EPCglobal Tag Data Standards Version 1.3

Ratified Specification

March 8, 2006

DOCUMENT HISTORY

Document Number:	
Document Version:	1.3
Document Date :	2005-11-21

Document Summary

Document Title:	**EPC™ Tag Data Standards Version 1.3**
Owner:	Tag Data Standard Work Group
Status:	(*check one box*) ☐ DRAFT X Approved

Document Change History

Date of Change	Version	Reason for Change	Summary of Change
			•
			•

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

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.

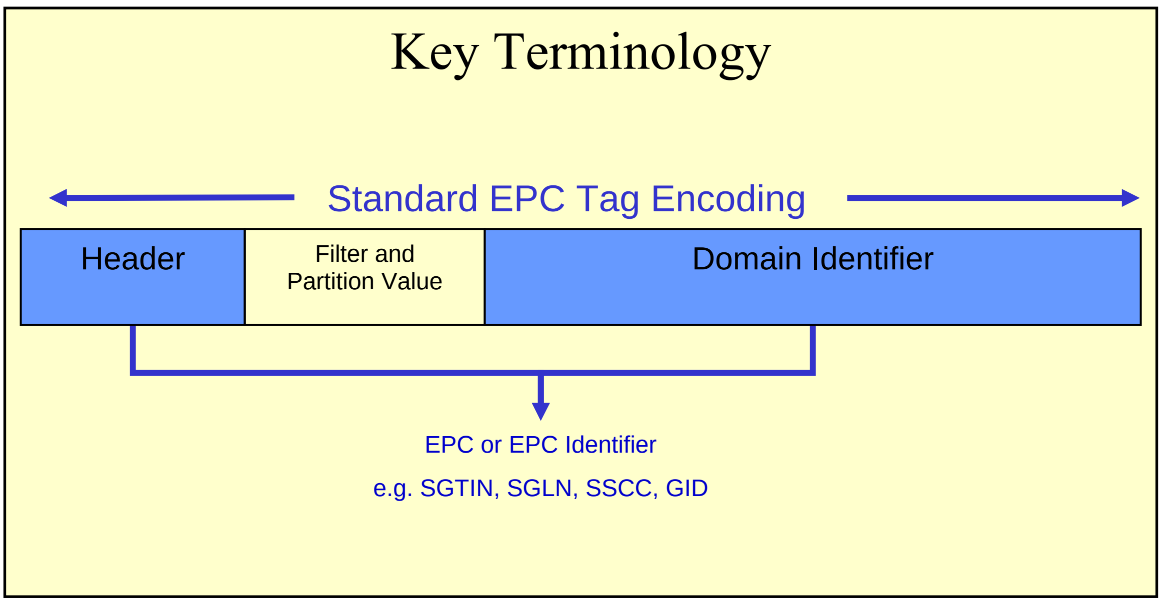


Figure A. EPC Terminology

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.

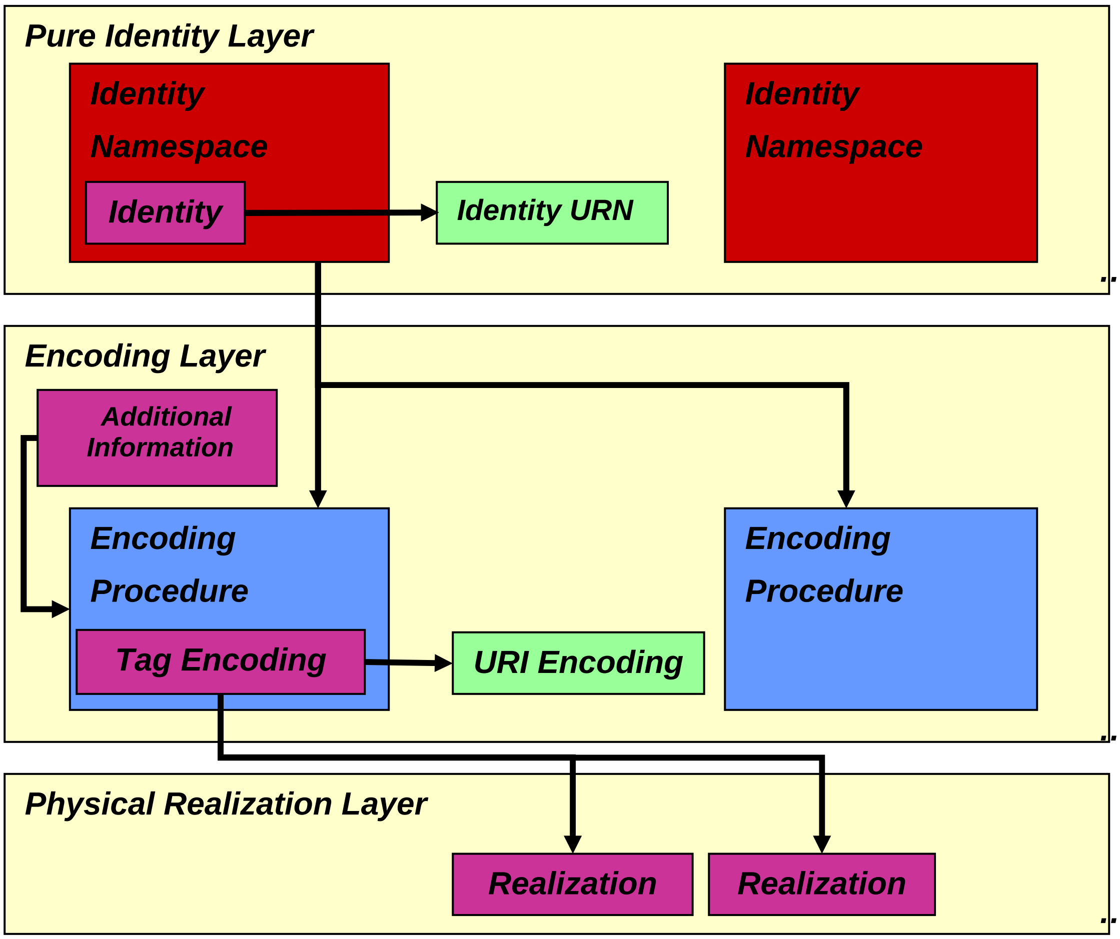


Figure B. Defined Identity Namespaces, Encodings, and Realizations.

- 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.

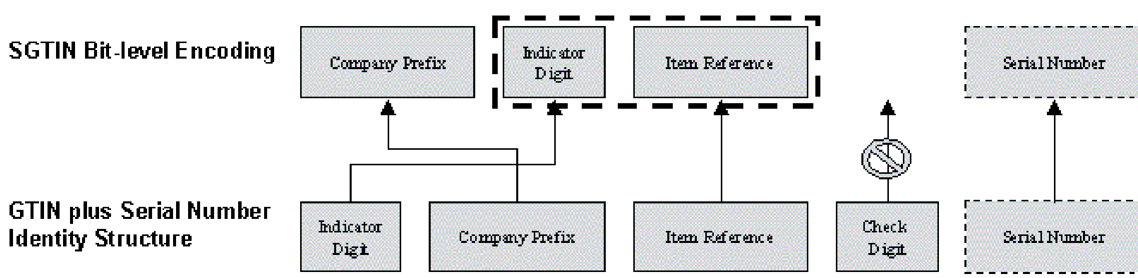


Figure C. How the parts of the decimal SGTIN are extracted, rearranged, and augmented for encoding.

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.

Explanation (non-normative): The restrictions are necessary for 96-bit tags in order for serial numbers to fit within the small number of bits available in earlier Class 1 Generation 1 tags. The serial number range is restricted to numeric values and alphanumeric serial numbers are disallowed. Leading zeros are forbidden so that the serial number 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, serial numbers can't have leading zeros. Then, when the bits 0000000000000000000010010 on the tag are seen, the serial number as "34" (not "034" or "00034") is decoded.

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.

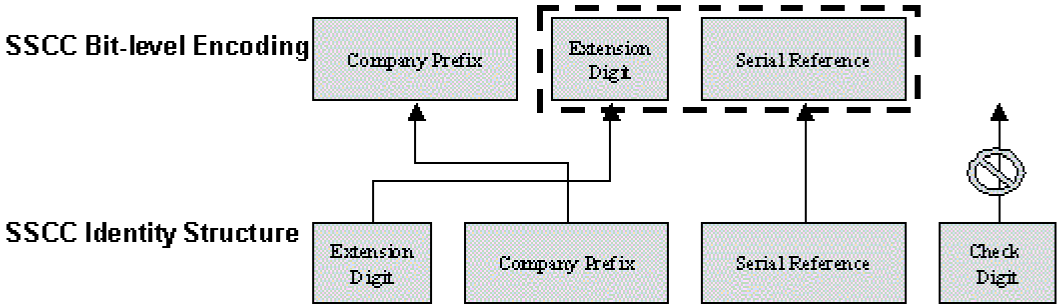


Figure D. How the parts of the decimal SSCC are extracted and rearranged for encoding.

2.1.2.3 Serialized Global Location Number (SGLN)

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

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

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

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

The SGLN 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 GLN decimal code.
- The *Location Reference*, assigned uniquely by the managing entity to an aggregate or specific physical location.
- The *GLN Extension*, assigned by the managing entity to an individual unique location.
  - ➢ The use of the GLN Extension is intended for internal purposes. For communication between trading partners a GLN will be used. The rules defining the use of the SGLN are described in Section 3.7.

.

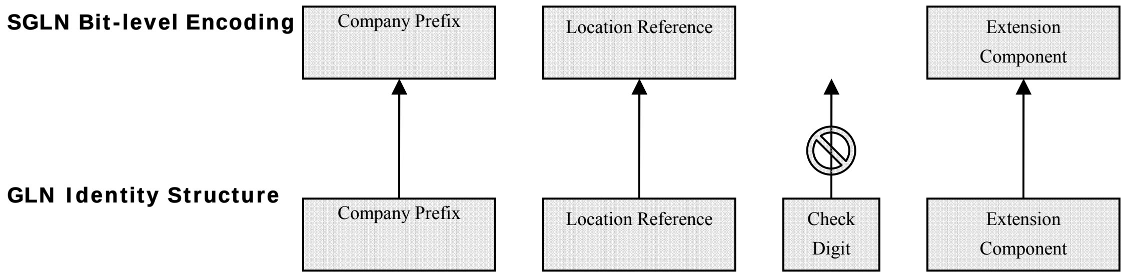


Figure E. How the parts of the decimal SGLN are extracted and rearranged for encoding

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 is (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.

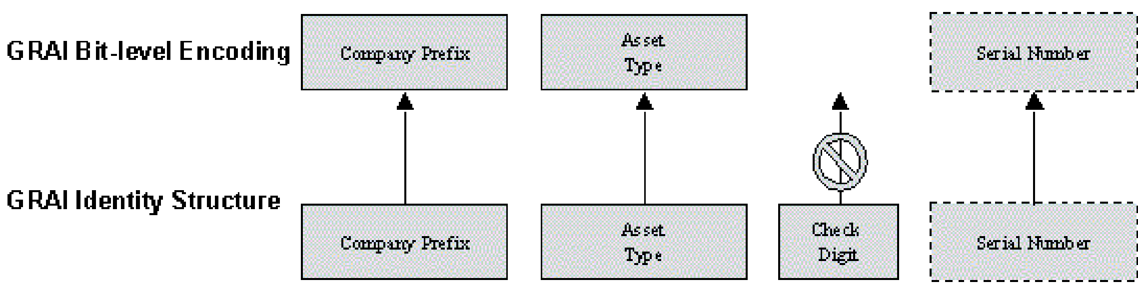


Figure F. How the parts of the decimal GRAI are extracted and rearranged for encoding.

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.

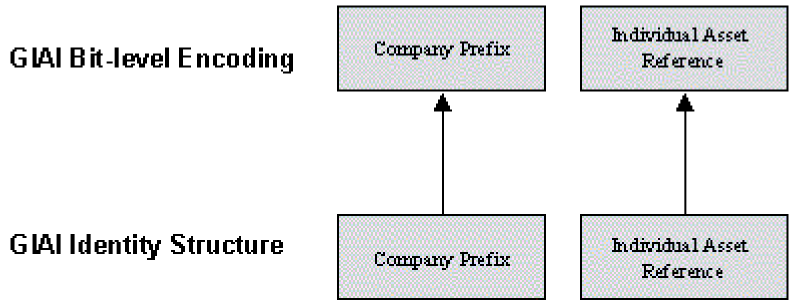


Figure G. How the parts of the decimal GIAI are extracted and rearranged for encoding.

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/supplierguide.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.

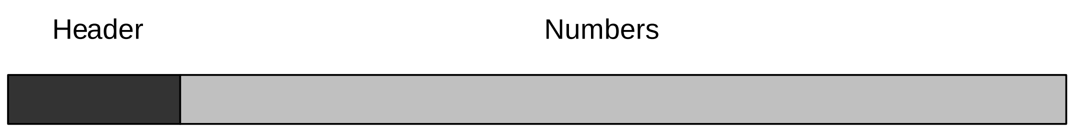


Figure H. The general structure of EPC encodings is as a string of bits, consisting of a fixed length header followed by a series of value fields, whose overall length, structure, and function are completely determined by the header value.

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
0000 0001	01	NA	Reserved for Future Use
0000 001x	02,03	NA	Reserved for Future Use
0000 01xx	04,05	NA	Reserved for Future Use
	06,07	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
0000 1100 to 0000 1111	0C to 0F		Reserved until 64 bit Sunset Due to 64 bit encoding rule in Gen 1
0001 0000 to 0010 1110	10 to 2E	NA NA	Reserved for Future Use
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	36	198	SGTIN-198
0011 0111	37	170	GRAI-170
0011 1000	38	202	GIAI-202
0011 1001	39	195	SGLN-195
0011 1010 to 0011 1111	3A to 3F		Reserved for future Header values
0100 0000 to 0111 1111	40 to 7F		Reserved until 64 bit Sunset
1000 0000 to 1011 1111	80 to BF	64	Reserved until 64 bit Sunset <SGTIN-64> (64 header values)
1100 0000 to 1100 1101	C0 to CD		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>
1100 1111 to 1111 1110	CF to FE		Reserved until 64 bit Sunset
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 contans 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:

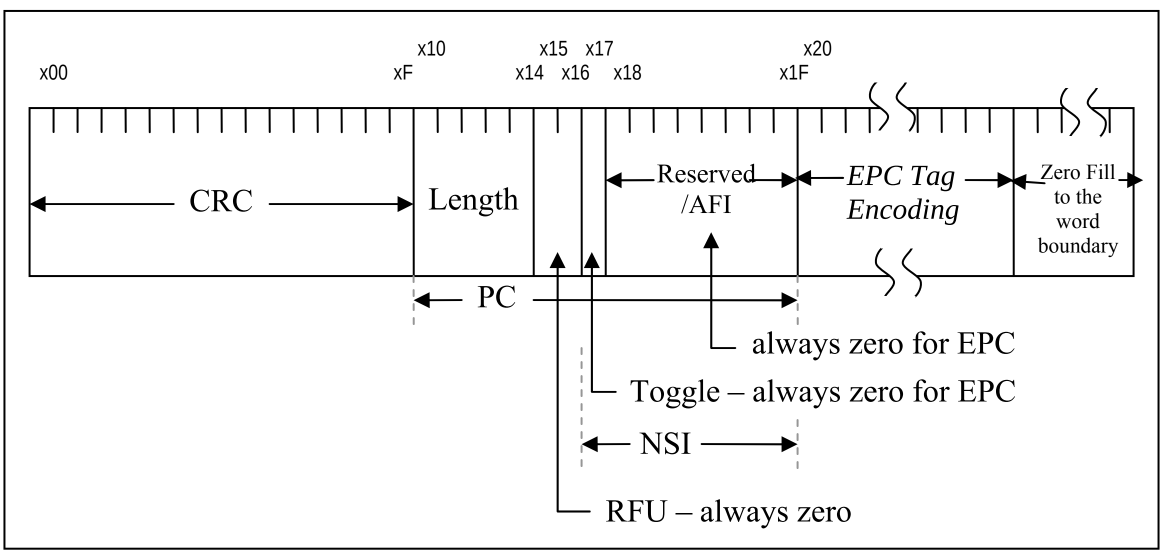


Figure I. Complete contents of EPCmemory bank of a Gen 2 Tag.

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.

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

3.2.2 The Length Bits

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

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

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

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

3.3 Notational Conventions

In the remainder of this section, EPC Tag Encoding schemes are depicted using the 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)

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

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

Procedure:

1. Construct the General Manager Number by considering digits $d_1d_2\ldots d_8$ 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 00110101$b_{87}b_{86}\ldots b_0$ (where the first eight bits 00110101 are the header)

Yields:

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

Procedure:

1. Bits $b_{87}b_{86}\ldots b_{60}$, considered as an unsigned integer, are the General Manager Number.
2. Bits $b_{59}b_{58}\ldots b_{36}$, considered as an unsigned integer, are the Object Class.
3. Bits $b_{35}b_{34}\ldots b_0$, 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 $d_1d_2 \ldots d_{14}$

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

- A Serial Number S where $0 \leq S < 2^{38}$, *or* an EAN.UCC-128 Application Identifier 21 consisting of characters $s_1 s_2 \ldots s_K$,.
- 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-96.

2. Construct the Company Prefix by concatenating digits $d_2 d_3 \ldots d_{(L+1)}$ and considering the result to be a decimal integer, C.

3. Construct the Indicator Digit and Item Reference by concatenating digits $d_1 d_{(L+2)} d_{(L+3)} \ldots d_{13}$ and considering the result to be a decimal integer, I.

4. When the Serial Number is provided directly as an integer S where $0 \leq S < 2^{38}$, proceed to Step 5. Otherwise, when the Serial Number is provided as an EAN.UCC-128 Application Identifier 21 consisting of characters $s_1 s_2 \ldots s_K$, construct the Serial Number by concatenating digits $s_1 s_2 \ldots s_K$. If any of these characters is not a digit, stop: this Serial Number cannot be encoded in the SGTIN-96 encoding. Also, if $K > 1$ and $s_1 = 0$, stop: this Serial Number cannot be encoded in the SGTIN-96 encoding (because leading zeros are not permitted except in the case where the Serial Number consists of a single zero digit). Otherwise, consider the result to be a decimal integer, S. If $S \geq 2^{38}$, stop: this Serial Number cannot be encoded in the SGTIN-96 encoding.

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

3.5.1.2 SGTIN-96 Decoding Procedure

Given:

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

Yields:

- An EAN.UCC GTIN-14
- A Serial Number
- A Filter Value

Procedure:

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

2. Extract the Partition Value P by considering bits $b_{84} b_{83} b_{82}$ as an unsigned integer. If $P = 7$, 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 $b_{81}b_{80}\ldots b_{(82\text{-M})}$ 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 $p_1p_2\ldots p_L$, adding leading zeros as necessary to make up *L* digits in total.

5. Extract the Item Reference and Indicator by considering bits $b_{(81\text{-M})}\,b_{(80\text{-M})}\ldots b_{38}$ as an unsigned integer. If this integer is greater than or equal to $10^{(13\text{-L})}$, stop: the input bit string is not a legal SGTIN-96 encoding. Otherwise, convert this integer to a (13-L)-digit decimal number $i_1i_2\ldots i_{(13\text{-L})}$, adding leading zeros as necessary to make (13-L) digits.

6. Construct a 13-digit number $d_1d_2\ldots d_{13}$ where $d_1 = i_1$ from Step 5, $d_2d_3\ldots d_{(L+1)} = p_1p_2\ldots p_L$ from Step 4, and $d_{(L+2)}d_{(L+3)}\ldots d_{13} = i_2\,i_3\ldots i_{(13\text{-L})}$ 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: $d_1d_2\ldots d_{14}$.

9. Bits $b_{37}b_{36}\ldots b_0$, 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 $d_1d_2\ldots d_{14}$
- The length L of the Company Prefix portion of the GTIN
- An EAN.UCC-128 Application Identifier 21 consisting of characters $s_1s_2\ldots s_K$, 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 $d_2d_3\ldots d_{(L+1)}$ and considering the result to be a decimal integer, C.

3. Construct the Indicator Digit and Item Reference by concatenating digits $d_1 d_{(L+2)} d_{(L+3)} \ldots d_{13}$ and considering the result to be a decimal integer, I.

4. . Check that each of the characters $s_1 s_2 \ldots s_K$ 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 SGTIN-198. Otherwise construct the Serial Number by concatenating the 7-bit code, as given in Appendix G, for each of the characters $s_1 s_2 \ldots s_K$, yielding 7K bits total. If K < 20, concatenate additional zero bits to the right to make a total of 140 bits.

5. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header 00110110 (8 bits), Filter Value F (3 bits), Partition Value P from Step 1 (3 bits), Company Prefix C from Step 2 (M bits), Item Reference from Step 3 (N bits), Serial Number from Step 4 (140 bits). Note that $M+N = 44$ bits for all P.

3.5.2.2 SGTIN-198 Decoding Procedure

Given:

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

Yields:

- An EAN.UCC GTIN-14
- A Serial Number
- A Filter Value

Procedure:

1. Bits $b_{189} b_{188} b_{187}$, considered as an unsigned integer, are the Filter Value.

2. Extract the Partition Value P by considering bits $b_{186} b_{185} b_{184}$ as an unsigned integer. If $P = 7$, stop: this bit string cannot be decoded as an SGTIN-198.

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 $b_{183} b_{182} \ldots b_{(184-M)}$ 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-198 encoding. Otherwise, convert this integer into a decimal number $p_1 p_2 \ldots p_L$, adding leading zeros as necessary to make up L digits in total.

5. Extract the Item Reference and Indicator by considering bits $b_{(183-M)}\, b_{(182-M)} \ldots b_{140}$ 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-198 encoding. Otherwise, convert this integer to a (13-L)-digit decimal number $i_1 i_2 \ldots i_{(13-L)}$, adding leading zeros as necessary to make (13-L) digits.

6. Construct a 13-digit number $d_1 d_2 \ldots d_{13}$ where $d_1 = i_1$ from Step 5, $d_2 d_3 \ldots d_{(L+1)} = p_1 p_2 \ldots p_L$ from Step 4, and $d_{(L+2)} d_{(L+3)} \ldots d_{13} = i_2\, i_3 \ldots i_{(13-L)}$ 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: $d_1 d_2 \ldots d_{14}$.

9. Divide the remaining bits $b_{139}b_{138}\ldots b_0$ 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. : $d_1d_2\ldots d_{14}\ s_1s_2\ldots s_K$

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 $d_1d_2\ldots d_{18}$
- 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 $d_2d_3\ldots d_{(L+1)}$ and considering the result to be a decimal integer, C.

3. Construct the Extension Digit and the Serial Reference by concatenating digits $d_1d_{(L+2)}d_{(L+3)}\ldots d_{17}$ 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:

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

Yields:

- An EAN.UCC SSCC
- A Filter Value

Procedure:

1. Bits $b_{87}b_{86}b_{85}$, considered as an unsigned integer, are the Filter Value.
2. Extract the Partition Value P by considering bits $b_{84}b_{83}b_{82}$ as an unsigned integer. If $P = 7$, stop: this bit string cannot be decoded as an SSCC-96.
3. Look up the Partition Value P in Table 10 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 $b_{81}b_{80}\ldots b_{(82\text{-}M)}$ as an unsigned integer. If this integer is greater than or equal to 10^L, stop: the input bit string is not a legal SSCC-96 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\ldots p_L$, adding leading zeros as necessary to make up L digits in total.
5. Extract the Serial Reference by considering bits $b_{(81\text{-}M)}\, b_{(80\text{-}M)}\ldots b_{24}$ as an unsigned integer. If this integer is greater than or equal to $10^{(17\text{-}L)}$, stop: the input bit string is not a legal SSCC-96 encoding. Otherwise, convert this integer to a (17-L)-digit decimal number $i_1i_2\ldots i_{(17\text{-}L)}$, adding leading zeros as necessary to make (17-L) digits.
6. Construct a 17-digit number $d_1d_2\ldots d_{17}$ where $d_1 = s_1$ from Step 5, $d_2d_3\ldots d_{(L+1)} = p_1p_2\ldots p_L$ from Step 4, and $d_{(L+2)}d_{(L+3)}\ldots d_{17} = i_2\, i_3\ldots i_{(17\text{-}L)}$ from Step 5.
7. Calculate the check digit $d_{18} = (-3(d_1 + d_3 + d_5 + d_7 + d_9 + d_{11} + d_{13} + d_{15} + d_{17}) - (d_2 + d_4 + d_6 + d_8 + d_{10} + d_{12} + d_{14} + d_{16}))$ mod 10.
8. The EAN.UCC SSCC is the concatenation of digits from Steps 6 and 7: $d_1d_2\ldots d_{18}$.

3.7 Serialized Global Location Number (SGLN)

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

The use of the GLN Extension Component is intended for internal company purposes. For communication between trading partners a GLN will be used. Trading partners can only use the GLN Extension through mutual agreement but would have to establish an “out of band” exchange of master data describing the extensions. If the GLN only encoding is used, then the *Extension Component* shall be set to a fixed value of binary “0” for SGLN-96 and to binary 0110000 followed by 133 binary “0” bits for SGLN-195 encoding as described in the 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 .

- *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.
- *Extension Component* contains a serial number. If an *Extension Component* is not used this value shall be set to a binary value of 0 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000. The SGLN-96 encoding is only capable of representing integer-valued Extension Components with limited range. The EAN.UCC specifications permit a broader range of Extension Components. The EAN.UCC-128 barcode symbology provides for a 20-character alphanumeric Extension Component to be associated with a GLN using Application Identifier (AI) 254 [EAN.UCCGS]. It is possible to convert between the Extension Component in the SGLN-96 tag encoding and the Extension Component in AI 254 barcodes under certain conditions. Specifically, such interconversion is possible when the alphanumeric Extension Component in AI 254 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 SGLN-96 tag encoding. These considerations are reflected in the encoding and decoding procedures below.

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

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 $d_1d_2\ldots d_{13}$
- 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 $s_1s_2\ldots s_K$, 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 $d_1d_2\ldots d_L$ and considering the result to be a decimal integer, C.

3. If L < 12 construct the Location Reference by concatenating digits $d_{(L+1)}d_{(L+2)}\ldots d_{12}$ 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 $s_1s_2\ldots s_K$, construct the Extension Component by concatenating characters $s_1s_2\ldots s_K$. 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 00110010$b_{87}b_{86}\ldots b_0$ (where the first eight bits 00110010 are the header)

Yields:

- An EAN.UCC GLN
- An Extension Component
- A Filter Value

Procedure:

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

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

3. Look up the Partition Value P in Table 13 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 $b_{81}b_{80}\ldots b_{(82\text{-}M)}$ as an unsigned integer. If this integer is greater than or equal to 10^{L}, stop: the input bit string is not a legal SGLN-96 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\ldots p_L$, adding leading zeros as necessary to make up L digits in total.

5. If L < 12 extract the Location Reference by considering bits $b_{(81\text{-}M)}\ b_{(80\text{-}M)}\ldots b_{41}$ as an unsigned integer. If this integer is greater than or equal to $10^{(12\text{-}L)}$, stop: the input bit string is not a legal SGLN-96 encoding. Otherwise, convert this integer to a (12−L)-digit decimal number $i_1i_2\ldots i_{(12\text{-}L)}$, adding leading zeros as necessary to make (12−L) digits.

6. Construct a 12-digit number $d_1d_2\ldots d_{12}$ where $d_1d_2\ldots d_L = p_1p_2\ldots p_L$ from Step 4, and if L < 12 $d_{(L+1)}d_{(L+2)}\ldots d_{12} = i_1\ i_2\ldots i_{(12\text{-}L)}$ from Step 5.

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

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

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

10. (Optional) If it is desired to represent the Extension Component as a EAN.UCC-128 Application Identifier 254, 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`".

3.7.2 SGLN-195

In addition to a Header, the SGLN-195 is composed of five fields: the *Filter Value*, *Partition*, *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 – 999,999,999,999 (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 $d_1d_2\ldots d_{13}$
- The length L of the Company Prefix portion of the GLN

- An EAN.UCC-128 Application Identifier 254 consisting of characters $s_1s_2\ldots s_K$, where K ≤ 20.,. If the Application Identifier 254 consists of a single character 0 where K=1, this Encoding is considered to be a GLN only.
- A Filter Value F where $0 \le F < 8$

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-195.

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

3. If L < 12 construct the Location Reference by concatenating digits $d_{(L+1)}d_{(L+2)}\ldots d_{12}$ and considering the result to be a decimal integer, I. If L = 12 set b_{140} to 0 since there is no Location Reference digit.

4. . Check that each of the characters $s_1s_2\ldots s_K$ 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 SGLN-195. Otherwise construct the Extension Component by concatenating the 7-bit code, as given in Appendix G, for each of the characters $s_1s_2\ldots s_K$, yielding 7K bits total. If K < 20, concatenate additional zero bits to the right to make a total of 140 bits.

5. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header 00111001 (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 (140 bits). Note that $M+N$ = 41 bits for all P.

3.7.2.2 SGLN-195 Decoding Procedure

Given:

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

Yields:

- An EAN.UCC GLN
- An Extension Component
- A Filter Value

Procedure:

1. Bits $b_{186}b_{185}b_{184}$, considered as an unsigned integer, are the Filter Value.

2. Extract the Partition Value P by considering bits $b_{183}b_{182}b_{181}$ as an unsigned integer. If $P = 7$, stop: this bit string cannot be decoded as an SGLN-195.

3. Look up the Partition Value P in Table 13 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 $b_{180}b_{179}\ldots b_{(181\text{-M})}$ as an unsigned integer. If this integer is greater than or equal to 10^{L}, stop: the input bit string is not a legal SGLN-195 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\ldots p_L$, adding leading zeros as necessary to make up L digits in total.

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

6. Construct a 12-digit number $d_1d_2\ldots d_{12}$ where $d_1d_2\ldots d_L = p_1p_2\ldots p_L$ from Step 4, and if L < 12 $d_{(L+1)}d_{(L+2)}\ldots d_{12} = i_2\ i_3\ldots i_{(12\text{-L})}$ from Step 5.

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

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

9. Divide the remaining bits $b_{139}b_{138}\ldots b_0$ into 7-bit segments. The result should consist of K non-zero binary segments followed by 20-K binary zero segments. If this is not the case, stop: this bit string cannot be decoded as an SGLN-195. 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 SGLN-195. If K=1 and s_1=0, then this indicates a GLN only with no *Extension Component*. Otherwise, the K characters so obtained, considered as a character string $s_1s_2\ldots s_K$, is the value of the EAN.UCC AI 254.

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

3.8 Global Returnable Asset Identifier (GRAI)

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

3.8.1 GRAI-96

In addition to a Header, the GRAI-96 is composed of five fields: the *Filter Value*, *Partition*, *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 (*P*)	Company Prefix		Asset Type	
	Bits (*M*)	**Digits (*L*)**	**Bits (N)**	**Digits**
0	40	12	4	0
1	37	11	7	1
2	34	10	10	2
3	30	9	14	3
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 $0d_2d_3\ldots d_K$, where $15 \le K\ 3\le 0$.
- The length L of the Company Prefix portion of the GRAI
- A Filter Value F where $0 \le F < 8$

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 $d_2d_3\ldots d_{(L+1)}$ and considering the result to be a decimal integer, C.

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

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

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

3.8.1.2 GRAI-96 Decoding Procedure

Given:

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

Yields:

- An EAN.UCC GRAI
- A Filter Value

Procedure:

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

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

3. Look up the Partition Value *P* in Table 17 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 $b_{81}b_{80}\ldots b_{(82-M)}$ as an unsigned integer. If this integer is greater than or equal to 10^L, stop: the input bit string is not a legal GRAI-96 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\ldots p_L$, adding leading zeros as necessary to make up *L* digits in total.

5. If L < 12 extract the Asset Type by considering bits $b_{(81-M)}\ b_{(80-M)}\ldots b_{38}$ as an unsigned integer. If this integer is greater than or equal to $10^{(12-L)}$, stop: the input bit string is not a legal GRAI-96 encoding. Otherwise, convert this integer to a (12-L)-digit decimal number $i_1i_2\ldots i_{(12-L)}$, adding leading zeros as necessary to make (12-L) digits.

6. Construct a 13-digit number $0d_2d_3\ldots d_{13}$ where $d_2d_3\ldots d_{(L+1)} = p_1p_2\ldots p_L$ from Step 4, and $d_{(L+2)}d_{(L+3)}\ldots d_{13} = i_1\ i_2\ldots i_{(12-L)}$ from Step 5.

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

8. Extract the Serial Number by considering bits $b_{37}b_{36}\ldots b_0$ as an unsigned integer. Convert this integer to a decimal number $d_{15}d_{16}\ldots d_K$, 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: $0d_2d_3\ldots d_K$.

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,

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

3.8.2.1 GRAI-170 Encoding Procedure

The following procedure creates a GRAI-170 encoding.

Given:

- An EAN.UCC GRAI consisting of digits $0d_2d_3\ldots d_{14}$, and a variable length alphanumeric serial number $s_{15}s_{16}\ldots s_K$ 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 $d_2d_3\ldots d_{(L+1)}$ and considering the result to be a decimal integer, *C*.

3. If $L < 12$ construct the Asset Type by concatenating digits $d_{(L+2)}d_{(L+3)}\ldots d_{13}$ and considering the result to be a decimal integer, *I*. Otherwise set bits $b_{115,}b_{114}\,,b_{113}\,,b_{112}$ to 0000.

4. Check that each of the characters $s_{15}s_{16}\ldots s_K$ 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 GRAI-170. Otherwise construct the Serial Number by concatenating the 7-bit code, as given in Appendix G, for each of the characters $s_{15}s_{16}\ldots s_K$, yielding 7*(K-14) bits total. If K < 30, concatenate additional zero bits to the right to make a total of 112 bits.

5. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header 00110111 (8 bits), Filter Value *F* (3 bits), Partition Value *P* from Step 1 (3 bits), Company Prefix *C* from Step 2 (*M* bits), Asset Type *I* from Step 3 (*N* bits), Serial Number *S* from Step 4 (112 bits). Note that M+N = 44 bits for all *P*.

3.8.2.2 GRAI-170 Decoding Procedure

Given:

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

Yields:

- An EAN.UCC GRAI
- A Filter Value

Procedure:

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

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

3. Look up the Partition Value P in Table 17 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 $b_{156}b_{155}\ldots b_{(157\text{-}M)}$ as an unsigned integer. If this integer is greater than or equal to 10^L, stop: the input bit string is not a legal GRAI-170 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\ldots p_L$, adding leading zeros as necessary to make up L digits in total.

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

6. Construct a 13-digit number $0d_2d_3\ldots d_{13}$ where $d_2d_3\ldots d_{(L+1)} = p_1p_2\ldots p_L$ from Step 4, and if $L < 12$ $d_{(L+2)}d_{(L+3)}\ldots d_{13} = i_1\ i_2\ldots i_{(12\text{-}L)}$ from Step 5.

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

8. Divide the remaining bits $b_{111}b_{110}\ldots b_0$ into 7-bit segments. This string should consist of K non-zero segments followed by 16-K zero segments. If this is not the case, stop: this bit string cannot be decoded as an GRAI-170. 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 GRAI-170. Otherwise, the first K characters considered as a character string is the serial number $s_{15}s_{16}\ldots s_K$.

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

3.9 Global Individual Asset Identifier (GIAI)

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

3.9.1 GIAI-96

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

	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 (*P*)	Company Prefix		Individual Asset Reference	
	Bits (*M*)	Digits (*L*)	Bits (*N*)	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 $d_1d_2\ldots d_K$, 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 $d_1d_2\ldots d_L$ and considering the result to be a decimal integer, C.

3. Construct the Individual Asset Reference by concatenating digits $d_{(L+1)}d_{(L+2)}\ldots d_K$. 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

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

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

3.9.1.2 GIAI-96 Decoding Procedure

Given:

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

Yields:

- An EAN.UCC GIAI
- A Filter Value

Procedure:

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

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

3. Look up the Partition Value P in Table 21 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 $b_{81}b_{80}\ldots b_{(82\text{-}M)}$ 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-96 encoding. Otherwise, convert this integer into a decimal number $p_1p_2\ldots p_L$, adding leading zeros as necessary to make up L digits in total.

5. Extract the Individual Asset Reference by considering bits $b_{(81\text{-}M)}\ b_{(80\text{-}M)}\ldots b_0$ as an unsigned integer. If this integer is greater than or equal to $10^{(30\text{-}L)}$, stop: the input bit string is not a legal GIAI-96 encoding. Otherwise, convert this integer to a decimal number $s_1s_2\ldots s_J$, with no leading zeros (exception: if the integer is equal to zero, convert it to a single zero digit).

6. Construct a K-digit number $d_1d_2\ldots d_K$ where $d_1d_2\ldots d_L = p_1p_2\ldots p_L$ from Step 4, and $d_{(L+1)}d_{(L+2)}\ldots d_K = s_1s_2\ldots s_J$ from Step 5. This K-digit number, where $K \leq 30$, is the EAN.UCC GIAI.

3.9.2 GIAI-202

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

	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

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

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

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

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

Table 23. GIAI-202 Partitions.

- *Company Prefix* contains a literal embedding of the EAN.UCC Company Prefix.
- *Individual Asset Reference* contains a mandatory alphanumeric asset reference number. The GIAI-202 encoding is capable of representing alphanumeric serial numbers of up to 24 characters, permitting the full range of serial numbers available in the EAN.UCC-128 barcode symbology using Application Identifier (AI) 8004 [EAN.UCCGS].
- *Company Prefix* and *Individual Asset Reference* should never total more than 30 characters.

3.9.2.1 GIAI-202 Encoding Procedure

The following procedure creates a GIAI-202 encoding.

Given:

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

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

3\. Check that each of the characters $s_{(L+1)}s_{(L+2)}\ldots s_K$ 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 $s_{(L+1)}s_{(L+2)}\ldots s_K$ 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.

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),

3.9.2.2 GIAI-202 Decoding Procedure

Given:

- A GIAI-202 as a 202-bit bit string 00111000$b_{193}b_{192}\ldots b_0$ (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 $b_{187}b_{186}\ldots b_{(188\text{-}M)}$ 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 $p_1p_2\ldots p_L$, adding leading zeros as necessary to make up L digits in total.

5. Extract the Individual Asset Reference by dividing the remaining bits $b_{(187\text{-}M)}\, b_{(186\text{-}M)}\ldots b_0$ into 7 bit segments beginning with the segment $b_{(187\text{-}M)}\, b_{(186\text{-}M)}\ldots b_{(181\text{-}M)}$, 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 $s_{(1)}s_{(2)}\ldots s_J$.

6. Construct a K-character string $s_1s_2\ldots s_K$ where $s_1s_2\ldots s_L = p_1p_2\ldots p_L$ from Step 4, and where $s_{(L+1)}s_{(L+2)}\ldots s_K = s_{(1)}s_{(2)}\ldots s_J$ 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/supplierguide.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`.

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

4.1 URI Forms for Pure Identities

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

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

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

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

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

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

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

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

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

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

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

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

The `ItemReference`, `SerialReference`, `LocationReference`, and `AssetType` fields correspond to the similar fields of the GTIN, SSCC, GLN, and GRAI, respectively. Like the *`CompanyPrefix`* field, the number of digits in these fields is significant, and leading zeros are included as necessary. The number of digits in these fields, when added to the number of digits in the *`CompanyPrefix`* field, always total the same number of digits according to the identity type: 13 digits total for SGTIN, 17 digits total for SSCC, 12 digits total for SGLN, and 12 characters total for the GRAI. (The *`ItemReference`* field of the SGTIN includes the GTIN Indicator (PI) digit, appended to the beginning of the item reference. The *SerialReference* field includes the SSCC Extension Digit (ED), followed by the serial reference. In no case are check digits included in URI representations.)

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

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

`urn:epc:id:sgtin:0652642.800031.400`

`urn:epc:id:sscc:0652642.0123456789`

`urn:epc:id:sgln:0652642.12345.40` (Use this form when Extension Component is used)

`urn:epc:id:sgln:0652642.12345.0` (Use this form when Extension Component is not used)

`urn:epc:id:grai:0652642.12345.1234`

`urn:epc:id:giai:0652642.123456`

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

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

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

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

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

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

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


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

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

4.2 URI Forms for Related Data Types

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

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

```
urn:epc:type:typeSpecificPart
```

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

4.2.1 URIs for EPC Tags

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

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

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

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

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

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

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

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

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

The tag URI for the DoD identifier is as follows:

`urn:epc:tag:`*`tagType`*`:`*`filter`*`.`*`CAGECodeOrDODAAC`*`.`*`serialNumber`*

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

4.2.2 URIs for Raw Bit Strings Arising From Invalid Tags

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

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

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

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

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

`0000000000000000000100100011010011011110101011011011111011101111`

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

`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 ≤ *itemReference* ≤ 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
```

```
NonZeroDigit ::= “1” | “2” | “3” | “4”
               | “5” | “6” | “7” | “8” | “9”

UpperAlpha ::= “A” | “B” | “C” | “D” | “E” | “F” | “G”
             | “H” | “I” | “J” | “K” | “L” | “M” | “N”
             | “O” | “P” | “Q” | “R” | “S” | “T” | “U”
             | “V” | “W” | “X” | “Y” | “Z”

LowerAlpha ::= “a” | “b” | “c” | “d” | “e” | “f” | “g”
             | “h” | “i” | “j” | “k” | “l” | “m” | “n”
             | “o” | “p” | “q” | “r” | “s” | “t” | “u”
             | “v” | “w” | “x” | “y” | “z”

OtherChar ::= “!” | “’” | “(“ | “)“ | “*” | “+” | “,” | “-“
            | “.” | “:” | “;” | “=” | “_”

UpperHexChar ::= Digit | “A” | “B” | “C” | “D” | “E” | “F”

HexComponent ::= UpperHexChar+

Escape ::= “%” HexChar HexChar

HexChar ::= UpperHexChar | “a” | “b” | “c” | “d” | “e” | “f”

GS3A3Char ::= Digit | UpperAlpha | LowerAlpha | OtherChar
            | Escape

GS3A3Component ::= GS3A3Char+
```

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

4.3.2 EPCGID-URI

```
EPCGID-URI ::= “urn:epc:id:gid:” 2*(NumericComponent “.”)
NumericComponent
```

4.3.3 SGTIN-URI

```
SGTIN-URI ::= “urn:epc:id:sgtin:” SGTINURIBody

SGTINURIBody ::= 2*(PaddedNumericComponent “.”) GS3A3Component
```

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

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

4.3.4 SSCC-URI

```
SSCC-URI ::= “urn:epc:id:sscc:” SSCCURIBody
SSCCURIBody ::= PaddedNumericComponent “.”
PaddedNumericComponent
```

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

4.3.5 SGLN-URI

```
SGLN-URI ::= “urn:epc:id:sgln:” SGLNURIBody
SGLNURIBody ::= 2*(PaddedNumericComponent “.”) GS3A3Component
```

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

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

4.3.6 GRAI-URI

```
GRAI-URI ::= “urn:epc:id:grai:” GRAIURIBody
GRAIURIBody ::= 2*(PaddedNumericComponent “.”) GS3A3Component
```

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

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

4.3.7 GIAI-URI

```
GIAI-URI ::= “urn:epc:id:giai:” GIAIURIBody
```

```
GIAIURIBody ::= PaddedNumericComponent "." GS3A3Component
```

The total number of characters in the `PaddedNumericComponent` and `GS3A3Component` fields must not exceed 30 (not including the dot character that seprates the two fields).

The Individual Asset Reference field of the GIAI-URI is expressed as a `GS3A3Component`, which permits the representation of all characters permitted in the Individual Asset Reference field of the GIAI according to the EAN.UCC General Specifications. GIAI-URIs that is derived from 96-bit tag encodings, however, will have Individual Asset References that consist only of digit characters and which have no leading zeros. These limitations are described in the encoding procedures, and in Appendix F.

4.3.8 EPC Tag URI

```
TagURI ::= "urn:epc:tag:" TagURIBody

TagURIBody ::= GIDTagURIBody | SGTINSGLNGRAI96TagURIBody |
SGTINSGLNGRAIAlphaTagURIBody | SSCCTagURIBody |
GIAI96TagURIBody | GIAIAlphaTagURIBody

GIDTagURIBody ::= GIDTagEncName ":" 2*(NumericComponent ".")
NumericComponent

GIDTagEncName ::= "gid-96"

SGTINSGLNGRAITag96URIBody ::= SGTINSGLNGRAI96TagEncName ":"
NumericComponent "." 2*(PaddedNumericComponent ".")
NumericComponent

SGTINSGLNGRAITagAlphaURIBody ::= SGTINSGLNGRAIAlphaTagEncName
":" NumericComponent "." 2*(PaddedNumericComponent ".")
GS3A3Component

SGTINSGLNGRAI96TagEncName ::= "sgtin-96" | "sgln-96"| "grai-
96"

SGTINSGLNGRAIAlphaTagEncName ::= "sgtin-198" | "sgln-195"|
"grai-170"

SSCCTagURIBody ::= SSCCTagEncName ":" NumericComponent 2*("."
PaddedNumericComponent)

SSCCTagEncName ::= "sscc-96"

GIAI96TagURIBody ::= GIAI96TagEncName ":" NumericComponent "."
PaddedNumericComponent "." NumericComponent

GIAIAlphaTagURIBody ::= GIAIAlphaTagEncName ":"
NumericComponent "." PaddedNumericComponent "." GS3A3Component

GIAI96TagEncName ::= "giai-96"

GIAIAlphaTagEncName ::= "giai-202"
```

4.3.9 Raw Tag URI

```
RawURI ::= “urn:epc:raw:” RawURIBody

RawURIBody ::=  DecimalRawURIBody | HexRawURIBody |
AFIRawURIBody)

DecimalRawURIBody ::= NonZeroComponent “.” NumericComponent

HexRawURIBody ::= NonZeroComponent “.x” HexComponent

AFIRawURIBody ::= NonZeroComponent “.x” HexComponent “.x”
HexComponent
```

4.3.10 EPC Pattern URI

```
PatURI ::= “urn:epc:pat:” PatBody

PatBody ::= GIDPatURIBody | SGTINSGLNGRAI96PatURIBody |
SGTINSGLNGRAIAlphaPatURIBody | SSCCPatURIBody |
GIAI96PatURIBody | GIAIAlphaPatURIBody

GIDPatURIBody ::= GIDTagEncName “:” 2*(PatComponent “.”)
PatComponent

SGTINSGLNGRAI96PatURIBody ::= SGTINSGLNGRAI96TagEncName “:”
PatComponent “.” GS1PatBody “.” PatComponent

SGTINSGLNGRAIAlphaPatURIBody ::= SGTINSGLNGRAIAlphaTagEncName
“:” PatComponent “.” GS1PatBody “.” GS3A3PatComponent

SSCCPatURIBody ::= SSCCTagEncName “:” PatComponent “.”
GS1PatBody

GIAI96PatURIBody ::= GIAI96TagEncName “:” PatComponent “.”
GS1PatBody

GIAIAlphaPatURIBody ::= GIAIAlphaTagEncName “:” PatComponent
“.” GS1GS3A3PatBody

GS1PatBody ::= “*.*” | ( PaddedNumericComponent “.”
PatComponent )

GS1GS3A3PatBody ::= “*.*” | ( PaddedNumericComponent “.”
GS3A3PatComponent )

PatComponent ::= NumericComponent
               | StarComponent
               | RangeComponent

GS3A3PatComponent ::= GS3A3Component | StarComponent

StarComponent ::= “*”

RangeComponent ::= “[“ NumericComponent “-“
                       NumericComponent “]”
```

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

4.3.11 EPC Identity Pattern URI

```
IDPatURI ::= “urn:epc:idpat:” IDPatBody

IDPatBody ::= GIDIDPatURIBody | SGTINIDPatURIBody |
SGLNIDPatURIBody | GIAIIDPatURIBody | SSCCIDPatURIBody |
GRAIIDPatURIBody

GIDIDPatURIBody ::= “gid:” GIDIDPatURIMain

GIDIDPatURIMain ::=
    2*(NumericComponent “.”) NumericComponent
  | 2*(NumericComponent “.”) “*”
  | NumericComponent “.*.*”
  | “*.*.*”

SGTINIDPatURIBody ::= “sgtin:” SGTINSGLNGRAIIDPatURIMain

GRAIIDPatURIBody ::= “grai:” SGTINSGLNGRAIIDPatURIMain

SGLNIDPatURIBody ::= “sgln:” SGTINSGLNGRAIIDPatURIMain

SGTINSGLNGRAIIDPatURIMain ::=
    2*(PaddedNumericComponent “.”) GS3A3Component
  | 2*(PaddedNumericComponent “.”) “*”
  | PaddedNumericComponent “.*.*”
  | “*.*.*”

SCCIDPatURIBody ::= “sscc:” SSCCIDPatURIMain

SSCCIDPatURIMain ::=
    PaddedNumericComponent “.” PaddedNumericComponent
  | PaddedNumericComponent “.*”
  | “*.*”

GIAIIDPatURIBody ::= “giai:” GIAIIDPatURIMain

GIAIIDPatURIMain ::=
    PaddedNumericComponent “.” GS3A3Component
  | PaddedNumericComponent “.*”
  | “*.*”
```

4.3.12 DoD Construct URI

```
DOD-URI ::= “urn:epc:id:usdod:” CAGECodeOrDODAAC “.”
DoDSerialNumber

DODTagURI ::= “urn:epc:tag:” DoDTagType “:” DoDFilter “.”
CAGECodeOrDODAAC “.” DoDSerialNumber

DODPatURI ::= “urn:epc:pat:” DoDTagType “:” DoDFilterPat “.”
CAGECodeOrDODAACPat “.” DoDSerialNumberPat
```

```
DODIDPatURI ::= “urn:epc:idpat:usdod:” DODIDPatMain

DODIDPatMain ::=
    CAGECodeOrDODAAC “.” DoDSerialNumber
  | CAGECodeOrDODAAC “.*”
  | “*.*”

DoDTagType ::= “usdod-96”

DoDFilter ::= NumericComponent

CAGECodeOrDODAAC ::= CAGECode | DODAAC

CAGECode ::= CAGECodeOrDODAACChar*5

DODAAC ::= CAGECodeOrDODAACChar*6

DoDSerialNumber ::= NumericComponent

DoDFilterPat ::= PatComponent

CAGECodeOrDODAACPat ::= CAGECodeOrDODAAC | StarComponent

DoDSerialNumberPat ::= PatComponent

CAGECodeOrDODAACChar ::= Digit | “A” | “B” | “C” | “D” | “E” |
“F” | “G” | “H” | “J” | “K” | “L” | “M” | “N” | “P” | “Q” |
“R” | “S” | “T” | “U” | “V” | “W” | “X” | “Y” | “Z”
```

4.3.13 Summary (non-normative)

The syntax rules above can be summarized informally as follows:

urn:epc:id:gid:*MMM.CCC.SSS*

urn:epc:id:sgtin:*PPP.III.AAA*

urn:epc:id:sscc:*PPP.III*

urn:epc:id:sgln:*PPP.III.AAA*

urn:epc:id:grai:*PPP.III.AAA*

urn:epc:id:giai:*PPP.AAA*

urn:epc:id:usdod:*TTT.SSS*

urn:epc:tag:gid-96:*MMM.CCC.SSS*

urn:epc:tag:sgtin-96:*FFF.PPP.III.SSS*

urn:epc:tag:sgtin-198:*FFF.PPP.III.AAA*

urn:epc:tag:sscc-96:*FFF.PPP.III*

urn:epc:tag:sgln-96:*FFF.PPP.III.SSS*

urn:epc:tag:sgln-195:*FFF.PPP.III.AAA*

```
urn:epc:tag:grai-96:FFF.PPP.III.SSS
urn:epc:tag:grai-170:FFF.PPP.III.AAA
urn:epc:tag:giai-96:FFF.PPP.SSS
urn:epc:tag:giai-202:FFF.PPP.AAA
urn:epc:tag:usdod-96:FFF.TTT.SSS

urn:epc:raw:LLL.BBB
urn:epc:raw:LLL.HHH
urn:epc:raw:LLL.HHH.HHH

urn:epc:idpat:gid:MMM.CCC.SSS
urn:epc:idpat:gid:MMM.CCC.*
urn:epc:idpat:gid:MMM.*.*
urn:epc:idpat:gid:*.*.*
urn:epc:idpat:sgtin:PPP.III.AAA
urn:epc:idpat:sgtin:PPP.III.*
urn:epc:idpat:sgtin:PPP.*.*
urn:epc:idpat:sgtin:*.*.*
urn:epc:idpat:sscc:PPP.III
urn:epc:idpat:sscc:PPP.*
urn:epc:idpat:sscc:*.*
urn:epc:idpat:sgln:PPP.III.AAA
urn:epc:idpat:sgln:PPP.III.*
urn:epc:idpat:sgln:PPP.*.*
urn:epc:idpat:sgln:*.*.*
urn:epc:idpat:grai:PPP.III.AAA
urn:epc:idpat:grai:PPP.III.*
urn:epc:idpat:grai:PPP.*.*
urn:epc:idpat:grai:*.*.*
urn:epc:idpat:giai:PPP.AAA
urn:epc:idpat:giai:PPP.*
urn:epc:idpat:giai:*.*
urn:epc:idpat:usdod:TTT.SSS
```

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

urn:epc:pat:gid-96:MMMpat.CCCpat.SSSpat
urn:epc:pat:sgtin-96:FFFpat.PPP.IIIpat.SSSpat
urn:epc:pat:sgtin-96:FFFpat.*.*.SSSpat
urn:epc:pat:sgtin-198:FFFpat.PPP.IIIpat.AAApat
urn:epc:pat:sgtin-198:FFFpat.*.*.AAApat
urn:epc:pat:sscc-96:FFFpat.PPP.IIIpat
urn:epc:pat:sscc-96:FFFpat.*.*
urn:epc:pat:sgln-96:FFFpat.PPP.IIIpat.SSSpat
urn:epc:pat:sgln-96:FFFpat.*.*.SSSpat
urn:epc:pat:sgln-195:FFFpat.PPP.IIIpat.AAApat
urn:epc:pat:sgln-195:FFFpat.*.*.AAApat
urn:epc:pat:grai-96:FFFpat.PPP.IIIpat.SSSpat
urn:epc:pat:grai-96:FFFpat.*.*.SSSpat
urn:epc:pat:grai-170:FFFpat.PPP.IIIpat.AAApat
urn:epc:pat:grai-170:FFFpat.*.*.AAApat
urn:epc:pat:giai-96:FFFpat.PPP.SSSpat
urn:epc:pat:giai-96:FFFpat.*.*
urn:epc:pat:giai-202:FFFpat.PPP.AAApat
urn:epc:pat:giai-202:FFFpat.*.*
urn:epc:pat:usdod-96:FFFpat.TTT.SSSpat
urn:epc:pat:usdod-96:FFFpat.*.SSSpat
```

where

MMM denotes a General Manager Number

CCC denotes an Object Class number

SSS denotes a numeric Serial Number or GIAI Individual Asset Reference

AAA denotes an alphanumeric Serial Number or GIAI Individual Asset reference

PPP denotes an EAN.UCC Company Prefix

TTT denotes a US DoD assigned CAGE code or DODAAC

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

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

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

`LLL` denotes the number of bits of an uninterpreted bit sequence

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

`HHH` denotes the literal value of an uninterpreted bit sequence converted to hexadecimal and preceded by the character ‘x’.

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

Exceptions:

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

5 Translation between EPC-URI and Other EPC Representations

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

5.1 Bit string into EPC-URI (pure identity)

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

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

Step 11; if the encoding scheme is GIAI-96 or GIAI-202, proceed to Step 14; if the encoding scheme is GID-96, proceed to Step 17.

2. Follow the decoding procedure given in Section 3.5.1.2 (for SGTIN-96) or in Section 3.5.2.2 (for SGTIN-198) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal Item Reference and Indicator $i_1i_2…i_{(13-L)}$, and the Serial Number S. If the decoding procedure fails, stop: the bit-level encoding cannot be translated into an EPC-URI.

3. Create an EPC-URI by concatenating the following: the string `urn:epc:id:sgtin:`, the Company Prefix $p_1p_2...p_L$ where each digit (including any leading zeros) becomes the corresponding ASCII digit character, a dot (`.`) character, the Item Reference and Indicator $i_1i_2…i_{(13-L)}$ (handled similarly), a dot (`.`) character, and the Serial Number S as a decimal integer (SGTIN-96) or alphanumeric character (SGTIN-198). For SGTIN-96 the portion corresponding to the Serial Number must have no leading zeros, except where the Serial Number is itself zero in which case the corresponding URI portion must consist of a single zero character.

4. Go to Step 19.

5. Follow the decoding procedure given in Section 3.6.1.2 (for SSCC-96) to obtain the decimal Company Prefix $p_1p_2...p_L$, and the decimal Serial Reference $s_1s_2…s_{(17-L)}$. If the decoding procedure fails, stop: the bit-level encoding cannot be translated into an EPC-URI.

6. Create an EPC-URI by concatenating the following: the string `urn:epc:id:sscc:`, the Company Prefix $p_1p_2...p_L$ where each digit (including any leading zeros) becomes the corresponding ASCII digit character, a dot (`.`) character, and the Serial Reference $s_1s_2…s_{(17-L)}$ (handled similarly).

7. Go to Step 19.

8. Follow the decoding procedure given in Section 3.7.1.2 (for SGLN-96) or in Section 3.7.2.2 (for SGLN-195) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal Location Reference $i_1i_2…i_{(12-L)}$, and the Extension Component S. If the decoding procedure fails, stop: the bit-level encoding cannot be translated into an EPC-URI.

9. Create an EPC-URI by concatenating the following: the string `urn:epc:id:sgln:`, the Company Prefix $p_1p_2...p_L$ where each digit (including any leading zeros) becomes the corresponding ASCII digit character, a dot (`.`) character, for $L < 12$ the Location Reference, $i_1i_2…i_{(12-L)}$ (handled similarly), a dot (`.`) character, and the Extension Component S as a decimal integer (SGLN-96) or alphanumeric character (SGLN-195). For SGLN-96 the portion corresponding to the Extension Component must have no leading zeros, except where the Extension Component is itself zero in which case the corresponding URI portion must consist of a single zero character. If a Location Reference does not exist (where $L = 12$), leave no blank space between the two dot (.) characters.

10. Go to Step 19.

11. Follow the decoding procedure given in Section 3.8.1.2 (for GRAI-96) or in Section 3.8.2.2 (for GRAI-170) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal

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

12. Create an EPC-URI by concatenating the following: the string `urn:epc:id:grai:`, the Company Prefix $p_1p_2\ldots p_L$ where each digit (including any leading zeros) becomes the corresponding ASCII digit character, a dot (`.`) character, for $L < 12$ the Asset Type $i_1i_2\ldots i_{(12\text{-}L)}$ (handled similarly), a dot (`.`) character, and the Serial Number *S* as a decimal integer (GRAI-96) or alphanumeric character (GRAI-170). For GRAI-96 the portion corresponding to the Serial Number must have no leading zeros, except where the Serial Number is itself zero in which case the corresponding URI portion must consist of a single zero character. If an Asset Type does not exist (where $L = 12$), leave no blank space between the two dot (.) characters.
13. Go to Step 19.
14. Follow the decoding procedure given in Section 3.9.1.2 (for GIAI-96) or in 3.9.2.2 (for GIAI-202) to obtain the decimal Company Prefix $p_1p_2\ldots p_L$, and the Individual Asset Reference *S*. If the decoding procedure fails, stop: the bit-level encoding cannot be translated into an EPC-URI.
15. Create an EPC-URI by concatenating the following: the string `urn:epc:id:giai:`, the Company Prefix $p_1p_2\ldots p_L$ where each digit (including any leading zeros) becomes the corresponding ASCII digit character, a dot (`.`) character, and the Individual Asset Reference *S* as a decimal integer (GIAI-96) or alphanumeric character (GIAI-202). For GIAI-96 the portion corresponding to the Individual Asset Reference must have no leading zeros, except where the Individual Asset Reference is itself zero in which case the corresponding URI portion must consist of a single zero character.
16. Go to Step 19.
17. Follow the decoding procedure given in Section 3.4.1.2 to obtain the General Manager Number *M*, the Object Class *C*, and the Serial Number *S*.
18. Create an EPC-URI by concatenating the following: the string `urn:epc:id:gid:`, the General Manager Number as a decimal integer, a dot (`.`) character, the Object Class as a decimal integer, a dot (`.`) character, and the Serial Number *S* as a decimal integer. Each decimal number must have no leading zeros, except where the integer is itself zero in which case the corresponding URI portion must consist of a single zero character.
19. The translation is now complete.

5.2 Bit String into Tag or Raw URI

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

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

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

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

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

4. Go to Step 21.

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

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

7. Go to Step 21.

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

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

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

10. Go to Step 21.

11. Follow the decoding procedure given in Section 3.8.1.2 (for GRAI-96) or 3.8.2.2 (for GRAI-170) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal Asset Type $i_1i_2\ldots i_{(12\text{-}L)}$, the Filter Value F, and the Serial Number $d_{15}d_2\ldots d_K$. If the decoding procedure fails, proceed to Step 20, otherwise proceed to the next step.

12. Create an EPC Tag URI by concatenating the following: the string `urn:epc:tag:`, the encoding scheme (`grai-96` or `grai-170`), a colon (`:`) character, the Filter Value F as a decimal integer, a dot (`.`) character, the Company Prefix $p_1p_2...p_L$ where each digit (including any leading zeros) becomes the corresponding ASCII digit character, a dot (`.`) character, for L < 12 the Asset Type $i_1i_2\ldots i_{(12\text{-}L)}$ (handled similarly), a dot (`.`) character, and the Serial Number $d_{15}d_2\ldots d_K$ as a decimal integer (GRAI-96) or alphanumeric character (GRAI-170). For GRAI-96 the portions corresponding to the Filter Value and Serial Number must have no leading zeros, except where the corresponding integer is itself zero in which case a single zero character is used. If an Asset Type does not exist where L = 12 leave no blank space between the two dot (.) characters.

13. Got to Step 21.

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

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

16. Go to Step 21.

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

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

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

19. Go to Step 21.
20. This tag is not a recognized EPC Tag Encoding, therefore create an EPC Raw URI by concatenating the following: the string `urn:epc:raw:`, the length of the bit string (N) expressed as a decimal integer with no leading zeros, a dot (`.`) character, a lowercase `x` character, and the value of the bit string considered as a single hexadecimal integer. The value must have a number of characters equal to the length (N) divided by four and rounded up to the nearest whole number, and must only use uppercase letters for the hexadecimal digits A, B, C, D, E, and F.
21. The translation is now complete.

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

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

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

5.4 Gen 2 Tag EPC Memory into Tag or Raw URI

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

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

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

5.5 URI into Bit String

The following procedure translates a URI into a bit string:

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

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

8. Go to Step 53.
9. Let the URI be written as `urn:epc:tag:`*`encName`*`:`$f_1f_2…f_F$`.`$p_1p_2…p_L$`.`$i_1i_2…i_{(13-L)}$`.`$s_1s_2…s_S$.
10. Interpret $f_1f_2…f_F$ as a decimal integer *F*.
11. Interpret $s_1s_2…s_S$ as an alphanumeric string *S*.
12. Carry out the encoding procedure defined in Section 3.5.2.1 (SGTIN-198) using $i_1p_1p_2…p_Li_2…i_{(13-L)}0$ as the EAN.UCC GTIN-14 (the trailing zero is a dummy check digit, which is ignored by the encoding procedure), L as the length of the EAN.UCC company prefix, *F* from Step 10 as the Filter Value, and *S* from Step 11 as the Serial Number. If the encoding procedure fails because an input is out of range, or because the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.
13. Go to Step 53.
14. Let the URI be written as `urn:epc:tag:`*`encName`*`:`$f_1f_2…f_F$`.`$p_1p_2…p_L$`.`$i_1i_2…i_{(17-L)}$.
15. Interpret $f_1f_2…f_F$ as a decimal integer *F*.
16. Carry out the encoding procedure defined in Section 3.6.1.1 (SSCC-96), using $i_1p_1p_2…p_Li_2i_3…i_{(17-L)}0$ as the EAN.UCC SSCC (the trailing zero is a dummy check digit, which is ignored by the encoding procedure), L as the length of the EAN.UCC company prefix, and *F* from Step 15 as the Filter Value. If the encoding procedure fails because an input is out of range, or because the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.
17. Go to Step 53.
18. Let the URI be written as `urn:epc:tag:`*`encName`*`:`$f_1f_2…f_F$`.`$p_1p_2…p_L$`.`$i_1i_2…i_{(12-L)}$`.`$s_1s_2…s_S$.
19. Interpret $f_1f_2…f_F$ as a decimal integer *F*.
20. Interpret $s_1s_2…s_S$ as a decimal integer *S*.
21. Carry out the encoding procedure defined in Section 3.7.1.1 (SGLN-96), using $p_1p_2…p_Li_1i_2…i_{(12-L)}0$ as the EAN.UCC GLN (the trailing zero is a dummy check digit, which is ignored by the encoding procedure), L as the length of the EAN.UCC company prefix, *F* from Step 19 as the Filter Value, and *S* from Step 20 as the Extension Component. If the encoding procedure fails because an input is out of range, or because the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.
22. Go to Step 53.

23. Let the URI be written as urn:epc:tag:*encName*:$f_1f_2 \ldots f_F.p_1p_2 \ldots p_L.i_1i_2 \ldots i_{(12\text{-}L)}.s_1s_2 \ldots s_S$.

24. Interpret $f_1f_2 \ldots f_F$ as a decimal integer *F*.

25. Interpret $s_1s_2 \ldots s_S$ as an alphanumeric string *S*.

26. Carry out the encoding procedure defined in Section 3.7.2.1 (SGLN-195), using $p_1p_2 \ldots p_Li_1i_2 \ldots i_{(12\text{-}L)}0$ as the EAN.UCC GLN (the trailing zero is a dummy check digit, which is ignored by the encoding procedure), L as the length of the EAN.UCC company prefix, *F* from Step 24 as the Filter Value, and *S* from Step 25 as the Extension Component. If the encoding procedure fails because an input is out of range, or because the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.

27. Go to Step 53.

28. Let the URI be written as urn:epc:tag:*encName*:$f_1f_2 \ldots f_F.p_1p_2 \ldots p_L.i_1i_2 \ldots i_{(12\text{-}L)}.s_1s_2 \ldots s_S$.

29. Interpret $f_1f_2 \ldots f_F$ as a decimal integer *F*

30. Interpret $s_1s_2 \ldots s_S$ as a decimal integer *S*.

31. Carry out the encoding procedure defined in Section 3.8.1.1 (GRAI-96),using $0p_1p_2 \ldots p_Li_1i_2 \ldots i_{(12\text{-}L)}0s_1s_2 \ldots s_S$ as the EAN.UCC GRAI (the second zero is a dummy check digit, which is ignored by the encoding procedure), L as the length of the EAN.UCC company prefix, and *F* from Step 29 as the Filter Value, and *S* from Step 30 as the Serial Number. If the encoding procedure fails because an input is out of range, or because the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.

32. Go to Step 53.

33. Let the URI be written as urn:epc:tag:*encName*:$f_1f_2 \ldots f_F.p_1p_2 \ldots p_L.i_1i_2 \ldots i_{(12\text{-}L)}.s_1s_2 \ldots s_S$.

34. Interpret $f_1f_2 \ldots f_F$ as a decimal integer *F*.

35. Interpret $s_1s_2 \ldots s_S$ as an alphanumeric string *S*.

36. Carry out the encoding procedure defined in Section 3.8.2.1 (GRAI-170) using $0p_1p_2 \ldots p_Li_1i_2 \ldots i_{(12\text{-}L)}0s_1s_2 \ldots s_S$ as the EAN.UCC GRAI (the second zero is a dummy check digit, which is ignored by the encoding procedure), L as the length of the EAN.UCC company prefix, and *F* from Step 34 as the Filter Value, and *S* from Step 35 as the Serial Number. If the encoding procedure fails because an input is out of range, or because the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.

37. Go to Step 53.

38. Let the URI be written as urn:epc:tag:*encName*:$f_1f_2 \ldots f_F.p_1p_2 \ldots p_L.s_1s_2 \ldots s_S$.

39. Interpret $f_1f_2 \ldots f_F$ as a decimal integer *F*

40. Interpret $s_1s_2 \ldots s_S$ as a decimal integer *S*.

41. Carry out the encoding procedure defined in Section 3.9.1.1 (GIAI-96), using $p_1p_2 \ldots p_L s_1s_2 \ldots s_S$ as the EAN.UCC GIAI, L as the length of the EAN.UCC company prefix, and *F* from Step 39 as the Filter Value, and *S* from Step 40 as the Serial Number. If the encoding procedure fails because an input is out of range, or because the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.

42. Go to Step 53.

43. Let the URI be written as `urn:epc:tag:`*`encName`*`:`$f_1f_2 \ldots f_F$`.`$p_1p_2 \ldots p_L$`.`$s_1s_2 \ldots s_S$.

44. Interpret $f_1f_2 \ldots f_F$ as a decimal integer *F*.

45. Interpret $s_1s_2 \ldots s_S$ as an alphanumeric string *S*.

46. Carry out the encoding procedure defined in Section 3.9.2.1 (GIAI-202) using $p_1p_2 \ldots p_L s_1s_2 \ldots s_S$ as the EAN.UCC GIAI, L as the length of the EAN.UCC company prefix, and *F* from Step 44 as the Filter Value, and *S* from Step 45 as the Serial Number. If the encoding procedure fails because an input is out of range, or because the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.

47. Go to Step 53.

48. Let the URI be written as `urn:epc:tag:`*`encName`*`:`$m_1m_2 \ldots m_L$`.`$c_1c_2 \ldots c_K$`.`$s_1s_2 \ldots s_S$.

49. Interpret $m_1m_2 \ldots m_L$ as a decimal integer *M*.

50. Interpret $c_1c_2 \ldots c_K$ as a decimal integer *C*.

51. Interpret $s_1s_2 \ldots s_S$ as a decimal integer *S*.

52. Carry out the encoding procedure defined in Section 3.4.1.1 using *M* from Step 49 as the General Manager Number, *C* from Step 50 as the Object Class, and *S* from Step 51 as the Serial Number. If the encoding procedure fails because an input is out of range, or because the procedure indicates a failure, stop: this URI cannot be encoded into a bit string.

53. The translation is complete.

5.6 URI into Gen 2 Tag EPC Memory

The following procedure converts a URI into a sequence of bits suitable for writing into the EPC memory of a Gen 2 Tag, starting with bit 10x (i.e., not including the CRC).

1. If the URI is a Raw Tag URI of the form `urn:epc:raw:`*`N.A.V`*, calculate the value L, where L = N/16 rounded up to the nearest whole number. If L ≥ 32, stop: this URI cannot be encoded into the EPC memory of a Gen 2 Tag. If A ≥ 256 or if the value V is too large to be expressed as an N-bit binary integer, stop: this URI cannot be encoded into the EPC memory of a Gen 2 Tag. Otherwise, construct the contents of EPC memory by concatenating the following bit strings: the value L (five bits), two zero bits (00), a single one bit (1), the value A (eight bits), and the value V (16L bits).

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

6 Semantics of EPC Pattern URIs

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

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

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

First, any EPC Tag URI component `C`*`i`* is said to *match* the corresponding EPC Pattern URI component `P`*`i`* if:

- `P`*`i`* is a `NumericComponent`, and `C`*`i`* is equal to `P`*`i`*; or
- `P`*`i`* is a `PaddedNumericComponent`, and `C`*`i`* is equal to `P`*`i`* both in numeric value as well as in length; or
- `P`*`i`* is a `GS3A3Component`, and `C`*`i`* is equal to `P`*`i`*, character for character; or
- `P`*`i`* is a `CAGECodeOrDODAAC`, and `C`*`i`* is equal to `P`*`i`*; or
- `P`*`i`* is a `RangeComponent [`*`lo`*`-`*`hi`*`]`, and *`lo`* ≤ `C`*`i`* ≤ *`hi`*; or
- `P`*`i`* is a `StarComponent` (and `C`*`i`* is anything at all)

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

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

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

Then the EPC Pure Identity URI is a member of the set denoted by the EPC Pure Identity Pattern URI if and only if `Ci` matches `Pi` for all $1 \le i \le n$, where “matches” is as defined above.

7 Background Information (non-normative)

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

The following papers capture the contributions of these individuals:

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

8 References

[EAN.UCCGS] “General EAN.UCC Specifications.” Version 6.0, EAN.UCC, Inc™.

[MIT-TR009] D. Engels, “The Use of the Electronic Product Code™,” MIT Auto-ID Center Technical Report MIT-TR007, February 2003, http://www.autoidlabs.com/whitepapers/mit-autoid-tr009.pdf

[RFC2141] R. Moats, “URN Syntax,” Internet Engineering Task Force Request for Comments RFC-2141, May 1997, http://www.ietf.org/rfc/rfc2141.txt.

[DOD Constructs] “United States Department of Defense Suppliers’ Passive RFID Information Guide,” http://www.dodrfid.org/supplierguide.htm

[Gen2 Specification] “EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF 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)

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

SGTIN Partition Table

Partition Value	Company Prefix		Indicator Digit and Item Reference	
	Bits	**Digits**	**Bits**	**Digit**
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

*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.

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

*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)

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

SGLN Partition Table

Partition Value	Company Prefix		Location Reference	
	Bits	**Digits**	**Bits**	**Digit**
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

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

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

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)

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

GRAI Partition Table

Partition Value	Company Prefix		Asset Type***	
	Bits	**Digits**	**Bits**	**Digit**
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

*Range of Asset Type field varies with Company Prefix.

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

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

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

GIAI Partition Table

Partition Value	**Company Prefix**		**Individual Asset Reference**	
	Bits	**Digits**	**Bits**	**Digits**
<GIAI-96>				
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
<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

*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	
EPC Memory Bank		CRC-16	Protocol Control Bits				EPC Bits				
Protocol Control Bits			Length bits	RFU	Numbering Systems Identifier						
Bit Field / EPC Identity Names	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	Total bits required
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

Notes:

GIAI-202 may have shorter Domain Identifier bits (Company Prefix and Individual Asset Reference) which will shorten the total bit requirement to 302 bits.
All the bits except for CRC-16 in the EPC Memory Bank requires encoding by application or process

This table illustrates the total number of bits required in the three logical memories (TID, Reserved and EPC) to support the EAN.UCC identities listed. User memory is set to zero required bits to load a single identity in the tag. As larger memories are defined and the User memory method of allocation is defined in this standard, additional bits can be assigned to user memory.

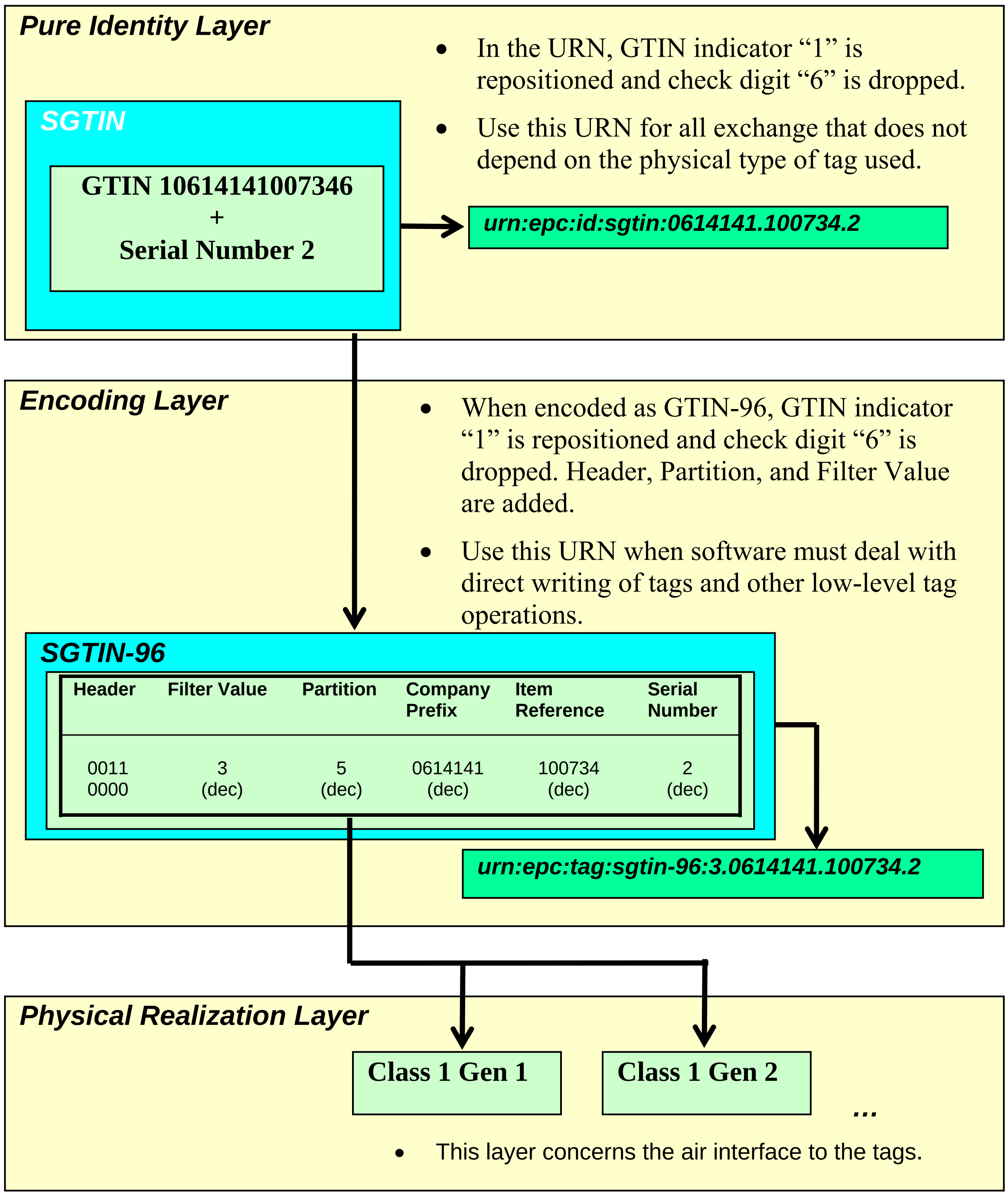
The EPC bits includes the extra bits required to round up to a fill the last 16 bit word.

The four identities; SGTIN-198, SGLN-195, GRAI-170 and GIAI-202 have been included in this standard to indicate to hardware vendors the user requirements for tag sizes and memory allocation required to support these longer identities. Please note that all three required more than 256 bits to contain all the fields required.

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



(01) 1 0614141 00734 6 (21) 2

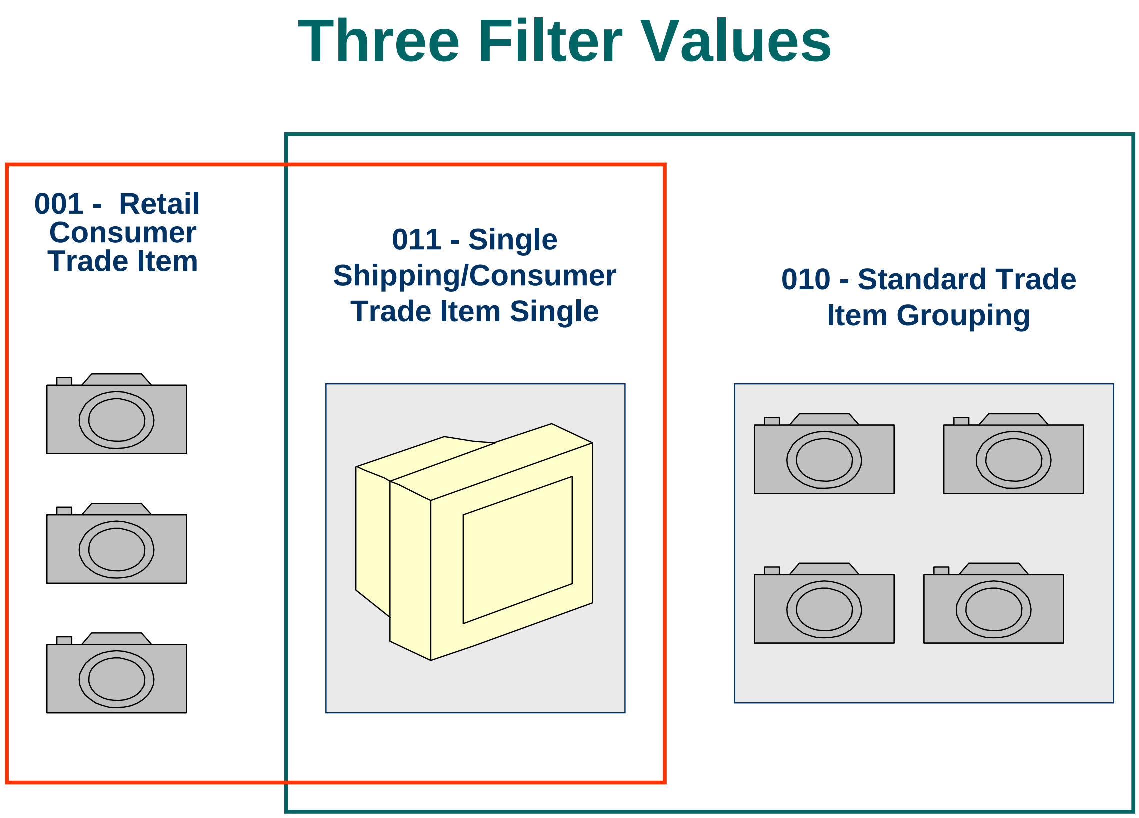
	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
SGTIN-96	8 bits	3 bits	3 bits	24 bits	20 bits	38 bits
	0011 0000 (Binary value)	3 (Decimal value)	5 (Decimal value)	0614141 (Decimal value)	100734 (Decimal value)	2 (Decimal value)

- (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.

Explanation of SGTIN Filter Values (non-normative).

SGTINs can be assigned at several levels, including: item, inner pack, case, and pallet. RFID can read through cardboard, and reading un-needed tags can slow us down, so Filter Values are used to “filter in” desired tags, or “filter out” unwanted tags. Filter values are used within the key type (i.e. SGTIN). While it is possible that filter values for several levels of packaging may be defined in the future, it was decided to use a minimum of values for now until the community gains more practical experience in their use. Therefore the three major categories of SGTIN filter values can be thought of in the following high level terms:

- Single Unit: A Retail Consumer Trade Item
- Not-a-single unit: A Standard Trade Item Grouping
- Items that could be included in both categories: For example, a Single Shipping container that contains a Single Consumer Trade Item



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

n	(2^n)$_{10}$	n	(2^n)$_{10}$
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		

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 then 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)

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

Notes

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

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

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

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