EPC™ Tag Data Standards Version 1.1 Rev.1.26

Standard Specification
19 November 2004

DOCUMENT HISTORY

Document Number:	1.1
Document Version:	1.26
Document Date :	2004-11-19

Document Summary

Document Title:	**Tag Data Specification**
Owner:	Tag Data Standard Work Group Chairperson: Yuichiro Hanawa, Mitsui & Co.
Status:	(*check one box*) ☐ DRAFT X Approved

Document Change History

Date of Change	Version	Reason for Change	Summary of Change
03-31-2004	1.24	Update errata	Comments and errata identified during public review
08-12-2004	1.25	Update errata	Further errata identified after release of v1.24 – especially inconsistencies regarding SSCC-96 and GRAI-96 partition tables
09-16-2004	1.25	Update errata	Correct errors re numeric range of Asset Type of GRAI-96 and include further reference to Appendix F in section 5 – i.e. need to check valid numeric ranges for 64-bit and 96-bit tags
09-30-2004	1.26		Added clarification to restrictions to serial numbers in SGTIN, GRAI, GIAI

Abstract

This document defines the EPC Tag Data Standards. 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 optional 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, 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 1.1 Rev.1.26.

Comments on this document should be sent to EPCglobal at epcinfo@epcglobalinc.org.

Changes from Previous Versions

Version 1.1, as the first formally specified version, serves as the basis for assignment and use of EPC numbers in standard, open systems applications. Previous versions, consisting of technical reports and working drafts, recommended certain headers, tag lengths, and EPC data structures. Many of these constructs have been modified in the development of

Version 1.1, and are generally not preserved for standard usage. Specifically, Version 1.1 supersedes all previous definitions of EPC Tag Data Standards.

Beyond the new content in Version 1.1 (such as the addition of new coding formats), the most significant changes to prior versions include the following:

1. Redefinition and clarification of the rules for assigning Header values: (i) to allow various Header lengths for a given length tag, to support more encoding options in a given length tag; and (ii) to indicate the tag length via the left-most ("preamble") portion of the Header, to support maximum reader efficiency.
2. Withdrawal of the 64-bit Universal Identifier format Types I-III, previously identified by specific 2-bit Headers. The Header assigned to the previous Universal Type II is now assigned to the 64-bit SGTIN encoding. The Type I and III Headers have not been reassigned to other encodings, but are rather simply designated as "reserved." The Headers associated with Types I and III will remain reserved for a yet-to-be-determined period of time to support tags that have previously used them, unless a clear need for them arises (as was the case with the SGTIN), in which case they will be considered for reassignment.
3. Renumbering of the 96-bit Universal Identifier Header to fit within the revised Header rules, and renaming this code the "General Identifier" to avoid confusion with the Unique Identifier (UID) that will be introduced by the US Department of Defense and its suppliers.

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 data consists of an EPC (or EPC Identifier) that uniquely identifies an individual object, as well as an optional Filter Value when judged to be necessary to enable effective and efficient reading of the EPC tags. In addition to this standardized data, certain Classes of EPC tags will allow user-defined data. The EPC Tag Data Standards will define the length and position of this data, without defining its content. Currently no user-defined data specifications exist since the related Class tags have not been defined.

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.

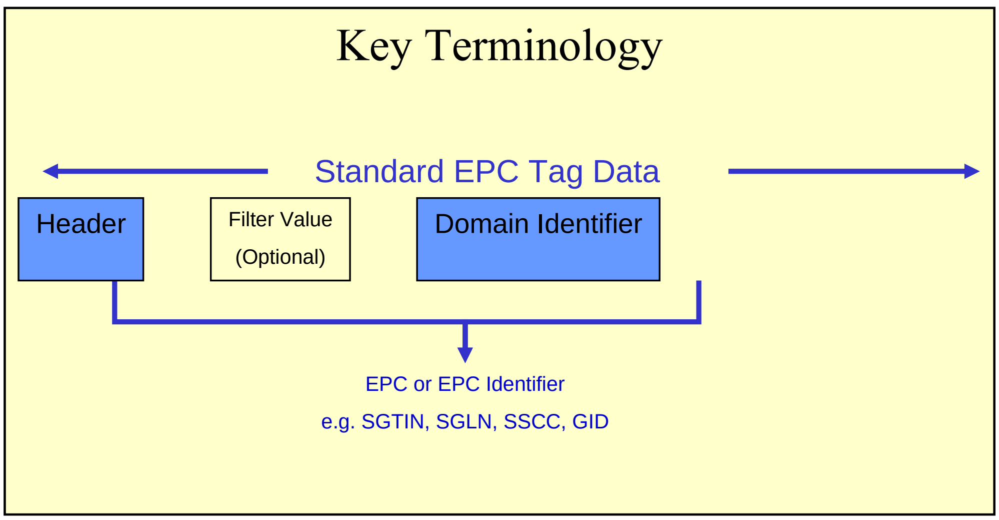


Figure A. EPC Terminology.

In this version of the EPC – EPC Version 1.1 – 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®), and the EAN.UCC Global Individual Asset Identifier (GIAI®).

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 V1.1 R1.26 has been approved by EAN.UCC 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 found online at www.epcglobalinc.org.

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.

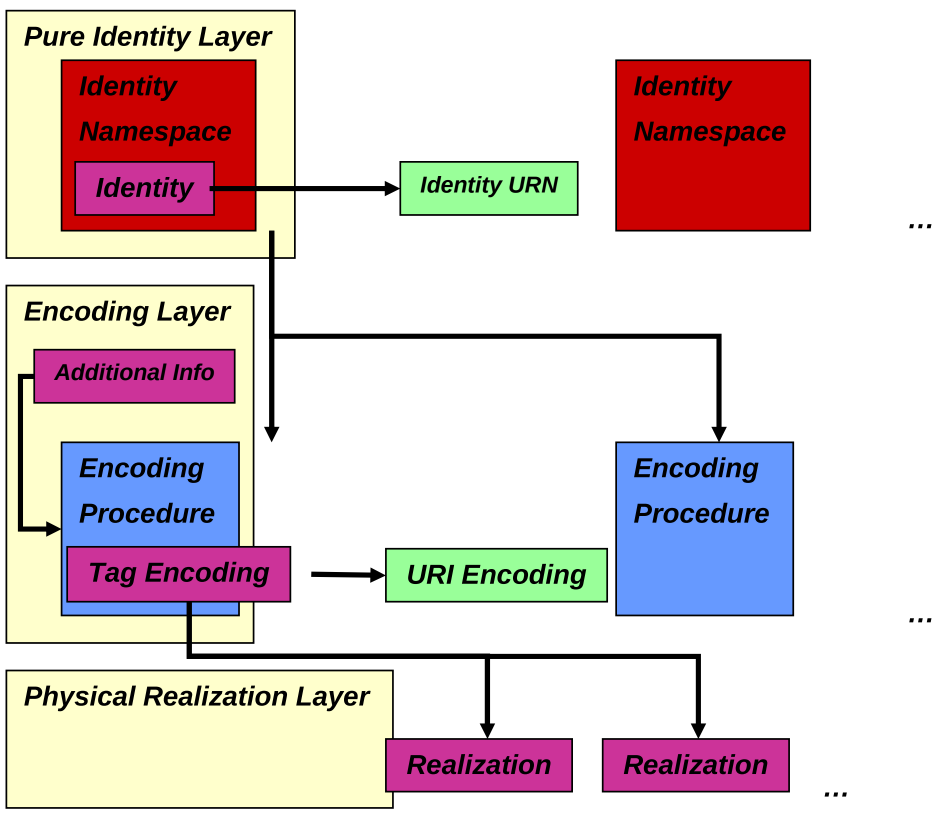


Figure B. Defined Identity Namespaces, Encodings, and Realizations.

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. For example, only 16,384 company prefixes can be encoded in the 64-bit SSCC scheme. 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.

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 EAN or UCC 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 EAN.UCC 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 encodings for EAN.UCC code types specified herein deviate from the aforementioned recommendations in the following ways:

EPC 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 encoding requires independent knowledge of the length of the Company Prefix.

EPC encodings do not include the check digit. Hence, converting from an EPC 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 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 EAN or UCC 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 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.

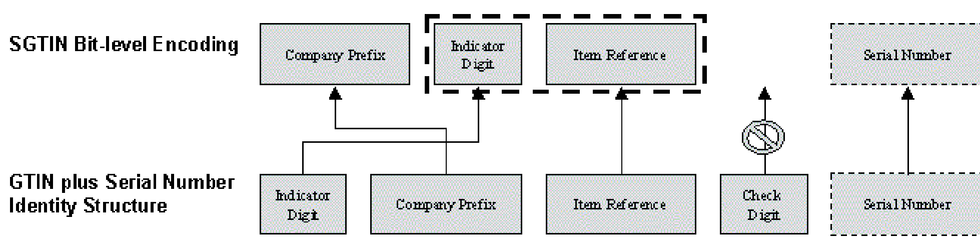


Figure C. How the parts of the decimal SGTIN are extracted, rearranged, and augmented for encoding.

The SGTIN is not explicitly defined in the EAN.UCC General Specifications. However, it may be considered equivalent to a UCC/EAN-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 AI 21 syntax is supported by the pure identity URI syntax specified in Section 4.3.3.

When representing serial numbers in 64- and 96-bit tags, however, only a subset of the serial number 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 characters, with no leading zeros, and whose value when considered as an integer fits within the range restrictions of the 64- and 96-bit tag encodings.

While these limitations exist for 64- and 96-bit tag encodings, future tag encodings may 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 accomodate further expansions of the Tag Data Standard.

Explanation (non-normative): The restrictions are necessary for 64- and 96-bit tags in order for serial numbers to fit within the small number of bits we have available. So we restrict the range, and also disallow alphabetic characters. The reason we also forbid leading zeros is that on these tags we're encoding the serial number value by considering it to be a decimal integer then encoding the integer value in binary. By considering it to be a decimal integer, we can't distinguish between "00034", "034", or "34" (for example) -- they all have the same value when considered as an integer rather than a character string. In order to insure that every encoded value can be decoded uniquely, we arbitrarily say that serial numbers can't have leading zeros. Then, when we see the bits 000000000000000000000010010 on the tag, we decode the serial number as "34" (not "034" or "00034").

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 that 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 EAN or UCC 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 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.

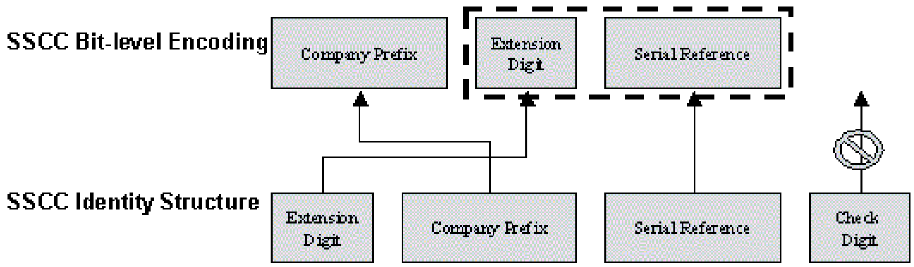


Figure D. How the parts of the decimal SSCC are extracted and rearranged for encoding.

2.1.2.3 Serialized Global Location Number (SGLN)

The Global Location Number (GLN) is defined by the General EAN.UCC Specifications. 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.

Recognizing these variables, the EPC GLN is meant to apply only to the physical location sub-type of GLN.

- The serial number field is reserved and should not be used, until the EAN.UCC community determines the appropriate way, if any, for extending GLN.

The SGLN consists of the following information elements:

The *Company Prefix*, assigned by EAN or UCC 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 *Serial Number*, assigned by the managing entity to an individual unique location.

- The serial number should not be used until specified by the EAN.UCC General Specifications .

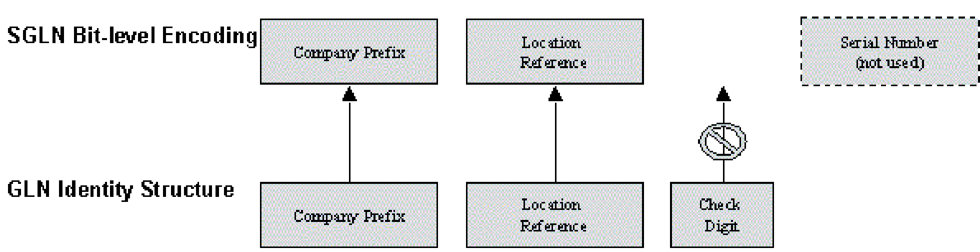


Figure E. How the parts of the decimal SGLN are extracted and rearranged for encoding.

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 EAN or UCC 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 EPC 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].

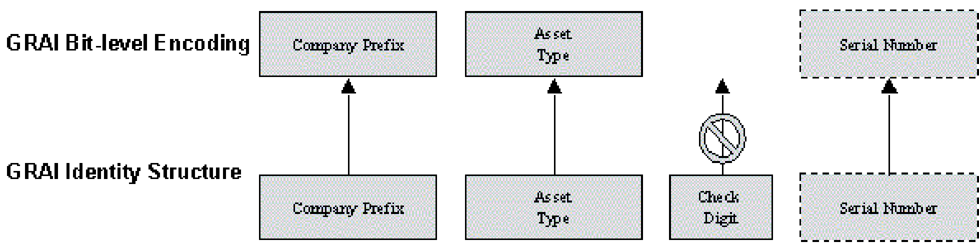


Figure F. How the parts of the decimal GRAI are extracted and rearranged for encoding.

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 EAN or UCC 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 EPC 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.

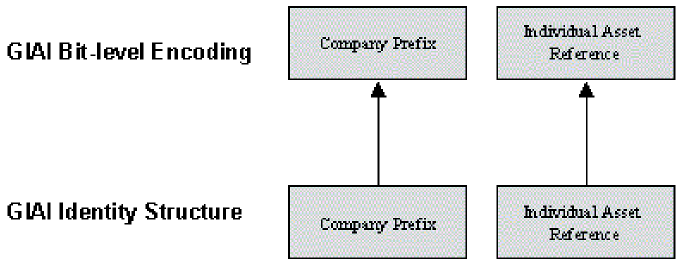


Figure G. How the parts of the decimal GIAI are extracted and rearranged for encoding.

3 EPC Tag Bit-level Encodings

The general structure of EPC encodings on a tag is as a string of bits (i.e., a binary representation), consisting of a tiered, variable length header followed by a series of numeric fields (Figure H) whose overall length, structure, and function are completely determined by the header value.

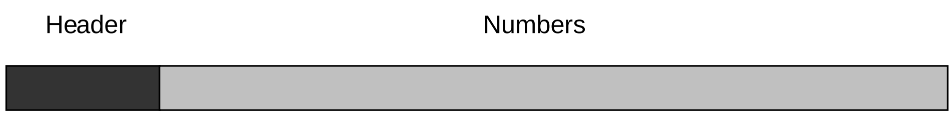


Figure H. The general structure of EPC encodings is as a string of bits, consisting of a variable length header followed by a series of value fields, whose overall length, structure, and function are completely determined by the header value.

3.1 Headers

As previously stated, the Header defines the overall length, identity type, and structure of the EPC Tag Encoding, including its Filter Value, if any. The header is of variable length, using a tiered approach in which a zero value in each tier indicates that the header is drawn from the next longer tier. For the encodings defined in this specification, headers are 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 can have 3 possible values (01, 10, and 11, not 00), and the 8-bit header can have 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).

Explanation (non-normative): The tiered scheme is designed to simplify the Header processing required by the Reader in order to determine the tag data format, particularly *the location of the Filter Value, while attempting to conserve bits for data values in the 64-bit tag. In the not-too-distant future, we expect to be able to "reclaim" the 2-bit tier when 64-bit tags are no longer needed, thereby expanding the 8-bit Header from 63 possible values to 255.*

The assignment of Header values has been designed so that the tag length may be easily discerned by examining the leftmost (or Preamble) bits of the Header. Moreover, the design is aimed at having as few Preambles per tag length as possible, ideally 1 but certainly no more than 2 or 3. This latter objective prompts us to avoid, if it all possible, using those Preambles that allow very few Header values (as noted in italics in Table 1 below). The purpose of this Preamble-to-Tag-Length design is so that RFID readers may easily determine a tag's length. See Appendix B for a detailed discussion of why this is important.

The currently assigned Headers are such that a tag may be inferred to be 64 bits if either the first two bits are non-zero *or* the first five bits are equal to 00001; otherwise, the Header indicates the tag is 96 bits. In the future, unassigned Headers may be assigned for these and other tag lengths.

Certain Preambles aren't currently tied to a particular tag length to leave open the option for additional tag lengths, especially longer ones that can accommodate longer coding schemes such as the Unique ID (UID) being pursued by suppliers to the US Department of Defense.

Eleven encoding schemes have been defined in this version of the EPC Tag Data Standard, as shown in Table 1 below.

Header Value (binary)	**Tag Length (bits)**	**EPC Encoding Scheme**
01	64	[Reserved 64-bit scheme]
10	64	SGTIN-64
11	64	[Reserved 64-bit scheme]
0000 0001 *0000 001x* *0000 01xx*	*na* *na* *Na*	*[1 reserved scheme]* *[2 reserved schemes]* *[4 reserved schemes]*
0000 1000	64	SSCC-64
0000 1001	64	GLN-64
0000 1010	64	GRAI-64
0000 1011	64	GIAI-64
0000 1100 … 0000 1111	64	[4 reserved 64-bit schemes]

Header Value (binary)	Tag Length (bits)	EPC Encoding Scheme
0001 0000 … 0010 1111	Na	[32 reserved schemes]
0011 0000	96	SGTIN-96
0011 0001	96	SSCC-96
0011 0010	96	GLN-96
0011 0011	96	GRAI-96
0011 0100	96	GIAI-96
0011 0101	96	GID-96
0011 0110 … 0011 1111	96	[10 reserved 96-bit schemes]
0000 0000 …		[reserved for future headers longer than 8 bits]

Table 1. Electronic Product Code Headers

3.2 Notational Conventions

In the remainder of this section, tag-encoding schemes are depicted using the following notation (See Table 2).

	Header	Filter Value	Company Prefix *Index*	Item Reference	Serial Number
SGTIN-64	2	3	14	20	25
	10 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	9 -1,048,575 (Decimal capacity*)	33,554,431 (Decimal capacity)

*Capacity 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 number of possible values that are permitted within that field. The number of possible values in the field can be either a specified limit, or simply two to the power of the number of bits in 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 decimal capacity is shown. In the example above, the decimal capacity for the Item Reference field depends on the length of the Company Prefix field; hence the decimal capacity 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.

3.3 General Identifier (GID-96)

The *General Identifier* is defined for a 96-bit EPC, and is independent of any existing identity specification or convention. 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, as shown in Table 3.

	Header	General Manager Number	Object Class	Serial Number
GID-96	8	28	24	36
	0011 0101 (Binary value)	268,435,456 (Decimal capacity)	16,777,216 (Decimal capacity)	68,719,476,736 (Decimal capacity)

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 *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.

The third component is *Object Class*, and 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.

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 code.

3.3.1.1 GID-96 Encoding Procedure

The following procedure creates a GID-96 encoding.

Given:

An 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 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).

3.3.1.2 GID-96 Decoding Procedure

Given:

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

Yields:

An 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.

3.4 Serialized Global Trade Item Number (SGTIN)

The EPC 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. Two encoding schemes are specified, SGTIN-64 (64 bits) and SGTIN-96 (96 bits).

In the SGTIN-64 encoding, the limited number of bits prohibits a literal embedding of the GTIN. As a partial solution, a Company Prefix *Index* is used. This Index, which can accommodate up to 16,384 codes, is assigned to companies that need to use the 64 bit tags, in addition to their existing EAN.UCC Company Prefixes. The Index is encoded on the tag instead of the Company Prefix, and is subsequently translated to the Company Prefix at low levels of the EPC system components (i.e. the Reader or Savant). While this means that only a limited number of Company Prefixes can be represented in the 64-bit tag, this is a transitional step to full accommodation in 96-bit and additional encoding schemes. The 64-bit company prefix index table can be found at http://www.onsepc.com.

3.4.1 SGTIN-64

The SGTIN-64 includes *five* fields – *Header, Filter Value, Company Prefix Index, Item Reference,* and *Serial Number*, as shown in Table 4.

	Header	Filter Value	Company Prefix *Index*	Item Reference	Serial Number
SGTIN-64	2	3	14	20	25
	10 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	9 -1,048,575 (Decimal capacity*)	33,554,431 (Decimal capacity)

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

Table 4. The EPC SGTIN-64 bit allocation, header, and decimal capacity.

Header is 2 bits, with a binary value of 10.

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, such as items, inner packs, cases and pallets. The Filter Values for 64-bit and 96-bit SGTIN are the same. The normative specifications for Filter Values are specified in Table 5. The value of 000 means “unspecified”. That is, a filter value of 000 means that the tag provides no information as to the logistics type of the object to which the tag is affixed. In particular, tags conforming to earlier versions of this specification, in which 000 was the only value approved for use, will have filter equal to 000. In general, filter 000 should not be used for SGTIN EPC tags intended for use in the supply chain following ratification of this standard. 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
Unspecified	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 .

Company Prefix Index encodes the EAN.UCC Company Prefix. The value of this field is not the Company Prefix itself, but rather an index into a table that provides the Company Prefix as well as an indication of the Company Prefix's length. The means by which hardware or software may obtain the contents of the translation table is specified in [Translation of 64-bit Tag Encoding Company Prefix Indices Into EAN.UCC Company Prefixes].

Item Reference encodes the GTIN Item Reference number and Indicator Digit. 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.

Formatted: Bulleted + Level: 1 + Aligned at: 18 pt + Tab after: 36 pt + Indent at: 36 pt, Tabs: 18 pt, Left

Serial Number contains a serial number. The SGTIN-64 encoding is only capable of representing integer-valued serial numbers with limited range. Other EAN.UCC specifications permit a broader range of serial numbers. In particular, the EAN-128 barcode symbology provides for a 20-character alphanumeric serial number to be associated with a GTIN using Application Identifier (AI) 21 [EANUCCGS]. It is possible to convert between the serial numbers in the SGTIN-64 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 digit characters, with no leading zeros, and whose value when interpreted as an integer falls within the range limitations of the SGTIN-64 tag encoding. These considerations are reflected in the encoding and decoding procedures below.

3.4.1.1 SGTIN-64 Encoding Procedure

The following procedure creates an SGTIN-64 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 $0 \leq S < 2^{25}$, *or* an UCC/EAN-128 Application Identifier 21 consisting of characters $s_1s_2 \ldots s_K$.

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

Procedure:

1. Extract the EAN.UCC Company Prefix $d_2d_3 \ldots d_{(L+1)}$

2. Do a reverse lookup of the Company Prefix in the Company Prefix Translation Table to obtain the corresponding Company Prefix Index, C. If the Company Prefix was not found in the Company Prefix Translation Table, stop: this GTIN cannot be encoded in the SGTIN-64 encoding.

3. Construct the Item Reference by concatenating digits $d_1d_{(L+2)}d_{(L+3)} \ldots d_{13}$ and considering the result to be a decimal integer, I. If $I \geq 2^{20}$, stop: this GTIN cannot be encoded in the SGTIN-64 encoding.

4. When the Serial Number is provided directly as an integer S where $0 \leq S < 2^{25}$, proceed to Step 5. Otherwise, when the Serial Number is provided as an UCC/EAN-128 Application Identifier 21 consisting of characters $s_1s_2 \ldots s_K$, construct the Serial Number by concatenating digits $s_1s_2 \ldots s_K$. If any of these characters is not a digit, stop: this Serial Number cannot be encoded in the SGTIN-64 encoding. Also, if $K > 1$ and $s_1 = 0$, stop: this Serial Number cannot be encoded in the SGTIN-64 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 $S \geq 2^{25}$, stop: this Serial Number cannot be encoded in the SGTIN-64 encoding.

5. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header 10 (2 bits), Filter Value F (3 bits), Company Prefix Index C from Step 2 (14 bits), Item Reference from Step 3 (20 bits), Serial Number S from Step 4 (25 bits).

3.4.1.2 SGTIN-64 Decoding Procedure

Given:

An SGTIN-64 as a 64-bit bit string $10b_{61}b_{60} \ldots b_0$ (where the first two bits 10 are the header)

Yields:

An EAN.UCC GTIN-14

A Serial Number

A Filter Value

Procedure:

1. Bits $b_{61}b_{60}b_{59}$, considered as an unsigned integer, are the Filter Value.

2. Extract the Company Prefix Index C by considering bits $b_{58}b_{57}\ldots b_{45}$ as an unsigned integer.

3. Look up the Company Prefix Index C in the Company Prefix Translation Table to obtain the EAN.UCC Company Prefix $p_1p_2\ldots p_L$ consisting of L decimal digits (the value of L is also obtained from the table). If the Company Prefix Index C is not found in the Company Prefix Translation Table, stop: this bit string cannot be decoded as an SGTIN-64.

4. Consider bits $b_{44}b_{43}\ldots b_{25}$ 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-64 encoding. Otherwise, convert this integer to a (13-L)-digit decimal number $i_1i_2\ldots i_{(13-L)}$, adding leading zeros as necessary to make (13-L) digits.

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

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

7. The EAN.UCC GTIN-14 is the concatenation of digits from Steps 5 and 6: $d_1d_2\ldots d_{14}$.

8. Bits $b_{24}b_{23}\ldots b_0$, considered as an unsigned integer, are the Serial Number.

9. (Optional) If it is desired to represent the serial number as a UCC/EAN-128 Application Identifier 21, convert the integer from Step 8 to a decimal string with no leading zeros. If the integer in Step 8 is zero, convert it to a string consisting of the single character "`0`".

3.4.2 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 6.

	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
SGTIN-96	8	3	3	20-40	24-4	38
	0011 0000 (Binary value)	8 (Decimal capacity)	8 (Decimal capacity)	999,999 – 999,999,999,999 (Decimal capacity*)	9,999,999 – 9 (Decimal capacity*)	274,877,906,943 (Decimal capacity)

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

Table 6. The EPC SGTIN-96 bit allocation, header, and decimal capacity.

Header is 8-bits, with a binary value of 0011 0000.

Filter Value is not part of the GTIN or EPC identifier, but is used for fast filtering and pre-selection of basic logistics types, such as items, inner packs, cases and pallets. The Filter Values for 64-bit and 96-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 (plus 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 7.

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. Other EAN.UCC specifications permit a broader range of serial numbers. In particular, the EAN-128 barcode symbology provides for a 20-character alphanumeric serial number to be associated with a GTIN using Application Identifier (AI) 21 [EANUCCGS]. 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 digit characters, 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		Item Reference and Indicator Digit	
	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 7. SGTIN-96 Partitions.

3.4.2.1 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 $0 \le S < 2^{38}$, *or* an UCC/EAN-128 Application Identifier 21 consisting of characters $s_1s_2\ldots s_K$.

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 7) 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 7, stop: this GTIN cannot be encoded in an SGTIN-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 Item Reference by concatenating digits $d_1d_{(L+2)}d_{(L+3)}\ldots d_{13}$ and considering the result to be a decimal integer, I.

4. When the Serial Number is provided directly as an integer S where $0 \le S < 2^{38}$, proceed to Step 5. Otherwise, when the Serial Number is provided as an UCC/EAN-128 Application Identifier 21 consisting of characters $s_1s_2\ldots s_K$, construct the Serial Number by concatenating digits $s_1s_2\ldots s_K$. 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*.

3.4.2.2 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 7 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 UCC/EAN-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 Serial Shipping Container Code (SSCC)

The EPC 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. Two encoding schemes are specified, SSCC-64 (64 bits) and SSCC-96 (96 bits).

In the 64-bit EPC, the limited number of bits prohibits a literal embedding of the EAN.UCC Company Prefix. As a partial solution, a Company Prefix *Index* is used. This Index, which can accommodate up to 16,384 codes, is assigned to companies that need to use the 64 bit tags, in addition to their existing Company Prefixes. The Index is encoded on the tag instead of the Company Prefix, and is subsequently translated to the Company Prefix at low levels of the EPC system components (i.e. the Reader or Savant). While this means a limited number of Company Prefixes can be represented in the 64-bit tag, this is a transitional step to full accommodation in 96-bit and additional encoding schemes.

3.5.1 SSCC-64

In addition to a Header, the EPC SSCC-64 is composed of three fields: the *Filter Value*, *Company Prefix Index,* and *Serial Reference*, as shown in Table 8.

	Header	Filter Value	Company Prefix *Index*	Serial Reference
SSCC-64	8	3	14	39
	0000 1000 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	99,999 - 99,999,999,999 (Decimal capacity*)

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

Table 8. The EPC 64-bit SSCC bit allocation, header, and decimal capacity.

Header is 8-bits, with a binary value of 0000 1000.

Filter Value is not part of the SSCC or EPC identifier, but is used for fast filtering and pre-selection of basic logistics types, such as cases and pallets. The Filter Values for 64-bit and 96-bit SSCC are the same. The normative specifications for Filter Values are specified in Table 9. The value of 000 means "unspecified". That is, a filter value of 000 means that the tag provides no information as to the logistics type of the object to which the tag is affixed. In particular, tags conforming to earlier versions of this specification, in which 000 was the only value approved for use, will have filter equal to 000. In general, filter 000 should not be used for SSCC EPC tags intended for use in the supply chain following ratification of this standard.

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

Table 9. SSCC Filter Values

Company Prefix Index encodes the EAN.UCC Company Prefix. The value of this field is not the Company Prefix itself, but rather an index into a table that provides the Company Prefix as well as an indication of the Company Prefix's length. The means by which hardware or software may obtain the contents of the translation table is specified in [Translation of 64-bit Tag Encoding Company Prefix Indices Into EAN.UCC Company Prefixes].

Serial Reference is a unique number for each instance, comprised of the Serial Reference and the Extension digit. 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.

Formatted: Indent: Left: 0 pt, First line: 0 pt, Bulleted + Level: 1 + Aligned at: 18 pt + Tab after: 36 pt + Indent at: 36 pt, Tabs: 0 pt, List tab + Not at 36 pt

3.5.1.1 SSCC-64 Encoding Procedure

The following procedure creates an SSCC-64 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 $0 \le F < 8$

Procedure:

1. Extract the EAN.UCC Company Prefix $d_2d_3 \ldots d_{(L+1)}$

2. Do a reverse lookup of the Company Prefix in the Company Prefix Translation Table to obtain the corresponding Company Prefix Index, *C*. If the Company Prefix was not found in the Company Prefix Translation Table, stop: this SSCC cannot be encoded in the SSCC-64 encoding.

3. Construct the Serial Reference by concatenating digits $d_1 d_{(L+2)} d_{(L+3)} \ldots d_{17}$ and considering the result to be a decimal integer, *I*. If $I \geq 2^{39}$, stop: this SSCC cannot be encoded in the SSCC-64 encoding.

4. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header 00001000 (8 bits), Filter Value *F* (3 bits), Company Prefix Index *C* from Step 2 (14 bits), Serial Reference from Step 3 (39 bits).

3.5.1.2 SSCC-64 Decoding Procedure

Given:

An SSCC-64 as a 64-bit bit string $00001000 b_{55} b_{54} \ldots b_0$ (where the first eight bits 00001000 are the header)

Yields:

An EAN.UCC SSCC

A Filter Value

Procedure:

1. Bits $b_{55}b_{54}b_{53}$, considered as an unsigned integer, are the Filter Value.

2. Extract the Company Prefix Index *C* by considering bits $b_{52}b_{51} \ldots b_{39}$ as an unsigned integer.

3. Look up the Company Prefix Index *C* in the Company Prefix Translation Table to obtain the EAN.UCC Company Prefix $p_1p_2 \ldots p_L$ consisting of L decimal digits (the value of L is also obtained from the table). If the Company Prefix Index *C* is not found in the Company Prefix Translation Table, stop: this bit string cannot be decoded as an SSCC-64.

4. Consider bits $b_{38}b_{37} \ldots b_0$ as an unsigned integer. If this integer is greater than or equal to $10^{(17-L)}$, stop: the input bit string is not a legal SSCC-64 encoding. Otherwise, convert this integer to a (17-L)-digit decimal number $i_1 i_2 \ldots i_{(17-L)}$, adding leading zeros as necessary to make (17-L) digits.

5. Construct a 17-digit number $d_1d_2 \ldots d_{17}$ where $d_1 = s_1$ from Step 4, $d_2d_3 \ldots d_{(L+1)} = p_1p_2 \ldots p_L$ from Step 3, and $d_{(L+2)}d_{(L+3)} \ldots d_{17} = i_2\, i_3 \ldots \mathrm{i}_{(17-L)}$ from Step 4.

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

7. The EAN.UCC SSCC is the concatenation of digits from Steps 5 and 6: $d_1d_2 \ldots d_{18}$.

3.5.2 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 10.

	Header	Filter Value	Partition	Company Prefix	Serial Reference	Unallocated
SSCC-96	8	3	3	20-40	38-18	24
	0011 0001 (Binary value)	8 (Decimal capacity)	8 (Decimal capacity)	999,999 – 999,999,999,999 (Decimal capacity*)	99,999,999,999 – 99,999 (Decimal capacity*)	[Not Used]

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

Table 10. The EPC 96-bit SSCC bit allocation, header, and decimal capacity.

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, such as cases and pallets. The Filter Values for 64-bit and 96-bit SSCC are the same. See Table 9.

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 (including 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 digit(s). Table 11 shows allowed values of the partition value and the corresponding lengths of the company prefix and serial reference.

Partition Value (*P*)	**Company Prefix**		**Serial Reference and Extension Digit**	
	Bits (*M*)	**Digits (*L*)**	**Bits (*N*)**	**Digits**
0	40	12	18	5
1	37	11	21	6
2	34	10	24	7
3	30	9	28	8
4	27	8	31	9
5	24	7	34	10

Partition Value (*P*)	Company Prefix		Serial Reference and Extension Digit	
	Bits (*M*)	Digits (*L*)	Bits (*N*)	Digits
6	20	6	38	11

Table 11. 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 Serial Reference and the Extension digit. 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.

3.5.2.1 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 11) to determine the Partition Value, P, the number of bits M in the Company Prefix field, and the number of bits N in the Serial Reference and Extension Digit field. If L is not found in any row of Table 11, 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 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.

3.5.2.2 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 11 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 $d_{(L+2)}d_{(L+3)}\ldots d_{17} = \mathrm{i}_2\ i_3\ldots\mathrm{i}_{(17\text{-L})}$ 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: $d_1d_2\ldots d_{18}$.

3.6 Serialized Global Location Number (SGLN)

The EPC encoding scheme for GLN permits the direct embedding of EAN.UCC System standard GLN on EPC tags. The serial number field is not used. In all cases the check digit is not encoded. Two encoding schemes are specified, SGLN-64 (64 bits) and SGLN-96 (96 bits).

In the SGLN-64 encoding, the limited number of bits prohibits a literal embedding of the GLN. As a partial solution, a Company Prefix *Index* is used. This *index*, which can accommodate up to 16,384 codes, is assigned to companies that need to use the 64 bit tags, in addition to their existing EAN.UCC Company Prefixes. The *index* is encoded on the tag instead of the Company Prefix, and is subsequently translated to the Company Prefix at low levels of the EPC system components (i.e. the Reader or Savant).

While this means a limited number of Company Prefixes can be represented in the 64-bit tag, this is a transitional step to full accommodation in 96-bit and additional encoding schemes.

3.6.1 SGLN-64

The SGLN-64 includes *four* fields in addition to the header – *Filter Value, Company Prefix Index, Location Reference,* and *Serial Number*, as shown in Table 12.

	Header	Filter Value	Company Prefix Index	Location Reference	Serial Number
SGLN-64	8	3	14	20	19
	0000 1001 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	999,999 - 0 (Decimal capacity*)	524,288 (Decimal capacity) [Not Used]

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

Table 12. The EPC SGLN-64 bit allocation, header, and decimal capacity.

Header is 8 bits, with a binary value of 0000 1001.

Filter Value is not part of the SGLN pure identity, but is additional data that is used for fast filtering and pre-selection of basic location types. The Filter Values for 64-bit and 96-bit SGLN are the same. See Table 13 for currently defined filter values.

Company Prefix Index encodes the EAN.UCC Company Prefix. The value of this field is not the Company Prefix itself, but rather an index into a table that provides the Company Prefix as well as an indication of the Company Prefix's length. The means by which hardware or software may obtain the contents of the translation table is specified in [Translation of 64-bit Tag Encoding Company Prefix Indices Into EAN.UCC Company Prefixes].

Location Reference encodes the GLN Location Reference number.

Serial Number contains a serial number. Note: The serial number field is reserved and should not be used, until the EAN.UCC community determines the appropriate way, if any, for extending GLN.

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

Table 13. SGLN Filter Values .

3.6.1.1 SGLN-64 Encoding Procedure

The following procedure creates an SGLN-64 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

A Serial Number S where $0 \leq S < 2^{19}$

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

Procedure:

1. Extract the EAN.UCC Company Prefix $d_1d_2\ldots d_L$

2. Do a reverse lookup of the Company Prefix in the Company Prefix Translation Table to obtain the corresponding Company Prefix Index, *C*. If the Company Prefix was not found in the Company Prefix Translation Table, stop: this GLN cannot be encoded in the SGLN-64 encoding.

3. 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 $I \geq 2^{20}$, stop: this GLN cannot be encoded in the SGLN-64 encoding.

4. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header 00001001 (8 bits), Filter Value *F* (3 bits), Company Prefix Index *C* from Step 2 (14 bits), Location Reference from Step 3 (20 bits), Serial Number *S* (19 bits).

3.6.1.2 SGLN-64 Decoding Procedure

Given:

An SGLN-64 as a 64-bit bit string 00001001$b_{55}b_{54}\ldots b_0$ (where the first eight bits 00001001 are the header)

Yields:

An EAN.UCC GLN

A Serial Number

A Filter Value

Procedure:

1. Bits $b_{55}b_{54}b_{53}$, considered as an unsigned integer, are the Filter Value.

2. Extract the Company Prefix Index C by considering bits $b_{52}b_{51}\ldots b_{39}$ as an unsigned integer.

3. Look up the Company Prefix Index C in the Company Prefix Translation Table to obtain the EAN.UCC Company Prefix $p_1p_2\ldots p_L$ consisting of L decimal digits (the value of L is also obtained from the table). If the Company Prefix Index C is not found in the Company Prefix Translation Table, stop: this bit string cannot be decoded as an SGLN-64.

4. Consider bits $b_{38}b_{37}\ldots b_{19}$ 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 SGLN-64 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.

5. 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 3, and $d_{(L+1)}d_{(L+2)}\ldots d_{12} = i_1\, i_2\ldots i_{(12-L)}$ from Step 4.

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

7. The EAN.UCC GLN is the concatenation of digits from Steps 5 and 6: $d_1d_2\ldots d_{13}$.

8. Bits $b_{18}b_{17}\ldots b_0$, considered as an unsigned integer, are the Serial Number.

3.6.2 SGLN-96

In addition to a Header, the SGLN-96 is composed of five fields: the *Filter Value*, *Partition*, *Company Prefix, Location Reference*, and *Serial Number*, as shown in Table 14.

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 64-bit and 96-bit GLN are the same. See Table 13.

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 15.

	Header	Filter Value	Partition	Company Prefix	Location Reference	Serial Number
SGLN-96	8	3	3	20-40	21-1	41
	0011 0010 (Binary value)	8 (Decimal capacity)	8 (Decimal capacity)	999,999 – 999,999,999,999 (Decimal capacity*)	999,999 – 0 (Decimal capacity*)	2,199,023,255,552 (Decimal capacity) [Not Used]

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

Table 14. The EPC SGLN-96 bit allocation, header, and decimal capacity.

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

Location Reference encodes the GLN Location Reference number.

Serial Number contains a serial number. Note: The serial number field is reserved and should not be used, until the EAN.UCC community determines the appropriate way, if any, for extending GLN.

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 15. SGLN-96 Partitions.

3.6.2.1 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

A Serial Number S where $0 \leq S < 2^{41}$

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 15) 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 15, 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. 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.

4. 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 from Step 3 (N bits), Serial Number S (41 bits). Note that $M+N = 41$ bits for all P.

3.6.2.2 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

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 SGLN-96.

3. Look up the Partition Value P in Table 15 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. 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 $d_{(L+1)}d_{(L+2)}\ldots d_{12} = i_2\, i_3\ldots i_{(12\text{-L})}$ 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. Bits $b_{40}b_{39}\ldots b_0$, considered as an unsigned integer, are the Serial Number.

3.7 Global Returnable Asset Identifier (GRAI)

The EPC 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. Two encoding schemes are specified, GRAI-64 (64 bits) and GRAI-96 (96 bits).

In the GRAI-64 encoding, the limited number of bits prohibits a literal embedding of the GRAI. As a partial solution, a Company Prefix *Index* is used. This Index, which can accommodate up to 16,384 codes, is assigned to companies that need to use the 64 bit tags, in addition to their existing EAN.UCC Company Prefixes. The Index is encoded on the tag instead of the Company Prefix, and is subsequently translated to the Company Prefix at low levels of the EPC system components (i.e. the Reader or Savant). While this means that only a limited number of Company Prefixes can be represented in the 64-bit tag, this is a transitional step to full accommodation in 96-bit and additional encoding schemes.

3.7.1 GRAI-64

The GRAI-64 includes *four* fields in addition to the Header – *Filter Value, Company Prefix Index, Asset Type,* and *Serial Number*, as shown in Table 16.

	Header	Filter Value	Company Prefix Index	Asset Type	Serial Number
GRAI-64	8	3	14	20	19
	0000 1010 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	999,999 - 0 (Decimal capacity*)	524,288 (Decimal capacity)

*Capacity of Asset Type field varies with Company Prefix.

Table 16. The EPC GRAI-64 bit allocation, header, and decimal capacity.

Header is 8 bits, with a binary value of 0000 1010.

Filter Value is not part of the GRAI pure identity, but is additional data that is used for fast filtering and pre-selection of basic asset types. The Filter Values for 64-bit and 96-bit GRAI are the same. See Table 17 for currently defined GRAI filter values. This specification anticipates that valuable Filter Values will be determined once there has been time to consider the possible use cases.

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

Table 17. GRAI Filter Values

Company Prefix Index encodes the EAN.UCC Company Prefix. The value of this field is not the Company Prefix itself, but rather an index into a table that provides the Company

Prefix as well as an indication of the Company Prefix's length. The means by which hardware or software may obtain the contents of the translation table is specified in [Translation of 64-bit Tag Encoding Company Prefix Indices Into EAN.UCC Company Prefixes].

Asset Type encodes the GRAI Asset Type number.

Serial Number contains a serial number. The 64-bit and 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.

3.7.1.1 GRAI-64 Encoding Procedure

The following procedure creates a GRAI-64 encoding.

Given:

An EAN.UCC GRAI consisting of digits $0d_2 \ldots d_K$, where $15 \leq K \leq 30$.

The length L of the company prefix portion of the GRAI

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

Procedure:

1. Extract the EAN.UCC Company Prefix $d_2d_3 \ldots d_{L+1}$

2. Do a reverse lookup of the Company Prefix in the Company Prefix Translation Table to obtain the corresponding Company Prefix Index, C. If the Company Prefix was not found in the Company Prefix Translation Table, stop: this GRAI cannot be encoded in the GRAI-64 encoding.

3. 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. If $I \geq 2^{20}$, stop: this GRAI cannot be encoded in the GRAI-64 encoding.

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-64 encoding. Otherwise, consider the result to be a decimal integer, S. If $S \geq 2^{19}$, stop: this GRAI cannot be encoded in the GRAI-64 encoding. Also, if $K > 15$ and $d_{15} = 0$, stop: this GRAI cannot be encoded in the GRAI-64 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 00001010 (8 bits), Filter Value F (3 bits), Company Prefix Index C from Step 2 (14 bits), Asset Type I from Step 3 (20 bits), Serial Number S from Step 4 (19 bits).

3.7.1.2 GRAI-64 Decoding Procedure

Given:

An GRAI-64 as a 64-bit bit string 00001010$b_{55}b_{54}\ldots b_0$ (where the first eight bits 00001010 are the header)

Yields:

An EAN.UCC GRAI

A Filter Value

Procedure:

1. Bits $b_{55}b_{54}b_{53}$, considered as an unsigned integer, are the Filter Value.

2. Extract the Company Prefix Index C by considering bits $b_{52}b_{51}\ldots b_{39}$ as an unsigned integer.

3. Look up the Company Prefix Index C in the Company Prefix Translation Table to obtain the EAN.UCC Company Prefix $p_1p_2\ldots p_L$ consisting of L decimal digits (the value of L is also obtained from the table). If the Company Prefix Index C is not found in the Company Prefix Translation Table, stop: this bit string cannot be decoded as a GRAI-64.

4. Consider bits $b_{38}b_{37}\ldots b_{19}$ 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-64 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.

5. 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 3, and $d_{(L+2)}d_{(L+3)}\ldots d_{13} = i_1\ i_2\ldots i_{(12-L)}$ from Step 4.

6. 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.

7. Consider bits $b_{18}b_{17}\ldots b_0$ as an unsigned integer. Convert this integer into 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).

8. The EAN.UCC GRAI is the concatenation of the digits from Steps 5, 6, and 7: $0d_2d_3\ldots d_K$.

3.7.2 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 18.

	Header	Filter Value	Partition	Company Prefix	Asset Type	Serial Number
GRAI-96	8	3	3	20-40	24-4	38
	0011 0011 (Binary value)	8 (Decimal capacity)	8 (Decimal capacity)	999,999 – 999,999,999,999 (Decimal capacity*)	999,999 – 0 (Decimal capacity*)	274,877,906,943 (Decimal capacity)

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

Table 18. The EPC GRAI-96 bit allocation, header, and decimal capacity.

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 64-bit and 96-bit GRAI are the same. See Table 17.

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 19.

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 19. GRAI-96 Partitions.

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

Asset Type encodes the GRAI Asset Type number.

Serial Number contains a serial number. The 64-bit and 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.

3.7.2.1 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 $15 \leq K \leq 30$.

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 19) 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 19, 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. 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.

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.

3.7.2.2 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 19 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. 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 Global Individual Asset Identifier (GIAI)

The EPC encoding scheme for GIAI permits the direct embedding of EAN.UCC System standard GIAI codes on EPC tags (except as noted below for 64-bit tags). Two encoding schemes are specified, GIAI-64 (64 bits) and GIAI-96 (96 bits).

In the 64-bit EPC, the limited number of bits prohibits a literal embedding of the EAN.UCC Company Prefix. As a partial solution, a Company Prefix *Index* is used. In addition to their existing Company Prefixes, this Index, which can accommodate up to 16,384 codes, is assigned to companies that need to use the 64 bit tags. The Index is encoded on the tag instead of the Company Prefix, and is subsequently translated to the Company Prefix at low levels of the EPC system components (i.e. the Reader or Savant). While this means a limited number of Company Prefixes can be represented in the 64-bit tag, this is a transitional step to full accommodation in 96-bit and additional encoding schemes.

3.8.1 GIAI-64

In addition to a Header, the EPC GIAI-64 is composed of three fields: the *Filter Value*, *Company Prefix Index,* and *Individual Asset Reference*, as shown in Table 20.

	Header	Filter Value	Company Prefix *Index*	Individual Asset Reference
GIAI-64	8	3	14	39

	0000 1011 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	549,755,813,888 (Decimal capacity)

Table 20. The EPC 64-bit GIAI bit allocation, header, and decimal capacity.

Header is 8-bits, with a binary value of 0000 1011.

Filter Value is not part of the GIAI pure identity, but is additional data that is used for fast filtering and pre-selection of basic asset types. The Filter Values for 64-bit and 96-bit GIAI are the same. See Table 21 for currently defined GIAI filter values. This specification anticipates that valuable Filter Values will be determined once there has been time to consider the possible use cases.

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

Table 21. GIAI Filter Values

Company Prefix Index encodes the EAN.UCC Company Prefix. The value of this field is not the Company Prefix itself, but rather an index into a table that provides the Company Prefix as well as an indication of the Company Prefix's length. The means by which hardware or software may obtain the contents of the translation table is specified in [Translation of 64-bit Tag Encoding Company Prefix Indices Into EAN.UCC Company Prefixes].

Individual Asset Reference is a unique number for each instance. The 64-bit and 96-bit tag encodings are 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.

3.8.1.1 GIAI-64 Encoding Procedure

The following procedure creates a GIAI-64 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. Extract the EAN.UCC Company Prefix $d_1d_2\ldots d_L$

2. Do a reverse lookup of the Company Prefix in the Company Prefix Translation Table to obtain the corresponding Company Prefix Index, C. If the Company Prefix was not found in the Company Prefix Translation Table, stop: this GIAI cannot be encoded in the GIAI-64 encoding.

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-64 encoding. Otherwise, consider the result to be a decimal integer, I. If $I \geq 2^{39}$, stop: this GIAI cannot be encoded in the GIAI-64 encoding. Also, if $K > L+1$ and $d_{(L+1)} = 0$, stop: this GIAI cannot be encoded in the GIAI-64 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 00001011 (8 bits), Filter Value F (3 bits), Company Prefix Index C from Step 2 (14 bits), Individual Asset Reference from Step 3 (39 bits).

3.8.1.2 GIAI-64 Decoding Procedure

Given:

An GIAI-64 as a 64-bit bit string 00001011$b_{55}b_{54}\ldots b_0$ (where the first eight bits 00001011 are the header)

Yields:

An EAN.UCC GIAI

A Filter Value

Procedure:

1. Bits $b_{55}b_{54}b_{53}$, considered as an unsigned integer, are the Filter Value.

2. Extract the Company Prefix Index C by considering bits $b_{52}b_{51}\ldots b_{39}$ as an unsigned integer.

3. Look up the Company Prefix Index C in the Company Prefix Translation Table to obtain the EAN.UCC Company Prefix $p_1p_2\ldots p_L$ consisting of L decimal digits (the value

of L is also obtained from the table). If the Company Prefix Index *C* is not found in the Company Prefix Translation Table, stop: this bit string cannot be decoded as a GIAI-64.

4. Consider bits $b_{38}b_{37}\ldots b_0$ as an unsigned integer. If this integer is greater than or equal to $10^{(30-L)}$, stop: the input bit string is not a legal GIAI-64 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).

5. 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 3, and $d_{(L+1)}d_{(L+2)}\ldots d_K = s_1\, s_2\ldots s_J$ from Step 4. This K-digit number, where $K \leq 30$, is the EAN.UCC GIAI.

3.8.2 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 22.

	Header	Filter Value	Partition	Company Prefix	Individual Asset Reference
GIAI-96	8	3	3	20-40	62-42
	0011 0100 (Binary value)	8 (Decimal capacity)	8 (Decimal capacity)	999,999 – 999,999,999,999 (Decimal capacity*)	4,611,686,018,427,387,904 – 4,398,046,511,103 (Decimal capacity*)

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

Table 22. The EPC 96-bit GIAI bit allocation, header, and decimal capacity.

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 64-bit and 96-bit GIAI are the same. See Table 21.

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 23.

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 23. GIAI-96 Partitions.

Company Prefix contains a literal embedding of the Company Prefix.

Individual Asset Reference is a 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.

3.8.2.1 GIAI-96 Encoding Procedure

The following procedure creates a GIAI-96 encoding.

Given:

An EAN.UCC GIAI consisting of digits $d_1d_2…d_K$, 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 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-96.

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

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

3.8.2.2 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 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 $b_{81}b_{80}\ldots b_{(82-M)}$ as an unsigned integer. If this integer is greater than or equal to 10^L, stop: the input bit string is not a legal 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 $b_{(81-M)}\, b_{(80-M)}\ldots b_0$ as an unsigned integer. If this integer is greater than or equal to $10^{(30-L)}$, 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.

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.

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 are 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 are 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 Reference 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 4.3 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 Universal 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.SerialNumber`*

```
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 the UCC or EAN. (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), appended at the beginning of the serial reference. In no case are check digits included in URI representations.)

In contrast to the other fields, the *`SerialNumber`* field of the SGLN is a pure integer, with no leading zeros. The *`SerialNumber`* field of the SGTIN and GRAI, as well as the `IndividualAssetReference` field of the GIAI, may include digits, letters, and certain other characters. In order for an SGTIN, GRAI, or GIAI to be encodable on a 64-bit and 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:0652642.12345.400
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 example, the corresponding GLN is 0652642123458, where the last digit (8) is the check digit, not included in the URI.

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

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

Note that all five 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.

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 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 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-64`, corresponding to the 96-bit encoding and the 64-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 64-bit encoding:

`urn:epc:tag:sgtin-64:3.0652642.800031.400`

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

4.2.2 URIs for Raw Bit Strings Arising From Invalid Tags

Certain bit strings do not correspond to legal encodings. For example, if the most significant bits cannot be recognized as a valid EPC header, the bit-level pattern is not a legal EPC. For a second example, if the binary value of a field in a 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. Nevertheless, software may wish to report such invalid bit-level patterns to users or to other software, and so 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 decimal number. 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:64.20018283527919`

It is intended that this URI form be used only when reporting errors associated with reading invalid tags. 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.

4.2.3 URIs for EPC Patterns

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

```
urn:epc:pat:sgtin-64:3.0652642.[1024-2047].*
```

This pattern refers to any EPC SGTIN Identifier 64-bit tag, whose Filter field is 3, whose Company Prefix is 0652642, whose Item Reference is in the range 1024 ≤ *itemReference* ≤ 2047, 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, and SGLN 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`*, or *`LocationReference`*) 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. Also, in the case of the SGTIN-64, SSCC-64, and GLN-64 encodings, the tag contains a manager index which maps into a company prefix but not in a way that preserves contiguous ranges. 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.

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 ::= “!” | “'” | “(“ | “)“ | “*” | “+” | “,” | “-“
            | “.” | “:” | “;” | “=” | “_”
```

```
Escape ::= “%” HexChar HexChar
```

```
HexChar ::= Digit | “A” | “B” | “C” | “D” | “E” | “F”
          | “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. 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 UCC/EAN-128 Application Identifier 21 Serial Number according to the EAN.UCC General

Specfications. SGTIN-URIs that are derived from 64-bit and 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.

4.3.4 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 “.”)
NumericComponent
```

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

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 64-bit and 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 are derived from 64-bit and 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 | SGTINSGLNGRAITagURIBody |
SSCCGIAITagURIBody

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

GIDTagEncName ::= "gid-96"

SGTINSGLNGRAITagURIBody ::= SGTINSGLNGRAITagEncName ":"
NumericComponent "." 2*(PaddedNumericComponent ".")
NumericComponent

SGTINSGLNGRAITagEncName ::= "sgtin-96" | "sgtin-64" | "sgln-
96" | "sgln-64" | "grai-96" | "grai-64"

SSCCGIAITagURIBody ::= SSCCGIAITagEncName ":"
NumericComponent 2*("." PaddedNumericComponent)

SSCCGIAITagEncName ::= "sscc-96" | "sscc-64" | "giai-96" |
"giai-64"
```

4.3.9 Raw Tag URI

```
RawURI ::= "urn:epc:raw:" RawURIBody

RawURIBody ::= NonZeroComponent "." NumericComponent
```

4.3.10 EPC Pattern URI

```
PatURI ::= "urn:epc:pat:" PatBody

PatBody ::= GIDPatURIBody | SGTINSGLNGRAIPatURIBody |
SSCCGIAIPatURIBody

GIDPatURIBody ::= GIDTagEncName ":" 2*(PatComponent ".")
PatComponent

SGTINSGLNGRAIPatURIBody ::= SGTINSGLNGRAITagEncName ":"
PatComponent "." GS1PatBody "." PatComponent

SSCCGIAIPatURIBody ::= SSCCGIAITagEncName ":" PatComponent
"." GS1PatBody
```

```
GS1PatBody ::= "*.*" | ( PaddedNumericComponent "."
PatComponent )

PatComponent ::= NumericComponent
               | StarComponent
               | RangeComponent

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 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.SSS*

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

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

`urn:epc:id:grai:`*PPP.III.SSS*

`urn:epc:id:giai:`*PPP.SSS*

`urn:epc:tag:sgtin-64:`*FFF.PPP.III.SSS*

`urn:epc:tag:sscc-64:`*FFF.PPP.III*

`urn:epc:tag:sgln-64:`*FFF.PPP.III.SSS*

`urn:epc:tag:grai-64:`*FFF.PPP.III.SSS*

`urn:epc:tag:giai-64:`*FFF.PPP.SSS*

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

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

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

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

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

`urn:epc:tag:giai-96:`*FFF.PPP.SSS*

`urn:epc:raw:`*LLL.BBB*

```
urn:epc:pat:sgtin-64:FFFpat.PPP.IIIpat.SSSpat
urn:epc:pat:sgtin-64:FFFpat.*.*.SSSpat
urn:epc:pat:sscc-64:FFFpat.PPP.IIIpat
urn:epc:pat:sscc-64:FFFpat.*.*
urn:epc:pat:sgln-64:FFFpat.PPP.IIIpat.SSSpat
urn:epc:pat:sgln-64:FFFpat.*.*.SSSpat
urn:epc:pat:grai-64:FFFpat.PPP.IIIpat.SSSpat
urn:epc:pat:grai-64:FFFpat.*.*.SSSpat
urn:epc:pat:giai-64:FFFpat.PPP.SSSpat
urn:epc:pat:giai-64:FFFpat.*.*
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: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:grai-96:FFFpat.PPP.IIIpat.SSSpat
urn:epc:pat:grai-96:FFFpat.*.*.SSSpat
urn:epc:pat:giai-96:FFFpat.PPP.SSSpat
urn:epc:pat:giai-96:FFFpat.*.*
```

where

MMM denotes a General Manager Number

CCC denotes an Object Class number

SSS denotes a Serial Number or GIAI Individual Asset Reference

PPP denotes an EAN.UCC Company Prefix

III denotes an SGTIN Item Reference (with Indicator Digit appended to the beginning), an SSCC Shipping Container Serial Number (with the Extension (ED) digit appended at the beginning), a SGLN Location Reference, or a GRAI Asset Type.

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

XXXpat is the same as *XXX* but allowing `*` and `[lo-hi]` pattern syntax in addition

LLL denotes the number of bits of an uninterpreted bit sequence

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

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). Exception: 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 EAN or UCC. The length of `III` plus the length of `PPP` must equal 13 for SGTIN, 17 for SSCC, 12 for GLN, or 12 for GRAI.

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 encodings and vice versa.

The following procedure translates a bit-level encoding of an EPC 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. Otherwise, if the encoding scheme is SGTIN-64 or SGTIN-96, proceed to Step 2; if the encoding scheme is SSCC-64 or SSCC-96, proceed to Step 5; if the encoding scheme is SGLN-64 or SGLN-96, proceed to Step 8; if the encoding scheme is GRAI-64 or GRAI-96, proceed to Step 11; if the encoding scheme is GIAI-64 or GIAI-96, proceed to Step 14; if the encoding scheme is GID-96, proceed to Step 17.

2. Follow the decoding procedure given in Section 3.4.1.2 (for SGTIN-64) or in Section 3.4.2.2 (for SGTIN-96) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal Item Reference and Indicator $i_1i_2\ldots i_{(13-L)}$, 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 $i_1i_2\ldots i_{(13-L)}$ (handled similarly), a dot (`.`) character, and the Serial Number S as a decimal integer. 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 3.5.1.2 (for SSCC-64) or in Section 3.5.2.2 (for SSCC-96) to obtain the decimal Company Prefix $p_1p_2...p_L$, and the decimal Serial Reference $s_1s_2\ldots s_{(17-L)}$. 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 $s_1s_2\ldots s_{(17-L)}$ (handled similarly).
7. Go to Step 19.
8. Follow the decoding procedure given in Section 3.6.1.2 (for SGLN-64) or in Section 3.6.2.2 (for SGLN-96) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal Location Reference $i_1i_2\ldots i_{(12-L)}$, and the Serial Number *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, the Location Reference $i_1i_2\ldots i_{(12-L)}$ (handled similarly), a dot (`.`) character, and the Serial Number *S* as a decimal integer. 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.
10. Go to Step 19.
11. Follow the decoding procedure given in Section 3.7.1.2 (for GRAI-64) or in Section 3.7.2.2 (for GRAI-96) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal Asset Type $i_1i_2\ldots i_{(12-L)}$, 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 $p_1p_2...p_L$ where each digit (including any leading zeros) becomes the corresponding ASCII digit character, a dot (`.`) character, the Asset Type $i_1i_2\ldots i_{(12-L)}$ (handled similarly), a dot (`.`) character, and the Serial Number *S* as a decimal integer. 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.
13. Go to Step 19.
14. Follow the decoding procedure given in Section 3.8.1.2 (for GIAI-64) or in Section 3.8.2.2 (for GIAI-96) to obtain the decimal Company Prefix $p_1p_2...p_L$, 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 $p_1p_2...p_L$ 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. 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 3.3.1.2 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.

The following procedure translates a bit-level tag encoding into either an EPC Tag URI or a Raw Tag 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 encoding scheme is SGTIN-64 or SGTIN-96, proceed to Step 2; if the encoding scheme is SSCC-64 or SSCC-96, proceed to Step 5; if the encoding scheme is SGLN-64 or SGLN-96, proceed to Step 8; if the encoding scheme is GRAI-64 or GRAI-96, proceed to Step 11, if the encoding scheme is GIAI-64 or GIAI-96, 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 3.4.1.2 (for SGTIN-64) or in Section 3.4.2.2 (for SGTIN-96) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal Item Reference and Indicator $i_1i_2…i_{(13-L)}$, 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-64` or `sgtin-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, 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. 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 3.5.1.2 (for SSCC-64) or in Section 3.5.2.2 (for SSCC-96) to obtain the decimal Company Prefix $p_1p_2...p_L$, and the decimal Serial Reference $i_1i_2…s_{(17-L)}$, 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-64` or `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 3.6.1.2 (for SGLN-64) or in Section 3.6.2.2 (for SGLN-96) 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 Serial Number 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-64` or `sgln-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, the Location Reference $i_1i_2 \ldots i_{(12-L)}$ (handled similarly), a dot (`.`) character, and the Serial Number S as a decimal integer. 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.

10. Go to Step 21.

11. Follow the decoding procedure given in Section 3.7.1.2 (for GRAI-64) or in Section 3.7.2.2 (for GRAI-96) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal Asset Type $i_1i_2 \ldots i_{(12-L)}$, 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-64` or `grai-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, the Asset Type $s_1s_2 \ldots s_{(12-L)}$ (handled similarly), a dot (`.`) character, and the Serial Number $d_{15}d_2 \ldots d_K$ as a decimal integer. 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.

13. Got to Step 21.

14. Follow the decoding procedure given in Section 3.8.1.2 (for GIAI-64) or in Section 3.8.2.2 (for GIAI-96) to obtain the decimal Company Prefix $p_1p_2...p_L$, the decimal 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-64` or `giai-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 Individual Asset Reference $i_1i_2\ldots i_J$ (handled similarly). The portion corresponding to the Filter Value 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 3.3.1.2 to obtain the EPC 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 encoding, therefore create an EPC Raw URI by concatenating the following: the string `urn:epc:raw:`, the length of the bit string, a dot (`.`) character, and the value of the bit string considered as a single decimal integer. Both the length and the value must have no leading zeros, except if the value is itself zero in which case a single zero character is used.

21. The translation is now complete.

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

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:`), or an EPC Pattern URI (`urn:epc:pat:`), the URI cannot be translated into a bit-level EPC.

2. If the URI is a Raw Tag URI (`urn:epc:raw:`), create the bit-level EPC by converting the second component of the Raw Tag URI into a binary integer, whose length is equal to the first component 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-level EPC.

3. If the URI is an EPC Tag URI (`urn:epc:tag:`*`encName`*`:`), parse the URI using the grammar for `TagURI` as given in Section 4.3.8. If the URI cannot be parsed using this grammar, stop: the URI is illegal and cannot be translated into a bit-level EPC. Otherwise, if *`encName`* is `sgtin-96` or `sgtin-64` go to Step 4,

if *encName* is `sscc-96` or `sscc-64` go to Step 9, if *encName* is `sgln-96` or `sgln-64` go to Step 13, if *encName* is `grai-96` or `grai-64` go to Step 18, if *encName* is `giai-96` or `giai-64` go to Step 22, or if *encName* is `gid-96` go to Step 26.

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 3.4.1.1 (SGTIN-64) or Section 3.4.2.1 (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 an EPC tag.
8. Go to Step 31.
9. Let the URI be written as
   `urn:epc:tag:`*encName*`:`$f_1f_2\ldots f_F.p_1p_2\ldots p_L.i_1i_2\ldots i_{(17-L)}$.
10. Interpret $f_1f_2\ldots f_F$ as a decimal integer *F*.
11. Carry out the encoding procedure defined in Section 3.5.1.1 (SSCC-64) or Section 3.5.2.1 (SSCC-96), using $i_1p_1p_2\ldots p_Li_2i_3\ldots i_{(17-L)}0$ as the EAN.UCC SSCC, L as the length of the EAN.UCC company prefix, and *F* from Step 10 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 an EPC tag.
12. Go to Step 31.
13. Let the URI be written as
   `urn:epc:tag:`*encName*`:`$f_1f_2\ldots f_F.p_1p_2\ldots p_L.i_1i_2\ldots i_{(12-L)}.s_1s_2\ldots s_S$.
14. Interpret $f_1f_2\ldots f_F$ as a decimal integer *F*.
15. Interpret $s_1s_2\ldots s_S$ as a decimal integer *S*.
16. Carry out the encoding procedure defined in Section 3.6.1.1 (SGLN-64) or Section 3.6.2.1 (SGLN-96), using $p_1p_2\ldots p_Li_1i_2\ldots i_{(12-L)}0$ 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 14 as the Filter Value, and *S* from Step 15 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 an EPC tag.
17. Go to Step 31.

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. Carry out the encoding procedure defined in Section 3.7.1.1 (GRAI-64) or Section 3.7.2.1 (GRAI-96), using $0p_1p_2…p_Li_1i_2…i_{(12-L)}0s_1s_2…s_S$ 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 19 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 an EPC tag.

21. Go to Step 31.

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

23. Interpret $f_1f_2…f_F$ as a decimal integer *F*.

24. Carry out the encoding procedure defined in Section 3.8.1.1 (GIAI-64) or Section 3.8.2.1 (GIAI-96), using $p_1p_2…p_Ls_1s_2…s_S$ as the EAN.UCC GIAI, L as the length of the EAN.UCC company prefix, and *F* from Step 23 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 an EPC tag.

25. Go to Step 31.

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

27. Interpret $m_1m_2…m_L$ as a decimal integer *M*.

28. Interpret $c_1c_2…c_K$ as a decimal integer *C*.

29. Interpret $s_1s_2…s_S$ as a decimal integer *S*.

30. Carry out the encoding procedure defined in Section 3.3.1.1 using *M* from Step 27 as the General Manager Number, *C* from Step 28 as the Object Class, and *S* from Step 29 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 an EPC tag.

31. The translation is complete.

6 Semantics of EPC Pattern URIs

The meaning of an EPC Pattern URI (`urn:epc:pat:`) can be formally defined as denoting a set of encoding-specific EPCs. 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 `RangeComponent [`*`lo-hi`*`]`, and $lo \le Ci \le hi$; 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 $1 \le i \le n$.

7 Background Information

This document represents the contributions of many people, especially the contributions of the Tag Data Standards Group and the URI Representation Group.

EPC Tag Data Standards Group

Bud Babcock	Procter & Gamble
Jason Carney	Ahold
Hal Charych	Symbol
Chris Cummins	Uniform Code Council
Falk Gernot	Metro
Yuichiro Hanawa	Mitsui, USA
Mark Harrison	Cambridge Auto-ID Lab
Larry Hilgert	Pepsico (Quaker)
André Frank	Sara Lee/DE
Jonathan Loretto	Cap Gemini Ernst & Young
Ron Moser	Wal-Mart
Don Mowery	Nestle
Steve Morris	Printronix
Bob Mytkowicz	Gillette
Doug Naal	Kraft
Juli Nackers	Kimberly-Clark
Robert Nonneman	UPS

Richard Probst	Nominum
Steve Rehling	Procter & Gamble
Rick Schendel	Target
Sylvia Stein	EAN Netherlands
Richard Swan	T3C1
Michael Szafranski	Kraft
Nancy Tai	Georgia Pacific
Ken Traub	Connecterra, Inc.

URI Representation Group

Dipan Anarkat	EPCglobal
Michael Mealling	VeriSign
Ken Traub	Connecterra, Inc.

This document also 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)

8 References

[EANUCCGS] "General EAN.UCC Specifications." Version 5.0, EAN International and the Uniform Code Council, Inc™, January 2004.

[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.

9 Appendix A: Encoding Scheme Summary Tables

SGTIN Summary

SGTIN-64	Header	Filter Value	Company Prefix Index	Item Reference	Serial Number
	2 bits	3 bits	14 bits	20 bits	25 bits
	10 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	9 - 1,048,575 (Decimal capacity*)	33,554,431 (Decimal capacity)

SGTIN-96	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
	8	3	3	20-40	24 - 4	38
	0011 0000 (Binary value)	8 (Decimal capacity)	8 (Decimal capacity)	999,999 – 999,999,999,999 (Decimal capacity**)	9,999,999 – 9 (Decimal capacity**)	274,877,906,943 (Decimal capacity)

Filter Values

(Non-normative)

Type	Binary Value
Unspecified	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		Item Reference and Indicator Digit	
	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

*Capacity of Item Reference field varies with the length of the Company Prefix
**Capacity of Company Prefix and Item Reference fields vary according to the contents of the Partition field.

SSCC Summary

SSCC-64	Header	Filter Value	Company Prefix Index	Serial Reference
	8	3	14	39
	0000 1000 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	99,999 - 99,999,999,999 (Decimal capacity*)

SSCC-96	Header	Filter Value	Partition	Company Prefix	Serial Reference	Unallocated
	8	3	3	20-40	38-18	24
	0011 0001 (Binary value)	8 (Decimal capacity)	8 (Decimal capacity)	999,999 – 999,999,999,999 (Decimal capacity**)	99,999,999,999 – 99,999 (Decimal capacity**)	[Not Used]

Filter Values (Non-normative)		SSCC Partition Table				
Type	**Binary Value**	**Partition Value**	**Company Prefix**		**Serial Reference and extension digit**	
Unspecified	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

*Capacity of Serial Reference field varies with the length of the Company Prefix
**Capacity of Company Prefix and Serial Reference fields vary according to the contents of the Partition field.

SGLN Summary					
SGLN-64	**Header**	**Filter Value**	**Company Prefix Index**	**Location Reference**	**Serial Number**
	8	3	14	20	19
	0000 1001 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	999,999 - 0 (Decimal capacity*)	524,288 (Decimal capacity) [Not Used]

SGLN-96	**Header**	**Filter Value**	**Partition**	**Company Prefix**	**Location Reference**	**Serial Number**
	8	3	3	20-40	21-1	41
	0011 0010 (Binary value)	8 (Decimal capacity)	8 (Decimal capacity)	999,999 – 999,999,999,999 (Decimal capacity**)	999,999 – 0 (Decimal capacity**)	2,199,023,255,552 (Decimal capacity) [Not Used]

Filter Values

(Non-normative)

Type	**Binary Value**
Unspecified	000
Reserved	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

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

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

GRAI Summary

GRAI-64	Header	Filter Value	Company Prefix Index	Asset Type	Serial Number
	8	3	14	20	19
	0000 1010 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	999,999 - 0 (Decimal capacity*)	524,288 (Decimal capacity)

GRAI-96	Header	Filter Value	Partition	Company Prefix	Asset Type	Serial Number
	8	3	3	20-40	24 – 4	38
	0011 0011 (Binary value)	8 (Decimal capacity)	8 (Decimal capacity)	999,999 – 999,999,999,999 (Decimal capacity**)	999,999 – 0 (Decimal capacity**)	274,877,906,943 (Decimal capacity)

Filter Values (Non-normative)

Type	Binary Value
Unspecified	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

*Capacity of Asset Type field varies with Company Prefix.

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

GIAI Summary

GIAI-64	Header	Filter Value	Company Prefix Index	Individual Asset Reference
	8	3	14	39
	0000 1011 (Binary value)	8 (Decimal capacity)	16,383 (Decimal capacity)	549,755,813,888 (Decimal capacity)

GIAI-96	Header	Filter Value	Partition	Company Prefix	Individual Asset Reference
	8	3	3	20-40	62-42
	0011 0100 (Binary value)	8 (Decimal capacity)	8 (Decimal capacity)	999,999 – 999,999,999,999 (Decimal capacity*)	4,611,686,018,427,387,904 - 4,398,046,511,103 (Decimal capacity*)

Filter Values

(To be confirmed)

Type	Binary Value
Unspecified	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
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

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

10 Appendix B: EPC Header Values and Tag Identity Lengths

With regards to tag identity lengths and EPC Header values: In the decoding process of a single tag: Having knowledge of the identifier length during the signal decoding process of the reader enables the reader to know when to stop trying to decode bit values. Knowing when to stop enables the readers to be more efficient in reading speed. For example, if the same Header value is used at 64 and 96 bits, the reader, upon finding that header value, must try to decode 96 bits. After decoding 96 bits, the reader must check the CRC (Cyclic Redundancy Check error check code) against both the 64-bit and 96-bit numbers it has decoded. If both error checks fail, the numbers are thrown away and the tag reread. If one of the numbers passes the error check, then that is reported as the valid number. Note that there is a non-zero, i.e., greater than zero but very small, probability that an erroneous number can be reported in this process. If both numbers pass the error check, then there is a problem. Note that there is a small probability that both a 64 bit EPC and 96-bit EPC whose first 64 bits are the same as the 64-bit EPC will have the same CRC. Other measures would have to be taken to determine which of the two numbers is valid (and perhaps both are). All of this slows down the reading process and introduces potential errors in identified numbers (erroneous numbers may be reported) and non-identified numbers (tags may be unread due to some of the above). These problems are primarily evident while reading weakly replying tags, which are often the tags furthest from the reader antenna and in noisy environments. Encoding the length within the Header eliminates virtually all of the error probabilities above and those that remain are reduced significantly in probability.

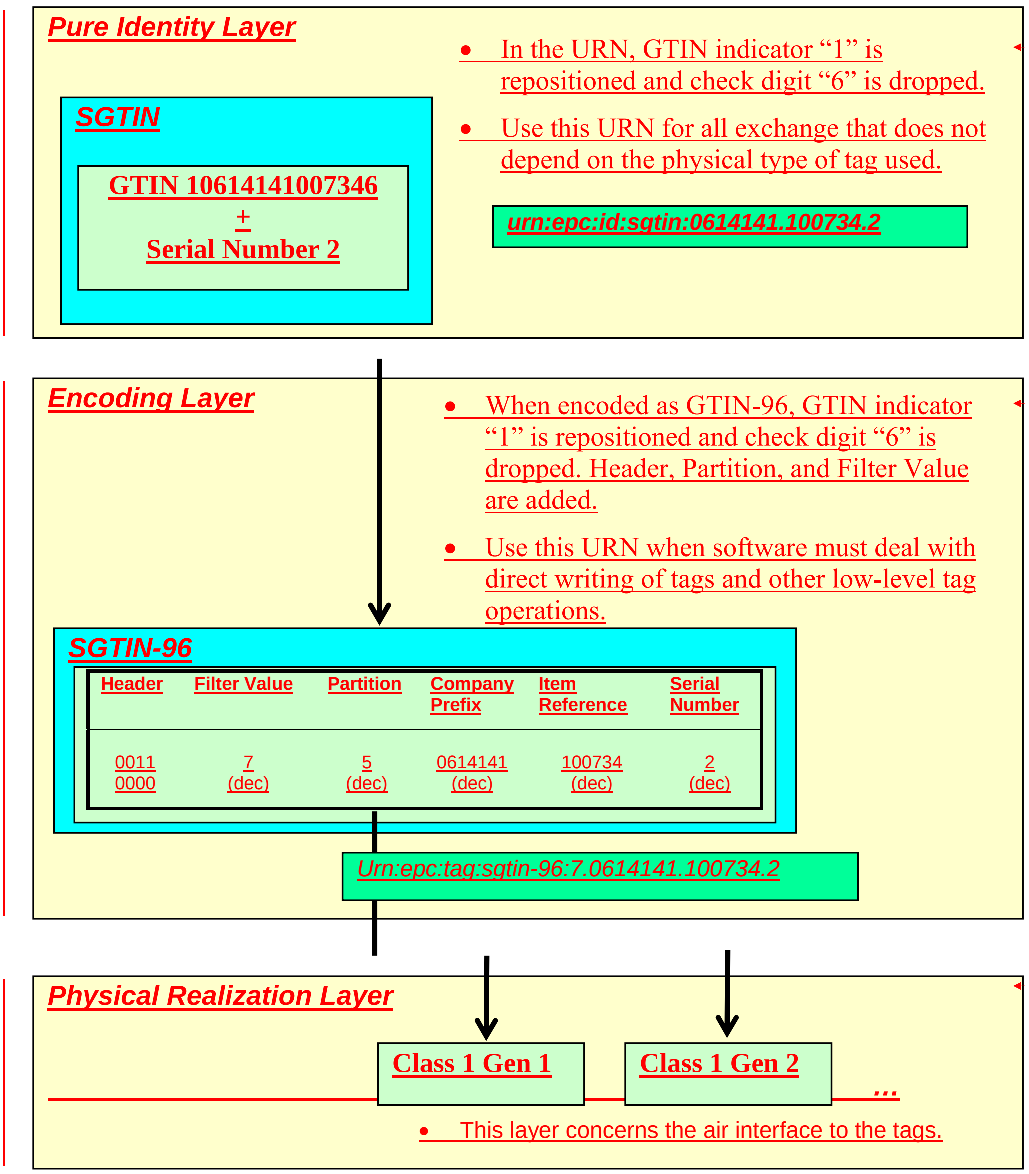
In the decoding process of multiple tags responding: When multiple tags respond at the same time their communications will overlap in time. Tags of the same length overlap almost completely bit for bit when the same reader controls them. Tags of different lengths will overlap almost completely over the first bits, but the longer tag will continue communicating after the shorter tag has stopped. Tags of very strong communication strength will mask tags responding with much weaker strength. The reader can use communication signal strength as a determiner of when to stop looking to decode bits. Tags of almost equal communication strength will tend to interfere almost completely with one another over the first bits before the shorter tag stops. The reader can usually detect these collisions, but not always when weak signals are trying to be pulled out of noise, as is the case for the distant tags. When the tags reply with close, but not equal strength, it may be possible to decode the stronger signal. When the short tag has the stronger signal, it may be possible to decode the weaker longer tag signal without being able to definitively say that a second tag is responding due to changes in signal strength. These problems are primarily evident in weakly replying tags. Encoding the length in the Header enables the reader to know when to stop pulling out the numbers, which enables it to more efficiently determine the validity of the numbers.

In the identification process: The reader can "select" what length tags it wishes to communicate with. This eliminates the decoding problems encountered above, since all communicating tags are of the same length and the reader knows what that length is a priori. For efficiency reasons, a single selection for a length is preferred, but two can be workable. More than two becomes very inefficient.

The net effect of encoding the length within the Header is to reduce the probabilities of error in the decoding process and to increase the efficiency of the identification process.

11 Appendix C: Example of a Specific Trade Item (SGTIN)

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.



Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

(01) 1 0614141 00734 6 (21) 2

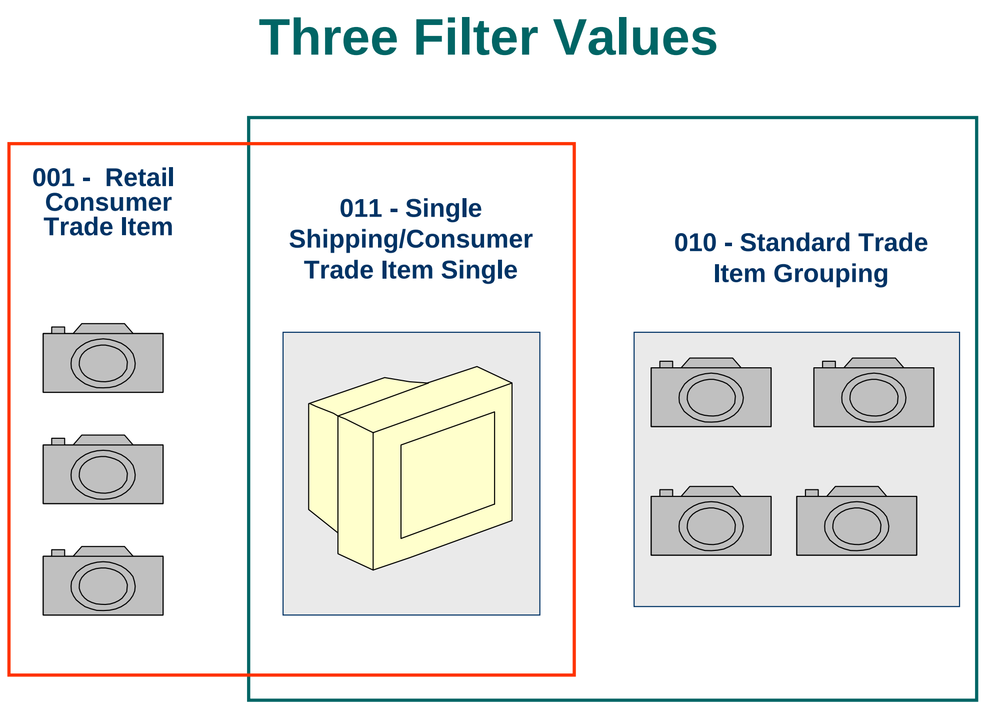
	Header	Filter Value	Partition	Company Prefix	Item Reference	Serial Number
SGTIN-96	8 bits	3 bits	3 bits	24 bits	20 bits	38 bits
	0011 0000 (Binary value)	7 (Decimal value)	5 (Decimal value)	0614141 (Decimal value)	100734 (Decimal value)	2 (Decimal value)

- (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 was not defined when this example was created , so 7 is a notional value.
- 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: Binary Digit Capacity Tables

Length in Binary Digits	Decimal Capacity	Length in Binary Digits	Decimal Capacity
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
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

(Section 3.0 Definition of Element Strings and Section 3.7 EPCglobal Tag Data Standard.)

This section provides EAN.UCC 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 EPC 64-bit tags, the following applies:

- 64-bit tag application is limited to 16,383 EAN.UCC Company Prefixes and therefore EAN.UCC EPCglobal implementation strategies will focus on tag capacity that can accommodate all EAN.UCC member companies. The 64-bit tag will be approved for use by EAN.UCC member companies with the restrictions that follow:
- **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 33,554,431 numeric-only serial numbers.
- **AI (41n) GLN + AI (21) Serial Number:** The Tag Data Standard V1.1 R1.23 is approved with a complete restriction on GLN serialization because this question has not been resolved by GSMP at this time.
- **AI (8003) GRAI Serial Number:** The Section 3.6.49 Global Returnable Asset Identifier definition is restricted to permit assignment of 524,288 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 549,755,813,888 numeric-only serial numbers.

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 (41n) GLN + AI (21) Serial Number:** The Tag Data Standard V1.1 R1.23 is approved with a complete restriction on GLN serialization because this question has not been resolved by GSMP at this time.
- **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.

