prefix* contains a literal embedding of the EAN.UCC Company Prefix.

1455

- *Individual Asset Reference* contains a mandatory alphanumeric asset reference number.

1456

The GIAI-202 encoding is capable of representing alphanumeric serial numbers of up

1457

to 24 characters, permitting the full range of serial numbers available in the EAN.UCC-

1458

128 barcode symbology using Application Identifier (AI) 8004 [EAN.UCCGS].

1459

- *Company Prefix* and *Individual Asset Reference* should never total more than 30 characters.

1460

1461

3.9.2.1 GIAI-202 Encoding Procedure

1462

1463

The following procedure creates a GIAI-202 encoding.

1464

Given:

1465

- An EAN.UCC GIAI consisting of digits $d_1d_2d_3\dots d_L$, and a variable length alphanumeric serial number $s_{L+1}s_{L+2}\dots s_K$ where $L+1 \leq K \leq 30$.

1466

1467

- The length L of the Company Prefix portion of the GIAI

1468

- A Filter Value F where $0 \leq F < 8$

1469

Procedure:

1470

1. . Look up the length L of the Company Prefix in the “Company Prefix Digits” column of the Partition Table (Table 23) to determine the Partition Value, P , the number of bits M in the Company Prefix field, and the number of bits N in the Individual Asset Reference field. If L is not found in any row of Table 23, stop: this GIAI cannot be encoded in a GIAI-202.

1471

1472

1473

1474

2. Construct the Company Prefix by concatenating digits $d_1d_2\dots d_L$ and considering the result to be a decimal integer, C .

1475

1476

3. Check that each of the characters $s_{(L+1)}s_{(L+2)}\dots s_K$ is one of the 82 characters listed in the table in Appendix G. If this is not the case, stop: this character string cannot be encoded as an GIAI-202. Otherwise construct the Individual Asset Reference by concatenating the 7-bit code, as given in Appendix G, for each of the characters $s_{(L+1)}s_{(L+2)}\dots s_K$ yielding $7*(K-L)$ bits total. Concatenate additional zero bits to the right, if necessary, to make a total of $(188-M)$ bits, where M is the number of bits in the Company Prefix portion as determined in Step 1.

1477

1478

1479

1480

1481

1482

1483

4. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header 00111000 (8 bits), Filter Value F (3 bits), Partition Value P from Step 1 (3 bits), Company Prefix C from Step 2 (M bits), Individual Asset Number S from Step 3 ($188-M$ bits),

1484

1485

1486

1487

3.9.2.2 GIAI-202 Decoding Procedure

Given:

- A GIAI-202 as a 202-bit bit string $00111000b_{193}b_{192}\dots b_0$ (where the first eight bits 00111000 are the header)

Yields:

- An EAN.UCC GIAI
- A Filter Value

Procedure:

1. Bits $b_{193}b_{192}b_{191}$, considered as an unsigned integer, are the Filter Value.
2. Extract the Partition Value P by considering bits $b_{190}b_{189}b_{188}$ as an unsigned integer. If $P = 7$, stop: this bit string cannot be decoded as a GIAI-202.
3. Look up the Partition Value P in Table 23 to obtain the number of bits M in the Company Prefix and the number of digits L in the Company Prefix.
4. Extract the Company Prefix C by considering bits $b_{187}b_{186}\dots b_{(188-M)}$ as an unsigned integer. If this integer is greater than or equal to 10^L , stop: the input bit string is not a legal GIAI-202 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\dots p_L$, adding leading zeros as necessary to make up L digits in total.
5. Extract the Individual Asset Reference by dividing the remaining bits $b_{(187-M)}b_{(186-M)}\dots b_0$ into 7 bit segments beginning with the segment $b_{(187-M)}b_{(186-M)}\dots b_{(181-M)}$, and continuing as far as possible (there may be up to four bits left over at the end).. The result should consist of J non-zero segments followed by zero or more zero-valued segments, with any remaining bits also being zeros. If this is not the case, stop: this bit string cannot be decoded as a GIAI-202. Otherwise, look up each of the non-zero 7-bit segments in Appendix G to obtain a corresponding character. If any of the non-zero 7-bit segments has a value that is not in Appendix G, stop: this bit string cannot be decoded as a GIAI-202. Otherwise, the first J characters considered as a character string is the Asset Reference Number $s_{(1)}s_{(2)}\dots s_J$.
6. Construct a K -character string $s_1s_2\dots s_K$ where $s_1s_2\dots s_L = p_1p_2\dots p_L$ from Step 4, and where $s_{(L+1)}s_{(L+2)}\dots s_K = s_{(1)}s_{(2)}\dots s_J$ from Step 5. This K -character string, where $K \leq 30$, is the EAN.UCC GIAI.

3.10 DoD Tag Data Constructs

3.10.1 DoD-96

This tag data construct may be used to encode Class 1 tags for shipping goods to the United States Department of Defense by an entity who has already been assigned a CAGE (Commercial and Government Entity) code.

At the time of this writing, the details of what information to encode into these fields is explained in a document titled "United States Department of Defense Supplier's Passive

RFID Information Guide" that can be obtained at the United States Department of Defense's web site (<http://www.dodrfid.org/suppliergus.htm>).

The current encoding structure of DoD-96 Tag Data Construct is shown in Table 24 below.

	Header	Filter Value	Government Managed Identifier	Serial Number
DoD-96	8	4	48	36
	0010 1111 (Binary value)	(Consult proper US Dept. Defense document for details)	Encoded with supplier CAGE code in 8-bit ASCII format (Consult US Dept. Defense doc for details)	68,719,476,735 (Max. decimal value)

Table 24. The DoD-96 bit allocation, header, and maximum decimal values

4 URI Representation

This section defines standards for the encoding of the Electronic Product Code™ as a Uniform Resource Identifier (URI). The URI Encoding complements the EPC Tag Encodings defined for use within RFID tags and other low-level architectural components. URIs provide a means for application software to manipulate Electronic Product Codes in a way that is independent of any particular tag-level representation, decoupling application logic from the way in which a particular Electronic Product Code was obtained from a tag.

Explanation (non-normative): The pure identity URI for a given EPC is the same regardless of the encoding. For example, the following pure identity URI `urn:epc:id:sgtin:0064141.112345.400` is the same regardless of whether it is encoded into a tag as an SGTIN-96 or an SGTIN-198. Other representations than the pure identity URI for use above the reader or middleware layer shall not be used, because they can lead to misinterpretations in the information system. Exclusively on the reader layer and below the encoding schemes including header, filter value and partition must be considered for filtering or writing processes.

This section defines four categories of URI. The first are URIs for pure identities, sometimes called "canonical forms." These contain only the unique information that identifies a specific physical object, and are independent of tag encodings. The second category is URIs that represent specific tag encodings. These are used in software applications where the encoding scheme is relevant, as when commanding software to write a tag. The third category is URIs that represent patterns, or sets of EPCs. These are used when instructing software how to filter tag data. The last category is a URI representation for raw tag information, generally used only for error reporting purposes.

All categories of URIs are represented as Uniform Resource Names (URNs) as defined by [RFC2141], where the URN Namespace is `epc`.

1555 This section complements Section 3, EPC Bit-level Encodings, which specifies the currently
1556 defined tag-level representations of the Electronic Product Code.

1557 **4.1 URI Forms for Pure Identities**

1558 (This section is non-normative; the formal specifications for the URI types are given in
1559 Sections 4.2.4 and 5.)

1560 URI forms are provided for pure identities, which contain just the EPC fields that serve to
1561 distinguish one object from another. These URIs take the form of Uniform Resource Names
1562 (URNs), with a different URN namespace allocated for each pure identity type.

1563 For the EPC General Identifier (Section 2.1.1), the pure identity URI representation is as
1564 follows:

1565 `urn:epc:id:gid:GeneralManagerNumber.ObjectClass.SerialNumber`

1566 In this representation, the three fields *GeneralManagerNumber*, *ObjectClass*, and
1567 *SerialNumber* correspond to the three components of an EPC General Identifier as
1568 described in Section 2.1.1. In the URI representation, each field is expressed as a decimal
1569 integer, with no leading zeros (except where a field's value is equal to zero, in which case a
1570 single zero digit is used).

1571 There are also pure identity URI forms defined for identity types corresponding to certain
1572 types within the EAN.UCC System family of codes as defined in Section 2.1.2; namely, the
1573 Serialized Global Trade Item Number (SGTIN), the Serial Shipping Container Code (SSCC),
1574 the Serialized Global Location Number (SGLN), the Global Reusable Asset Identifier
1575 (GRAI), and the Global Individual Asset Identifier (GIAI). The URI representations
1576 corresponding to these identifiers are as follows:

1577 `urn:epc:id:sgtin:CompanyPrefix.ItemReference.SerialNumber`

1578 `urn:epc:id:sscc:CompanyPrefix.SerialReference`

1579 `urn:epc:id:sgln:CompanyPrefix.LocationReference.ExtensionComponent`

1580 `urn:epc:id:grai:CompanyPrefix.AssetType.SerialNumber`

1581 `urn:epc:id:giai:CompanyPrefix.IndividualAssetReference`

1582 In these representations, *CompanyPrefix* corresponds to an EAN.UCC company prefix
1583 assigned to a manufacturer by GS1. (A UCC company prefix is converted to an EAN.UCC
1584 company prefix by adding one leading zero at the beginning.) The number of digits in this
1585 field is significant, and leading zeros are included as necessary.

1586 The *ItemReference*, *SerialReference*, *LocationReference*, and
1587 *AssetType* fields correspond to the similar fields of the GTIN, SSCC, GLN, and GRAI,
1588 respectively. Like the *CompanyPrefix* field, the number of digits in these fields is
1589 significant, and leading zeros are included as necessary. The number of digits in these fields,
1590 when added to the number of digits in the *CompanyPrefix* field, always total the same
1591 number of digits according to the identity type: 13 digits total for SGTIN, 17 digits total for
1592 SSCC, 12 digits total for SGLN, and 12 characters total for the GRAI. (The
1593 *ItemReference* field of the SGTIN includes the GTIN Indicator (PI) digit, appended to

1594 the beginning of the item reference. The *SerialReference* field includes the SSCC
 1595 Extension Digit (ED), followed by the serial reference. In no case are check digits included
 1596 in URI representations.)

1597 The *SerialNumber* field of the SGTIN and GRAI, the *ExtensionComponent* of the
 1598 SGLN, as well as the *IndividualAssetReference* field of the GIAI, may include
 1599 digits, letters, and certain other characters. In order for an SGTIN, SGLN, GRAI, or GIAI to
 1600 be encoded on a 96-bit tag, however, these fields must consist only of digits with no leading
 1601 zeros. These restrictions are defined in the encoding procedures for these types, as well as in
 1602 Appendix F.

1603 An SGTIN, SSCC, etc in this form is said to be in SGTIN-URI form, SSCC-URI form, etc
 1604 form, respectively. Here are examples:

1605 urn:epc:id:sgtin:0652642.800031.400
 1606 urn:epc:id:sscc:0652642.0123456789
 1607 urn:epc:id:sgln:0652642.12345.40 (Use this form when Extension
 1608 Component is used)
 1609 urn:epc:id:sgln:0652642.12345.0 (Use this form when Extension
 1610 Component is not used)
 1611 urn:epc:id:grai:0652642.12345.1234
 1612 urn:epc:id:giai:0652642.123456

1613 Referring to the first example, the corresponding GTIN-14 code is 80652642000311. This
 1614 divides as follows: the first digit (8) is the PI digit, which appears as the first digit of the
 1615 *ItemReference* field in the URI, the next seven digits (0652642) are the
 1616 *CompanyPrefix*, the next five digits (00031) are the remainder of the *ItemReference*,
 1617 and the last digit (1) is the check digit, which is not included in the URI.

1618 Referring to the second example, the corresponding SSCC is 006526421234567896 and the
 1619 last digit (6) is the check digit, not included in the URI.

1620 Referring to the third and fourth examples, the corresponding GLN is 0652642123458,
 1621 where the last digit (8) is the check digit, not included in the URI.

1622 Referring to the fifth example, the corresponding GRAI is 006526421234581234, where the
 1623 digit (8) is the check digit, not included in the URI.

1624 Referring to the sixth example, the corresponding GIAI is 0652642123456. (GIAI codes do
 1625 not include a check digit.)

1626 Note that all six URI forms have an explicit indication of the division between the company
 1627 prefix and the remainder of the code. This is necessary so that the URI representation may
 1628 be converted into tag encodings. In general, the URI representation may be converted to the
 1629 corresponding EAN.UCC numeric form (by combining digits and calculating the check
 1630 digit), but converting from the EAN.UCC numeric form to the corresponding URI
 1631 representation requires independent knowledge of the length of the company prefix.

1632 For the DoD identifier as defined in Section 3.9, the pure identity URI representation is as
 1633 follows:

1634 `urn:epc:id:usdod:CAGECodeOrDODAAC.serialNumber`

1635 where *CAGECodeOrDODAAC* is the five-character CAGE code or six-character DoDAAC,
1636 and *serialNumber* is the serial number represented as a decimal integer with no leading
1637 zeros (except that a serial number whose value is zero should be represented as a single zero
1638 digit). Note that a space character is never included as part of *CAGECodeOrDODAAC* in the
1639 URI form, even though on a 96-bit tag a space character is used to pad the five-character
1640 CAGE code to fit into the six-character field on the tag.

1641

1642 **4.2 URI Forms for Related Data Types**

1643 (This section is non-normative; the formal specifications for the URI types are given in
1644 Sections 4.3 and Section 5.)

1645 There are several data types that commonly occur in applications that manipulate Electronic
1646 Product Codes, which are not themselves Electronic Product Codes but are closely related.
1647 This specification provides URI forms for those as well. The general form of the *epc* URN
1648 Namespace is

1649 `urn:epc:type:typeSpecificPart`

1650 The *type* field identifies a particular data type, and *typeSpecificPart* encodes
1651 information appropriate for that data type. Currently, there are three possibilities defined for
1652 *type*, discussed in the next three sections.

1653 **4.2.1 URIs for EPC Tags**

1654 In some cases, it is desirable to encode in URI form a specific tag encoding of an EPC. For
1655 example, an application may wish to report to an operator what kinds of tags have been read.
1656 In another example, an application responsible for programming tags needs to be told not
1657 only what Electronic Product Code to put on a tag, but also the encoding scheme to be used.
1658 Finally, applications that wish to manipulate any additional data fields on tags need some
1659 representation other than the pure identity forms.

1660 EPC Tag URIs are encoded by setting the *type* field to *tag*, with the entire URI having
1661 this form:

1662 `urn:epc:tag:EncName:EncodingSpecificFields`

1663 where *EncName* is the name of an EPC Tag Encoding scheme, and
1664 *EncodingSpecificFields* denotes the data fields required by that encoding scheme,
1665 separated by dot characters. Exactly what fields are present depends on the specific
1666 encoding scheme used.

1667 In general, there are one or more encoding schemes (and corresponding *EncName* values)
1668 defined for each pure identity type. For example, the SGTIN Identifier has two encodings
1669 defined: *sgtin-96* and *sgtin-198*, corresponding to the 96-bit encoding and the 198-
1670 bit encoding. Note that these encoding scheme names are in one-to-one correspondence with
1671 unique tag Header values, which are used to represent the encoding schemes on the tag itself.

1672 The *EncodingSpecificFields*, in general, include all the fields of the corresponding
1673 pure identity type, possibly with additional restrictions on numeric range, plus additional
1674 fields supported by the encoding. For example, all of the defined encodings for the
1675 Serialized GTIN include an additional Filter Value that applications use to do tag filtering
1676 based on object characteristics associated with (but not encoded within) an object's pure
1677 identity.

1678 Here is an example: a Serialized GTIN 96-bit encoding:

1679 urn:epc:tag:sgtin-96:3.0652642.800031.400

1680 In this example, the number 3 is the Filter Value.

1681 The tag URI for the DoD identifier is as follows:

1682 urn:epc:tag:tagType:filter.CAGECodeOrDODAAC.serialNumber

1683 where *tagType* is *usdod-96*, *filter* is the filter value represented as two decimal
1684 digits, and the other two fields are as defined above in 4.1.

1685

1686 4.2.2 URIs for Raw Bit Strings Arising From Invalid Tags

1687 Certain bit strings do not correspond to legal encodings. Here are several examples:

- 1688 • If the most significant bits of a bit string cannot be recognized as a valid EPC header, the
1689 bit-level pattern is not a legal EPC Tag Encoding.
- 1690 • If the most significant bits of a bit string are recognized as a valid EPC header, but the
1691 binary value of a field in the corresponding tag encoding is greater than the value that
1692 can be contained in the number of decimal digits in that field in the URI form, the bit
1693 level pattern is not a legal EPC Tag Encoding.
- 1694 • A Gen 2 Tag whose “toggle bit” is set to one (see Section 3.2) by definition does not
1695 contain an EPC Tag Encoding.

1696 While in these situations a bit string is not a legal EPC Tag Encoding, software may wish to
1697 report such invalid bit-level patterns to users or to other software. For such cases, a
1698 representation of invalid bit-level patterns as URIs is provided. The *raw* form of the URI has
1699 this general form:

1700 urn:epc:raw:BitLength.Value

1701 where *BitLength* is the number of bits in the invalid representation, and *Value* is the
1702 entire bit-level representation converted to a single hexadecimal number and preceded by the
1703 letter “x”. For example, this bit string:

1704 000000000000000000000000100100011010011011110101011011011111011101111

1705 which is invalid because no valid header begins with 0000 0000, corresponds to this raw
1706 URI:

1707 urn:epc:raw:64.x00001234DEADBEEF

In order to ensure that a given bit string has only one possible raw URI representation, the number of digits in the hexadecimal value is required to be equal to the *BitLength* divided by four and rounded up to the nearest whole number. Moreover, only uppercase letters are permitted for the hexadecimal digits A, B, C, D, E, and F.

It is intended that this URI form be used only when reporting errors associated with reading invalid tags and when representing partially written tag. It is *not* intended to be a general mechanism for communicating arbitrary bit strings for other purposes.

Explanation (non-normative): The reason for recommending against using the raw URI for general purposes is to avoid having an alternative representation for legal tag encodings.

Earlier versions of this specification described a decimal, as opposed to hexadecimal, version of the raw URI. This is still supported for back-compatibility, but its use is no longer recommended. The “x” character is included so that software may distinguish between the decimal and hexadecimal forms.

4.2.2.1 Use of the Raw URI with Gen 2 Tags

The EPC memory of a Gen 2 Tag may contain either an EPC Tag Encoding or a value from a different numbering system for which an ISO Application Family Identifier (AFI) has been assigned. The “toggle” bit (bit 17x) of EPC memory distinguishes between these two possibilities (see Section 3.2).

The Raw URI as described above is intended primarily to represent undecodable EPC Tag Encodings or partially written tags. For a Gen 2 Tag, therefore, the Raw URI described above is used only when the toggle bit is a zero, indicating that the tag is supposed to contain an EPC Tag Encoding.

For completeness, an alternative form of the Raw URI is provided to represent the contents of a UHF Class 1 Gen 2 Tag whose toggle bit is a one. It has the following form:

`urn:epc:raw:BitLength.AFI.Value`

where *BitLength* is the number of bits in the non-EPC representation (not including the AFI), AFI is the Application Family Identifier represented as a two-digit hexadecimal number and preceded by the letter “x”, and *Value* is the remainder of EPC memory converted to a single hexadecimal number and preceded by the letter “x”.

4.2.2.2 The Length Field of a Raw URI when using Gen 2 Tags (non-normative)

(This non-normative section explains a subtle interaction between the Raw URI and the length indication on Gen 2 Tags.)

Unlike earlier generations of RFID tags, the Gen 2 Tag is designed so that the length of the EPC Tag Encoding stored on the tag is not necessarily the same as the total length of EPC memory provided. The Gen 2 Specification provides a five-bit length indication, that indicates the length of the EPC memory to the nearest multiple of 16 bits (see Section 3.2.2).

Because of the way the EPC Tag Encoding aligns in the Gen 2 Tag’s EPC memory, the five-bit length indication does not necessarily indicate the length of the EPC Tag Encoding. This is because the length indication is limited to expressing multiples of 16 bits, including the

first 16 bits in the protocol control (PC) bits which is not part of the EPC Tag Encoding. For example, if a Gen 2 Tag contains an SGTIN-198 EPC, the EPC Tag Encoding is 198 bits, which means there are total of 214 bits is considered when calculating the length indicator (198 EPC Tag Encoding bits plus the 16 PC bits). The nearest round up length indicator value is 01101 (binary), which indicates a total length of 224 bits. Working in the other direction, if a length indicator of 01101 is read from a Gen 2 Tag, it indicates a total of 224 bits including the 16 PC bits, and therefore appears to indicate an EPC Tag Encoding of 208 bits.

This does not present a problem when a Gen 2 Tag contains a valid EPC. The procedures in Sections 5.3 and 5.4 use the header table in Section 3.1 to determine the length of the EPC, and discard any extra bits that may be implied by the length indication. When the contents of a Gen 2 Tag are converted to a Raw URI, however, the length indication on the tag is used to calculate the length in the URI. Therefore the length representation in the raw URI will have different bit length to the EPC Tag Encoding bits. Also one must consider the fact that value field in the raw URI may be different, because the values from Gen 2 tags may also include excess bits that are filled with zeros up to the word boundary.

For these and other reasons, Raw URIs should never be used within information systems to represent valid EPCs.

4.2.3 URIs for EPC Patterns

Certain software applications need to specify rules for filtering lists of tags according to various criteria. This specification provides a *pattern* URI form for this purpose. A pattern URI does not represent a single tag encoding, but rather refers to a set of tag encodings. A typical pattern looks like this:

`urn:epc:pat:sgtin-96:3.0652642.[102400-204700].*`

This pattern refers to any EPC SGTIN Identifier 96-bit tag, whose Filter field is 3, whose Company Prefix is 0652642, whose Item Reference is in the range $102400 \leq \text{itemReference} \leq 204700$, and whose Serial Number may be anything at all.

In general, there is a pattern form corresponding to each tag encoding form (Section 4.2.1), whose syntax is essentially identical except that ranges or the star (*) character may be used in each field.

For the SGTIN, SSCC, SGLN, GRAI and GIAI patterns, the pattern syntax slightly restricts how wildcards and ranges may be combined. Only two possibilities are permitted for the *CompanyPrefix* field. One, it may be a star (*), in which case the following field (*ItemReference*, *SerialReference*, *LocationReference*, *AssetType* or *IndividualAssetReference*) must also be a star. Two, it may be a specific company prefix, in which case the following field may be a number, a range, or a star. A range may not be specified for the *CompanyPrefix*.

Explanation (non-normative): Because the company prefix is variable length, a range may not be specified, as the range might span different lengths. When a particular company prefix is specified, however, it is possible to match ranges or all values of the following field, because its length is fixed for a given company prefix. The other case that is allowed is when

both fields are a star, which works for all tag encodings because the corresponding tag fields (including the Partition field, where present) are simply ignored.

The pattern URI for the DoD Construct is as follows:

`urn:epc:pat:tagType:filterPat.CAGECodeOrDODAACPat.serialNumberPat`

where *tagType* is as defined above in 4.2.1, *filterPat* is either a filter value, a range of the form [*lo-hi*], or a * character; *CAGECodeOrDODAACPat* is either a CAGE Code/DODAAC or a * character; and *serialNumberPat* is either a serial number, a range of the form [*lo-hi*], or a * character.

4.2.4 URIs for EPC Pure Identity Patterns

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 looks like this:

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

This pattern refers to any EPC SGTIN, whose Company Prefix is 0652642, 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.

In general, there is a pattern form corresponding to each pure identity form (Section 4.1), whose syntax is essentially identical except any number of fields starting at the right may be a star (*). This is more restrictive than tag patterns (Section 4.2.3), in that the star characters must occupy adjacent rightmost fields and the range syntax is not allowed at all.

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

`urn:epc:idpat:usdod:CAGECodeOrDODAACPat.serialNumberPat`

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

4.3 Syntax

The syntax of the EPC-URI and the URI forms for related data types are defined by the following grammar.

4.3.1 Common Grammar Elements

`NumericComponent ::= ZeroComponent | NonZeroComponent`

`ZeroComponent ::= "0"`

`NonZeroComponent ::= NonZeroDigit Digit*`

`PaddedNumericComponent ::= Digit+`

`Digit ::= "0" | NonZeroDigit`

```

1822 NonZeroDigit ::= "1" | "2" | "3" | "4"
1823                | "5" | "6" | "7" | "8" | "9"
1824 UpperAlpha  ::= "A" | "B" | "C" | "D" | "E" | "F" | "G"
1825                | "H" | "I" | "J" | "K" | "L" | "M" | "N"
1826                | "O" | "P" | "Q" | "R" | "S" | "T" | "U"
1827                | "V" | "W" | "X" | "Y" | "Z"
1828 LowerAlpha  ::= "a" | "b" | "c" | "d" | "e" | "f" | "g"
1829                | "h" | "i" | "j" | "k" | "l" | "m" | "n"
1830                | "o" | "p" | "q" | "r" | "s" | "t" | "u"
1831                | "v" | "w" | "x" | "y" | "z"
1832 OtherChar   ::= "!" | "'" | "(" | ")" | "*" | "+" | "," | "-"
1833                | "." | ":" | ";" | "=" | "_"
1834 UpperHexChar ::= Digit | "A" | "B" | "C" | "D" | "E" | "F"
1835 HexComponent ::= UpperHexChar+
1836 Escape      ::= "%" HexChar HexChar
1837 HexChar     ::= UpperHexChar | "a" | "b" | "c" | "d" | "e" | "f"
1838 GS3A3Char   ::= Digit | UpperAlpha | LowerAlpha | OtherChar
1839                | Escape
1840 GS3A3Component ::= GS3A3Char+

```

1841 The syntactic construct **GS3A3Component** is used to represent fields of EAN.UCC codes
 1842 that permit alphanumeric and other characters as specified in Figure 3A3-1 of the EAN.UCC
 1843 General Specifications (see Appendix G). Owing to restrictions on URN syntax as defined
 1844 by [RFC2141], not all characters permitted in the EAN.UCC General Specifications may be
 1845 represented directly in a URN. Specifically, the characters " (double quote), % (percent), &
 1846 (ampersand), / (forward slash), < (less than), > (greater than), and ? (question mark) are
 1847 permitted in the General Specifications but may not be included directly in a URN. To
 1848 represent one of these characters in a URN, escape notation must be used in which the
 1849 character is represented by a percent sign, followed by two hexadecimal digits that give the
 1850 ASCII character code for the character.

1851 **4.3.2 EPCGID-URI**

```

1852 EPCGID-URI ::= "urn:epc:id:gid:" 2*(NumericComponent ".")
1853 NumericComponent

```

1854 **4.3.3 SGTIN-URI**

```

1855 SGTIN-URI ::= "urn:epc:id:sgtin:" SGTINURIBody
1856 SGTINURIBody ::= 2*(PaddedNumericComponent ".") GS3A3Component

```

1857 The number of characters in the two **PaddedNumericComponent** fields must total 13
 1858 (not including any of the dot characters).

1859 The Serial Number field of the SGTIN-URI is expressed as a **GS3A3Component**, which
1860 permits the representation of all characters permitted in the EAN.UCC-128 Application
1861 Identifier 21 Serial Number according to the EAN.UCC General Specifications. SGTIN-
1862 URIs that are derived from 96-bit tag encodings, however, will have Serial Numbers that
1863 consist only of digits and which have no leading zeros. These limitations are described in
1864 the encoding procedures, and in Appendix F.

1865 **4.3.4 SSCC-URI**

1866 **SSCC-URI ::= "urn:epc:id:sscc:" SSCCURIBody**

1867 **SSCCURIBody ::= PaddedNumericComponent "."**

1868 **PaddedNumericComponent**

1869 The number of characters in the two **PaddedNumericComponent** fields must total 17
1870 (not including any of the dot characters).

1871 **4.3.5 SGLN-URI**

1872 **SGLN-URI ::= "urn:epc:id:sgln:" SGLNURIBody**

1873 **SGLNURIBody ::= 2*(PaddedNumericComponent ".") GS3A3Component**

1874 The number of characters in the two **PaddedNumericComponent** fields must total 12
1875 (not including any of the dot characters).

1876 The *GLN Extension Component* field of the SGLN-URI is expressed as a
1877 **GS3A3Component**, which permits the representation of all characters permitted in the
1878 EAN.UCC-128 Application Identifier 254 Extension Component according to the EAN.UCC
1879 General Specifications. SGLN-URIs that are derived from 96-bit tag encodings, however,
1880 will have Extension Component that consist only of digits and which have no leading zeros.
1881 These limitations are described in the encoding procedures, and in Appendix F

1882 **4.3.6 GRAI-URI**

1883 **GRAI-URI ::= "urn:epc:id:grai:" GRAIURIBody**

1884 **GRAIURIBody ::= 2*(PaddedNumericComponent ".") GS3A3Component**

1885 The number of characters in the two **PaddedNumericComponent** fields must total 12
1886 (not including any of the dot characters).

1887 The Serial Number field of the GRAI-URI is expressed as a **GS3A3Component**, which
1888 permits the representation of all characters permitted in the Serial Number field of the GRAI
1889 according to the EAN.UCC General Specifications. GRAI-URIs that are derived from 96-bit
1890 tag encodings, however, will have Serial Numbers that consist only of digit characters and
1891 which have no leading zeros. These limitations are described in the encoding procedures,
1892 and in Appendix F.

1893 **4.3.7 GIAI-URI**

1894 **GIAI-URI ::= "urn:epc:id:giai:" GIAIURIBody**

1895 GIAIURIBody ::= PaddedNumericComponent "." GS3A3Component
 1896 The total number of characters in the PaddedNumericComponent and
 1897 GS3A3Component fields must not exceed 30 (not including the dot character that separates
 1898 the two fields).
 1899 The Individual Asset Reference field of the GIAI-URI is expressed as a GS3A3Component,
 1900 which permits the representation of all characters permitted in the Individual Asset
 1901 Reference field of the GIAI according to the EAN.UCC General Specifications. GIAI-URIs
 1902 that is derived from 96-bit tag encodings, however, will have Individual Asset References
 1903 that consist only of digit characters and which have no leading zeros. These limitations are
 1904 described in the encoding procedures, and in Appendix F.

1905 **4.3.8 EPC Tag URI**

1906 TagURI ::= "urn:epc:tag:" TagURIBody
 1907 TagURIBody ::= GIDTagURIBody | SGTINSGLNGRAI96TagURIBody |
 1908 SGTINSGLNGRAIAlphaTagURIBody | SSCCTagURIBody |
 1909 GIAI96TagURIBody | GIAIAlphaTagURIBody
 1910 GIDTagURIBody ::= GIDTagEncName ":" 2*(NumericComponent ".")
 1911 NumericComponent
 1912 GIDTagEncName ::= "gid-96"
 1913 SGTINSGLNGRAITag96URIBody ::= SGTINSGLNGRAI96TagEncName ":"
 1914 NumericComponent "." 2*(PaddedNumericComponent ".")
 1915 NumericComponent
 1916 SGTINSGLNGRAITagAlphaURIBody ::= SGTINSGLNGRAIAlphaTagEncName
 1917 ":" NumericComponent "." 2*(PaddedNumericComponent ".")
 1918 GS3A3Component
 1919 SGTINSGLNGRAI96TagEncName ::= "sgtin-96" | "sgln-96" | "grai-
 1920 96"
 1921 SGTINSGLNGRAIAlphaTagEncName ::= "sgtin-198" | "sgln-195" |
 1922 "grai-170"
 1923 SSCCTagURIBody ::= SSCCTagEncName ":" NumericComponent 2*(".")
 1924 PaddedNumericComponent)
 1925 SSCCTagEncName ::= "sscc-96"
 1926 GIAI96TagURIBody ::= GIAI96TagEncName ":" NumericComponent "."
 1927 PaddedNumericComponent "." NumericComponent
 1928 GIAIAlphaTagURIBody ::= GIAIAlphaTagEncName ":"
 1929 NumericComponent "." PaddedNumericComponent "." GS3A3Component
 1930 GIAI96TagEncName ::= "giai-96"
 1931 GIAIAlphaTagEncName ::= "giai-202"

1932 **4.3.9 Raw Tag URI**

1933 RawURI ::= "urn:epc:raw:" RawURIBody
1934 RawURIBody ::= DecimalRawURIBody | HexRawURIBody |
1935 AFIRawURIBody)
1936 DecimalRawURIBody ::= NonZeroComponent "." NumericComponent
1937 HexRawURIBody ::= NonZeroComponent ".x" HexComponent
1938 AFIRawURIBody ::= NonZeroComponent ".x" HexComponent ".x"
1939 HexComponent

1940 **4.3.10 EPC Pattern URI**

1941 PatURI ::= "urn:epc:pat:" PatBody
1942 PatBody ::= GIDPatURIBody | SGTINSGLNGRAI96PatURIBody |
1943 SGTINSGLNGRAIAlphaPatURIBody | SSCCPatURIBody |
1944 GIAI96PatURIBody | GIAIAlphaPatURIBody
1945 GIDPatURIBody ::= GIDTagEncName ":" 2*(PatComponent ".")
1946 PatComponent
1947 SGTINSGLNGRAI96PatURIBody ::= SGTINSGLNGRAI96TagEncName ":"
1948 PatComponent "." GS1PatBody "." PatComponent
1949 SGTINSGLNGRAIAlphaPatURIBody ::= SGTINSGLNGRAIAlphaTagEncName
1950 ":" PatComponent "." GS1PatBody "." GS3A3PatComponent
1951 SSCCPatURIBody ::= SSCCTagEncName ":" PatComponent "."
1952 GS1PatBody
1953 GIAI96PatURIBody ::= GIAI96TagEncName ":" PatComponent "."
1954 GS1PatBody
1955 GIAIAlphaPatURIBody ::= GIAIAlphaTagEncName ":" PatComponent
1956 "." GS1GS3A3PatBody
1957 GS1PatBody ::= "*.*)" | (PaddedNumericComponent ".")
1958 PatComponent)
1959 GS1GS3A3PatBody ::= "*.*)" | (PaddedNumericComponent ".")
1960 GS3A3PatComponent)
1961 PatComponent ::= NumericComponent
1962 | StarComponent
1963 | RangeComponent
1964 GS3A3PatComponent ::= GS3A3Component | StarComponent
1965 StarComponent ::= "*"
1966 RangeComponent ::= "[" NumericComponent "-"
1967 NumericComponent "]"

1968 For a RangeComponent to be legal, the numeric value of the first NumericComponent
1969 must be less than or equal to the numeric value of the second NumericComponent.

1970 **4.3.11 EPC Identity Pattern URI**

```
1971 IDPatURI ::= "urn:epc:idpat:" IDPatBody
1972 IDPatBody ::= GIDIDPatURIBody | SGTINIDPatURIBody |
1973 SGLNIDPatURIBody | GIAIIDPatURIBody | SSCCIDPatURIBody |
1974 GRAIIDPatURIBody
1975 GIDIDPatURIBody ::= "gid:" GIDIDPatURIMain
1976 GIDIDPatURIMain ::=
1977     2*(NumericComponent ".") NumericComponent
1978     | 2*(NumericComponent ".") "*"
1979     | NumericComponent ".*.*"
1980     | ".*.*"
1981 SGTINIDPatURIBody ::= "sgtin:" SGTINSGLNGRAIIDPatURIMain
1982 GRAIIDPatURIBody ::= "grai:" SGTINSGLNGRAIIDPatURIMain
1983 SGLNIDPatURIBody ::= "sgln:" SGTINSGLNGRAIIDPatURIMain
1984 SGTINSGLNGRAIIDPatURIMain ::=
1985     2*(PaddedNumericComponent ".") GS3A3Component
1986     | 2*(PaddedNumericComponent ".") "*"
1987     | PaddedNumericComponent ".*.*"
1988     | ".*.*"
1989 SSCCIDPatURIBody ::= "sscc:" SSCCIDPatURIMain
1990 SSCCIDPatURIMain ::=
1991     PaddedNumericComponent "." PaddedNumericComponent
1992     | PaddedNumericComponent ".*"
1993     | ".*"
1994 GIAIIDPatURIBody ::= "giai:" GIAIIDPatURIMain
1995 GIAIIDPatURIMain ::=
1996     PaddedNumericComponent "." GS3A3Component
1997     | PaddedNumericComponent ".*"
1998     | ".*"
```

1999 **4.3.12 DoD Construct URI**

```
2000 DOD-URI ::= "urn:epc:id:usdod:" CAGECodeOrDODAAC "."
2001 DoDSerialNumber
2002 DODTagURI ::= "urn:epc:tag:" DoDTagType ":" DoDFilter "."
2003 CAGECodeOrDODAAC "." DoDSerialNumber
2004 DODPatURI ::= "urn:epc:pat:" DoDTagType ":" DoDFilterPat "."
2005 CAGECodeOrDODAACPat "." DoDSerialNumberPat
```

```

2006 DODIDPatURI ::= "urn:epc:idpat:usdod:" DODIDPatMain
2007 DODIDPatMain ::=
2008     CAGECodeOrDODAAC "." DoDSerialNumber
2009     | CAGECodeOrDODAAC ".*"
2010     | ".*.*"
2011 DoDTagType ::= "usdod-96"
2012 DoDFilter ::= NumericComponent
2013 CAGECodeOrDODAAC ::= CAGECode | DODAAC
2014 CAGECode ::= CAGECodeOrDODAACChar*5
2015 DODAAC ::= CAGECodeOrDODAACChar*6
2016 DoDSerialNumber ::= NumericComponent
2017 DoDFilterPat ::= PatComponent
2018 CAGECodeOrDODAACPat ::= CAGECodeOrDODAAC | StarComponent
2019 DoDSerialNumberPat ::= PatComponent
2020 CAGECodeOrDODAACChar ::= Digit | "A" | "B" | "C" | "D" | "E" |
2021 "F" | "G" | "H" | "J" | "K" | "L" | "M" | "N" | "P" | "Q" |
2022 "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"

```

2023

2024 **4.3.13 Summary (non-normative)**

2025 The syntax rules above can be summarized informally as follows:

```

2026 urn:epc:id:gid:MMM.CCC.SSS
2027 urn:epc:id:sgtin:PPP.III.AAA
2028 urn:epc:id:sscc:PPP.III
2029 urn:epc:id:sgln:PPP.III.AAA
2030 urn:epc:id:grai:PPP.III.AAA
2031 urn:epc:id:giai:PPP.AAA
2032 urn:epc:id:usdod:TTT.SSS
2033
2034 urn:epc:tag:gid-96:MMM.CCC.SSS
2035 urn:epc:tag:sgtin-96:FFF.PPP.III.SSS
2036 urn:epc:tag:sgtin-198:FFF.PPP.III.AAA
2037 urn:epc:tag:sscc-96:FFF.PPP.III
2038 urn:epc:tag:sgln-96:FFF.PPP.III.SSS
2039 urn:epc:tag:sgln-195:FFF.PPP.III.AAA

```

2040 urn:epc:tag:grai-96:FFF.PPP.III.SSS
 2041 urn:epc:tag:grai-170:FFF.PPP.III.AAA
 2042 urn:epc:tag:giai-96:FFF.PPP.SSS
 2043 urn:epc:tag:giai-202:FFF.PPP.AAA
 2044 urn:epc:tag:usdod-96:FFF.TTT.SSS
 2045
 2046 urn:epc:raw:LLL.BBB
 2047 urn:epc:raw:LLL.HHH
 2048 urn:epc:raw:LLL.HHH.HHH
 2049
 2050 urn:epc:idpat:gid:MMM.CCC.SSS
 2051 urn:epc:idpat:gid:MMM.CCC.*
 2052 urn:epc:idpat:gid:MMM.*.*
 2053 urn:epc:idpat:gid:*. *.*
 2054 urn:epc:idpat:sgtin:PPP.III.AAA
 2055 urn:epc:idpat:sgtin:PPP.III.*
 2056 urn:epc:idpat:sgtin:PPP.*.*
 2057 urn:epc:idpat:sgtin:*. *.*
 2058 urn:epc:idpat:sscc:PPP.III
 2059 urn:epc:idpat:sscc:PPP.*
 2060 urn:epc:idpat:sscc:*. *
 2061 urn:epc:idpat:sgln:PPP.III.AAA
 2062 urn:epc:idpat:sgln:PPP.III.*
 2063 urn:epc:idpat:sgln:PPP.*.*
 2064 urn:epc:idpat:sgln:*. *.*
 2065 urn:epc:idpat:grai:PPP.III.AAA
 2066 urn:epc:idpat:grai:PPP.III.*
 2067 urn:epc:idpat:grai:PPP.*.*
 2068 urn:epc:idpat:grai:*. *.*
 2069 urn:epc:idpat:giai:PPP.AAA
 2070 urn:epc:idpat:giai:PPP.*
 2071 urn:epc:idpat:giai:*. *
 2072 urn:epc:idpat:usdod:TTT.SSS

2073

2074 urn:epc:idpat:usdod:TTT.*

2075 urn:epc:idpat:usdod:*.*

2076

2077 urn:epc:pat:gid-96:MMMpat.CCCpat.SSSpat

2078 urn:epc:pat:sgtin-96:FFFpat.PPP.IIIpat.SSSpat

2079 urn:epc:pat:sgtin-96:FFFpat.*.*.SSSpat

2080 urn:epc:pat:sgtin-198:FFFpat.PPP.IIIpat.AAApat

2081 urn:epc:pat:sgtin-198:FFFpat.*.*.AAApot

2082 urn:epc:pat:sscc-96:FFFpat.PPP.IIIpat

2083 urn:epc:pat:sscc-96:FFFpat.*.*

2084 urn:epc:pat:sgln-96:FFFpat.PPP.IIIpat.SSSpat

2085 urn:epc:pat:sgln-96:FFFpat.*.*.SSSpat

2086 urn:epc:pat:sgln-195:FFFpat.PPP.IIIpat.AAApat

2087 urn:epc:pat:sgln-195:FFFpat.*.*.AAApot

2088 urn:epc:pat:grai-96:FFFpat.PPP.IIIpat.SSSpat

2089 urn:epc:pat:grai-96:FFFpat.*.*.SSSpat

2090 urn:epc:pat:grai-170:FFFpat.PPP.IIIpat.AAApat

2091 urn:epc:pat:grai-170:FFFpat.*.*.AAApot

2092 urn:epc:pat:giai-96:FFFpat.PPP.SSSpat

2093 urn:epc:pat:giai-96:FFFpat.*.*

2094 urn:epc:pat:giai-202:FFFpat.PPP.AAApat

2095 urn:epc:pat:giai-202:FFFpat.*.*

2096 urn:epc:pat:usdod-96:FFFpat.TTT.SSSpat

2097 urn:epc:pat:usdod-96:FFFpat.*.*.SSSpat

2098 where

2099 *MMM* denotes a General Manager Number

2100 *CCC* denotes an Object Class number

2101 *SSS* denotes a numeric Serial Number or GIAI Individual Asset Reference

2102 *AAA* denotes an alphanumeric Serial Number or GIAI Individual Asset reference

2103 *PPP* denotes an EAN.UCC Company Prefix

2104 *TTT* denotes a US DoD assigned CAGE code or DODAAC

2105 *III* denotes an SGTIN Item Reference (prefixed by the Indicator Digit), an SSCC
2106 Shipping Container Serial Number (prefixed by the Extension Digit (ED)), a SGLN Location
2107 Reference, or a GRAI Asset Type.

2108 *FFF* denotes a filter code as used by the SGTIN, SSCC, SGLN, GRAI, GIAI, and DoD tag
2109 encodings

2110 *XXXpat* is the same as *XXX* but allowing * and [lo - hi] pattern syntax in addition
2111 (exception: [lo - hi] syntax is not allowed for *AAAp*at).

2112 *LLL* denotes the number of bits of an uninterpreted bit sequence

2113 *BBB* denotes the literal value of an uninterpreted bit sequence converted to decimal

2114 *HHH* denotes the literal value of an uninterpreted bit sequence converted to hexadecimal
2115 and preceded by the character 'x'.

2116 and where all numeric fields are in decimal with no leading zeros (unless the overall value of
2117 the field is zero, in which case it is represented with a single 0 character), with the exception
2118 of the hexadecimal raw representation.

2119 Exceptions:

- 2120 1. The length of *PPP* and *III* is significant, and leading zeros are used as necessary.
2121 The length of *PPP* is the length of the company prefix as assigned by GS1. The
2122 length of *III* plus the length of *PPP* must equal 13 for SGTIN, 17 for SSCC, 12 for
2123 GLN, or 12 for GRAI.
- 2124 2. The Value field of urn:epc:raw is expressed in hexadecimal if the value is
2125 preceded by the character 'x'.

2126 **5 Translation between EPC-URI and Other EPC** 2127 **Representations**

2128 This section defines the semantics of EPC-URI encodings, by defining how they are
2129 translated into other EPC representations and vice versa.

2130 **5.1 Bit string into EPC-URI (pure identity)**

2131 The following procedure translates a bit-level encoding into an EPC-URI:

- 2132 1. Determine the identity type and encoding scheme by finding the row in Table 1
2133 (Section 3.1) that matches the most significant bits of the bit string. If the most
2134 significant bits do not match any row of the table, stop: the bit string is invalid and
2135 cannot be translated into an EPC-URI. If the encoding scheme indicates one of the
2136 DoD Tag Data Constructs, consult the appropriate U.S. Department of Defense
2137 document for specific encoding and decoding rules. Otherwise, if the encoding
2138 scheme is SGTIN-96 or SGTIN-198, proceed to Step 2; if the encoding scheme is
2139 SSCC-96, proceed to Step 5; if the encoding scheme is SGLN-96 pr SGLN-195,
2140 proceed to Step 8; if the encoding scheme is GRAI-96 or GRAI-170, proceed to

- 2141 Step 11; if the encoding scheme is GIAI-96 or GIAI-202, proceed to Step 14; if the
2142 encoding scheme is GID-96, proceed to Step 17.
- 2143 2. Follow the decoding procedure given in Section 3.5.1.2 (for SGTIN-96) or in
2144 Section 3.5.2.2 (for SGTIN-198) to obtain the decimal Company Prefix $p_1p_2...p_L$, the
2145 decimal Item Reference and Indicator $i_1i_2...i_{(13-L)}$, and the Serial Number S . If the
2146 decoding procedure fails, stop: the bit-level encoding cannot be translated into an
2147 EPC-URI.
- 2148 3. Create an EPC-URI by concatenating the following: the string
2149 `urn:epc:id:sgtin:`, the Company Prefix $p_1p_2...p_L$ where each digit (including
2150 any leading zeros) becomes the corresponding ASCII digit character, a dot (.)
2151 character, the Item Reference and Indicator $i_1i_2...i_{(13-L)}$ (handled similarly), a dot (.)
2152 character, and the Serial Number S as a decimal integer (SGTIN-96) or alphanumeric
2153 character (SGTIN-198). For SGTIN-96 the portion corresponding to the Serial
2154 Number must have no leading zeros, except where the Serial Number is itself zero in
2155 which case the corresponding URI portion must consist of a single zero character.
- 2156 4. Go to Step 19.
- 2157 5. Follow the decoding procedure given in Section 3.6.1.2 (for SSCC-96) to obtain the
2158 decimal Company Prefix $p_1p_2...p_L$, and the decimal Serial Reference $s_1s_2...s_{(17-L)}$. If
2159 the decoding procedure fails, stop: the bit-level encoding cannot be translated into an
2160 EPC-URI.
- 2161 6. Create an EPC-URI by concatenating the following: the string
2162 `urn:epc:id:sscc:`, the Company Prefix $p_1p_2...p_L$ where each digit (including
2163 any leading zeros) becomes the corresponding ASCII digit character, a dot (.)
2164 character, and the Serial Reference $s_1s_2...s_{(17-L)}$ (handled similarly).
- 2165 7. Go to Step 19.
- 2166 8. Follow the decoding procedure given in Section 3.7.1.2 (for SGLN-96) or in Section
2167 3.7.2.2 (for SGLN-195) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal
2168 Location Reference $i_1i_2...i_{(12-L)}$, and the Extension Component S . If the decoding
2169 procedure fails, stop: the bit-level encoding cannot be translated into an EPC-URI.
- 2170 9. Create an EPC-URI by concatenating the following: the string
2171 `urn:epc:id:sgln:`, the Company Prefix $p_1p_2...p_L$ where each digit (including
2172 any leading zeros) becomes the corresponding ASCII digit character, a dot (.)
2173 character, for $L < 12$ the Location Reference, $i_1i_2...i_{(12-L)}$ (handled similarly), a dot
2174 (.) character, and the Extension Component S as a decimal integer (SGLN-96) or
2175 alphanumeric character (SGLN-195). For SGLN-96 the portion corresponding to the
2176 Extension Component must have no leading zeros, except where the Extension
2177 Component is itself zero in which case the corresponding URI portion must consist of
2178 a single zero character. If a Location Reference does not exist (where $L = 12$), leave
2179 no blank space between the two dot (.) characters.
- 2180 10. Go to Step 19.
- 2181 11. Follow the decoding procedure given in Section 3.8.1.2 (for GRAI-96) or in Section
2182 3.8.2.2 (for GRAI-170) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal

2183 Asset Type $i_1i_2\dots i_{(12-L)}$, and the Serial Number S . If the decoding procedure fails,
 2184 stop: the bit-level encoding cannot be translated into an EPC-URI.

2185 12. Create an EPC-URI by concatenating the following: the string
 2186 `urn:epc:id:grai:`, the Company Prefix $p_1p_2\dots p_L$ where each digit (including
 2187 any leading zeros) becomes the corresponding ASCII digit character, a dot (.)
 2188 character, for $L < 12$ the Asset Type $i_1i_2\dots i_{(12-L)}$ (handled similarly), a dot (.)
 2189 character, and the Serial Number S as a decimal integer (GRAI-96) or alphanumeric
 2190 character (GRAI-170). For GRAI-96 the portion corresponding to the Serial Number
 2191 must have no leading zeros, except where the Serial Number is itself zero in which
 2192 case the corresponding URI portion must consist of a single zero character. If an
 2193 Asset Type does not exist (where $L = 12$), leave no blank space between the two dot
 2194 (.) characters.

2195 13. Go to Step 19.

2196 14. Follow the decoding procedure given in Section 3.9.1.2 (for GIAI-96) or in 3.9.2.2
 2197 (for GIAI-202) to obtain the decimal Company Prefix $p_1p_2\dots p_L$, and the Individual
 2198 Asset Reference S . If the decoding procedure fails, stop: the bit-level encoding
 2199 cannot be translated into an EPC-URI.

2200 15. Create an EPC-URI by concatenating the following: the string
 2201 `urn:epc:id:giai:`, the Company Prefix $p_1p_2\dots p_L$ where each digit (including
 2202 any leading zeros) becomes the corresponding ASCII digit character, a dot (.)
 2203 character, and the Individual Asset Reference S as a decimal integer (GIAI-96) or
 2204 alphanumeric character (GIAI-202). For GIAI-96 the portion corresponding to the
 2205 Individual Asset Reference must have no leading zeros, except where the Individual
 2206 Asset Reference is itself zero in which case the corresponding URI portion must
 2207 consist of a single zero character.

2208 16. Go to Step 19.

2209 17. Follow the decoding procedure given in Section 3.4.1.2 to obtain the General
 2210 Manager Number M , the Object Class C , and the Serial Number S .

2211 18. Create an EPC-URI by concatenating the following: the string `urn:epc:id:gid:`,
 2212 the General Manager Number as a decimal integer, a dot (.) character, the Object
 2213 Class as a decimal integer, a dot (.) character, and the Serial Number S as a decimal
 2214 integer. Each decimal number must have no leading zeros, except where the integer
 2215 is itself zero in which case the corresponding URI portion must consist of a single
 2216 zero character.

2217 19. The translation is now complete.

2218 5.2 Bit String into Tag or Raw URI

2219 The following procedure translates a bit string of N bits into either an EPC Tag URI or a
 2220 Raw Tag URI:

2221 1. Determine the identity type, encoding scheme, and encoding length (K) by finding
 2222 the row in Table 1 (Section 3.1) that matches the most significant bits of the bit string.

2223 If $N < K$, proceed to Step 20; otherwise, continue with the remainder of this
 2224 procedure, using the most significant K bits of the bit string. If the encoding scheme
 2225 indicates one of the DoD Tag Data Constructs, consult the appropriate U.S.
 2226 Department of Defense document for specific encoding and decoding rules. If the
 2227 encoding scheme is SGTIN-96 or SGTIN-198, proceed to Step 2; if the encoding
 2228 scheme is SSCC-96, proceed to Step 5; if the encoding scheme is SGLN-96 or
 2229 SGLN-195, proceed to Step 8; if the encoding scheme is GRAI-96 or GRAI-170,
 2230 proceed to Step 11, if the encoding scheme is GIAI-96 or GIAI-202, proceed to Step
 2231 14, if the encoding scheme is GID-96, proceed to Step 17; otherwise, proceed to Step
 2232 20.

2233 2. Follow the decoding procedure given in Section 3.5.1.2 (for SGTIN-96) or 3.5.2.2
 2234 (for SGTIN-198) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal Item
 2235 Reference and Indicator $i_1i_2...i_{(13-L)}$, the Filter Value F , and the Serial Number S . If
 2236 the decoding procedure fails, proceed to Step 20, otherwise proceed to the next step.

2237 3. Create an EPC Tag URI by concatenating the following: the string `urn:epc:tag:`,
 2238 the encoding scheme (`sgtin-96` or `sgtin-198`), a colon (`:`) character, the Filter
 2239 Value F as a decimal integer, a dot (`.`) character, the Company Prefix $p_1p_2...p_L$ where
 2240 each digit (including any leading zeros) becomes the corresponding ASCII digit
 2241 character, a dot (`.`) character, the Item Reference and Indicator $i_1i_2...i_{(13-L)}$ (handled
 2242 similarly), a dot (`.`) character, and the Serial Number S as a decimal integer (SGTIN-
 2243 96) or alphanumeric character (SGTIN-198). For SGTIN-96 the portions
 2244 corresponding to the Filter Value and Serial Number must have no leading zeros,
 2245 except where the corresponding integer is itself zero in which case a single zero
 2246 character is used.

2247 4. Go to Step 21.

2248 5. Follow the decoding procedure given in Section 3.6.1.2 (for SSCC-96) to obtain the
 2249 decimal Company Prefix $p_1p_2...p_L$, and the decimal Serial Reference $i_1i_2...i_{(17-L)}$, and
 2250 the Filter Value F . If the decoding procedure fails, proceed to Step 20, otherwise
 2251 proceed to the next step.

2252 6. Create an EPC Tag URI by concatenating the following: the string `urn:epc:tag:`,
 2253 the encoding scheme (`sscc-96`), a colon (`:`) character, the Filter Value F as a
 2254 decimal integer, a dot (`.`) character, the Company Prefix $p_1p_2...p_L$ where each digit
 2255 (including any leading zeros) becomes the corresponding ASCII digit character, a dot
 2256 (`.`) character, and the Serial Reference $i_1i_2...i_{(17-L)}$ (handled similarly).

2257 7. Go to Step 21.

2258 8. Follow the decoding procedure given in Section 3.7.1.2 (for SGLN-96) or Section
 2259 3.7.2.2 (for SGLN-195) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal
 2260 Location Reference $i_1i_2...i_{(12-L)}$, the Filter Value F , and the Extension Component S .
 2261 If the decoding procedure fails, proceed to Step 20, otherwise proceed to the next step.

2262 9. Create an EPC Tag URI by concatenating the following: the string `urn:epc:tag:`,
 2263 the encoding scheme (`sgln-96` or `sgln-195`), a colon (`:`) character, the Filter
 2264 Value F as a decimal integer, a dot (`.`) character, the Company Prefix $p_1p_2...p_L$ where

2265 each digit (including any leading zeros) becomes the corresponding ASCII digit
 2266 character, a dot (.) character, when $L < 12$ the Location Reference $i_1i_2...i_{(12-L)}$
 2267 (handled similarly), a dot (.) character, and the Extension Component S as a decimal
 2268 integer (SGLN-96) or alphanumeric character (SGLN-198). For SGLN-96 the
 2269 portions corresponding to the Filter Value and Extension Component must have no
 2270 leading zeros, except where the corresponding integer is itself zero in which case a
 2271 single zero character is used. If a Location Reference does not exist where $L = 12$
 2272 leave no blank space between the two dot (.) characters.

2273 10. Go to Step 21.

2274 11. Follow the decoding procedure given in Section 3.8.1.2 (for GRAI-96) or 3.8.2.2 (for
 2275 GRAI-170) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal Asset Type
 2276 $i_1i_2...i_{(12-L)}$, the Filter Value F , and the Serial Number $d_1d_2...d_K$. If the decoding
 2277 procedure fails, proceed to Step 20, otherwise proceed to the next step.

2278 12. Create an EPC Tag URI by concatenating the following: the string `urn:epc:tag:`,
 2279 the encoding scheme (`grai-96` or `grai-170`), a colon (:) character, the Filter
 2280 Value F as a decimal integer, a dot (.) character, the Company Prefix $p_1p_2...p_L$ where
 2281 each digit (including any leading zeros) becomes the corresponding ASCII digit
 2282 character, a dot (.) character, for $L < 12$ the Asset Type $i_1i_2...i_{(12-L)}$ (handled
 2283 similarly), a dot (.) character, and the Serial Number $d_1d_2...d_K$ as a decimal integer
 2284 (GRAI-96) or alphanumeric character (GRAI-170). For GRAI-96 the portions
 2285 corresponding to the Filter Value and Serial Number must have no leading zeros,
 2286 except where the corresponding integer is itself zero in which case a single zero
 2287 character is used. If an Asset Type does not exist where $L = 12$ leave no blank space
 2288 between the two dot (.) characters.

2289 13. Got to Step 21.

2290 14. Follow the decoding procedure given in Section 3.9.1.2 (for GIAI-96) or 3.9.2.2 (for
 2291 GIAI-202) to obtain the decimal Company Prefix $p_1p_2...p_L$, the Individual Asset
 2292 Reference $s_1s_2...s_J$, and the Filter Value F . If the decoding procedure fails, proceed
 2293 to Step 20, otherwise proceed to the next step.

2294 15. Create an EPC Tag URI by concatenating the following: the string `urn:epc:tag:`,
 2295 the encoding scheme (`giai-96` or `giai-202`), a colon (:) character, the Filter
 2296 Value F as a decimal integer, a dot (.) character, the Company Prefix $p_1p_2...p_L$ where
 2297 each digit (including any leading zeros) becomes the corresponding ASCII digit
 2298 character, a dot (.) character, and the Individual Asset Reference $s_1s_2...s_J$ (handled
 2299 similarly). For GIAI-96 the portion corresponding to the Filter Value and the
 2300 Individual Asset Reference must have no leading zeros, except where the
 2301 corresponding integer is itself zero in which case a single zero character is used.

2302 16. Go to Step 21.

2303 17. Follow the decoding procedure given in Section 3.4.1.2 to obtain the General
 2304 Manager Number, the Object Class, and the Serial Number.

2305 18. Create an EPC Tag URI by concatenating the following: the string
 2306 `urn:epc:tag:gid-96:`, the General Manager Number as a decimal number, a

2307 dot (.) character, the Object Class as a decimal number, a dot (.) character, and the
2308 Serial Number as a decimal number. Each decimal number must have no leading
2309 zeros, except where the integer is itself zero in which case the corresponding URI
2310 portion must consist of a single zero character.

2311 19. Go to Step 21.

2312 20. This tag is not a recognized EPC Tag Encoding, therefore create an EPC Raw URI by
2313 concatenating the following: the string `urn:epc:raw:`, the length of the bit string
2314 (N) expressed as a decimal integer with no leading zeros, a dot (.) character, a
2315 lowercase x character, and the value of the bit string considered as a single
2316 hexadecimal integer. The value must have a number of characters equal to the length
2317 (N) divided by four and rounded up to the nearest whole number, and must only use
2318 uppercase letters for the hexadecimal digits A, B, C, D, E, and F.

2319 21. The translation is now complete.

2320

2321 **5.3 Gen 2 Tag EPC Memory into EPC-URI (pure identity)**

2322 The following procedure translates the contents of the EPC Memory of a Gen 2 Tag into an
2323 EPC-URI:

- 2324 1. Consider bits 10x through 14x (inclusive) as a five-bit binary integer, L.
- 2325 2. Examine the “toggle” bit, bit 17x. If the toggle bit is a one, stop: this bit string
2326 cannot be converted into an EPC-URI. Otherwise, continue with Step 3.
- 2327 3. Extract N bits beginning with bit 20x, where N = 16L.
- 2328 4. Finish by proceeding with the procedure in Section 5.1, using the N-bit string
2329 extracted in Step 3.

2330 **5.4 Gen 2 Tag EPC Memory into Tag or Raw URI**

2331 The following procedure translates the contents of the EPC Memory of a Gen 2 Tag into
2332 either an EPC Tag URI or a Raw Tag URI:

- 2333 1. Consider bits 10x through 14x (inclusive) as a five-bit binary integer, L.
- 2334 2. Examine the “toggle” bit, bit 17x. If the toggle bit is a one, go to Step 5. Otherwise,
2335 continue with Step 3.
- 2336 3. Extract N bits beginning with bit 20x, where N = 16L.
- 2337 4. Finish by proceeding with the procedure in Section 5.2, using the N-bit string
2338 extracted in Step 3.
- 2339 5. This tag has an AFI, and is therefore by definition not an EPC Tag Encoding.
2340 Continue with the following steps.
- 2341 6. Extract bits 18x through 1Fx (inclusive) as an eight-bit binary integer, A (this is the
2342 AFI).

- 2343 7. Extract N bits beginning with bit 20x, where N = 16L.
- 2344 8. Create an EPC Raw URI by concatenating the following: the string
- 2345 urn:epc:raw:, the number N from Step 7 expressed as a decimal integer with no
- 2346 leading zeros, a dot (.) character, a lowercase x character, the value A from Step 6
- 2347 expressed as a two-character hexadecimal integer, a lowercase x character, and the
- 2348 value of the N-bit string from Step 7 considered as a single hexadecimal integer. The
- 2349 value must have a number of characters equal to the length (N) divided by four. Both
- 2350 the AFI and the value must only use uppercase letters for the hexadecimal digits A, B,
- 2351 C, D, E, and F.

2352 5.5 URI into Bit String

2353 The following procedure translates a URI into a bit string:

- 2354 1. If the URI is an SGTIN-URI (urn:epc:id:sgtin:), an SSCC-URI
- 2355 (urn:epc:id:sscc:), an SGLN-URI (urn:epc:id:sgln:), a GRAI-URI
- 2356 (urn:epc:id:grai:), a GIAI-URI (urn:epc:id:giai:), a GID-URI
- 2357 (urn:epc:id:gid:), a DOD-URI (urn:epc:id:usdod:) or an EPC Pattern
- 2358 URI (urn:epc:pat:), the URI cannot be translated into a bit string.
- 2359 2. If the URI is a Raw Tag URI of the form urn:epc:raw:N.V, create the bit string
- 2360 by converting the second component (V) of the Raw Tag URI into a binary integer,
- 2361 whose length is equal to the first component (N) of the Raw Tag URI. If the value of
- 2362 the second component is too large to fit into a binary integer of that size, the URI
- 2363 cannot be translated into a bit string. If the URI is a Raw Tag URI of the form
- 2364 urn:epc:raw:N.A.V, the URI cannot be translated into a bit string (but see the
- 2365 related procedure in Section 5.6).
- 2366 3. If the URI is an EPC Tag URI or US DoD Tag URI (urn:epc:tag:encName:),
- 2367 parse the URI using the grammar for TagURI as given in Section 4.3.8 or for
- 2368 DODTagURI as given in Section 4.3.11. If the URI cannot be parsed using these
- 2369 grammars, stop: the URI is illegal and cannot be translated into a bit string. If
- 2370 encName is usdod-96, consult the appropriate U.S. Department of Defense
- 2371 document for specific translation rules. Otherwise, if encName is sgtin-96 go to
- 2372 Step 4, if sgtin-198 go to Step 9, if encName is sscc-96 go to Step 14, if
- 2373 encName is sgln-96 go to Step 18 or sgln-195 go to Step 23, if encName is
- 2374 grai-96 go to Step 28 or grai-170 go to Step 33, if encName is giai-96 go
- 2375 to Step 38 or giai-202 go to Step 43, or if encName is gid-96 go to Step 48.
- 2376 4. Let the URI be written as
- 2377 urn:epc:tag:encName: $f_1f_2...f_F.p_1p_2...p_L.i_1i_2...i_{(13-L)}.s_1s_2...s_S$.
- 2378 5. Interpret $f_1f_2...f_F$ as a decimal integer F.
- 2379 6. Interpret $s_1s_2...s_S$ as a decimal integer S.
- 2380 7. Carry out the encoding procedure defined in Section 3.5.1.1 (SGTIN-96), using
- 2381 $i_1p_1p_2...p_Li_2...i_{(13-L)}0$ as the EAN.UCC GTIN-14 (the trailing zero is a dummy
- 2382 check digit, which is ignored by the encoding procedure), L as the length of the

2383 EAN.UCC company prefix, F from Step 5 as the Filter Value, and S from Step 6 as
 2384 the Serial Number. If the encoding procedure fails because an input is out of range,
 2385 or because the procedure indicates a failure, stop: this URI cannot be encoded into a
 2386 bit string.

2387 8. Go to Step 53.

2388 9. Let the URI be written as
 2389 $\text{urn:epc:tag:encName} : f_1 f_2 \dots f_F \cdot p_1 p_2 \dots p_L \cdot i_1 i_2 \dots i_{(13-L)} \cdot s_1 s_2 \dots s_S$.

2390 10. Interpret $f_1 f_2 \dots f_F$ as a decimal integer F .

2391 11. Interpret $s_1 s_2 \dots s_S$ as an alphanumeric string S .

2392 12. Carry out the encoding procedure defined in Section 3.5.2.1 (SGTIN-198) using
 2393 $i_1 p_1 p_2 \dots p_L i_2 \dots i_{(13-L)} 0$ as the EAN.UCC GTIN-14 (the trailing zero is a dummy
 2394 check digit, which is ignored by the encoding procedure), L as the length of the
 2395 EAN.UCC company prefix, F from Step 10 as the Filter Value, and S from Step 11
 2396 as the Serial Number. If the encoding procedure fails because an input is out of range,
 2397 or because the procedure indicates a failure, stop: this URI cannot be encoded into a
 2398 bit string.

2399 13. Go to Step 53.

2400 14. Let the URI be written as
 2401 $\text{urn:epc:tag:encName} : f_1 f_2 \dots f_F \cdot p_1 p_2 \dots p_L \cdot i_1 i_2 \dots i_{(17-L)}$.

2402 15. Interpret $f_1 f_2 \dots f_F$ as a decimal integer F .

2403 16. Carry out the encoding procedure defined in Section 3.6.1.1 (SSCC-96), using
 2404 $i_1 p_1 p_2 \dots p_L i_2 i_3 \dots i_{(17-L)} 0$ as the EAN.UCC SSCC (the trailing zero is a dummy
 2405 check digit, which is ignored by the encoding procedure), L as the length of the
 2406 EAN.UCC company prefix, and F from Step 15 as the Filter Value. If the encoding
 2407 procedure fails because an input is out of range, or because the procedure indicates a
 2408 failure, stop: this URI cannot be encoded into a bit string.

2409 17. Go to Step 53.

2410 18. Let the URI be written as
 2411 $\text{urn:epc:tag:encName} : f_1 f_2 \dots f_F \cdot p_1 p_2 \dots p_L \cdot i_1 i_2 \dots i_{(12-L)} \cdot s_1 s_2 \dots s_S$.

2412 19. Interpret $f_1 f_2 \dots f_F$ as a decimal integer F .

2413 20. Interpret $s_1 s_2 \dots s_S$ as a decimal integer S .

2414 21. Carry out the encoding procedure defined in Section 3.7.1.1 (SGLN-96), using
 2415 $p_1 p_2 \dots p_L i_1 i_2 \dots i_{(12-L)} 0$ as the EAN.UCC GLN (the trailing zero is a dummy check
 2416 digit, which is ignored by the encoding procedure), L as the length of the EAN.UCC
 2417 company prefix, F from Step 19 as the Filter Value, and S from Step 20 as the
 2418 Extension Component. If the encoding procedure fails because an input is out of
 2419 range, or because the procedure indicates a failure, stop: this URI cannot be encoded
 2420 into a bit string.

2421 22. Go to Step 53.

2422 23. Let the URI be written as
2423 urn:epc:tag:encName: $f_1f_2...f_F \cdot p_1p_2...p_L \cdot i_1i_2...i_{(12-L)} \cdot s_1s_2...s_S$.
2424 24. Interpret $f_1f_2...f_F$ as a decimal integer F .
2425 25. Interpret $s_1s_2...s_S$ as an alphanumeric string S .
2426 26. Carry out the encoding procedure defined in Section 3.7.2.1 (SGLN-195), using
2427 $p_1p_2...p_L i_1i_2...i_{(12-L)}0$ as the EAN.UCC GLN (the trailing zero is a dummy check
2428 digit, which is ignored by the encoding procedure), L as the length of the EAN.UCC
2429 company prefix, F from Step 24 as the Filter Value, and S from Step 25 as the
2430 Extension Component. If the encoding procedure fails because an input is out of
2431 range, or because the procedure indicates a failure, stop: this URI cannot be encoded
2432 into a bit string.
2433 27. Go to Step 53.
2434 28. Let the URI be written as
2435 urn:epc:tag:encName: $f_1f_2...f_F \cdot p_1p_2...p_L \cdot i_1i_2...i_{(12-L)} \cdot s_1s_2...s_S$.
2436 29. Interpret $f_1f_2...f_F$ as a decimal integer F .
2437 30. Interpret $s_1s_2...s_S$ as a decimal integer S .
2438 31. Carry out the encoding procedure defined in Section 3.8.1.1 (GRAI-96), using
2439 $0p_1p_2...p_L i_1i_2...i_{(12-L)}0s_1s_2...s_S$ as the EAN.UCC GRAI (the second zero is a
2440 dummy check digit, which is ignored by the encoding procedure), L as the length of
2441 the EAN.UCC company prefix, and F from Step 29 as the Filter Value, and S from
2442 Step 30 as the Serial Number. If the encoding procedure fails because an input is out
2443 of range, or because the procedure indicates a failure, stop: this URI cannot be
2444 encoded into a bit string.
2445 32. Go to Step 53.
2446 33. Let the URI be written as
2447 urn:epc:tag:encName: $f_1f_2...f_F \cdot p_1p_2...p_L \cdot i_1i_2...i_{(12-L)} \cdot s_1s_2...s_S$.
2448 34. Interpret $f_1f_2...f_F$ as a decimal integer F .
2449 35. Interpret $s_1s_2...s_S$ as an alphanumeric string S .
2450 36. Carry out the encoding procedure defined in Section 3.8.2.1 (GRAI-170) using
2451 $0p_1p_2...p_L i_1i_2...i_{(12-L)}0s_1s_2...s_S$ as the EAN.UCC GRAI (the second zero is a
2452 dummy check digit, which is ignored by the encoding procedure), L as the length of
2453 the EAN.UCC company prefix, and F from Step 34 as the Filter Value, and S from
2454 Step 35 as the Serial Number. If the encoding procedure fails because an input is out
2455 of range, or because the procedure indicates a failure, stop: this URI cannot be
2456 encoded into a bit string.
2457 37. Go to Step 53.
2458 38. Let the URI be written as urn:epc:tag:encName: $f_1f_2...f_F \cdot p_1p_2...p_L \cdot s_1s_2...s_S$.
2459 39. Interpret $f_1f_2...f_F$ as a decimal integer F .

2460 40. Interpret $S_1S_2...S_S$ as a decimal integer S .

2461 41. Carry out the encoding procedure defined in Section 3.9.1.1 (GIAI-96), using

2462 $p_1p_2...p_LS_1S_2...S_S$ as the EAN.UCC GIAI, L as the length of the EAN.UCC company

2463 prefix, and F from Step 39 as the Filter Value, and S from Step 40 as the Serial

2464 Number. If the encoding procedure fails because an input is out of range, or because

2465 the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.

2466 42. Go to Step 53.

2467 43. Let the URI be written as $urn:epc:tag:encName:f_1f_2...f_F.p_1p_2...p_L.S_1S_2...S_S$.

2468 44. Interpret $f_1f_2...f_F$ as a decimal integer F .

2469 45. Interpret $S_1S_2...S_S$ as an alphanumeric string S .

2470 46. Carry out the encoding procedure defined in Section 3.9.2.1 (GIAI-202) using

2471 $p_1p_2...p_LS_1S_2...S_S$ as the EAN.UCC GIAI, L as the length of the EAN.UCC company

2472 prefix, and F from Step 44 as the Filter Value, and S from Step 45 as the Serial

2473 Number. If the encoding procedure fails because an input is out of range, or because

2474 the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.

2475 47. Go to Step 53.

2476 48. Let the URI be written as $urn:epc:tag:encName:m_1m_2...m_L.C_1C_2...C_K.S_1S_2...S_S$.

2477 49. Interpret $m_1m_2...m_L$ as a decimal integer M .

2478 50. Interpret $C_1C_2...C_K$ as a decimal integer C .

2479 51. Interpret $S_1S_2...S_S$ as a decimal integer S .

2480 52. Carry out the encoding procedure defined in Section 3.4.1.1 using M from Step 49 as

2481 the General Manager Number, C from Step 50 as the Object Class, and S from

2482 Step 51 as the Serial Number. If the encoding procedure fails because an input is out

2483 of range, or because the procedure indicates a failure, stop: this URI cannot be

2484 encoded into a bit string.

2485 53. The translation is complete.

2486 5.6 URI into Gen 2 Tag EPC Memory

2487 The following procedure converts a URI into a sequence of bits suitable for writing into the

2488 EPC memory of a Gen 2 Tag, starting with bit 10x (i.e., not including the CRC).

2489 1. If the URI is a Raw Tag URI of the form $urn:epc:raw:N.A.V$, calculate the

2490 value L , where $L = N/16$ rounded up to the nearest whole number. If $L \geq 32$, stop:

2491 this URI cannot be encoded into the EPC memory of a Gen 2 Tag. If $A \geq 256$ or if

2492 the value V is too large to be expressed as an N -bit binary integer, stop: this URI

2493 cannot be encoded into the EPC memory of a Gen 2 Tag. Otherwise, construct the

2494 contents of EPC memory by concatenating the following bit strings: the value L (five

2495 bits), two zero bits (00), a single one bit (1), the value A (eight bits), and the value V

2496 (16L bits).

2497 2. Otherwise, apply the procedure of Section 5.5 to obtain an N-bit string, V. If the
2498 procedure of Section 5.5 fails, stop: this URI cannot be encoded into the EPC
2499 memory of a Gen 2 Tag. Otherwise, calculate $L = N/16$ rounded up to the nearest
2500 whole number. Construct the contents of EPC memory by concatenating the
2501 following bit strings: the value L (five bits), eleven zero bits (00000000000), the
2502 value V (N bits), and as many zero bits as required to make a total of $16(L+1)$ bits.

2503 6 Semantics of EPC Pattern URIs

2504 The meaning of an EPC Pattern URI (`urn:epc:pat:`) or EPC Pure Identity Pattern URI
2505 (`urn:epc:idpat:`) can be formally defined as denoting a set of encoding-specific EPCs
2506 or a set of pure identity EPCs, respectively.

2507 The set of EPCs denoted by a specific EPC Pattern URI is defined by the following decision
2508 procedure, which says whether a given EPC Tag URI belongs to the set denoted by the EPC
2509 Pattern URI.

2510 Let `urn:epc:pat:EncName:P1.P2...Pn` be an EPC Pattern URI. Let
2511 `urn:epc:tag:EncName:C1.C2...Cn` be an EPC Tag URI, where the *EncName* field
2512 of both URIs is the same. The number of components (*n*) depends on the value of
2513 *EncName*.

2514 First, any EPC Tag URI component *Ci* is said to *match* the corresponding EPC Pattern URI
2515 component *Pi* if:

- 2516 • *Pi* is a *NumericComponent*, and *Ci* is equal to *Pi*; or
- 2517 • *Pi* is a *PaddedNumericComponent*, and *Ci* is equal to *Pi* both in numeric value as
2518 well as in length; or
- 2519 • *Pi* is a *GS3A3Component*, and *Ci* is equal to *Pi*, character for character; or
- 2520 • *Pi* is a *CAGECodeOrDODAAC*, and *Ci* is equal to *Pi*; or
- 2521 • *Pi* is a *RangeComponent* [*lo-hi*], and $lo \leq Ci \leq hi$; or
- 2522 • *Pi* is a *StarComponent* (and *Ci* is anything at all)

2523 Then the EPC Tag URI is a member of the set denoted by the EPC Pattern URI if and only if
2524 *Ci* matches *Pi* for all $1 \leq i \leq n$.

2525 The set of pure identity EPCs denoted by a specific EPC Pure Identity URI is defined by a
2526 similar decision procedure, which says whether a given EPC Pure Identity URI belongs to
2527 the set denoted by the EPC Pure Identity Pattern URI.

2528 Let `urn:epc:idpat:SchemeName:P1.P2...Pn` be an EPC Pure Identity Pattern
2529 URI. Let `urn:epc:id:SchemeName:C1.C2...Cn` be an EPC Pure Identity URI,
2530 where the *SchemeName* field of both URIs is the same. The number of components (*n*)
2531 depends on the value of *SchemeName*.

2532 Then the EPC Pure Identity URI is a member of the set denoted by the EPC Pure Identity
2533 Pattern URI if and only if C_i matches P_i for all $1 \leq i \leq n$, where “matches” is as defined
2534 above.

2535 **7 Background Information (non-normative)**

2536 This document draws from the previous work at the Auto-ID Center, and we recognize the
2537 contribution of the following individuals: David Brock (MIT), Joe Foley (MIT), Sunny Siu
2538 (MIT), Sanjay Sarma (MIT), and Dan Engels (MIT). In addition, we recognize the
2539 contribution from Steve Rehling (P&G) on EPC to GTIN mapping.

2540 The following papers capture the contributions of these individuals:

- 2541 • Engels, D., Foley, J., Waldrop, J., Sarma, S. and Brock, D., "The Networked Physical
2542 World: An Automated Identification Architecture"
2543 2nd IEEE Workshop on Internet Applications (WIAPP '01),
2544 (<http://csdl.computer.org/comp/proceedings/wiapp/2001/1137/00/11370076.pdf>)
- 2545 • Brock, David. "The Electronic Product Code (EPC), A Naming Scheme for Physical
2546 Objects", 2001. (<http://www.autoidlabs.org/whitepapers/MIT-AUTOID-WH-002.pdf>)
- 2547 • Brock, David. "The Compact Electronic Product Code; A 64-bit Representation of the
2548 Electronic Product Code", 2001. ([http://www.autoidlabs.com/whitepapers/MIT-](http://www.autoidlabs.com/whitepapers/MIT-AUTOID-WH-008.pdf)
2549 [AUTOID-WH-008.pdf](http://www.autoidlabs.com/whitepapers/MIT-AUTOID-WH-008.pdf))
- 2550 • D. Engels, "The Use of the Electronic Product Code™," MIT Auto-ID Center Technical
2551 Report MIT-TR007, February 2003, ([http://www.autoidlabs.com/whitepapers/mit-](http://www.autoidlabs.com/whitepapers/mit-autoid-tr009.pdf)
2552 [autoid-tr009.pdf](http://www.autoidlabs.com/whitepapers/mit-autoid-tr009.pdf))
- 2553 • R. Moats, "URN Syntax," Internet Engineering Task Force Request for Comments RFC-
2554 2141, May 1997, (<http://www.ietf.org/rfc/rfc2141.txt>)

2555 **8 References**

- 2556 [EAN.UCCGS] "General EAN.UCC Specifications." Version 6.0, EAN.UCC, IncTM.
- 2557 [MIT-TR009] D. Engels, "The Use of the Electronic Product Code™," MIT Auto-ID Center
2558 Technical Report MIT-TR007, February 2003, [http://www.autoidlabs.com/whitepapers/mit-](http://www.autoidlabs.com/whitepapers/mit-autoid-tr009.pdf)
2559 [autoid-tr009.pdf](http://www.autoidlabs.com/whitepapers/mit-autoid-tr009.pdf)
- 2560 [RFC2141] R. Moats, "URN Syntax," Internet Engineering Task Force Request for
2561 Comments RFC-2141, May 1997, <http://www.ietf.org/rfc/rfc2141.txt>.
- 2562 [DOD Constructs] "United States Department of Defense Suppliers' Passive RFID
2563 Information Guide," <http://www.dodrfid.org/supplierguide.htm>
- 2564 [Gen2 Specification] "EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF
2565 RFID Protocol for Communications at 860 MHz-960MHz Version 1.0.9"

9 Appendix A: Encoding Scheme Summary Tables (non-normative)

SGTIN Summary						
SGTIN-96	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
	8	3	3	20-40	24 - 4	38
	0011 0000 (Binary value)	(Refer to Table below for values)	(Refer to Table below for values)	999,999 – 999,999,999,999 (Max. decimal range**)	9,999,999 – 9 (Max .decimal range**)	274,877,906,943 (Max .decimal value)
SGTIN-198	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
	8	3	3	20-40	24 - 4	140
	0011 0110 (Binary value)	(Refer to Table below for values)	(Refer to Table below for values)	999,999 – 999,999,999,999 (Max. decimal range**)	9,999,999 – 9 (Max .decimal range**)	Up to 20 alphanumeric characters
Filter Values (Non-normative)		SGTIN Partition Table				
Type	Binary Value	Partition Value	Company Prefix		Indicator Digit and Item Reference	
All Others	000		Bits	Digits	Bits	Digit
Retail Consumer Trade Item	001	0	40	12	4	1
Standard Trade Item Grouping	010	1	37	11	7	2
Single Shipping / Consumer Trade Item	011	2	34	10	10	3
Reserved	100	3	30	9	14	4
Reserved	101	4	27	8	17	5
Reserved	110	5	24	7	20	6
Reserved	111	6	20	6	24	7

*Range of Item Reference field varies with the length of the Company Prefix

**Range of Company Prefix and Item Reference fields vary according to the contents of the Partition field.

2572

SSCC Summary						
SSCC-96	Header	Filter Value	Partition	Company Prefix	Serial Reference	Unallocated
	8	3	3	20-40	38-18	24
	0011 0001 (Binary value)	(Refer to Table below for values)	(Refer to Table below for values)	999,999 – 999,999,999,999 (Max. decimal range**)	99,999,999,999 – 99,999 (Max. decimal range**)	[Not Used]
Filter Values (Non-normative)		SSCC Partition Table				
Type	Binary Value	Partition Value	Company Prefix		Extension Digit and Serial Reference	
All Others	000		Bits	Digits	Bits	Digits
Undefined	001	0	40	12	18	5
Logistical / Shipping Unit	010	1	37	11	21	6
Reserved	011	2	34	10	24	7
Reserved	100	3	30	9	28	8
Reserved	101	4	27	8	31	9
Reserved	110	5	24	7	34	10
Reserved	111	6	20	6	38	11

2573

2574

*Range of Serial Reference field varies with the length of the Company Prefix

**Range of Company Prefix and Serial Reference fields vary according to the contents of the Partition field.

SGLN Summary						
SGLN-96	Header	Filter Value	Partition	Company Prefix	Location Reference	Extension Component
	8	3	3	20-40	21-1	41
	0011 0010 (Binary value)	(Refer to Table below for values)	(Refer to Table below for values)	999,999 – 999,999,999,999 (Max. decimal range**)	999,999 – 0 (Max. decimal range**)	2,199,023,255,551 (Max Decimal Value) Recommend: Min=1 Max=999,999,999,999 Reserved=0 All bits shall be set to 0 when an Extension Component is not encoded signifying GLN only.
SGLN-195	Header	Filter Value	Partition	Company Prefix	Location Reference	Extension component
	8	3	3	20-40	21-1	140
	0011 1001 (Binary value)	(Refer to Table below for values)	(Refer to Table below for values)	999,999 – 999,999,999,999 (Max. decimal range**)	999,999 – 0 (Max. decimal range**)	Up to 20 alphanumeric characters If Extension Component is not used these 140 bits shall all be set to binary 0
Filter Values (Non-normative)		SGLN Partition Table				
Type	Binary Value	Partition Value	Company Prefix		Location Reference	
			Bits	Digits	Bits	Digit
All Others	000					
Physical Location	001	0	40	12	1	0
Reserved	010	1	37	11	4	1
Reserved	011	2	34	10	7	2
Reserved	100	3	30	9	11	3
Reserved	101	4	27	8	14	4
Reserved	110	5	24	7	17	5
Reserved	111	6	20	6	21	6

2576 *Range of Location Reference field varies with the length of the Company Prefix

2577 **Range of Company Prefix and Location Reference fields vary according to contents of the Partition field.

2578

GRAI Summary						
GRAI-96	Header	Filter Value	Partition	Company Prefix	Asset Type	Serial Number
	8	3	3	20-40	24 – 4	38
	0011 0011 (Binary value)	(Refer to Table below for values)	(Refer to Table below for values)	999,999 – 999,999,999,999 (Max. decimal range**)	999,999 – 0 (Max. decimal range**)	274,877,906,943 (Max. decimal value)
GRAI-170	Header	Filter Value	Partition	Company Prefix	Asset Type	Serial Number
	8	3	3	20-40	24 – 4	112
	0011 0111 (Binary value)	(Refer to Table below for values)	(Refer to Table below for values)	999,999 – 999,999,999,999 (Max. decimal range**)	999,999 – 0 (Max. decimal range**)	Up to 16 alphanumeric characters
Filter Values (Non-normative)		GRAI Partition Table				
Type	Binary Value	Partition Value	Company Prefix		Asset Type***	
All Others	000		Bits	Digits	Bits	Digit
Reserved	001	0	40	12	4	0
Reserved	010	1	37	11	7	1
Reserved	011	2	34	10	10	2
Reserved	100	3	30	9	14	3
Reserved	101	4	27	8	17	4
Reserved	110	5	24	7	20	5
Reserved	111	6	20	6	24	6

2579 *Range of Asset Type field varies with Company Prefix.

2580 **Range of Company Prefix and Asset Type fields vary according to contents of the Partition field.

2581 *** Explanation (non-normative): The Asset Type field of the GRAI-96 has four more bits than necessary given
2582 the capacity of that field.

GIAI Summary					
GIAI-96	Header	Filter Value	Partition	Company Prefix	Individual Asset Reference
	8	3	3	20-40	62-42
	0011 0100 (Binary value)	(Refer to Table below for values)	(Refer to Table below for values)	999,999 – 999,999,999,999 (Max. decimal range*)	4,611,686,018,427,387,903 - 4,398,046,511,103 (Max. decimal range*)
GIAI-202	Header	Filter Value	Partition	Company Prefix	Individual Asset Reference
	8	3	3	20-40	168-126
	0011 1000 (Binary value)	(Refer to Table below for values)	(Refer to Table below for values)	999,999 – 999,999,999,999 (Max. decimal range*)	Up to 24 alphanumeric characters
Filter Values (To be confirmed)		GIAI Partition Table			
Type	Binary Value	Partition Value	Company Prefix		Individual Asset Reference
All Others	000		Bits	Digits	Bits Digits
Reserved	001	<GIAI-96>			
Reserved	010	0	40	12	42 12
Reserved	011	1	37	11	45 13
Reserved	100	2	34	10	48 14
Reserved	101	3	30	9	52 15
Reserved	110	4	27	8	55 16
Reserved	111	5	24	7	58 17
		6	20	6	62 18
		<GIAI-202>			
		0	40	12	126 18
		1	37	11	133 19
		2	34	10	140 20
		3	30	9	147 21
		4	27	8	154 22
		5	24	7	161 23
		6	20	6	168 24

2584 *Range of Company Prefix and Individual Asset Reference fields vary according to contents of the Partition field.

10 Appendix B: TDS 1.3 EAN.UCC Identities Bit Allocation and Required Physical Tag Bit Length for Encoding (non-normative)

Memory Bank Names	Reserved Memory Bank	EPC Memory Bank							TID Memory Bank	User Memory Bank	Total bits required
EPC Memory Bank		CRC-16	Protocol Control Bits				EPC Bits				
Protocol Control Bits			Length bits	RFU	Numbering Systems Identifier						
EPC Identity Names \ Bit Field	Reserved Memory bits	CRC-16 bits	Length bits	RFU bits	EPC/ISO Toggle bit	Reserved / AFI bits	EPC Header + Filter value bits+ Partition value bits + Domain Identifier bits	Word Boundary Filler bits	TID bits	User Memory bits	
GID-96	64	16	5	2	1	8	96	0	32	0	224
SGTIN-96	64	16	5	2	1	8	96	0	32	0	224
SGTIN-198	64	16	5	2	1	8	198	10	32	0	336
SSCC-96	64	16	5	2	1	8	96	0	32	0	224
SGLN-96	64	16	5	2	1	8	96	0	32	0	224
SGLN-195	64	16	5	2	1	8	195	13	32	0	333
GRAI-96	64	16	5	2	1	8	96	0	32	0	224
GRAI-170	64	16	5	2	1	8	170	6	32	0	304
GIAI-96	64	16	5	2	1	8	96	0	32	0	224
GIAI-202	64	16	5	2	1	8	202	6	32	0	336

2590 Notes:

2591 GIAI-202 may have shorter Domain Identifier bits (Company Prefix and Individual Asset

2592 Reference) which will shorten the total bit requirement to 302 bits.

2593 All the bits except for CRC-16 in the EPC Memory Bank requires encoding by application or

2594 process

2595 This table illustrates the total number of bits required in the three logical memories (TID,

2596 Reserved and EPC) to support the EAN.UCC identities listed. User memory is set to zero

2597 required bits to load a single identity in the tag. As larger memories are defined and the User

2598 memory method of allocation is defined in this standard, additional bits can be assigned to

2599 user memory.

2600 The EPC bits includes the extra bits required to round up to a fill the last 16 bit word.

2601 The four identities; SGTIN-198, SGLN-195, GRAI-170 and GIAI-202 have been included in

2602 this standard to indicate to hardware vendors the user requirements for tag sizes and memory

2603 allocation required to support these longer identities. Please note that all three required more

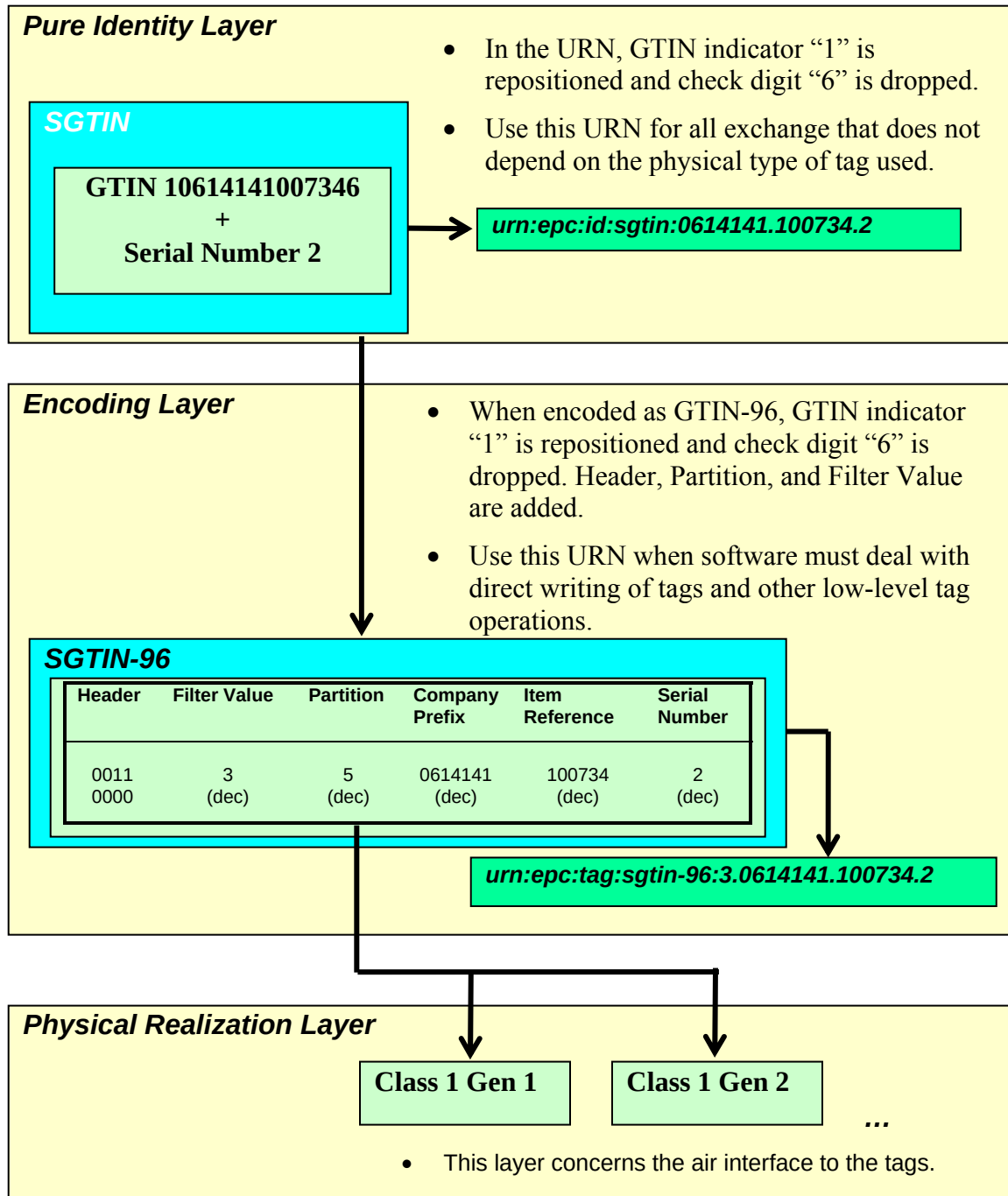
2604 than 256 bits to contain all the fields required.

2605 The Generation two protocol allows for reserved commands that are anticipated to provide

2606 dynamic assignment of memory as well as fixed static memory assignment.

11 Appendix C: Example of a Specific Trade Item <SGTIN> (non-normative)

This section presents an example of a specific trade item using SGTIN (Serialized GTIN). Each representation serves a distinct purpose in the software stack. Generally, the highest applicable level should be used. The GTIN used in the example is 10614141007346.



2612



2613
2614
2615
2616

	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
SGTIN-96	8 bits	3 bits	3 bits	24 bits	20 bits	38 bits
	0011 0000 (Binary value)	3 (Decimal value)	5 (Decimal value)	0614141 (Decimal value)	100734 (Decimal value)	2 (Decimal value)

2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628

- (01) is the Application Identifier for GTIN, and (21) is the Application Identifier for Serial Number. Application Identifiers are used in certain bar codes. The header fulfills this function (and others) in EPC.
- Header for SGTIN-96 is 00110000.
- Filter Value of 3 (Single Shipping/ Consumer Trade Item) was chosen for this example.
- Since the Company Prefix is seven-digits long (0614141), the Partition value is 5. This means Company Prefix has 24 bits and Item Reference has 20 bits.
- Indicator digit 1 is repositioned as the first digit in the Item Reference.
- Check digit 6 is dropped.

2629 Explanation of SGTIN Filter Values (non-normative).

2630 SGTINs can be assigned at several levels, including: item, inner pack, case, and pallet.

2631 RFID can read through cardboard, and reading un-needed tags can slow us down, so Filter

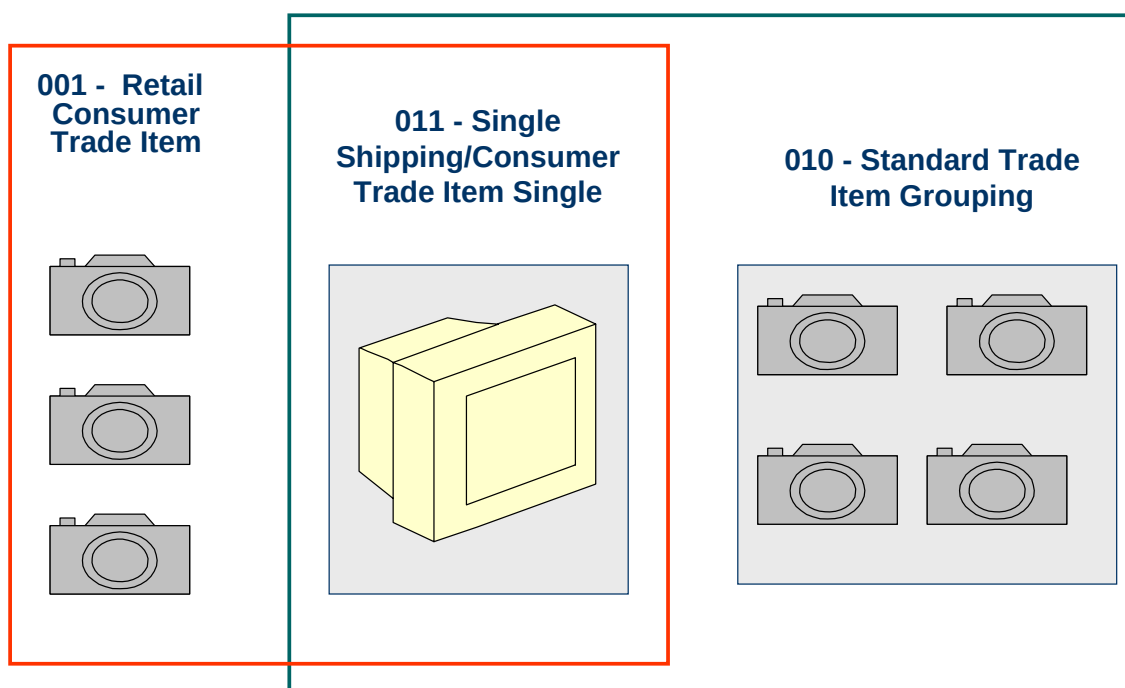
2632 Values are used to “filter in” desired tags, or “filter out” unwanted tags. Filter values are

2633 used within the key type (i.e. SGTIN). While it is possible that filter values for several levels

2634 of packaging may be defined in the future, it was decided to use a minimum of values for

2635 now until the community gains more practical experience in their use. Therefore the three
2636 major categories of SGTIN filter values can be thought of in the following high level terms:
2637 • Single Unit: A Retail Consumer Trade Item
2638 • Not-a-single unit: A Standard Trade Item Grouping
2639 • Items that could be included in both categories: For example, a Single Shipping
2640 container that contains a Single Consumer Trade Item

Three Filter Values



2641

2642

2643

2644

2645

2646

12 Appendix D: Decimal values of powers of 2 Table (non-normative)

n	(2ⁿ)₁₀	n	(2ⁿ)₁₀
0	1	33	8,589,934,592
1	2	34	17,179,869,184
2	4	35	34,359,738,368
3	8	36	68,719,476,736
4	16	37	137,438,953,472
5	32	38	274,877,906,944
6	64	39	549,755,813,888
7	128	40	1,099,511,627,776
8	256	41	2,199,023,255,552
9	512	42	4,398,046,511,104
10	1,024	43	8,796,093,022,208
11	2,048	44	17,592,186,044,416
12	4,096	45	35,184,372,088,832
13	8,192	46	70,368,744,177,664
14	16,384	47	140,737,488,355,328
15	32,768	48	281,474,976,710,656
16	65,536	49	562,949,953,421,312
17	131,072	50	1,125,899,906,842,624
18	262,144	51	2,251,799,813,685,248
19	524,288	52	4,503,599,627,370,496
20	1,048,576	53	9,007,199,254,740,992
21	2,097,152	54	18,014,398,509,481,984
22	4,194,304	55	36,028,797,018,963,968
23	8,388,608	56	72,057,594,037,927,936
24	16,777,216	57	144,115,188,075,855,872
25	33,554,432	58	288,230,376,151,711,744
26	67,108,864	59	576,460,752,303,423,488
27	143,217,728	60	1,152,921,504,606,846,976
28	268,435,456	61	2,305,843,009,213,693,952
29	536,870,912	62	4,611,686,018,427,387,904
30	1,073,741,824	63	9,223,372,036,854,775,808
31	2,147,483,648	64	18,446,744,073,709,551,616
32	4,294,967,296		

2647

13 Appendix E: List of Abbreviations

BAG	Business Action Group
EPC	Electronic Product Code
EPCIS	EPC Information Services
GIAI	Global Individual Asset Identifier
GID	General Identifier
GLN	Global Location Number
GRAI	Global Returnable Asset Identifier
GTIN	Global Trade Item Number
HAG	Hardware Action Group
ONS	Object Naming Service
RFID	Radio Frequency Identification
SAG	Software Action Group
SGLN	Serialized Global Location Number
SSCC	Serial Shipping Container Code
URI	Uniform Resource Identifier
URN	Uniform Resource Name

14 Appendix F: General EAN.UCC Specifications (non-normative)

(Section 3 Definition of Element Strings and Section 3.7 EPCglobal Tag Data Standard.)

This section provides GS1 approval of this version of the EPCglobal® Tag Data Standard with the following EAN.UCC Application Identifier definition restrictions:

Companies should use the EAN.UCC specifications to define the applicable fields in databases and other ICT-systems.

For EAN.UCC use of EPC96-bit tags, the following applies:

- **AI (00) SSCC** (no restrictions)
- **AI (01) GTIN + AI (21) Serial Number:** The Section 3.6.13 Serial Number definition is restricted to permit assignment of 274,877,906,943 numeric-only serial numbers)
- **AI (414) GLN + AI (254) GLN Extension Component:** The Tag Data Standard V1.1 R1.27 is approved for the use of GLN Extension with the restrictions specified in Section 2.4.6.1 of the General EAN.UCC Specifications..
- **AI (8003) GRAI Serial Number:** The Section 3.6.49 Global Returnable Asset Identifier definition is restricted to permit assignment of 274,877,906,943 numeric-only serial numbers and the serial number element is mandatory.
- **AI (8004) GIAI Serial Number:** The Section 3.6.50 Global Individual Asset Identifier definition is restricted to permit assignment of 4,611,686,018,427,387,904 numeric-only serial numbers.

For EAN.UCC use of EPC longer than 96-bit tags, the following applies:

- **AI (00) SSCC** (no restrictions)
- **AI (01) GTIN + AI (21) Serial Number:** (no restrictions)
- **AI (414) GLN + AI (254) Extension Component:** (no restrictions).
- **AI (8003) GRAI Serial Number:** (no restrictions)
- **AI (8004) GIAI Serial Number:** (no restrictions)

2678
2679

15 Appendix G: EAN.UCC Alphanumeric Character Set (Normative)

ISO/IEC 646 Subset

Unique Graphic Character Allocations					
Graphic Symbol	Name	Hex Coded Representation	Graphic Symbol	Name	Hex Coded Representation
!	Exclamation mark	21	M	Capital letter M	4D
"	Quotation mark	22	N	Capital letter N	4E
%	Percent sign	25	O	Capital letter O	4F
&	Ampersand	26	P	Capital letter P	50
'	Apostrophe	27	Q	Capital letter Q	51
(	Left parenthesis	28	R	Capital letter R	52
)	Right parenthesis	29	S	Capital letter S	53
*	Asterisk	2A	T	Capital letter T	54
+	Plus sign	2B	U	Capital letter U	55
,	Comma	2C	V	Capital letter V	56
-	Hyphen/Minus	2D	W	Capital letter W	57
.	Full stop	2E	X	Capital letter X	58
/	Solidus	2F	Y	Capital letter Y	59
0	Digit zero	30	Z	Capital letter Z	5A
1	Digit one	31	_	Low line	5F
2	Digit two	32	a	Small letter a	61
3	Digit three	33	b	Small letter b	62
4	Digit four	34	c	Small letter c	63
5	Digit five	35	d	Small letter d	64
6	Digit six	36	e	Small letter e	65
7	Digit seven	37	f	Small letter f	66
8	Digit eight	38	g	Small letter g	67
9	Digit nine	39	h	Small letter h	68
:	Colon	3A	i	Small letter i	69
;	Semicolon	3B	j	Small letter j	6A
<	Less-than sign	3C	k	Small letter k	6B
=	Equals sign	3D	l	Small letter l	6C
>	Greater-than sign	3E	m	Small letter m	6D
?	Question mark	3F	n	Small letter n	6E
A	Capital letter A	41	o	Small letter o	6F
B	Capital letter B	42	p	Small letter p	70

C	Capital letter C	43	q	Small letter q	71
D	Capital letter D	44	r	Small letter r	72
E	Capital letter E	45	s	Small letter s	73
F	Capital letter F	46	t	Small letter t	74
G	Capital letter G	47	u	Small letter u	75
H	Capital letter H	48	v	Small letter v	76
I	Capital letter I	49	w	Small letter w	77
J	Capital letter J	4A	x	Small letter x	78
K	Capital letter K	4B	y	Small letter y	79
L	Capital letter L	4C	z	Small letter z	7A

2680 Notes

2681 Readers should be aware that this table is derived from [EAN.UCCGS] and may include
2682 discrepancy with the original specification at any given time. Readers are advised to always
2683 consult the original specification upon implementation.

2684 This table specifies the allowed subset of ISO/IEC 646 characters that shall be used for
2685 encoding alphanumeric Serial Number/Extension Component in this standard. The SGTIN-
2686 198, SGLN-195, GRAI-170 and GIAI-202 encodings use this table.

2687 Each entry in this table gives a 7-bit code for a character, expressed in hexadecimal. For
2688 example, “Capital Letter K” has a 7-bit code of 1001011, expressed as “4B” in the table.

2689 The 7-bit codes in this table are identical to ISO/IEC 646 (ASCII) character codes.