EPC Identifiers for aerospace

Mark Harrison,
Auto-ID Lab, University of Cambridge, UK

Report Abstract: This paper is a technical report developed in conjunction with the ATA 'RFID on Parts' work group to express end-user requirements and propose a technical solution, aligned with existing EPCglobal Tag Data Standards.

Preface

There is now a growing critical mass of companies who are adopting the Electronic Product Code (EPC) technology for large-scale deployment of Radio-Frequency Identification (RFID) technology providing automated input into business information systems in order to improve the efficiency of business processes.

The early vision for the EPC Network architecture was developed by the Auto-ID Centre during 1999-2003, an industrially funded project involving academics at (then) six leading research institutes around the world, together with over 100 major industrial sponsors, consisting of both end-users and technology solution providers.

Since the transition of the Auto-ID Centre into Auto-ID Labs and EPCglobal in November 2003, the number of companies involved in this effort has increased almost ten-fold in only three years and the activities are reaching out to other industry sectors beyond the initial focus on fast-moving consumer goods and the retail sector. The EPC system is already being used by the US Department of Defense – and is also being developed for the healthcare and life sciences sector, to enable item-level identification of pharmaceuticals and support electronic pedigrees to try to eliminate counterfeits.

On the technical side, the original vision for the EPC Network architecture has been substantially overhauled, focusing primarily on defining standard interfaces at various layers of the architecture, ranging from the RFID air interface, to the low-level software interfaces to readers (Reader Protocol, Reader Management), access to filtered data (Application Level Events, ALE) and to information services (EPC Information Services, EPCIS). The Object Name Service is operational – and further developments of serial-level lookup services (Discovery Services) are underway. The collaborative standards development process involving both technology solution providers and end users has resulted in most of the standard interfaces required for multi-vendor interoperability being ratified as freely available open standards, which can be downloaded from the website of EPCglobal.

The aerospace sector already have a system of unique identifiers for marking aircraft parts. These unique identifiers used the CAGE (Commercial and Government Entity) code to identify the manufacturer or supplier, together with a serial number that is unique within the CAGE code. This is the basis for the Air Transport Association (ATA) Spec 2000 identifier, which is widely used for tracking parts and their life history. A number of organizations still use an older method of mass serializing, consisting of the CAGE code, an Original Part Number and a Serialized Part Sequence Number. This 3-part unique identifier is also known as UID Construct 2.

Traditionally, part marking on aircraft parts has been done via handwritten or embossed identifiers on name plates, then by the use of linear barcodes. The current activity within the industry is looking at using RFID tags to store not only a unique identifier, but also some additional data about the history of the part.

Given the likelihood of mass-adoption of EPC technology, it may be of significant benefit to the aerospace sector if they are able to express their existing identifiers in an EPC-compatible representation, so that they can make use of EPC-compliant readers, middleware and information systems that is being developed for multiple industry sectors and thereby use commoditized solutions rather than

requiring a bespoke solution for the aerospace sector. The ATA 'RFID on Parts' working group is currently in the process of defining standards for the aerospace sector about how to store this data on an RFID tag. Members of Auto-ID Labs within the Aerospace ID technologies programme have been working closely with this forum and this report is one of the results of that collaboration.

Beginning with a clear statement of the user requirements for wanting an EPC representation, while aligning with their existing unique identifiers that are already well entrenched, the remainder of the report provides a technical proposal about how the ATA Spec 2000 identifier and the UID construct 2 can be mapped into EPC representations, suitable for encoding the identifier on the RFID tag (compact binary format), for use with filtering middleware and information systems (URN formats).

This report will be submitted via Boeing (a subscriber of EPCglobal) to the EPCglobal Aerospace & Defense business action group, and subject to their endorsement, will be passed to the EPCglobal Tag Data Translation and Standards work group for their consideration about how it may be incorporated within a future revision of the EPCglobal Tag Data Standards.

In order to minimise the amount of additional technical work that needs to be done, we have taken care to align the technical sections as closely as possible with the format, terminology and methodology used in the existing Tag Data Standard.

We have already prepared two draft 'definition files' for EPCglobal Tag Data Translation, which will allow existing readers, integrated label printers, middleware etc., which uses the Tag Data Translation standard to automatically by able to handle the unique identifiers for the aerospace sector, with an upgrade as simple as reading in the new definition files that define the encoding and decoding rules in a machine-readable manner.

Furthermore, as soon as an EPC representation has been approved, we intend to provide free of charge a Tag Data Translation software library and software application to the aerospace community, to assist them with the translation between their existing notation for identifiers and the URN and binary representations that are used in the EPC system.

Acknowledgment:

We wish to thank members of the ATA RFID on Parts work group for useful discussions and in particular, the efforts of Joyce Polkinghorne (American Airlines) for further development and strengthening of the section on user requirements.

Contents:

User Requirements

Purpose

The purpose of this paper is to provide the background information and justification for the allocation of two EPC header codes for use by the aerospace industry. This will provide aerospace with a globally unique ID system for RFID tags which can integrate seamlessly with existing information systems, EDI messages and airworthiness certificates without the need to do an additional identifier lookup.

Background

The aerospace industry identifies organizations by CAGE code – a five character alphanumeric code that is assigned by the US Department of Defense (DoD), NATO and other governments. The CAGE code has been an integral part of communication within the airline industry for many decades to do procurement of aircraft parts, order repairs, and to designate design/manufacturing authority, etc. The marking of parts as unique individual entities has been another predominant use of the CAGE code in aviation.

In 1992 the Air Transport Association approved standards to formalize the use of the CAGE code in aircraft part marking practices. To give a part a unique individual identity which will never change during the life of the part, the CAGE code is combined with a serial number. This serial number is assigned by the manufacturer and unique within their CAGE code. This is often referred to as the ATA SPEC2000 identifier; however, for the purposes of this document it will be called Construct 1.

While most organizations within the aerospace industry can utilize the Construct 1 schema, there are a few companies who cannot. These companies serialize within part number, rather than at the higher level of entity wide CAGE code. To accommodate these companies and have them able to participate in unique part marking in a standardized manner, the ATA defined a secondary schema of permanent identification comprised of CAGE code, original part number and sequence number. In this document this schema will be called Construct 2.

Both Construct 1 and Construct 2 are comprised of data elements defined in the ATA Common Support Dictionary (CSDD) and are used in other applications and mediums of communication as aerospace companies do business.

The formats of the constructs are documented in the SPEC2000 International Specifications, Chapter 9. This body of standards for business practices and communication within the aerospace industry is governed and maintained by the ATA, and used by companies worldwide.

Requirements

New technology should be the servant of industry, enabling it to function more adroitly and economically. Where there are established efficiencies, those should not be lost with the move to new technology. Within the aerospace industry the use of strictly defined common data elements is a long established practice, encompassing many applications including the unique part ID schemas, Construct 1 and Construct 2. The new technology, in this case RFID, should enhance these current best practices. Therefore, the aerospace industry is best served by translating the established Unique ID schemas for parts into compatible codes for RFID.

Request

The aerospace industry recognizes the importance of the open standards for networked RFID which are being developed within the EPCglobal community – and the benefits in terms of interoperability, end-user choice of vendors and eventual lowering of costs (commoditization) of being able to use hardware and software which is certified to be compliant with the relevant EPCglobal standards. With this in mind, the aerospace industry is seeking an EPC representation of its existing well-established UID constructs 1 and 2.

To support the co-existence of both UID Construct 1 and Construct 2, the aerospace industry, represented by the ATA RFID on parts working group, requests that EPCglobal allocates at its earliest convenience:

- One 8-bit EPC header code for UID Construct 1
- One 8-bit EPC header code for UID Construct 2

The technical sections in the remainder of this document propose how the UID Constructs 1 and 2 may be represented as EPC identifiers, both in terms of URN notation, URN patterns for EPCs and binary representation for encoding on RFID tags.

The encoding and decoding rules in the technical sections that follow have been written using the same terminology and notation as found in the EPCglobal Tag Data Standard v1.3, in order to facilitate inclusion within a future revision of the EPCglobal Tag Data Standard that includes the new header values requested above, as well as to enable the ATA to harmonize with EPCglobal when the ATA is ready to include these details within a future revision of the ATA Spec 2000 standard.

ATA Spec 2000 UID Construct 1 (ATA1)

The ATA Spec 2000 identifier (UID Construct 1) is defined in Chapter 9 of the ATA e-Business Specifications.

In addition to a Header, the EPC representation, ATA1-132 is composed of three fields: the *Filter Value*, *CAGE code*, and *Serial Number*, as shown in Table 1.

	Header	*Filter Value*	*CAGE code (CAG, MFR or SPL)*	*Serial Number (SER)*
ata1-132	8	4	30	90
	xxxxxxxx (to be defined)	(Refer to table 2 for values)	5 characters (uppercase alphanumeric only)	Up to 15 characters (uppercase alphanumeric and hyphen)

Table 1. The 132-bit binary EPC representation for the ATA Spec 2000 identifier (UID Construct 1) including the header, and allowed character ranges for CAGE code and Serial Number.

• *Header* is 8-bits, with a binary value of xxxxxxxx (to be defined by EPCglobal)
• *Filter Value* is not part of the ATA Spec 2000 identifier, but is used for fast filtering and pre-selection of basic logistics types. The Filter Values are specified in Table 2.

Type	**Binary Value**
Pallet	0000
Case	0001
UID item	0010
Safety equipment	0011
Reserved for future use	all other values

Table 2. Filter Values for use with the ATA Spec 2000 identifier

• *CAGE code* contains a literal embedding of the CAGE code using 6-bit ASCII compaction.

• *Serial Number* is a unique string serial code for each instance. Leading zeros on the Serial Number are not significant.

ATA1-132 Encoding Procedure

The following procedure creates an ATA1-132 encoding.

Given:

- A 5-character CAGE code C consisting of uppercase alphanumeric characters $c_1c_2c_3c_4c_5$

- A Serial Number S up to 15 characters in length, consisting of any combination of the digits 0-9, the uppercase letters A-Z and the hyphen character

- A Filter Value F where $0 \leq F < 16$

Procedure:

1. Check that each of the characters $c_1c_2c_3c_4c_5$ of the CAGE code C is one of the characters listed in the table in Appendix H. If this is not the case, stop: this character string cannot be encoded as an ATA1-132. Otherwise, construct the CAGE code by concatenating the 6-bit code, as given in Appendix H, for each of the characters $c_1c_2c_3c_4c_5$, yielding 6*5 = 30 bits total. This is the binary representation of the CAGE code, denoted c.

2. Check that each of the characters $s_1s_2\ldots s_K$ of the Serial Number S is one of the characters listed in the table in Appendix H. If this is not the case, stop: this character string cannot be encoded as an ATA1-132. Otherwise, construct the Serial Number by concatenating the 6-bit code, as given in Appendix H, for each of the characters $s_1s_2\ldots s_K$, yielding 6*K bits. If K < 15, concatenate additional zero bits to the right to make a total of 90 bits. This is the binary representation of the serial number, denoted s.

3. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header xxxxxxxx (8 bits), Filter Value *F* (4 bits), CAGE code *c* from Step 1 (*30* bits), Serial Number *s* from Step 2 (*90* bits)

ATA1-132 Decoding Procedure

Given:

- An ATA1-132 as a 132-bit bit string xxxxxxxx$b_{123}b_{122}\ldots b_0$ (where the first eight bits xxxxxxxx are the header)

Yields:

- A 5-character CAGE code
- A Serial Number up to 15 characters
- A Filter Value

Procedure:

1. Bits $b_{123}b_{122}b_{121}b_{120}$, considered as an unsigned integer, are the Filter Value.

2. Divide the bits $b_{119}b_{118}\ldots b_{91}b_{90}$ into five 6-bit segments. The result should consist of five non-zero binary segments. Lookup each of the 6-bit segments in Appendix H to obtain a corresponding character. If any of the 6-bit segments has a value that is not in Appendix H, stop: this bit string cannot be decoded as an ATA1-132. The five characters so obtained, considered as a character string c1c2…ck, is the value of the CAGE code.

3. Divide the remaining bits $b_{89}b_{88}\ldots b_1b_0$ into fifteen 6-bit segments. The result should consist of K non-zero binary segments followed by 15-K binary zero segments, where 0< K <=15). If this is not the case, stop; this bit string cannot be decoded as an ATA1-132. Otherwise, lookup each of the non-zero 6-bit segments in Appendix H to obtain a corresponding character. If any of the non-zero 6-bit segments has a value that is not in Appendix H, stop: this bit string cannot be decoded as an ATA1-132.
The K characters so obtained, considered as a character string $s_1s_2\ldots s_k$, is the value of the Serial Number.

URN representations for ATA UID Construct 1

Pure-identity URI

For the ATA Spec 2000 identifier (UID construct 1), the pure identity URI representation is as follows:

`urn:epc:id:ata1:`*`CAGECode`*`.`*`serialNumber`*

where *`CAGECode`* is the five-character CAGE code and *`serialNumber`* is the serial number represented as a string of between 1 and 15 characters (consisting exclusively of any of the upper-case alphanumeric characters and the hyphen symbol '–').

Tag-encoding URI

For the ATA Spec 2000 identifier (UID construct 1), the tag-encoding URI representation is as follows:

`urn:epc:tag:ata1-132:`*`filter`*`.`*`CAGECode`*`.`*`serialNumber`*

where filter is the filter value represented as either one or two decimal digits, *`CAGECode`* is the five-character CAGE code and *`serialNumber`* is the serial number represented as a string of between 1 and 15 characters (consisting exclusively of any of the upper-case alphanumeric characters and the hyphen symbol '–').

Pattern URI

The pattern URI for the ATA Spec 2000 identifier (UID construct 1) is as follows:

`urn:epc:pat:`*`tagType`*`:`*`filterPat`*`.`*`CAGECodePat`*`.`*`serialNumberPat`*

where *`tagType`* is `ata1-132`, *`filterPat`* is either a filter value, a range of the form [*lo-hi*], or a * character; *`CAGECodePat`* is either a CAGE Code or a * character; and *`serialNumberPat`* is either a serial number or a * character.

Pure Identity Pattern URI

The pure identity pattern URI for the ATA Spec 2000 identifier (UID Construct 1) is as follows:

`urn:epc:idpat:ata1:`*`CAGECodePat`*`.`*`serialNumberPat`*

where *`CAGECodePat`* is either a CAGE Code or a * character and *`serialNumberPat`* is either a serial number or a * character with the proviso that `CAGECodePat` shall not be a * character if `serialNumberPat` is not a * character.

Syntax for ATA UID Construct 1

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

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


UpperAlphanumeric :: = Digit | UpperAlpha
UpperAlphanumericAndHyphen :: = Digit | UpperAlpha | Hyphen

StarComponent ::= “*”
RangeComponent ::= “[” NumericComponent “-”
                   NumericComponent “]”

ATATagType ::= “ata1-132”
ATAFilter ::= NumericComponent
CAGECode ::= CAGECodeChar*5
ATASerialNumber ::= UpperAlphanumericAndHyphen+
ATAFilterPat ::= PatComponent
CAGECodePat ::= CAGECode | StarComponent
ATASerialNumberPat ::= ATASerialNumber | StarComponent
```

Pure-identity URI

```
ATA1-URI ::= “urn:epc:id:ata1:” CAGECode “.” ATASerialNumber
```

Tag-encoding URI

```
ATA1TagURI ::= “urn:epc:tag:” ATA1TagType “:” ATAFilter “.”
CAGECode “.” ATASerialNumber
```

EPC Pattern URI

```
ATAPatURI ::= “urn:epc:pat:” ATA1TagType “:” ATAFilterPat “.”
CAGECodePat “.” ATASerialNumberPat
```

EPC Identity Pattern URI

```
ATAIDPatURI ::= “urn:epc:idpat:ata1:” ATAIDPatMain
ATAIDPatMain ::= CAGECode “.” ATASerialNumber
| CAGECode “.*” | “*.*”
```

Summary (non-normative) for ATA UID Construct 1

The syntax rules above can be summarized informally as follows:

`urn:epc:id:ata1:`*`TTT`*`.`*`BBB`*

`urn:epc:tag:ata1-132:`*`FFF`*`.`*`TTT`*`.`*`BBB`*

`urn:epc:idpat:ata1:`*`TTT`*`.`*`BBB`*

`urn:epc:pat:ata1-132:`*`FFFpat`*`.`*`TTT`*`.`*`BBBpat`*

`urn:epc:pat:ata1-132:`*`FFFpat`*`.*.`*`BBBpat`*

where

`BBB` denotes an alphanumeric Serial Number (including the hyphen)
`TTT` denotes a CAGE code assigned by the US DoD or NATO
`FFF` denotes a filter code as used by the SGTIN, SSCC, SGLN, GRAI, GIAI, DoD and ATA tag encodings
`BBBpat` denotes an alphanumeric Serial Number (including the hyphen) or `*`
`TTTpat` denotes a CAGE code (assigned by the US DoD or NATO) or `*`
`FFFpat` denotes a filter code as used by the SGTIN, SSCC, SGLN, GRAI, GIAI, DoD and ATA tag encodings but allowing `*` and `[lo-hi]` pattern syntax in addition

Translation between EPC-URI and Other EPC Representations

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

Bit string into EPC-URI (pure identity)

Create an EPC-URI by concatenating the following: the string urn:epc:id:ata1:, the CAGE code $c_1c_2c_3c_4c_5$, a dot (.) character and the Serial Number.
The Serial Number should 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.

Bit string into tag EPC-URI (tag-encoding)

Create an EPC tag URI by concatenating the following: the string `urn:epc:tag:`, the encoding scheme (`ata1-132`), the colon character (`:`), the Filter Value F as a decimal integer, a dot (`.`) character, the CAGE code $c_1c_2c_3c_4c_5$, a dot (`.`) character and the Serial Number.
The Serial Number should 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.

URI into Bit String

The following procedure translates a URI into a bit string:

1. If the URI is an ATA1-URI (`urn:epc:id:ata1:`), stop: the URI cannot be translated into a bit string.

2. If the URI is an EPC Tag URI (`urn:epc:tag:`*`encName`*`:`), parse the URI using the grammar for TagURI as given on p5.
If the URI cannot be parsed using these grammars, stop: the URI is illegal and cannot be translated into a bit string.
3. If *encName* is ata1-132, let the URI be written as
`urn:epc:tag:`*`encName`*`:`$f_1f_2\ldots f_F$`.`$c_1c_2\ldots c_5$`.`$s_1s_2\ldots s_K$
4. Interpret $f_1f_2\ldots f_F$ as a decimal integer filter value F
5. Interpret $c_1c_2\ldots c_5$ as the 5-character CAGE code C
6. Interpret $s_1s_2\ldots s_K$ as a string serial number S
7. Carry out the encoding procedure defined on p8.

ATA UID Construct 2 (ATA2)

In addition to a Header, the EPC representation, ATA2-222 is composed of four fields: the *Filter Value*, *CAGE code*, *Original Part Number* and *Sequential Part Serial Number*, as shown in Table 1.

	Header	*Filter Value*	*CAGE code (CAG, MFR or SPL)*	*Original Part Number (PNO)*	*Sequential Part Serial Number (SEQ)*
ata2-222	8	4	30	90	90
	yyyyyyyy (to be defined)	(Refer to table 2 for values)	5 characters (uppercase alphanumeric only)	Up to 15 characters (uppercase alphanumeric and hyphen)	Up to 15 characters (uppercase alphanumeric and hyphen)

Table 1. The 222-bit binary EPC representation for the ATA identifier based on UID Construct 2 including the header, and allowed character ranges for CAGE code, Original Part Number and Sequential Part Serial Number.

• *Header* is 8-bits, with a binary value of yyyyyyyy (to be defined by EPCglobal)
• *Filter Value* is not part of the ATA identifier, but is used for fast filtering and pre-selection of basic logistics types. The Filter Values are specified in Table 2.

Type	**Binary Value**
Pallet	0000
Case	0001
UID item	0010
Safety equipment	0011
Reserved for future use	all other values

Table 2. Filter Values for use with the ATA identifier

• *CAGE code* contains a literal embedding of the CAGE code using 6-bit ASCII compaction.

• *Original Part Number* is a literal embedding of the Original Part Number code using 6-bit ASCII compaction. Leading zeros on the Original Part Number are not significant.

• *Sequential Part Serial Number* is a literal embedding of the Sequential Part Serial Number using 6-bit ASCII compaction. The Sequential Part Serial Number is unique within the Original Part Number for each instance within a particular CAGE code. Leading zeros on the Sequential Part Serial Number are not significant.

ATA2-222 Encoding Procedure

The following procedure creates an ATA2-222 encoding.

Given:

- A 5-character CAGE code C consisting of uppercase alphanumeric characters $c_1c_2c_3c_4c_5$

- An Original Part Number P up to 15 characters in length, consisting of any combination of the digits 0-9, the uppercase letters A-Z and the hyphen character

- A Sequential Part Serial Number S up to 15 characters in length, consisting of any combination of the digits 0-9, the uppercase letters A-Z and the hyphen character

- A Filter Value *F* where $0 \leq F < 16$

Procedure:

1. Check that each of the characters $c_1c_2c_3c_4c_5$ of the CAGE code C is one of the characters listed in the table in Appendix H. If this is not the case, stop: this character string cannot be encoded as an ATA2-222. Otherwise, construct the CAGE code by concatenating the 6-bit code, as given in Appendix H, for each of the characters $c_1c_2c_3c_4c_5$, yielding 6*5 = 30 bits total. This is the binary representation of the CAGE code, denoted c.

2. Check that each of the characters $p_1p_2\ldots p_L$ of the Original Part Number P is one of the characters listed in the table in Appendix H. If this is not the case, stop: this character string cannot be encoded as an ATA2-222. Otherwise, construct the Original Part Number by concatenating the 6-bit code, as given in Appendix H, for each of the characters $p_1p_2\ldots p_L$, yielding 6*L bits. If L < 15, concatenate additional zero bits to the right to make a total of 90 bits. This is the binary representation of the original part number, denoted p.

3. Check that each of the characters $s_1s_2\ldots s_K$ of the Serial Number S is one of the characters listed in the table in Appendix H. If this is not the case, stop: this character string cannot be encoded as an ATA2-222. Otherwise, construct the Serial Number by concatenating the 6-bit code, as given in Appendix H, for each of the characters $s_1s_2\ldots s_K$, yielding 6*K bits. If K < 15, concatenate additional zero bits to the right to make a total of 90 bits. This is the binary representation of the serial number, denoted s.

4. Construct the final encoding by concatenating the following bit fields, from most significant to least significant: Header yyyyyyyy (8 bits), Filter Value *F* (4 bits), CAGE code *c* from Step 1 (*30* bits), Original Part Number *p* from Step 2 (90 bits), Sequential Part Serial Number *s* from Step 3 (90 bits)

ATA2-222 Decoding Procedure

Given:

- An ATA2-222 as a 222-bit bit string yyyyyyyy$b_{213}b_{212}\ldots b_0$ (where the first eight bits yyyyyyyy are the header)

Yields:

- A 5-character CAGE code
- An Original Part Number (PNO) up to 15 characters
- A Sequential Part Serial Number (SEQ) up to 15 characters
- A Filter Value

Procedure:

1. Bits $b_{213}b_{212}b_{211}b_{210}$, considered as an unsigned integer, are the Filter Value.

2. Divide the bits $b_{209}b_{208}\ldots b_{181}b_{180}$ into five 6-bit segments. The result should consist of five non-zero binary segments. Lookup each of the 6-bit segments in Appendix H to obtain a corresponding character. If any of the 6-bit segments has a value that is not in Appendix H, stop: this bit string cannot be decoded as an ATA2-222. The five characters so obtained, considered as a character string c1c2…ck, is the value of the CAGE code (CAG).

3. Divide the remaining bits $b_{179}b_{178}\ldots b_{91}b_{90}$ into fifteen 6-bit segments. The result should consist of L non-zero binary segments followed by 15-L binary zero segments, where 0< L <=15). If this is not the case, stop; this bit string cannot be decoded as an ATA2-222. Otherwise, lookup each of the non-zero 6-bit segments in Appendix H to obtain a corresponding character. If any of the non-zero 6-bit segments has a value that is not in Appendix H, stop: this bit string cannot be decoded as an ATA2-222.
The L characters so obtained, considered as a character string $p_1p_2\ldots p_L$, is the value of the Original Part Number (PNO).

4. Divide the remaining bits $b_{89}b_{88}\ldots b_1b_0$ into fifteen 6-bit segments. The result should consist of K non-zero binary segments followed by 15-K binary zero segments, where 0< K <=15). If this is not the case, stop; this bit string cannot be decoded as an ATA2-222. Otherwise, lookup each of the non-zero 6-bit segments in Appendix H to obtain a corresponding character. If any of the non-zero 6-bit segments has a value that is not in Appendix H, stop: this bit string cannot be decoded as an ATA2-222.
The K characters so obtained, considered as a character string $s_1s_2\ldots s_k$, is the value of the Sequential Part Serial Number (SEQ).

URN representations for ATA UID Construct 2

Pure-identity URI

For the ATA identifier based on UID construct 2, the pure identity URI representation is as follows:

`urn:epc:id:ata2:`*`CAGECode.origPartNumber.seqNumber`*

where *`CAGECode`* is the five-character CAGE code, *`origPartNumber`* is the original part number (PNO) and *`seqNumber`* is the sequential part serial number (SEQ), where both the original part number (PNO) and the sequential part serial number (SEQ) are represented as a string of between 1 and 15 characters (consisting exclusively of any of the upper-case alphanumeric characters and the hyphen symbol '–').

Tag-encoding URI

For the ATA identifier based on UID construct 2, the tag-encoding URI representation is as follows:

`urn:epc:tag:ata2-222:`*`filter.CAGECode.origPartNumber.seqNumber`*

where filter is the filter value represented as either one or two decimal digits, *`CAGECode`* is the five-character CAGE code, *`origPartNumber`* is the original part number (PNO) and *`seqNumber`* is the sequential part serial number (SEQ), where both the original part number (PNO) and the sequential part serial number (SEQ) are represented as a string of between 1 and 15 characters (consisting exclusively of any of the upper-case alphanumeric characters and the hyphen symbol '–').

Pattern URI

The pattern URI for the ATA identifier based on UID construct 2 is as follows:

`urn:epc:pat:`*`tagType:filterPat.CAGECodePat.origPartNumberPat.seqNumberPat`*

where *`tagType`* is `ata2-222`, *`filterPat`* is either a filter value, a range of the form [*lo-hi*], or a * character; *`CAGECodePat`* is either a CAGE Code or a * character; *`origPartNumberPat`* is either an original part number or a * character and *`seqNumberPat`* is either a serial number or a * character.

Pure Identity Pattern URI

The pure identity pattern URI for the ATA identifier based on UID Construct 2 is as follows:

`urn:epc:idpat:ata2:`*`CAGECodePat.origPartNumberPat.seqNumberPat`*

where *`CAGECodePat`* is either a CAGE Code or a * character; *`origPartNumberPat`* is either an original part number or a * character and *`seqNumberPat`* is either a serial number or a * character, with the proviso that:

- `CAGECodePat` shall not be a * character if either `seqNumberPat` or `origPartNumberPat` is not a * character.
- `origPartNumberPat` shall not be a * character if `seqNumberPat` is not a * character.

Syntax for ATA UID Construct 2

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

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


UpperAlphanumeric :: = Digit | UpperAlpha
UpperAlphanumericAndHyphen :: = Digit | UpperAlpha | Hyphen

StarComponent ::= “*”
RangeComponent ::= “[” NumericComponent “-”
                   NumericComponent “]”

ATATagType ::= “ata2-222”
ATAFilter ::= NumericComponent
CAGECode ::= CAGECodeChar*5
ATAPartNumber ::= UpperAlphanumericAndHyphen+
ATASeqNumber ::= UpperAlphanumericAndHyphen+
ATAFilterPat ::= PatComponent
CAGECodePat ::= CAGECode | StarComponent
ATAPartNumberPat ::= ATAPartNumber | StarComponent
ATASeqNumberPat ::= ATASeqNumber | StarComponent
```

Pure-identity URI

```
ATA2-URI ::= “urn:epc:id:ata2:” CAGECode “.” ATAPartNumber “.”
ATASeqNumber
```

Tag-encoding URI

```
ATA2TagURI ::= “urn:epc:tag:” ATA2TagType “:” ATAFilter “.”
CAGECode “.” ATAPartNumber “.” ATASeqNumber
```

EPC Pattern URI

```
ATAPatURI ::= “urn:epc:pat:” ATA2TagType “:” ATAFilterPat “.”
CAGECodePat “.” ATAPartNumberPat “.” ATASeqNumberPat
```

EPC Identity Pattern URI

```
ATAIDPatURI ::= “urn:epc:idpat:ata2:” ATAIDPatMain
ATAIDPatMain ::= CAGECode “.” ATAPartNumber “.” ATASeqNumber
| CAGECode “.” ATAPartNumber “.*” | CAGECode “.*.*” | “*.*.*”
```

Summary (non-normative) for ATA UID Construct 2

The syntax rules above can be summarized informally as follows:

`urn:epc:id:ata2:`*`TTT`*`.`*`BBB`*

`urn:epc:tag:ata2-222:`*`FFF`*`.`*`TTT`*`.`*`BBB`*

`urn:epc:idpat:ata2:`*`TTT`*`.`*`BBB`*

`urn:epc:pat:ata2-222:`*`FFFpat`*`.`*`TTT`*`.`*`BBBpat`*

`urn:epc:pat:ata2-222:`*`FFFpat`*`.*.`*`BBBpat`*

where

`BBB` denotes an alphanumeric Serial Number (including the hyphen)
`TTT` denotes a CAGE code assigned by the US DoD or NATO
`FFF` denotes a filter code as used by the SGTIN, SSCC, SGLN, GRAI, GIAI, DoD and ATA tag encodings
`BBBpat` denotes an alphanumeric Serial Number (including the hyphen) or `*`
`TTTpat` denotes a CAGE code (assigned by the US DoD or NATO) or `*`
`FFFpat` denotes a filter code as used by the SGTIN, SSCC, SGLN, GRAI, GIAI, DoD and ATA tag encodings but allowing `*` and `[lo-hi]` pattern syntax in addition

Translation between EPC-URI and Other EPC Representations

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

Bit string into EPC-URI (pure identity)

Create an EPC-URI by concatenating the following: the string urn:epc:id:ata2:, the CAGE code $c_1c_2c_3c_4c_5$, a dot (.) character, the Original Part Number p, a dot (.) character and the Sequential Part Serial Number s.
The Sequential Part Serial Number should have no leading zeros, except where the Sequential Part Serial Number is itself zero, in which case the corresponding URI portion must consist of a single zero character.

Bit string into EPC-URI (tag-encoding)

Create an EPC tag URI by concatenating the following: the string `urn:epc:tag:`, the encoding scheme (`ata2-222`), the colon character (`:`), the Filter Value F as a decimal integer, a dot (`.`) character, the CAGE code $c_1c_2c_3c_4c_5$, a dot (`.`) character, the Original Part Number p and the Sequential Part Serial Number s.
The Sequential Part Serial Number should have no leading zeros, except where the Sequential Part Serial Number is itself zero, in which case the corresponding URI portion must consist of a single zero character.

URI into Bit String

The following procedure translates a URI into a bit string:

1. If the URI is an ATA2-URI (`urn:epc:id:ata2:`), stop: the URI cannot be translated into a bit string.

2. If the URI is an EPC Tag URI (`urn:epc:tag:`*`encName`*`:`), parse the URI using the grammar for TagURI as given on p6.
If the URI cannot be parsed using these grammars, stop: the URI is illegal and cannot be translated into a bit string.
3. If *encName* is ata2-222, let the URI be written as
`urn:epc:tag:`*encName*`:`$f_1f_2\ldots f_F$`.`$c_1c_2\ldots c_5$`.`$s_1s_2\ldots s_K$
4. Interpret $f_1f_2\ldots f_F$ as a decimal integer filter value F
5. Interpret $c_1c_2\ldots c_5$ as the 5-character CAGE code C
6. Interpret $s_1s_2\ldots s_K$ as a string serial number S
7. Carry out the encoding procedure defined on p14.

Appendix A1: Encoding Scheme Summary Table (non-normative) for ATA UID Construct 1

ATA1 - ATA Spec 2000 identifier (UID Construct 1)				
ATA1-132	**Header**	**Filter Value**	**CAGE Code**	**Serial Number**
	8	4	30	90
	xxxx xxxx (Binary value)	(Refer to Table below for values)	5 upper-case alphanumeric characters	Up to 15 upper-case alphanumeric characters

Filter Values (Non-normative)	
Type	**Binary Value**
Pallet	0000
Case	0001
UID item	0010
Safety equipment	0011
Reserved for future use	All other values

Appendix A2: Encoding Scheme Summary Table (non-normative) for ATA UID Construct 2

ATA2 - ATA identifier based on UID Construct 2					
ATA2-222	**Header**	**Filter Value**	**CAGE Code (CAG)**	**Original Part Number (PNO)**	**Sequential Item Number (SEQ)**
	8	4	30	90	90
	yyyy yyyy (Binary value)	(Refer to Table below for values)	5 upper-case alphanumeric characters	Up to 15 upper-case alphanumeric characters	Up to 15 upper-case alphanumeric characters

Filter Values (Non-normative)	
Type	**Binary Value**
Pallet	0000
Case	0001
UID item	0010
Safety equipment	0011
Reserved for future use	All other values

Appendix H: Conversion between 6-bit compacted values and the subset of ASCII characters permitted for use in ATA Spec 2000 identifiers

Graphic Symbol	Name	6-bit code
–	Hyphen/Dash	101101
0	Digit zero	110000
1	Digit one	110001
2	Digit two	110010
3	Digit three	110011
4	Digit four	110100
5	Digit five	110101
6	Digit six	110110
7	Digit seven	110111
8	Digit eight	111000
9	Digit nine	111001

Graphic Symbol	Name	6-bit code
A	Capital letter A	000001
B	Capital letter B	000010
C	Capital letter C	000011
D	Capital letter D	000100
E	Capital letter E	000101
F	Capital letter F	000110
G	Capital letter G	000111
H	Capital letter H	001000
I	Capital letter I	001001
J	Capital letter J	001010
K	Capital letter K	001011
L	Capital letter L	001100
M	Capital letter M	001101
N	Capital letter N	001110
O	Capital letter O	001111
P	Capital letter P	010000
Q	Capital letter Q	010001
R	Capital letter R	010010
S	Capital letter S	010011
T	Capital letter T	010100
U	Capital letter U	010101
V	Capital letter V	010110
W	Capital letter W	010111
X	Capital letter X	011000
Y	Capital letter Y	011001
Z	Capital letter Z	011010

Note that the following characters shall not appear in the CAGE code:

Hyphen (–)
Capital letter I
Capital letter O