Interim Tag Data Standard for UHF Animal Identification

August 2016

Background

Radio frequency identification (RFID) technology has been available in the livestock industry for many years. Standardization of the RFID technology used in the identification devices has been and will remain critical to ensure compatibility of devices across manufacturers.

Most common in the market place has been low frequency identification devices operating at 134.2 kHz. Standards for low frequency (LF) RFID devices used for livestock were established in the 1990's through a Working Group of the International Organization for Standardization (ISO/TC23/SC19/WG3). Two primary standards were defined: one standard on the code structure in the transponder, and the other on the technology for the communication between reader and transponder. These standards are referenced below.

- ISO 11784. Agricultural Electronics—Radio Frequency Identification of Animals—Code Structure. International Organization for Standardization.
- ISO 11785. Radio Frequency Identification of Animals—Technical Concept. International Organization for Standardization.

USDA has required conformance to these standards for official identification devices that utilize low frequency RFID technology.

More recently, RFID identification devices using ultra high frequency (UHF) technology have become available on the market. USDA has approved several RFID eartags that incorporate UHF technology based on EPC Gen 2 (v1.2.0) ISO/IEC 18000-6C operating in the 902 MHz – 928 MHz range. While this standard addresses the communication protocol between the reader and the UHF tag, there is no standard for a common encoding scheme, or Tag Data Standard (TDS), for translating USDA animal numbering systems in UHF identification devices. A global standard is needed and highly preferred by USDA. However, as of this date, no standard has been defined and no standard appears to be on the horizon anytime soon.

In lieu of an established global standard for the encoding of animal identification numbers in UHF tags, USDA has defined an interim standard that would achieve uniformity across manufacturers authorized to encode USDA animal numbers into UHF identification devices. This action was warranted to ensure technical standardization is achieved as timely as possible across manufacturers already providing USDA animal identification devices utilizing UHF. Transition to a global standard(s) is strongly preferred and USDA acknowledges that this standard will be for interim use until such a standard evolves. When such a standard is available, USDA will work with approved manufacturers of official UHF identification devices to establish a timeline to transition to the recognized global standard.

UNITED STATES DEPARTMENT OF AGRICULTURE

This interim standard offers encoding flexibility for current tag types and is designed to accommodate new tag types in the future without the need for hardware or software updates. In addition, this interim TDS does not dictate the size of the memory chip used in the UHF device but leaves that decision up to the manufacturer based on the ID type, encoding format and any additional information they elect to encode in the chip. A User Memory Indicator (UMI) also referred to as a Value Added Indicator (VAI) has been added in the Electronic Product Code (EPC) header to indicate if information was encoded in the user memory at the time of tag manufacture.

The interim UHF TDS covers all USDA official animal identification numbers as well as the number format of the USDA Approved Backtags illustrated in the following chart.

USDA animal numbering systems

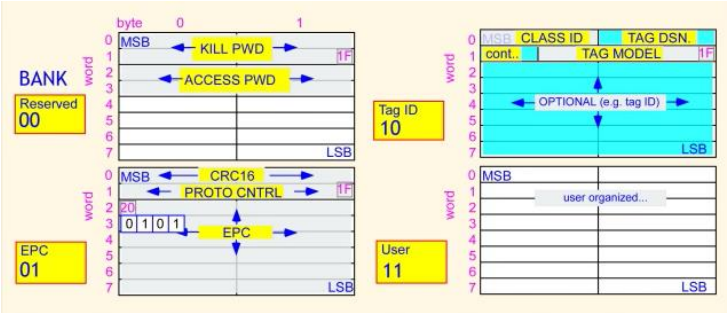
Number	Format of Animal Number	Number Examples
Animal identification number (AIN)	15 digits (fixed) - 840 are the first three digits (numeric code for USA)	840 003 123 456 789
National Uniform Eartagging System (NUES) - 9	9 alpha/numeric (fixed) - 2 State or Tribal[1] code - 3 alpha series - 4 digits in a sequential numerical series	23 ELV 4574 PA ELV 4574
National Uniform Eartagging System (NUES) - 8	8 alpha/numeric (fixed) - Swine and other species (except sheep and goats) o 2 numeric State or Tribal code o 2 alphabetical series o 4 digits in a numerical series	23 AB 4574
	- Sheep and goats (exclusive to scrapie program) o 2 alpha postal abbreviation o 2 alphabetical or alphanumeric series o 4 digits in a numerical series	PA AB 4574 or PA A2 4574
Flock-based number *with herd management number*	15 alpha/numeric (variable) - Flock identification number (maximum of 9 characters prefixed with State's postal abbreviation) with a unique herd management number (up to 6 characters). Does not include I, O, or Q except as part of a postal abbreviation.	MN0456 4275
Location-based number[2] *With the herd management number*	14 Alpha/numeric (variable) - Either a premises identification number (PIN) or location identification number (LID) with a unique herd management number PINs have 7 characters; LIDs may have 6, 7, or 8 characters; and the herd management number may have up to 6 characters.	006ER2A 4275
USDA Approved Backtag	8 Alpha/numeric (fixed) - 2 digit State numeric code - 2 alpha (2) - 4 digits	006ER2A 4275

[1] Tribal alpha and numeric codes are assigned by APHIS when requested by a Tribe (see ADT General Standards for listing: http://www.aphis.usda.gov/traceability/downloads/ADT_standards.pdf).

[2] Location identifiers include both the premises identification number (PIN) issued through the PIN allocator and the Location Identification (LID) numbers administered by the State or Tribe.

Interim Tag Data Standard Overview

Below is the Memory Map reference we have used to help establish our proposed interim TDS and it should be noted that this interim standard only covers the EPC in memory bank 01 of the map below. It is assumed that the use of all other memory banks along with the CRC and PC of the EPC will follow GS1 standards as this will eventually provide an easier transition to a recognized international standard when it becomes available.



The interim TDS described in this document divides the Class-1 Gen 2 EPC memory into three general sections (not including the CRC and PC): **Header**, which occupies the first three bytes, the **Content** section that holds the actual device identification and a **Flex** section that may or may not be used to store additional information.

EPC of Memory bank 01

Header	Content	Flex

The **Header** encompasses the first three Bytes of the EPC memory and contains information about the tag type, size of the Content and the data encoding method, ASCII or integer. In addition, the header contains a reissue counter, the check digit for the Content (animal identification section) and the Value Added Indicator.

The **Content** section of the memory contains the official identification such as an AIN or NUES ID and may be encoded as either ASCII or an integer, depending on the ID type. The Header section provides the reader/software information on the ID type, size and encoding method of the information contained in the Content. The Content size is incremented in bytes and when Content is an integer the encoding will follow little-endian format. https://en.wikipedia.org/wiki/Endianness

The **Flex** section may or may not exist and when it does exist can contain information encoded as ASCII or Integer. Information in the Header partition identifies if the Flex section contains data and the

encoding format. The available size of the Flex section will depend on the Content and the size of the chip.

Header

Byte One:

The information contained in the first byte of the Header will define the tag type, if the content is encoded as integer or ASCII and the Flex type.

1. Bits 1-6 identify the tag type (64 possible types)
    a. Types 0-31 = content is integer
    b. Types 32-63 = content is ASCII
2. Bits 7-8 identify the Flex type (4 possible types)
    a. 0 = No Flex
    b. 1 = Management ID (INT)
    c. 2 = Management ID (ASCII)
    d. 3 = Manufacture Number (INT) e.g., date of manufacture

Byte 1	Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6	**Content type** (0-63 inclusive) Highest bit = 0 indicates Integer (<=31) content encoding Highest bit = 1 indicates ASCII (>=32) content encoding
	Bit 7 Bit 8	**Flex type** (0-3 inclusive) 0=No Flex, 1=MGT (INT), 2=MGT (ASCII), 3=MFG NUM (INT)
Byte 2	Bit 1 Bit 2 Bit 3 Bit 4	**Content Length** in Bytes (0-15 inclusive) If encoded as an integer then only 1,2,4,8 allowed
	Bit 5 Bit 6 Bit 7 Bit 8	**Flex Length** in Bytes (0-15 inclusive) If encoded as an integer then only 1,2,4,8 allowed
Byte 3	Bit 1	**User Memory Indicator (Value Added Indicator)**
	Bit 2 Bit 3 Bit 4	**Reissue Counter** (0-7 inclusive)
	Bit 5 Bit 6 Bit 7 Bit 8	**Check digit** (0-15 inclusive) Display as hex (0-F)

Byte Two:

Byte two identifies how many bytes of memory the Content and Flex sections occupy. Combining the information contained in bytes one and two the hardware/software should be able to read existing and new tag types without the need for updates each time a new tag type is added.

Byte Three:

Byte three holds the UMI/VAI, tag reissue counter and the check digit that is calculated from the ID contained in the content portion of memory.

The UMI/VAI is a one bit indicator within the EPC that provides a mechanism for the manufacturer to identify if information was written to the User Memory (bank 11) at the time of initial encoding.

The reissue counter will be set to 0 for all initial tag production and allows for up to 7 reissues. This is the same functionality as used in LF RFID animal identification devices.

The check character calculation is based on the credit card check digit algorithm and is demonstrated in the appendix of this document.

Content

The Content portion of tag will contain the actual official identification, e.g., 840 AIN or NUES. After encoding by the manufacturer the Content section will be “read only” without the ability to alter. As defined in byte one of the Header, the Content will be encoded as an integer or ASCII. Byte two of the Header will define the total size, in bytes, of the Content section. Please note that not all ID types will fit on the smaller 96 bit chips.

Flex

This section may or may not exist and if it does, it holds data as defined in byte one of the Header. The available size of the Flex area is determined by the total available memory less the Header and Content sections. Therefore, chips with more memory will accommodate larger Flex volumes than chips with smaller memory capacities.

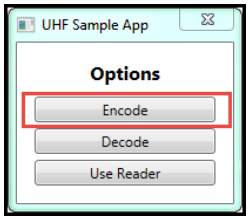
Examples of the TDS and tools on the following pages.

This illustration provides example encodings for multiple tag types and demonstrates how some ID types with flex data added would require larger chips.

Content Type	Tag Content	Ck Digit	Flex Type	Flex Content	Reissue #	User Mem?	96 Bits												128 Bits				192 Bits							
							1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
0 - AIN (Integer)	840000123456789	B	None		0	N	0x00	0x80	0x0B	0x15	0x4D	0xF8	0xC4	0xF9	0xFB	0x02	0x00	0x00												
32 - NUES (ASCII)	34AB1234	C	MGT TAG ASCII	3-B	0	N	0x82	0x83	0x0C	0x33	0x34	0x41	0x42	0x31	0x32	0x33	0x34	0x33	0x2D	0x42	0x00	0x00								
32 - NUES (ASCII)	84WAZ5678	6	MGT TAG Integer	548745	0	Y	0x81	0x94	0x86	0x38	0x34	0x57	0x41	0x5A	0x35	0x36	0x37	0x38	0x89	0x5F	0x08	0x00								
34 - Approved Back Tag (ASCII)	34GL8322	3	Mfn Num	200	0	N	0x8B	0x81	0x03	0x33	0x34	0x47	0x4C	0x38	0x33	0x32	0x32	0xC8												
0 - AIN (Integer)	840000123456789	B	MGT TAG ASCII	'Ole Bessie	3	Y	0x02	0x8B	0xBB	0x15	0x4D	0xF8	0xC4	0xF9	0xFB	0x02	0x00	0x27	0x4F	0x6C	0x65	0x20	0x42	0x65	0x73	0x73	0x69	0x65	0x00	0x00
33 - Location # Plus MGT # (ASCII)	IA123456 123456	0	MGT TAG ASCII	Ca$h	0	N	0x86	0xF4	0x00	0x49	0x41	0x31	0x32	0x33	0x34	0x35	0x36	0x20	0x31	0x32	0x33	0x34	0x35	0x36	0x43	0x61	0x24	0x68	0x00	0x00

The following examples use an application that was produced to validate the encoding and reading process using the interim tag data standard. A copy of this application can be obtained by contacting randy.d.munger@aphis.usda.gov

When the application opens, you are presented with three options Encode, Decode or Use Reader.



Example 1 - AIN:

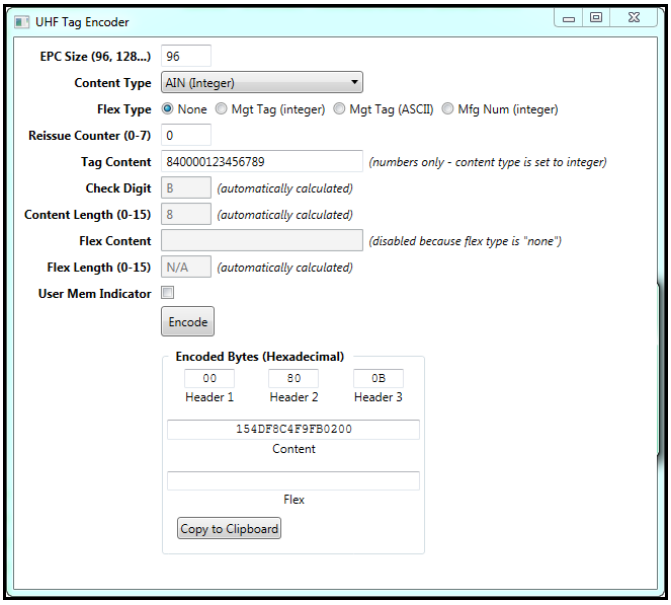
In this example we will demonstrate the encode/decode options entering the tag information from the first row of the spreadsheet above. Click the Option Encode to open the encoder.

Enter the information from the first row of the spreadsheet on page six of this document (also provided below) into the encoder.

Content Type	Tag Content	Ck Digit	Flex Type	Flex Content	Reissue #	User Mem?	96 Bits												128 Bits			
							1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0 - AIN (Integer)	840000123456789	B	None		0	N	0x00	0x80	0x0B	0x15	0x4D	0xF8	0xC4	0xF9	0xFB	0x02	0x00	0x00				

- Content type = AIN (integer)
- Flex = None
- Reissue counter = 0
- Tag Content = 840000123456789
- User Memory Indicator = none

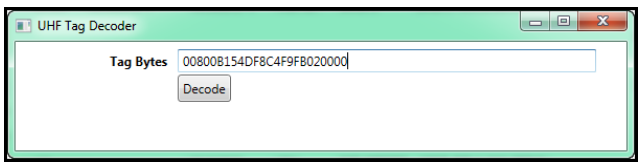
Click on Encode and the Encoded Bytes in Hexadecimal fill the Header, Content and Flex (if present) fields. In this example the result is 00800B154DF8C4F9FB020000.Remember this is in little-endian.



If you have the encoding for a tag and would like to reverse the process above, then you can use the Decode option in the sample app selector.

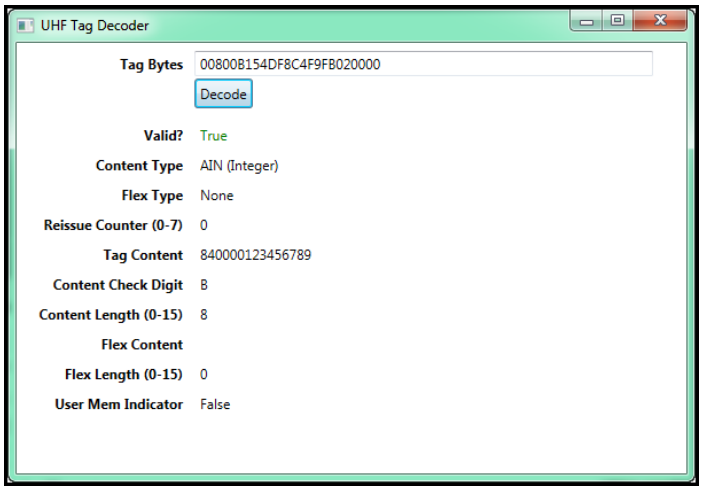
Clicking on Decode opens the app.

Type or paste in the tag encoding (use Copy to Clipboard button to easily use the information from the encoder):



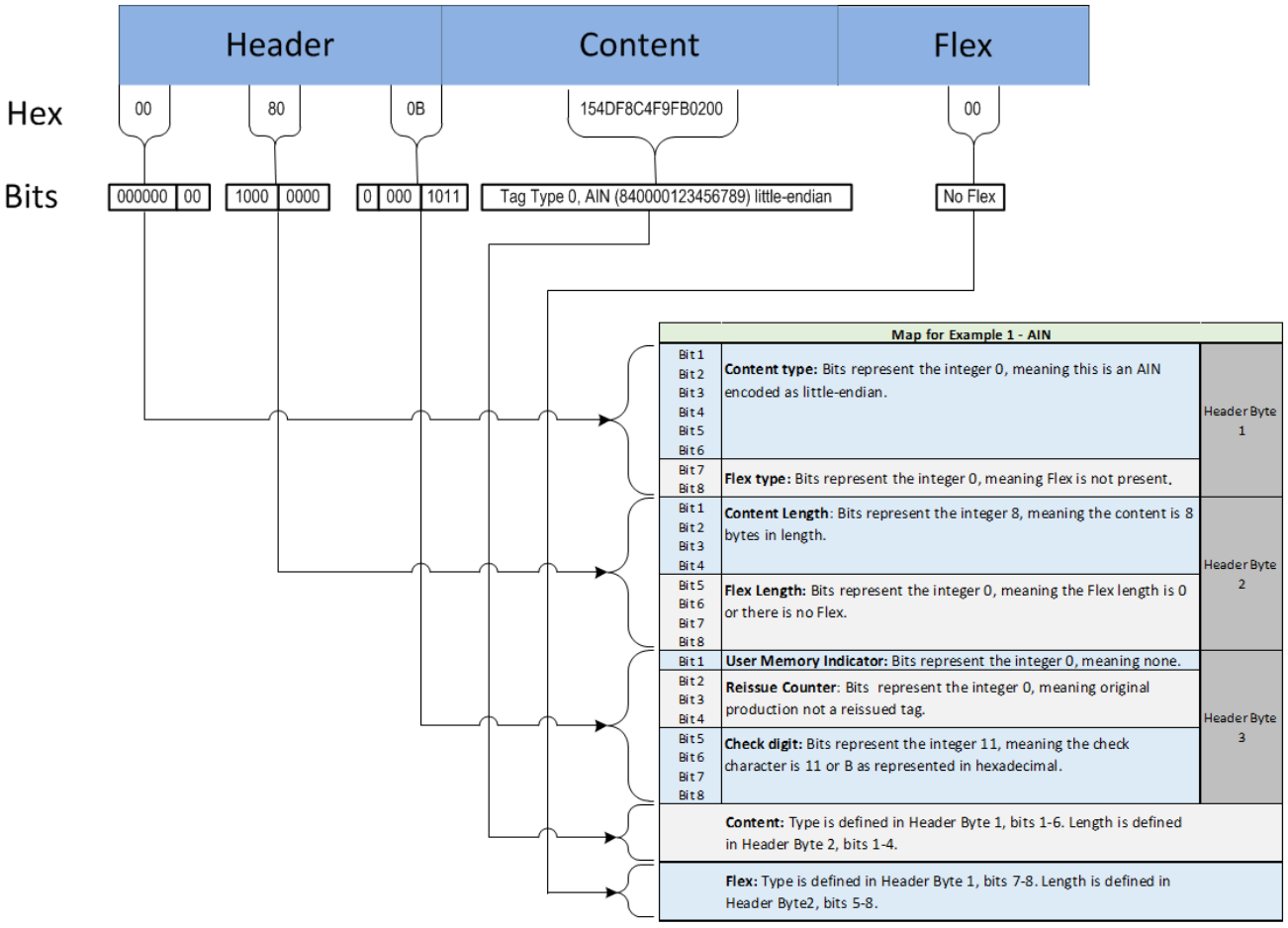
Here we have used the encoding produced in the example above.

Click the Decode button to review the tag information:



This is a reverse process of the encoder example and the information is exactly the same.

The following diagram is an examination of the hexadecimal value created in this example: 00800B154DF8C4F9FB020000



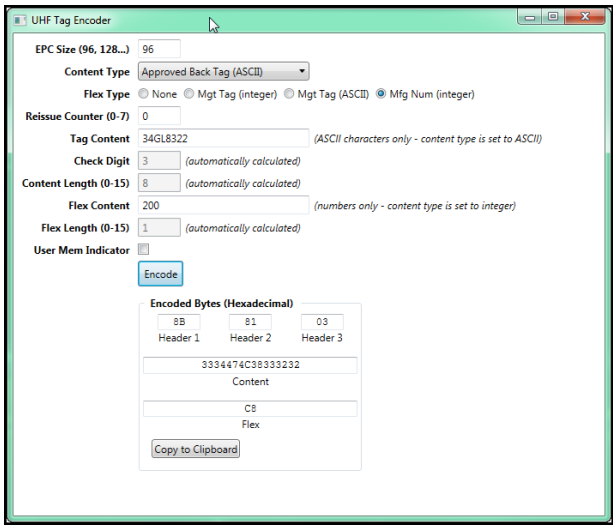
Example 2 – Approved Back tag with Flex information

In this second example we will also demonstrate the use of the encode/decode options while entering the tag information from the fourth row of the spreadsheet on page five of this document.

Enter the information from the fourth row of the spreadsheet above into the Encoder:

Content Type	Tag Content	Ck Digit	Flex Type	Flex Content	Reissue #	User Mem?	96 Bits												128 Bits			
							1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
34 - Approved Back Tag (ASCII)	34GL8322	3	Mfn Num	200	0	N	0x8B	0x81	0x03	0x33	0x34	0x47	0x4C	0x38	0x33	0x32	0x32	0xC8				

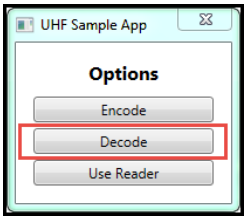
- Content type = Approved Back Tag (ASCII)
- Flex Type = Mfn. Num. (Integer)
- Flex Content = 200
- Reissue counter = 0
- Tag Content = 34GL8322
- User Memory Indicator = none



Click on Encode and the Encoded Bytes in Hexadecimal fill the Header, Content and Flex fields. In this example the result is 8B81033334474C38333232C8.

As shown in the first example, if you have the encoding value for a tag and would like to reverse the process above, then you can use the Decode option in the sample app selector.

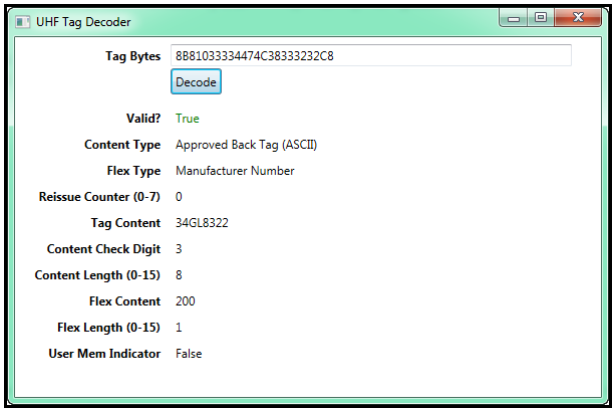
Clicking on Decode opens the UHF Tag Decoder app.



Type or paste in the tag encoding (use Copy to Clipboard button to easily use the information from the encoder):

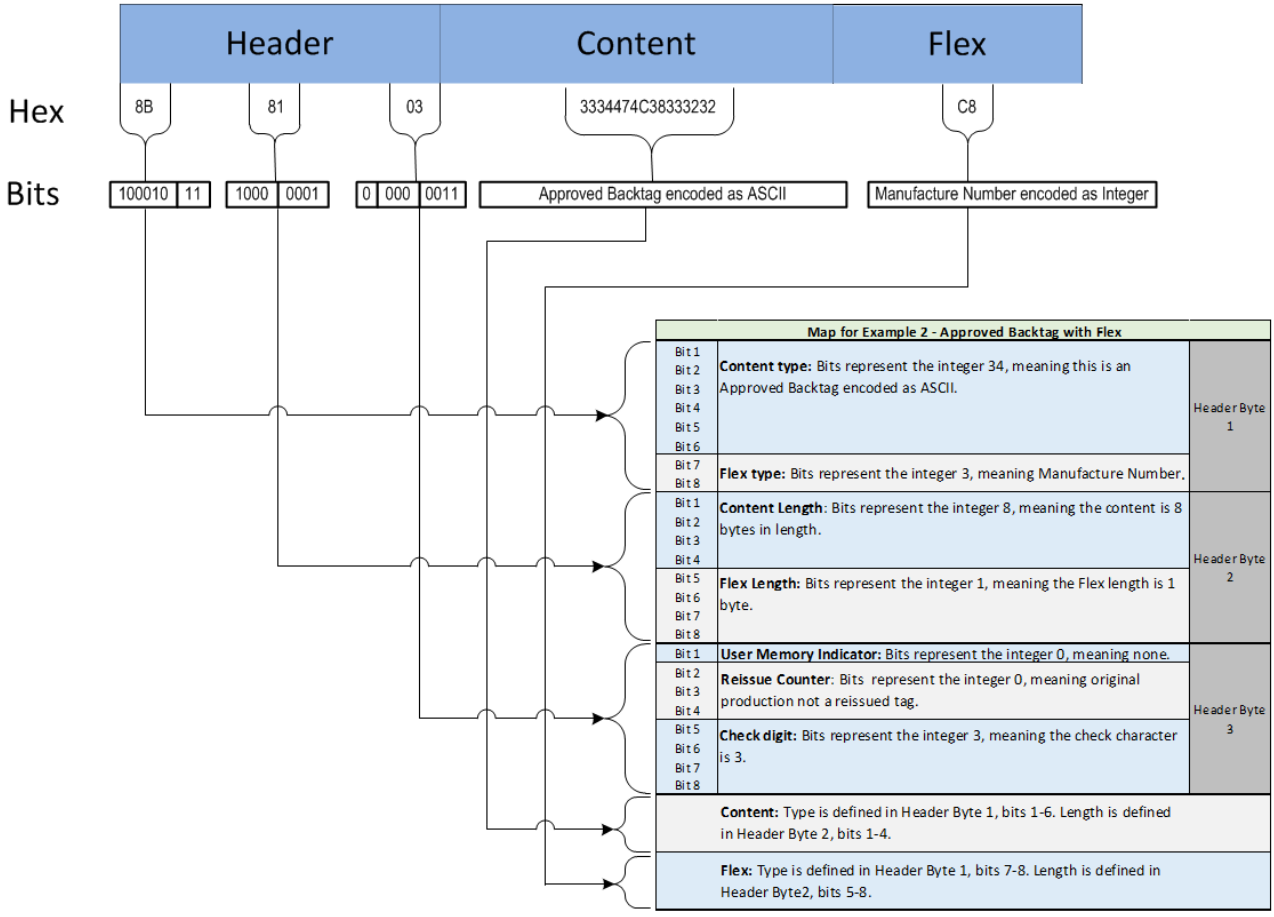
Here we have used the encoding produced in this Back tag example above.

Click on the Decode button to review the tag information:



This is just a reverse process of the encoder and the information is exactly the same.

The following diagram is an examination of the hexadecimal created in this example: 8B81033334474C38333232C8



Upon request we will also provide sample code in C# .NET to encode and decode tags using the proposed TDS. This code may be a useful guide to tag and software manufacturers. Please contact randy.d.munger@aphis .usda.gov for a copy of this code.

Appendix (Check Character Calculation):

```
// To get the ASCII integer code for a character, find the index then add 32
// This isn't really needed in C#/.NET because we could treat the CHAR as
// an integer but this makes the example more agnostic
private const String AsciiCharacters =
    " !\"#$%&'()*+,-./0123456789:;<=>?@" +
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`" +
    "abcdefghijklmnopqrstuvwxyz{|}~";

/// <summary>
///     Calculates the check digit using a modified version of the GS1 standard
///     http://www.gs1.org/how-calculate-check-digit-manually.
///     The modifications are:
///         1.) Map each character to the printable ASCII decimal values
///             This means any printable ASCII is allowed
///         2.) Divide the sum by 16 and return the remainder instead
///             of returning 0-9 return 0-15 (to make full use of 4 bits)
/// </summary>
/// <param name="identifier">The identifier.</param>
/// <returns>A checkdigit of 0-15</returns>
public static Byte CalculateCheckDigit(String identifier)
{
    Int32 sum = 0;

    for (Int32 i = 0; identifier != null && i < identifier.Length; i++)
    {
        // Get the ASCII decimal value of the character we are on
        Int32 dec = AsciiCharacters.IndexOf(identifier[identifier.Length - i - 1]) + 32;

        // 31 means the ASCII character wasn't found
        if (dec == 31)
```

```
            throw new Exception("Only ASCII between 32 and 126 (decimal value)
allowed.");


        sum += dec*(i%2 == 0 ? 3 : 1);
    }

    // Return the remainder of our sum divided by 16
    // to make full use of our 4 bits (range of result is 0-15)
    return (Byte) (sum%16);
}
```