MH10.8.4 – 2002

Unit Loads and Transport Packages RFID Tags for Returnable Containers

Approved: 9 August 2002

Abstract

This standard defines the Radio Frequency Identification (RFID) devices to be used with Returnable Containers. This standard is intended to allow for compatibility and to encourage interoperability of products for the growing RFID market in the United States.

Developed by:

MH10 Committee, Unit-Loads and Transport-Packages
Subcommittee 8, Coding & Labeling of Unit-Loads

Published by MH10 Secretariat:

Material Handling Industry
8720 Red Oak Blvd., Suite 201
Charlotte, NC 28217-3992
mhstandards@mhia.org

MATERIAL HANDLING INDUSTRY®

American National Standard

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer.

Consensus is established when, in the judgement of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that a concerted effort be made toward their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he has approved the standards or not, from manufacturing, marketing, purchasing, or using products, processes or procedures not conforming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give an interpretation of any American National Standard. Moreover, no person shall have the right or authority to issue an interpretation of an American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the sponsor whose name appears on the title page of this standard.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken periodically to reaffirm, revise or withdraw this standard. Purchasers of American National Standards may receive current information on all standards by calling or writing the American National Standards Institute.

ANSI MH10.8.4 – 2002

American National Standard

Unit Loads and Transport Packages
RFID Tags for Returnable Containers

Material Handling Industry

Approved 9 August 2002
American National Standards Institute, Inc.

Disclaimer

This standard, which was developed under the ANSI Committee method and approved by ANSI on August 9, 2002, represents suggested design practices and guidance for radio frequency identification (RFID) of returnable containers. It was developed with the sole intent of offering information to parties engaged in the manufacture, marketing, purchase, or use of automatic identification equipment software and services. This standard is advisory only and acceptance is voluntary and the standard should be regarded as a guide that the user may or may not choose to adopt, modify, or reject. The information does not constitute a comprehensive safety program and should not be relied upon as such. Such a program should be developed and an independent safety adviser consulted to do so.

Material Handling Industry (MHI), the MH10 Committee and its officers and members assume no responsibility and disclaim all liability of any kind, however arising, as a result of acceptance or use or alleged use of this standard. User specifically understands and agrees that MHI, the MH10 Committee, their officers, committee members, agents, and members shall not be liable under any legal theory of any kind for any action or failure to act with respect to the design, installation, manufacture, preparation for sale, sale, characteristics, features, or delivery of anything covered by this standard. Any use of this information must be determined by the user to be in accordance with applicable federal, state, and local laws and regulations.

MHI, the MH10 Committee and its officers and members make no warranties of any kind, express, implied, or statutory, in connection with the information in this standard. MHI and the MH10 Committee specifically disclaim all implied warranties of merchantability or of fitness for particular purpose.

By referring to or otherwise employing this standard, the user agrees to defend, protect, indemnify, and hold MHI, the MH10 Committee, their officers, committee members, agents, and members harmless from and against all claims, losses, expenses, damages, and liabilities, direct, incidental, or consequential, arising from acceptance or use or alleged use of this standard, including loss of profits and reasonable attorneys' fees which may arise out of the acceptance or use or alleged use of this standard. The intent of this provision and of the user is to absolve and protect MHI, the MH10 Committee, their officers, committee members, agents, and members from any and all loss relating in any way to this standard, including those resulting from the user's own negligence.

Foreword (This foreword is not part of American National Standard MH10.8.4-2002)

This standard defines the Radio Frequency Identification (RFID) devices to be used with Returnable Containers. This standard is intended to allow for compatibility and to encourage interoperability of products for the growing RFID market in the United States. This standard defines a single Application Programming Interface (API), which will be shared by all compliant RFID implementations and provide a common interface to application programs. The applications for returnable containers, including cable reels, addressed by this standard typically require ranges greater than one meter.

The goal of this standard is to serve current and future users and manufacturers by encouraging the development of open, dynamic systems.

Unit Loads and Transport Packages RFID Tags for Returnable Containers provides a single, common technical and data solution for equipping returnable transport items with RF tags. It relies upon the data semantic and syntax standards developed within ASC MH10. It relies upon the technology standards developed internationally through the work of ISO/IEC JTC 1/SC 31. The international equivalent of this standard is the international standard ISO 17367, developed by the Joint Working Group (JWG) between ISO Technical Committees 122 (Packaging) and 104 (Freight Containers). ANSI MH10.8.4 was developed prior to ISO 22389 and the ANS served as a source document for the international document.

At the time of approval, the MH10/SC 8 committee consisted of the following members:

Automotive Industry Action Group	Morris Brown
Boeing	Frank Goodell
Bruno Associates	Thomas Bruno
Canada Post	Ken Cavanagh
CEA	Brian Markwalter
DoD AIT Project Office	Maurice Stewart; Dan Kimball (Alternate); Eugene Bransfield (Alternate)
Federal Express	Mark Thomas
General Motors	Larry Graham
Handheld Products	Robert Hussey
HighTech Aid	Steve Halliday
IBM	Charles Milligan
Intermec Technologies	Michael Guillory
Mississippi Valley State University	Allan Gilligan
Q.E.D. Systems	Craig K. Harmon; Marsha A. Harmon (Alternate)
RVSI	Luis Flgarella
Rylander Associates	Robert Rylander
Symbol Technologies	Christina Barkan
Telcordia Technologies	Robert Fox
Texas Instruments	Dan Wikander
United Parcel Service	Mark Lewis
United States Air Force	Mark Reboulet; Howard English (Alternate)
United States Postal Service	Himesh Patel

At the date of approval of this standard, the MH10 Committee, Unit-Loads and Transport-Packages, consisted of the following members:

AIM, USA
American Trucking Associations
American Wood Packaging Association
APA – The Engineered Wood Association
Assoc. of Professional Material Handling Consultants
ASTM
Automotive Industry Action Group
Comp TIA
Containerization & Intermodal Institute
Electronics Industries Association
Fibre Box Association
Flexible Intermediate Bulk Containers Association
Food Marketing Institute
Glass Packaging Institute
Graphic Communications Association
IMC & WD, Product Section - Material Handling Industry
Institute of Packaging Professionals
Integrated Business Communications Alliance
International Cargo Handling Coordination Assoc.
International Safe Transit Association
Material Handling Industry
Material Handling Management Society
National Wholesale Grocer's Association
National Wooden Pallet & Container Association
Plastic Drum Institute
Q.E.D. Systems
Rack Manufacturers Institute
Reusable Industrial Packaging Association
Steel Shipping Container Institute
Textile Bag Manufacturers Association
The Soap & Detergent Association
U.S. Dept. of Agriculture
U.S. Dept. of Defense Logistics
U.S. Forest Products Laboratory
Uniform Code Council
United Fresh Fruit & Vegetable Association

Suggestions for improvement, and questions regarding interpretation of this standard will be welcome. They should be sent to: MH 10 Committee (MHIA), Material Handling Industry of America, 8720 Red Oak Blvd., Suite 201, Charlotte, NC, 28217-3992 or mhstandards@mhia.org.

Unit Loads and Transport Packages
RFID Tags for Returnable Containers

Table of Contents

UNIT LOADS AND TRANSPORT PACKAGES
RFID TAGS FOR RETURNABLE CONTAINERS

1 INTRODUCTION

This ANSI MH 10/SC 8 Standard defines the Radio Frequency Identification (RFID) standard for returnable containers. This standard is intended to allow for compatibility and to encourage interoperability of products for the growing RFID market in the United States. This standard defines a single Application Programming Interface (API), which will be shared by all compliant RFID implementations and provide a common interface to application programs.

The applications for returnable containers, including cable reels, addressed by this standard typically require ranges greater than one meter.

The goal of this standard is to serve current and future users and manufacturers by encouraging the development of open, dynamic systems.

This standard supports national and international standards for data semantics, data syntax, transfer syntax, and a radio-frequency air interface.

2 PURPOSE

This document establishes an application standard for RFID devices, specifically, one operating in accordance with US Code of Federal Regulations (CFR) Title 47, Chapter I, Part 15.

3 SCOPE

This document is intended to support applications of RFID for returnable containers, such as cable reels, pallets, gas cylinders, unit load devices, dolleys, and special containers.

3.1 Air Interface Definitions

This standard is intended to address RFID devices operating with Frequency Hopping Spread Spectrum modulation using ANSI NCITS 256:2001, Clause 4.2, modified to at least 50 channels each of 400 kHz in width between 902 MHz and 928 MHz. Annex A contains the physical layer parameters recommended by this standard.

3.1.1 Host Interface Definitions

This standard employs the host interface defined in NCITS 256:2001, Clause 2, ISO 15961, and ISO 15962. The data syntax complies with ISO 15434 and the data semantics with ISO 15418. This standard further defines a standard API (Annex B) and the standard air interface (Annex A) for wireless, non-contact information system equipment for Item Management applications.

3.1.2 RFID System Definition

The RFID system includes a host system and RFID equipment. The host system runs an application program that controls and interfaces with the RFID equipment. The RFID equipment is composed of two principal components: tags and interrogators. The tag is intended for attachment to an item that a user wishes to manage. It is capable of storing data regarding the tag or item and communicating this information to the interrogator. The interrogator is a device that communicates to tags in its field of view. The interrogator controls the protocol, reads from or writes to the tag, and ensures message delivery and validity.

3.1.2.1 Minimum Features

RFID systems defined by this standard provide the following minimum features:

- Identify tag in range
- Read data
- Write data
- Selection by group or address
- Graceful handling of multiple tags in the field of view
- Error detection

3.1.2.2 Additional Features

RFID systems defined by this standard provide the following additional features:

- Security

3.1.3 Document Structure and References

This standard addresses the following topics:

- Clause 6 of this standard provides direction for the physical layer parameters.
- Clause 7 of this standard provides guidance for the data structure used in this application for RF tags.
- Clause 8 of the standard defines the Tag Placement.
- Annex A provides the specific air interface to be employed.
- Annex B defines the Application Programming Interface (API) that is shared by all standard-compliant systems. This API is intended to be implementation independent.
- Annex C describes the memory layout of the tag.
- Annex D provides examples of tag location on containers.

3.1.4 Tag Identification Number

A tag identification number may be included in commands directed to a specific tag. Annex C describes the data structure of the tag including the Tag Identification number.

4 REFERENCED DOCUMENTS

4.1 Normative References

- ANSI NCITS 256:2001 – Radio Frequency Identification, Clauses 1, 2, and 4.2 (hereinafter referred to as NCITS 256)
- ISO/IEC 15434:1999 – Syntax for High Capacity ADC Media
- ISO/IEC 15418:1999 – EAN.UCC Application Identifiers and ANSI MH 10 Data Identifiers
- ISO/IEC 15961:WD2001 – RFID for Item Management – Host Interrogator – Tag functional commands and other syntax features
- ISO/IEC 15962:WD2001 – RFID for Item Management – Data Syntax
- ISO/IEC 7498-1:1994 – "Information Technology – Open Systems Interconnection – Basic Reference Model: The Basic Model"; International Standards Organization ISO/IEC JTC1/SC21; 1994
- ISO 14816: 2000 – Road Traffic and Transport Telematics – Automatic Vehicle and Equipment Identification
- US Code of Federal Regulations (CFR) Title 47, Chapter I, Part 15. "Radio Frequency Devices"; U.S. Federal Communications Commission, 1 Oct. 1995 (hereinafter referred to as FCC Part 15)

4.2 Informative References

- ISO 10374:1991, w/Amd 1:1995 – Freight containers – Automatic identification
- ISO 668:1995 – Series 1 freight containers – Classification, dimensions and ratings
- ISO 6346:1995 – Freight containers – Coding, identification and marking

5 ABBREVIATIONS AND DEFINITIONS

5.1 Abbreviations

AM	Amplitude Modulation
API	Application Programming
ASK	Amplitude Shift Keying
bps	Bits per second
BER	Bit Error Rate
CRC	Cyclic Redundancy Check
CW	Continuous Wave
EIRP	Equivalent Isotropic Radiation Power

DLL	Data Link Layer (OSI Model)
DSSS	Direct Sequence Spread Spectrum
FCC	Federal Communications Commission
FHSS	Frequency Hopping Spread Spectrum
FM	Frequency Modulation
FSK	Frequency Shift Keying
kbps	kilobits per second
kHz	kilohertz (10^3 Hertz)
LLC	Logical Link Control Sublayer of Data Link Layer
LSB	Least Significant Bit
MAC	Medium Access Control Sublayer of Data Link Layer
MHz	megahertz (10^6 Hertz)
MSB	Most Significant Bit
NCITS	National Committee for Information Technology Standards
NRZ	Non-Return to Zero. Coding scheme in which a binary "one" is carrier on, and a binary "zero" is carrier off.
NRZI	Non-Return to Zero, Invert on Ones. Coding scheme in which a binary "one" is a continuation of the previous carrier state (on or off), and a binary "zero" is a change of state (off-on or on-off).
OOK	On-Off Keying
OSI	Open Systems Interconnection
ppm	Parts per million (10^{-6})
PHY	Physical Layer
PM	Phase Modulation
PN	Pseudo-Noise (as in PN Code)
PSK	Phase Shift Keying
RAM	Random Access Memory
RF	Radio Frequency
RFID	Radio Frequency Identification

5.2 Definitions

5.2.1 Antenna Polarization – Locus of the tip of the vector of the electrical field strength in a plane perpendicular to the transmission vector. Examples are horizontal and vertical linear polarization and left and right hand circular polarization.

5.2.2 Awake – A gag is awake if the tag's receiver is powered and able to receive a transmission from a compliant interrogator.

5.2.3 Byte – Eight (8) bits of logical data operated on as a unit.

5.2.4 Collision Arbitration – The process by which two or more simultaneous tag transmissions are resolved by an individual interrogator.

5.2.5 Compatibility – Components (tags, readers) being capable of exchanging bit-level data properly with one another, regardless of type or origin.

5.2.6 Chip – Data coding element used in DSSS broadbanding techniques.

5.2.7 Data Transfer – The process by which a "selected" tag transmits requested data from its memory to an interrogator, or an interrogator transmits data to the tag memory.

5.2.8 Effective Isotropic Radiated Power – The maximum power gain of a transmitting antenna in any direction multiplied by the net power accepted by the antenna from the connected transmitter. Example: 36 dBm EIRP equals 4 @ transmitted into an isotropic antenna or 1 W transmitted into a 6dB antenna.

5.2.9 Family of Tags – A group of tags with differing capabilities which are nevertheless capable of communicating ID numbers and/or data with a common interrogator.

5.2.10 Field of View – The zone surrounding an interrogator in which the interrogator is capable of communicating with the tag.

5.2.11 Forward Link (Downlink) – Communications from interrogator to tag.

5.2.12 Host (System Controller) – An electronic computing device, such as a personal computer, which provides an interface between the user and the non-contact information system. The host is the Master in a master-slave relationship between the host, the interrogator, and the tags in the Field of View of the interrogator.

5.2.13 Interchangeability – Components (tags, readers) of a particular origin being capable of replacing a similar type component of a different origin and operating identically to the replaced component.

5.2.14 Interlace Half Duplex – Full duplex transmissions by the interrogator; half duplex operation by the tag.

5.2.15 Interoperability – Components (tags, readers) being capable of using standard messages to bi-directionally perform complete transactions with one another, regardless of type or origin.

5.2.16 Interrogator (Reader) – A fixed or mobile radio frequency electronic device which manages communications with one or more tags in the non-contact information system. The interrogator includes radio frequency transmit and receive devices, associated antennas, and modulation/demodulation hardware and software. The interrogator may or may not contain an integral host.

5.2.17 Item – The smallest identifiable entity within an application.

5.2.18 Logical Data Bit – A binary digit (bit) containing information. A single element (0-1) in a binary number.

5.2.19 Long – Thirty-two (32) bits of logical data operated on as a unit.

5.2.20 Message Broadcast – The process by which an interrogator sends data or configuration information to all tags in the field of view or to one or more subsets of tags in the field of view, based on a selection criteria.

5.2.21 MfrTagID – A reference number which uniquely identifies the tag.

5.2.22 Name Resolution – The process by which user-defined data fields are mapped to hardware addresses. Example: UPC-code.

5.2.23 Non-Interference – Components (tags, readers) of various types or of different origins co-existing within the same space (in conformance with established reused distance criteria) without having a serious detrimental effect on one another's performance. Does not require that the components communicate with one another as part of a common infrastructure, only that they peacefully co-exist.

5.2.24 Operating Frequency Range – The allowed range of operating frequency at which the RFID system may operate. This range may differ in various national regions.

5.2.25 Power Gain – IN a given direction, the field intensity radiated by the transmitting antenna referenced to the field intensity that would be radiated by an isotropic antenna provided the same input power. Includes efficiency losses, in contrast to directive gain. Does not include losses resulting from polarization mismatch.

5.2.26 Return Link (Uplink) – Communications from tag to interrogator.

5.2.27 Selection – The process by which an interrogator addresses communications to a specific tag or group of tags. A referenced tag is a "selected" tag.

5.2.28 Short – Sixteen (16) bits of logical data operated on as a unit.

5.2.29 Signal Element – Distinctly recognizable signal characteristics that can be interpreted as bit indications.

5.2.30 Tag (Transponder) – A radio frequency electronic device that may be attached to an item. The tag includes read/write memory capable of containing information about the item and electronic circuitry to enable communication of this information to the interrogator.

5.2.31 TagID – The generic reference to either a MfgTagID or UserTagID.

5.2.32 UserTagID – User-defined tag identifier. The UserTagID may not be a unique identifier.

5.2.33 VAR – A variable-length sequence of logical data, whose length is determined by information sent in the command.

5.2.34 Wake-Up – The process by which a tag transitions from a "sleep" (power conservation) state to an "awake" (ready to receive or transmit) state. Tags may be awake on a continuous basis or may be cycled from sleep to awake states.

6 PHYSICAL LAYER PARAMETERS

See Annex A, Clauses A-1 and A-2 for the physical layer parameter of the air interface.

A frequency hopping pattern, F, consists of a permutation of all frequency channels defined in Table 1, in which the channel center frequency is defined in sequential 400 kHz steps beginning with the first channel at 902.6 MHz and ending with the last channel at 927.4 MHz.

Table 6-1: Frequency Channels

Channel number	Frequency in MHz	Channel number	Frequency in MHz	Channel number	Frequency in MHz
1	902.6 MHz	21	910.6 MHz	41	918.6 MHz
2	903.0 MHz	22	911.0 MHz	42	919.0 MHz
3	903.4 MHz	23	911.4 MHz	43	919.4 MHz
4	903.8 MHz	24	911.8 MHz	44	919.8 MHz
5	904.2 MHz	25	912.2 MHz	45	920.2 MHz
6	904.6 MHz	26	912.6 MHz	46	920.6 MHz
7	905.0 MHz	27	913.0 MHz	47	921.0 MHz
8	905.4 MHz	28	913.4 MHz	48	921.4 MHz
9	905.8 MHz	29	913.8 MHz	49	921.8 MHz
10	906.2 MHz	30	914.2 MHz	50	922.2 MHz
11	906.6 MHz	31	914.6 MHz	51	922.6 MHz
12	907.0 MHz	32	915.0 MHz	52	923.0 MHz
13	907.4 MHz	33	915.4 MHz	53	923.4 MHz
14	907.8 MHz	34	915.8 MHz	54	923.8 MHz
15	908.2 MHz	35	916.2 MHz	55	924.2 MHz
16	908.6 MHz	36	916.6 MHz	56	924.6 MHz
17	909.0 MHz	37	917.0 MHz	57	925.0 MHz
18	909.4 MHz	38	917.4 MHz	58	925.4 MHz
19	909.8 MHz	39	917.8 MHz	59	925.8 MHz
20	910.2 MHz	40	918.2 MHz	60	926.2 MHz
				61	926.6 MHz
				62	927.0 MHz
				63	927.4 MHz

7 DATA STRUCTURE

All data structures employed by this standard shall use the semantics defined in ISO/IEC 15418 and the syntax defined in ISO/IEC 15434.

7.1 Mandatory Data Fields

To provide complete uniqueness to the returnable container number, a serial number that is not reused, assigned by the owner of the *contains*, must be combined with the identity of the owner. It is recommended that his uniqueness be provided as shown in Data Identifier "7B" or Application Identifier "8003" or Application Identifier "8004". If DI "1B" employed a second discrete field identifying the container owner ("8B") is required. Further, all returnable containers shall identify the container type and container tare weight.

Identifier	Data field	Data format Type/length	Description *(Examples show encoded characters. Spaces are shown for clarity but are not encoded.)*
DI "1B"	Returnable Container Number	an2+an6	Returnable container identification code assigned by the container owner or the appropriate regulatory agency (e.g., a metal tub, basket, reel, unit load device (ULD), trailer, tank, or intermodal container) (excludes gas cylinders) • Where DI "1B" is employed, "8B" is required. The combination of these two fields, as shown in "7B" is recommended.
DI "7B"	Unique Returnable Container Number	an2+an11	Identification of a returnable container owner assigned in cooperation with BIC, followed by a unique container identification assigned by the container owner, e.g., 8B OC EI CSN CD, where the OC is the owner code assigned in cooperation with BIC, the EI is the equipment category code assigned in cooperation with BIC, the CSN is unique container identification assigned by the equipment owner, and CD is a modulus 11 check digit calculated in accordance with Annex A, ISO 6346. • The Returnable Container Number shall be eleven (11) characters in length, comprised of - owner code (an3) - equipment category identifier (an1) - serial number: six numerals (an6) - check digit: one numeral (an1)
DI "8B"	Returnable Container Number Owner ID	an2+an..35	Identification of a returnable container owner assigned in cooperation with BIC • Where DI "1B" is employed, "8B" is required. The combination of these two fields, as shown in "7B" is recommended.
DI "9B"	Returnable Container Type	an2+an..35	Container Type as defined in (Annex K / ISO xxxxx / ANSI xxxxx)
DI "11Q"	Tare Weight	an3+an..35	Weight of an empty container

AI “8003”	Returnable Container Number	n4+n14+ an...16	The AI 8003 indicates that the data fields contain the Global Returnable Asset Identifier. 5 The EAN/UCC company prefix is the one allocated to the owner of the asset. It is a component of the EAN/UCC numbering structure to make the number unique worldwide. The asset type is a number assigned by the owner of the asset to identify uniquely each type of asset. The zero in the leftmost position is added to generate 14 digits in the field "asset identification number". 6 The check digit is an EAN/UCC Mod 10. Its verification, which must be carried out in the application software, ensures that the number is correctly composed. 7 The serial number is assigned by the owner of the asset. It identifies an individual asset within a given asset type number. The field is alphanumeric and it may contain up to and including 16 characters.
AI “8004”	Global Individual Asset Identifier	n4+n…14+ an...30	The AI 8004 indicates that the data field contains a Global Individual Asset Identifier. 8 The individual asset number uses the EAN/UCC company prefix of the company assigning the asset reference. The structure and numbering of the individual asset reference is determined by the holder of the company prefix. It may contain all characters contained in ISO/IEC 646. 9 The data transmitted from the bar code reader means that the element string Global Individual Asset Identifier has been captured. It may be processed according to the particular application requirements.
AI "908B"	Returnable Container Type	an2+an..35	Container Type as defined in (Annex K / ISO xxxxx / ANSI xxxxx)
AI "9011Q"	Tare Weight	an3+an..35	Weight of an empty container

7.2 Conditional Data Fields

In addition to the mandatory data fields described in Clause 7.1, the following fields are mandatory on all returnable containers/transport units:

- Part number(s) of contained material – Tag encoded
- Quantity – Tag encoded

7.2.1 Part Number Field

The part number data field should be in one of the following formats:

Identifier	Data field	Data characteristics Type/length	Description *(Examples show encoded characters. Spaces are shown for clarity but are not encoded.)*
DI "P"	Part Number	an1+an..25	Customer assigned part number Example: P ROF1234567R3A
DI "1P"	Part Number	an2+an..25	Supplier assigned part number
DI "3P"	UPC product identification	an2+n13	Combined 13-digit manufacturer identification code/item code using UCC/EAN formats (12 digit U.P.C. preceded by "0")
DI "7P"	CLEI code	an2+an7	Common Language Equipment Identifier number structured according to ANSI T1.213 specifications
DI "9P"	DUNS + Product ID	an2+n9+ an..16	Dun and Bradstreet DUNS (9) number followed by Item Identification
N/A	Part Number (Supplier/Item) UPC-12 /EAN-13	n2+n13	UPC-A/EAN-13 Symbology (Combination of Supplier & Item Identification)
AI "01"	GTIN	n2+n14	UCC/EAN SCC-14 (including UPC-A/EAN-13 expressed as a 14 digit field)
AI "241"	Part Number	n3+an...30	Customer Part Number

7.2.2 Supplier Identification Field

The supplier identification field should be in one of the following formats:

Identifier	Data field	Data characteristics Type/length	Description *(Examples show encoded characters. Spaces are shown for clarity but are not encoded.)*
DI "V"	Supplier Code	an1+n35	Supplier Code assigned by Customer
DI "3V"	UCC/EAN Company Prefix	an2+n7	6-digit Company Code as assigned by the Uniform Code Council (UCC) preceded by the digit zero (0) or 7-digit EAN Manufacturer ID
DI "12V"	Manufacturer DUNS	an3+n9	9-digit DUNS Number as assigned by Dun & Bradstreet to identify a manufacturer
DI "13V"	Supplier DUNS	an3+n9	9-digit DUNS Number as assigned by Dun & Bradstreet to identify a supplier
N/A	Part Number (Supplier/Item) UPC-12/EAN-13	n2+n13	UPC-A/EAN-13 Symbology (Combination of Supplier & Item Identification)
AI "01"	SCC-14 (Supplier/Item)	n2+n14	UCC/EAN SCC-14 (including UPC-A/EAN-13 expressed as a 14 digit field)

7.2.3 Quantity Data Field

The quantity data field should be in the following format:

Identifier	Data field	Data characteristics Type/length	Description *(Examples show encoded characters. Spaces are shown for clarity but are not encoded.)*
DI "Q"	Quantity in package	an1+n...14	The number of products (pcs) in the shipment container. Example: Q 2000
DI "2Q"	Actual Weight	an2+n...14	The actual weight of package (kilograms implied by convention) (This includes an encoded decimal point, if necessary)
DI "7Q"	Quantity with Unit of Measure	an2+n..14+an2	The Quantity with ANSI X12.3 Data Element Dictionary qualifier of products in the shipment container. (CR = Cubic Meter) Example: 7Q 1CR (This includes an encoded decimal point, if necessary)
AI "30"	Quantity in package	n2 + n..8	The number of products (pcs) in the shipment container. Example: 30 2000
AI "3nn*" (*) Plus one digit for decimal point indication	Quantity with specific unit of measure (with decimal point indication)	n4 + n6	Defined quantity and unit measure of the package Example: 3101 000025 equals 2.5 kilograms net weight

Note: Print only the significant digits for the human readable quantity. Do not print leading zeros.

7.3 Optional Data Fields

- Date of Last Maintenance
- Date of Next Maintenance
- Product traceability
- Other fields consistent with ISO/IEC 15418

7.3.1 Date of Last Maintenance

When used, the Date of Last Maintenance should be presented in the following format:

Data Identifier	Data field	Data format Type/length	Description *(Examples show encoded characters. Spaces are shown for clarity but are not encoded.)*
5D	Data of Last Maintenance	an2+n6+n3	The 5D identifier is used to identify the date followed by the type of date. - For Date of last maintenance the format shall be 5DYYMMDD150

7.3.2 Date of Next Maintenance

When used, the Date of Next Maintenance should be presented in the following format:

Data Identifier	Data field	Data format Type/length	Description *(Examples show encoded characters. Spaces are shown for clarity but are not encoded.)*
5D	Data of Next Maintenance	an2+n6+n3	The 5D identifier is used to identify the date followed by the type of date. - For Date of last maintenance the format shall be 5DYYMMDD151

7.3.3 Product Traceability Code Field

The product traceability code field must be in the following formats:

Identifier	Data field	Data format Type/length	Description *(Examples show encoded characters. Spaces are shown for clarity but are not encoded.)*
DI "S"	Serial Number	an1+an..18	Serial number Example: S A21567890
DI "1T"	Lot / Batch Number	an2+an..18	Example: 1T A21567890
AI "21"	Serial number	n2 + an..20	Serial number Example: 21 MH80312
AI "10"	Lot / Batch Number	n2 + an..20	Lot / Batch Number Example: 10 110780

Note on "serial number:" The format for the serial number is to be defined by the manufacturer.

7.4 In-Use Cable Reels

7.4.1 When Using ANSI MH10.8.2 Data Identifiers for the Identification of In-Use Cable Reels

The machine-readable manifest format accommodates both mandatory and optional data elements.

The number included in character count is exclusive of overhead characters such as start and stop characters, data identifiers or application identifiers and any other characters required by a standard symbology specification to properly encoded data.

All data elements so encoded in a machine-readable medium shall be preceded by the appropriate Data Identifier (DI) defined in ISO 15418 and ANSI MH10.8.2 *Data Identifiers*.

7.4.1.1 Mandatory Data Elements

- Cable Reel ID (Container ID)
- Part Number(s)
- Original Cable Length (Initial Quantity with Unit of Measure)
- Empty Reel Return Date (Expiration Date) (Container Return Date)

7.4.1.1.1 Cable Reel ID

The cable reel ID is assigned by the owner of the cable reel. The cable reel ID shall conform with TCIF-BCC-95-004 RI - Guidelines for the Identification and Bar Code Labeling of Cable Reels. The cable reel ID is preceded by the Data Identifier "IB". The cable reel ID is comprised of a two-character alpha owner code followed by a six-character unique alphanumeric serial number assigned by the owner.

7.4.1.1.2 Item Identification

Item Identification may be assigned by either the Supplier or the Customer. Either the Customer Item Identification or the Supplier Item Identification or both may be encoded on the manifest as agreed to between the trading partners.

It is recommended that the Item Identification assigned by the customer be the same as the item identification used on the purchase order.

In the absence of an agreement between trading partners, the supplier's part number is to be used for Item Identification.

The maximum length of this data element is 25 alphanumeric characters.

The item identification field should be in one of the following formats:

Data Identifier	Data Field	Data characteristics Type/length	Description
P	Part Number	an1+an...25	Customer Assigned Part Number
1P	Part Number	an2+an...25	Supplier Assigned Part Number
8P	Part Number	an2+n14	EAN.UCC GTIN
11P	Part Number	an3+an10	CLEI Code for telecommunications equipment
17P	Part Number	an3+an7+an...25	Combined UCC supplier identification and item code assigned by the supplier

7.4.1.1.3 Original Cable Length

The Original Cable Length is the length of cable provided by the supplier immediately prior to shipment. The original cable length shall be expressed in feet. The Original Cable Length shall be preceded by the Data Identifier "7Q" and followed by the unit of measure "FT". When length expressed is a partial foot, it will be expressed as a decimal. If used, the decimal point shall be encoded in the data. The Original Cable Length shall not exceed 10 characters.

7.4.1.1.4 Empty Reel Return Date

The Empty Reel Return Date is the date when the cable reel shall be returned to the owner. The Empty Reel Return Date shall be preceded by the Data Identifier "14D". The format date shall be YYYYMMDD.

7.4.1.2 Optional Data Elements When Using Data Identifiers

- Owner's Name
- Call-To-Return Phone Number (e.g., 800-RET-CABL)
- Starting Length (IN)
- Ending Length (OUT)
- Current Cable Length
- Work Order Number
- Return to Storage Location

7.4.1.2.1 Owner's Name

The Owner's Name is spelled out name of the owner of the cable reel, e.g., Verizon, General, etc. The Owner's Name data field shall be preceded by the Data Identifier "18V". The Owner's Name shall not exceed 25 characters.

7.4.1.2.2 Call-To-Return Phone Number

The Call-To-Return Phone Number is the telephone number to call to inform the owner that a cable reel is being returned. The Call-To-Return Phone Number data field shall be preceded by the Data Identifier "20L". The Call-To-Return Phone Number shall not exceed 15 characters and shall include the area code (toll free code) and if applicable, the country and city code.

7.4.1.2.3 Starting Length (IN)

The Starting Length (IN) is the length of cable resultant from the reels prior use. The Starting Length (IN) unit of measure shall be implied to be measured in feet. The Starting Length (IN) shall be preceded by the Data Identifier "14Q". When length expressed is a partial foot, it will be expressed as a decimal. If used, the decimal point shall be encoded in the data. The Starting Length (IN) shall not exceed 10 characters.

7.4.1.2.4 Ending Length (OUT)

The Ending Length (OUT) is the length of cable resultant from the reels current use. The Ending Length (OUT) unit of measure shall be implied to be measured in feet. The Ending Length (OUT) shall be preceded by the Data Identifier "15Q". When length expressed is a partial foot, it will be expressed as a decimal. If used, the decimal point shall be encoded in the data. The Ending Length (OUT) shall not exceed 10 characters.

7.4.1.2.5 Current Cable Length

The Current Length is the length removed from the reel. The Current Cable Length unit of measure shall be implied to be measured in feet. The Current Cable Length shall be preceded by the Data Identifier "Q". When length expressed is a partial foot, it will be expressed as a decimal. If used, the decimal point shall be encoded in the data. The Current Cable Length shall not exceed 10 characters.

7.4.1.2.6 Work Order Number

The Work Order Number is the company document authorizing the use of the cable. The Work Order Number shall be preceded by the Data Identifier "W". The Work Order Number shall not exceed 15 characters.

7.4.1.2.7 Return to Storage Location

The Return to Storage Location is the organizational storage location where the cable reel is to be returned following its use. The Return to Storage Location shall be preceded by the Data Identifier "L". The Return to Storage Location shall not exceed 10 characters.

7.4.2 When Using EAN.UCC Application Identifiers for the Identification of In-Use Cable Reels

The number included in character count is exclusive of overhead characters such as start and stop characters, data identifiers or application identifiers and any other characters required by a standard symbology specification to properly encoded data.

7.4.2.1 Mandatory Data Elements

- Cable Reel ID
- Part Number
- Original Cable Length
- Empty Reel Return Date (Expiration Date)

7.4.2.1.1 Cable Reel ID

The cable reel ID is assigned by the owner of the cable reel. The cable reel ID shall conform with TCIF-BCC-95-004 R1 – Guidelines for the Identification and Bar Code Labeling of Cable Reels. The cable reel ID is preceded by the Data Identifier "901B". The cable reel ID is comprised of a two-character alpha owner code followed by a six-character unique alphanumeric serial number assigned by the owner.

7.4.2.1.2 Item Identification

Item Identification may be assigned by either the Supplier or the Customer. Either the Customer Item Identification or the Supplier Item Identification or both may be encoded on the manifest as agreed to between the trading partners.

It is recommended that the Item Identification assigned by the customer be the same as the item identification used on the purchase order.

In the absence of an agreement between trading partners, the supplier's part number is to be used for Item Identification.

The maximum length of this data element is 25 alphanumeric characters.

The item identification data field should be in one of the following formats:

Application Identifier	Data Field	Data characteristics Type/length	Description
01	GTIN	N2+n14	Supplier Assigned EAN.UCC Product Code
241	Customer Part Number	an2+an...25	Customer Assigned Part Number
9011P	Part Number	N2+an3+an10	CLEI Code for telecommunications equipment

7.4.2.1.3 Original Cable Length

The Original Cable Length is the length of cable provided by the supplier immediately prior to shipment. The original cable length shall be expressed in feet. The Original Cable Length shall be preceded by the Application Identifier "907Q" and followed by the unit of measure "FT". When length expressed is in partial foot, it will be expressed as a decimal. If used, the decimal point shall be encoded in the data. The Original Cable Length shall not exceed 10 characters.

7.4.2.1.4 Empty Reel Return Date

The Empty Reel Return Date is the date when the cable reel shall be returned to the owner. The Empty Reel Return Date shall be preceded by Application Identifier "9014D". The format of date shall be YYYYMMDD.

7.4.2.2 Optional Data Elements for Cable Reels When Using Application Identifiers

- Owner's Name
- Call-To-Return Phone Number (e.g., 800-RET-CABL)
- Starting Length (IN)
- Ending Length (OUT)
- Current Cable Length
- Work Order Number
- Return to Storage Location

7.4.2.2.1 Owner's Name

The Owner's Name is spelled out name of the owner of the cable reel, e.g., Verizon, General, etc. The Owner's Name data field shall be preceded by the Application Identifier "9018V". The Owner's Name shall not exceed 25 characters.

7.4.2.2.2 Call-To-Return Phone Number

The Call-To-Return Phone Number is the telephone number to call to inform the owner that a cable reel is being returned. The Call-To-Return Phone Number data field shall be preceded by the Application Identifier "9020L". The Call-To-Return shall not exceed 15 characters and shall include the area code (toll free code) and if applicable, the country and city code.

7.4.2.2.3 Starting Length (IN)

The Starting Length (IN) is the length of cable resultant from the reels prior use. The Starting Length (IN) unit of measure shall be implied to be measured in feet. The Starting Length (IN) shall be preceded by the Application Identifier "9014Q". When length expressed is a partial foot, it will be expressed as a decimal. If used, the decimal point shall be encoded in the data. The Starting Length (IN) shall not exceed 10 characters.

7.4.2.2.4 Ending Length (OUT)

The Ending Length (OUT) is the length of cable resultant from the reels current use. The Ending Length (OUT) unit of measure shall be implied to be measured in feet. The Ending Length (OUT) shall be preceded by the Application Identifier "9015Q". When length expressed is a partial foot, it will be expressed as a decimal. If used, the decimal point shall be encoded in the data. The Ending Length (OUT) shall not exceed 10 characters.

7.4.2.2.5 Current Cable Length

The Current Cable Length is the length removed from the reel. The Current Cable Length unit of measure shall be implied to be measured in feet. The Current Cable Length shall be preceded by the Application Identifier "90Q". When length expressed is a partial foot, it will be expressed as a decimal. If used, the decimal point shall be encoded in the data. The Current Cable Length shall not exceed 10 characters.

7.4.2.2.6 Work Order Number

The Work Order Number is the company document authorizing the use of the cable. The Work Order Number shall be preceded by the Application Identifier "90W". The Work Order Number shall not exceed 15 characters.

7.4.2.2.7 Return to Storage Location

The Return to Storage Location is the organizational storage location where the cable reel is to be returned following its use. The Return to Storage Location shall be preceded by the Application Identifier "90L". The Return to Storage Location shall not exceed 10 characters.

8 TAG PLACEMENT

8.1 General Considerations

RF tags should be affixed at a suitable location where there is a minimum risk of damage.

Returnable containers should have identical RF tags affixed on two adjacent sides.

RF Tags shall be placed (embedded) 2 centimeters from the natural bottom of the returnable container.

8.2 Affixing Tags to Metal Containers

When affixed to metal containers, tags need to be have a separation of a specific distance from the metal surface to the tag insert. For the frequencies defined in this standard the spacer should be between 8 mm and 15 mm in thickness. The spacer can be made of polyurethane, polyethylene, polycarbonate, or other materials having a dielectric with a value of approximately "1". An increase in range can be achieved if the spacer is backed with a smooth, flat metal material (metal foil) and/or by doping the dielectric.

8.3 Tag Location

Annex D provides examples of the location for RF tags on various returnable containers.

ANNEX A

PHYSICAL PARAMETERS UHF AIR INTERFACE

For the purpose of this standard, the following parameter definitions apply. These parameters are referenced by both the parameter number and the parameter name. Note: Not all forward or return physical layer options reference every Forward or Return Link Parameter.

A.1 Forward Link Parameters

Table A-1: Physical Layer Parameters

Parameter Number	Parameter Name	Description
F 1	Operating Frequency Range	Range of frequencies over which the communications link will operate.
F 1a	Default Operating Frequency	Operating frequency at which the interrogator and tag establish communications. The value shown is the center frequency of the modulated signal. All compliant tags and interrogators must support operation at the default operating frequency.
F 1b	Operating Channels	Number and value of the forward link operating frequencies. The values provided are the center frequencies of the modulated signals.
F 1c	Operating Frequency Accuracy	Maximum deviation of the carrier frequency from the specified nominal frequency, expressed in ppm. Example: 1 ppm of a 2450 MHz carrier allows the carrier frequency to be in the range of 2450 MHz ± 2.45 kHz.
F 1d	Frequency Hop Rate	The inverse of the dwell time at an FHSS center frequency.
F 1e	Frequency Hop Sequence	A pseudo-randomly ordered list of hopping frequencies used by the FHSS transmitter to select an FHSS channel.

Parameter Number	Parameter Name	Description
F 2	Occupied Channel Bandwidth	The bandwidth of the communications signal occupying a specified channel. This bandwidth is not equivalent to channel spacing, although the channel spacing could equal the occupied channel bandwidth. (Allowed channel spacing for FHSS systems is regulated by FCC Part 15, section 15.247: the channel spacing must be the greater than or equal to the 20dB bandwidth of the signal, between the limits of 25 kHz and 1 MHz.) The occupied channel bandwidth may be narrower than the channel spacing to allow for frequency tolerance or to provide for other guard bands necessary for reliable communication links. For FHSS and narrowband operation, the occupied channel bandwidth is the maximum allowed 20 dB bandwidth of the modulated signal in an occupied channel. For DSSS operation, the occupied channel bandwidth is the maximum allowed null-to-null bandwidth (frequency difference between the main lobe nulls) of the DSSS signal in an occupied channel.
F 3	Interrogator Transmit Maximum EIRP	The maximum EIRP transmitted by the interrogator antenna, expressed in dBm.
F 4	Interrogator Transmit Spurious Emissions	Undesired frequency outputs, including harmonics, intermodulation products, cross modulation, and parasitic emission transmitted by the interrogator.
F 4a	Interrogator Transmit Spurious Emissions, In-Band	Spurious emissions that occur within the allowed range of carrier frequencies.
F 4b	Interrogator Transmit Spurious Emissions, Out-of-Band	Spurious emissions that occur outside the allowed range of carrier frequencies.
F 5	Interrogator Transmitter Spectrum Mask	Maximum power (density) emitted by a interrogator transmitter as a function of the frequency.
F 5a	Transmit to Receive Turn Around Time	The maximum time after the tag has completed transmission of a reply to an interrogation until the time the tag is ready to receive another interrogation.
F 5b	Receive to Transmit Turn Around Time	The maximum time after the tag has completed reception of an interrogation until the tag begins a reply transmission.
F 5c	Interrogator Transmit Power On Ramp	The maximum time required for the interrogator transmit power to increase from 10% to 90% of the steady-state transmit output power level.
F 5d	Interrogator Transmit Power Down Ramp	The maximum time required for the interrogator transmit power to decrease from 90% to 10% of the steady-state transmit output power.

Parameter Number	Parameter Name	Description
F 6	Modulation	Keying of the carrier wave by coded data. Examples: Amplitude Shift Keying (ASK), Phase Shift Keying (PSK) and Frequency Shift Keying (FSK).
F 6a	Spreading Sequence	The sequence of data coding elements (chips) used to encode each logical data bit.
F 6b	Chip Rate	The frequency at which the spreading sequence modulates the carrier.
F 6c	Chip Rate Accuracy	The allowed variation in chip rate, expressed in ppm.
F 6d	On-Off Ratio	For ASK modulation (including OOK): the ratio of peak amplitude to minimum amplitude of the ASK modulated signal.
F 6e	Duty Cycle	For OOK modulation: The ratio of ON period to OFF period.
F 6f	FM Deviation	For FM modulation: The peak difference between the instantaneous frequency of the modulated wave and its carrier frequency.
F 7	Data Coding	Baseband signal presentation, i.e. a mapping of logical bits to physical signals. Examples: two level schemes, NRZ and NRZI; and bi-phase schemes, Manchester and FM0.
F 8	Bit Rate	Number of logical bits per second, independent of the data coding.
F 8a	Bit Rate Accuracy	Maximum deviation of the bit rate from the specified nominal bit rate, expressed in ppm.
F 9	Interrogator Transmit Modulation Accuracy	The peak vector error magnitude measured during each chip period.
F 10	Tag Receiver Non-Destructive Input RF Level	The maximum input power level that can occur at the tag without causing damage to the tag.
F 11	Preamble	Specific Layer 1 address, independent of Layer 2. Ether an unmodulated carrier wave or a modulated carrier, in which case the requirement refers to the channel after coding.
F 11a	Bit Sync Sequence	A series of bits generated by the physical layer that a receiver uses to synchronize to the incoming bit stream.
F 11b	Frame Sync Sequence	A series of bits generated by the physical layer that indicates the start of a data link layer (Layer 2) message packet.
F 12	Scrambling	An operation performed on all bits transmitted by the physical layer for the purposes of bit timing generation and improving spectral quality.
F 13	Bit Transmission Order	

A.2 Return Link Parameters

Table A-2: Return Link Parameters

Parameter Number	Parameter Name	Description
R 1	Operating Frequency Range	Range of frequencies over which the communications link will operate.
R 1a	Default Operating Frequency	Operating frequency at which the interrogator and tag establish communications. The value shown is the center frequency of the modulated signal. All compliant tags and interrogators must support operation at the default operating frequency.
R 1b	Operating Channels	Number and value of the return link operating frequencies. The values provided are the center frequencies of the modulated signals.
R 1c	Operating Frequency Accuracy	Maximum deviation of the carrier frequency from the specified nominal frequency, expressed in ppm. Example: 1 ppm of a 2450 MHz carrier allows the carrier frequency to be in the range of 2450 MHz ± 2.45 kHz.
R 1d	Frequency Hop Rate	The inverse of the dwell time at an FHSS center frequency.
R 1e	Frequency Hop Sequence	A pseudo-randomly ordered list of hopping frequencies used by the FHSS transmitter to select an FHSS channel.
R 2	Occupied Channel Bandwidth	The bandwidth of the communications signal occupying a specified channel. For FHSS and narrowband operation, this bandwidth is the maximum allowed 20 dB bandwidth of the modulated signal in an occupied channel. For DSSS operation, this bandwidth is the maximum allowed null-to-null bandwidth (frequency difference between the main lobe nulls) of the DSSS signal in an occupied channel.
R 3	Transmit Maximum EIRP	The maximum EIRP from the transmitting antenna, expressed in dBm.
R 4	Transmit Spurious Emissions	Undesired frequency outputs, including harmonics, intermodulation products, cross modulation, and parasitic emission from the transmitter.
R 4a	Transmit Spurious Emissions, In-Band	Spurious emissions that occur within the allowed range of carrier frequencies.
R 4b	Transmit Spurious Emissions, Out-of-Band	Spurious emissions that occur outside the allowed range of carrier frequencies.
R5	Transmit Spectrum Mask	Maximum power (density) emitted by the transmitter as a function of frequency

Parameter Number	Parameter Name	Description
R 5a	Transmit to Receive Turn Around Time	The maximum time after the tag has completed transmission of a reply to an interrogation until the time the tag is ready to receive another interrogation.
R 5b	Receive to Transmit Turn Around Time	The maximum time after the tag has completed reception of an interrogation until the tag begins a reply transmission.
R 5c	Transmit Power On Ramp	The maximum time required for the transmit power to increase from 10% to 90% of the steady-state transmit output power level.
R 5d	Transmit Power Down Ramp	The maximum time required for the transmit power to decrease from 90% to 10% of the steady-state transmit output power.
R 6	Modulation	Keying of the carrier with coded data.
R 6a	Sub-carrier Frequency	Number and values of the return link sub-carrier frequency. Sub-carrier frequencies are described as the frequency distance of the center of the return link band to the corresponding forward link carrier, i.e. to the center of the corresponding forward link band.
R 6b	Sub-carrier Frequency Accuracy	Maximum deviation of the sub-carrier frequency, expressed in ppm of the sub-carrier frequency.
R 6c	Sub-Carrier Modulation	Keying of the subcarrier with coded data.
R 7	Data Coding	Baseband signal presentation, i.e. a mapping of logical bits to physical signals.
R 7a	Spreading Sequence	The sequence of data coding elements (chips) used to encode each logical data bit.
R 7b	Chip Rate	The frequency at which the spreading sequence modulates the carrier.
R 7c	Chip Rate Accuracy	The allowed variation in chip rate, expressed in ppm.
R 6d	On-Off Ratio	For ASK modulation (including OOK): the ratio of peak amplitude to minimum amplitude of the ASK modulated signal.
R 6e	Duty Cycle	For OOK modulation: The ratio of ON period to OFF period.
R 6f	FM Deviation	For FM modulation: The peak difference between the instantaneous frequency of the modulated wave and its carrier frequency.
R 8	Bit Rate	Number of logical bits per second, independent of the data coding.
R 8a	Bit Rate Accuracy	Maximum deviation of the bit rate from the specified nominal bit rate, expressed in ppm.
R 9	Tag Transmit Modulation Accuracy	The peak vector error magnitude measured during each chip period.

Parameter Number	Parameter Name	Description
R 10	Preamble	Specific Layer 1 address, independent of Layer 2. Either an unmodulated carrier wave or a modulated carrier, in which case the requirement refers to the channel after coding.
R 10a	Bit Sync Sequence	A series of bits generated by the physical layer that a receiver uses to synchronize to the incoming bit stream.
R 10b	Frame Sync Sequence	A series of bits generated by the physical layer that a receiver uses to synchronize to the incoming bit stream.
R 11	Scrambling	An operation performed on all bits transmitted by the physical layer for the purposes of bit timing generation and improving spectral quality.
R 12	Bit Transmission Order	

A.3 ANSI MH10.8.4 Physical Link Specifications

Table A-3 lists the physical link specifications form the reader to the tag (forward link).

Table A-3: Physical Link Specifications – Forward Link

Item	Parameter	Value
F1	Operating Frequency Range	902 - 928 MHz
F 1b	Operating Channels	63 channels centered at 902.6MHz to 927.4MHz with a spacing of 0.4 MHz
F 1c	Operating Frequency Accuracy	± 50 ppm maximum
F 1d	Frequency Hop Rate	If applicable, defined by reader in accordance to regulatory requirements
F 1e	Frequency Hop Sequences	If applicable, a pseudo-random sequence defined by reader and in accordance to regulatory compliance.
F 2	Occupied Channel Bandwidth	Less than 400kHz using 20dB bandwidth
F 3	Interrogator Transmit Maximum EIRP	Minimum of 4W (36 dBm) EIRP from the interrogator transmit antenna
F 4b	Interrogator Transmit Spurious Emissions, Out of Band	As required
F 5a	Receive to Transmit Turn Around Time	< 1 ms
F 5c	Interrogator Transmit Power-On Ramp	< 5 % of bit period
F 5d	Interrogator Transmit Power-Down Ramp	< 5 % of bit period
F 6	Modulation	On-off Keying (OOK)
F 6d	On/Off Ratio	> 40 dBc
F 6e	Duty Cycle	50% ± 5%
F 7	Data Coding	Manchester
F 8	Bit Rate	20–40 kbps
F 10	Tag Receiver Non-Destructive Input RF Level	≤ +13 dBm, in band
F 11	Preamble	See section A.4.2.3.2

Table A-4 lists the physical link specifications from the tag to the reader (backscatter return link).

Table A-4: Physical Link Specifications – Backscatter Return Link

Item	Parameter	Value
R1	Operating Frequency Range	As per F1
R 1b	Operating Channels	As per F1b
R 1c	Operating Frequency Accuracy	± 50 ppm maximum
R 1d	Frequency Hop Rate	As per F1d
R 1e	Frequency Hop Sequence	As per F1e
R 2	Occupied Channel Bandwidth	400 kHz using 20 dB bandwidth definition
R 3	Transmit Maximum EIRP	4W (36 dBm) EIRP from the interrogator transmit antenna
R 4b	Transmit Spurious Emissions, Out of Band	As required
R 5a	Receive-Transmit Turn Around Time	< 1 ms
R 6	Modulation	On-off keying (OOK)
R 6e	Duty Cycle	50% ± 5%
R 7	Data Coding	FM0
R 8	Bit Rate	Bit rate of return link should match permissible bit rate of forward link with less than 15% deviation.
R 11	Preamble	See section A.4.2.3.7

A.4 Protocol Description – Passive Backscatter RFID System

This part defines the RFID command/data level communication protocols. These protocols facilitate communication between compliant tags and compliant interrogators. The timing parameters and signal characteristics for the protocols are defined in the physical link specifications appearing below.

This portion of the standard describes a passive backscatter RFID system that supports the following system capabilities:

- System Protocol
  - Identify and communicate with multiple tags in the field
  - Select a subgroup of tags to identify or communicate with based on information that the user has stored in the tag
  - Read from and write data many times to individual tags

- Data Integrity Protection
  - Manchester bit-wise encoding and CRC-16 packet-level protection is applied to the forward link (reader to tag) data.
  - FM0 bit-wise encoding and CRC-16 packet-level protection is applied to the return link (tag to reader) data.

In this RFID system, readers both power and communicate with the tags that are within their range. Tags receive data as On-Off key amplitude modulation of the power/data signal from the reader. During the period of time that the tag communicates back to the reader, the reader broadcasts a steady RF power level, and the tag modulates the impedance of its RF load attached to the tag antenna terminals. The reader then receives the data back from the tag as a variation in a reflection of its transmitted power.

A.4.1 Functional Description

This section is divided into two parts:

- Introduction – provides a general overview of the passive backscatter RFID system functions
- RFID Tag Command Set – lists the tag command definitions, both high level descriptions and the bit patterns assigned to each command

The details of the communication protocol itself, such as start delimiters, preambles, data encoding techniques, and so forth, are discussed in section A.4.2.

A.4.1.1 Introduction

The backscatter option 2 RFID system includes a base station (interrogator) that runs the backscatter option 2 RFID protocol, as well as one or more tags. The tag itself includes a chip, an antenna tuned to the carrier frequency of the interrogator, and a package to hold the chip and antenna together.

When placed in the radio frequency field of an interrogator, the tag will begin to power up. If the field is strong enough (section A.4.2), the tag integrated circuit (IC) will execute a power-on reset and be ready to receive a WAKEUPMH10 command sequence. Each command begins with a preamble and start delimiter. Data to and from the tag is checked for errors using a CRC; therefore, CRC fields are present in all base station interrogations and in all tag responses. Additional data protection is provided by Manchester encoding on the forward link (reader-to-tag) and FM0 encoding on the return link (tag-to-reader).

By using the backscatter option 2 RFID command set, the interrogator can execute a number of functions on tags in its field.

A.4.1.2 RFID Command Set

A given command must include, at a minimum, the following elements (see figure A-1): reader-to-tag preamble detect field; reader-to-tag preamble; reader-to-tag start delimiter; command field; and CRC-16 field. It may also include subsets of the following, depending on the command: tag identification field; byte mask; address; byte data; and 8-byte word data. Tags may respond to commands, where all responses include a quiet time, a return preamble, either return data or an acknowledgment code, and CRC-16. These fields are described in detail in section A.4.2.

The description of the RFID tag command set in the following sections provides detail regarding the command field and return data/acknowledgment fields, if any. In addition, it covers additional high-level elements of the backscatter option 2RFID protocol, including how the multiple item identification algorithm works and byte ordering requirements. The more general aspects of the protocol (e.g., preambles, CRC-16) are covered in detail in section A.4.2.

A command sequence is a combination of individual commands, and a command sequence must be performed in entirety, i.e. individual commands of a command sequence cannot be executed by itself.

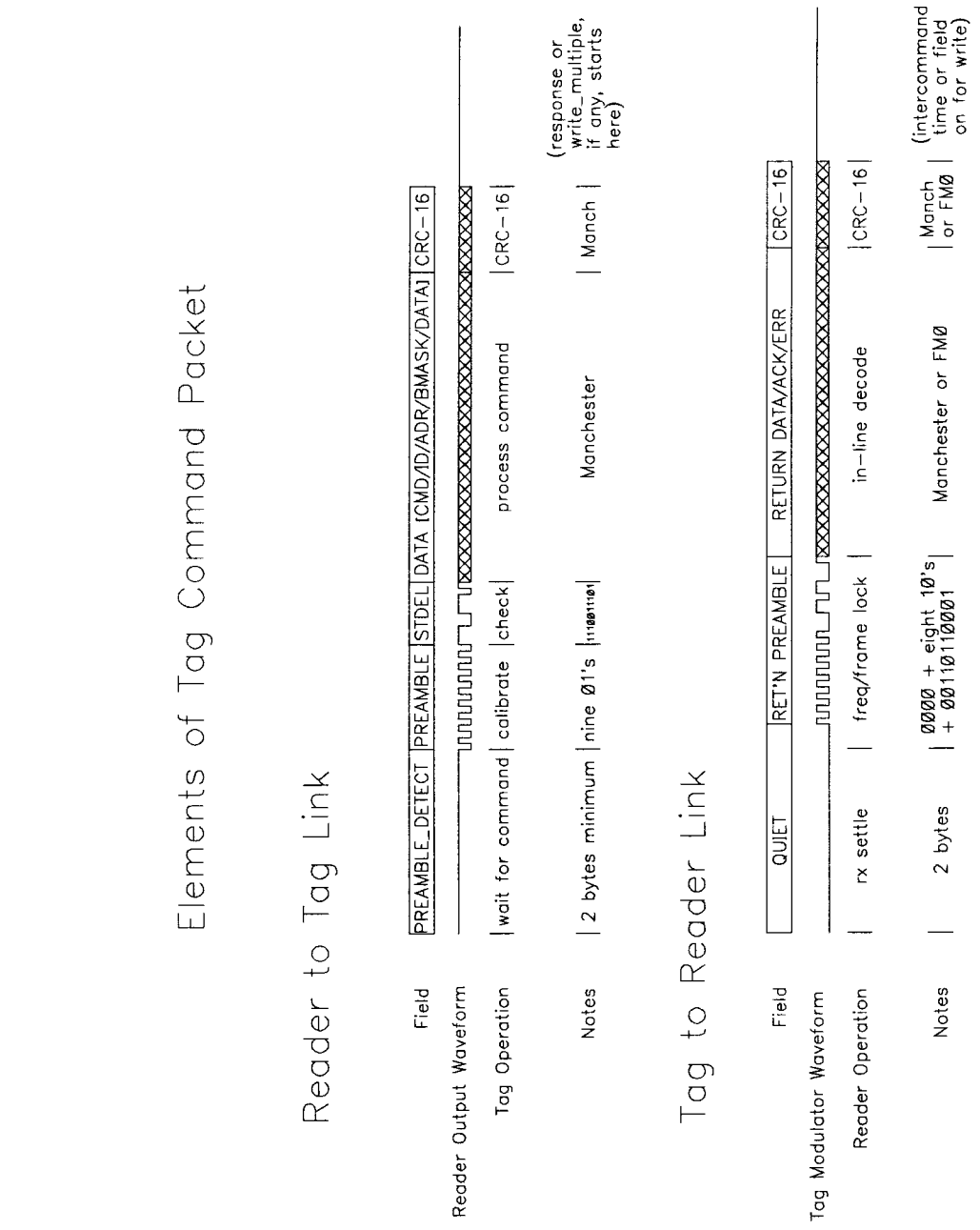


Figure A-1: Elements of Tag Command Packet

A.4.1.2. Command Types

Tag commands can be divided functionally into three groups.

- Wake-up command sequence for initialization of MH10 tags:

  WAKEUPMH10

- Identification commands to run the multiple tag identification protocol:

  FAIL
  SUCCESS
  RESEND
  DATA_READ

- Data Transfer commands read or write to Electronically Erasable, Programmable Read Only Memory (EEPROM) data:

  DATA_READ
  WRITE

A.4.1.2.2 Summary of Commands (Base Station to Tag)

FAIL			
SUCCESS			
RESEND			
WAKEUPMH10 (sequence)			
DATA_READ	ID	ADDRESS	
WRITE	ID	ADDRESS	BYTE_DATA

A.4.1.2.3 Summary of Responses (Tag to Base Station)

ACKNOWLEDGE	(write)
ERROR	(write)
WORD_DATA	(fail, success, resend, data_read)

A.4.1.2.4 Field Lengths

command	1 byte
address	1 byte
id	8 bytes
word_data	8 bytes
acknowledge	1 byte
error	1 byte

A.4.1.3 Address Map

There are 70 addressable locations, each containing one 8-bit data byte and an associated lock bit. The only hard coded addresses are 0-7, which contain the tag ID. Bytes 8 and 9 are reserved and bytes 10-17 are intended for information on tag capabilities, manufacturer ID number, and memory organization. Byte 12 of the Address Map defines Embedded Application Codes. Embedded Application Code "0B" (Binary "0000 1011") is used to define that the structure complies with the rules of this standard, those of ANSI MH 10/SC 8, and those of ISO TC 122.

A.4.1.4 Tag Major States

The tag has two major states:

ID	Tag is trying to identify itself to the base station.
DATA_EXCHANGE	Tag is known to the base station.

A.4.1.5 Command Description (Basic)

A.4.1.5.1 FAIL

The identification algorithm uses FAIL when more than one tag tried to identify itself at the same time. Some tags back off and some tags retransmit according to an algorithm described in section A.4.1.8. A maximum of 7 successive and consecutive FAIL commands can be transmitted between WAKEUPMH10 command sequences without an intervening SUCCESS command.

A.4.1.5.2 SUCCESS

SUCCESS initiates identification of the next set of tags. It is used in two cases:

- When all tags receiving FAIL backed off and did not transmit, SUCCESS causes those same tags to transmit again.
- After a DATA_READ moves an identified tag to DATA_EXCHANGE, SUCCESS causes the next subset of selected but unidentified tags to transmit.

A.4.1.5.3 RESEND

The identification algorithm uses RESEND when only one tag transmitted but the ID was received in error. The tag that transmitted resend its ID.

A.4.1.5.4 WAKEUPMH10 (command sequence)

WAKEUPMH10 instructs tags to participate in an identification algorithm. The WAKEUPMH10 command sequence is a combination of two or four command packets, and the second and (optional) subsequent command packets follows the first command packet. The data contents of the Reader to Tag (see Figure A-1) of the first command packet is one byte long, and the data contents of the second and (optional) subsequent packets are 11 bytes long. The WAKEUPMH10 command cannot be sent more frequently than once every 200 msec.

A.4.1.5.5 DATA_READ

From the ID or DATA_EXCHANGE state, DATA_READ reads the specified address of the specified tag and moves the tag to the DATA_EXCHANGE state. As part of the DATA_READ command, an address is specified, and the only valid addresses, in decimal notation, are 0, 10, and 18-118.

A.4.1.5.6 WRITE

From any state, WRITE writes to the specified address of the specified tag with the specified data byte and moves the tag to the DATA_EXCHANGE state. As part of the WRITE command, an address is specified, and the only valid addresses, in decimal notation, are 13-125.

A.4.1.6 Response Description (Basic)

A.4.1.6.1 ACKNOWLEDGE

ACKNOWLEDGE indicates a successful WRITE.

A.4.1.6.2 ERROR

ERROR indicates an error in the WRITE.

A.4.1.6.3 WORD_DATA

WORD_DATA is 8 bytes returned in response to a WAKEUPMH10, FAIL, SUCCESS, RESEND, READ, or DATA_READ command.

A.4.1.7 Transmission Errors

There are two types of transmission errors: modulation coding errors (detectable per bit) and CRC errors (detectable per command). Both errors cause any command to abort. The tag does not respond. For all CRC errors, the tag returns to the READY state. For all coding errors, the tag returns to the READY state if a valid start delimiter had been detected. Otherwise, it maintains its current state.

A.4.1.8 Identification Algorithm Application Note

The algorithm uses a WAKEUPMH10 command sequence to initialize tags to participate in the identification protocol. It then uses the identification commands to run the algorithm.

After initialization, the following loop is performed:

1. All tags in the ID state with the counter at 0 transmit their ID. This set initially includes all the tags.

2. If more than one tag transmitted, the base station receives an erroneous response. The FAIL command is sent. Up to seven successive and consecutive FAIL commands can be sent after a WAKEUPMH10 command sequence without an intervening SUCCESS command.

    - FAIL causes all tags with a count not equal to 0 to increment their counter. That is, they move further away from being able to transmit.

- FAIL causes all tags with a count of 0 (those that just transmitted) to generate a random number. Those that roll a 1 increment their counter and do not transmit. Those that roll a zero keep the counter at zero and try again.

One of four possibilities now occurs:

3. If more than one tag transmits, the FAIL step 2 repeats. (Possibility 1)

4. If all tags roll a 1, none transmits. The base station receives nothing. It sends the SUCCESS command. All the counter decrement, and the tags with a count of 0 transmit. Typically, this returns to step 2. (Possibility 2)

5. If only one tag transmits and the ID is received correctly, the base station sends the DATA_READ command with the ID. If the DATA_READ command is received correctly, that tag moves to the DATA_EXCHANGE state and transmits its data.

   The base station next sends SUCCESS, causing all other tags to decrement their counters. If only one tag has a count of 1 and transmits, step 5 or 6 repeats. If more than one tag transmits, step 2 repeats. (Possibility 3)

6. If only one tag transmits and the ID is received with an error, the base station sends the RESEND command. If the ID is received correctly, step 5 repeats. If the ID is received some variable number of times (this number can be set based on the level of error handling desired for the system), it is assumed that more than one tag is transmitting, and step 2 repeats. (Possibility 4)

A.4.1.9 Bit and Byte Ordering

In all byte fields, the MSB is transmitted first, proceeding to the LSB. In all word (8-byte) data fields, the MSB is transmitted first.

The MSB is the byte at the specified address. The LSB is the byte at the specified address plus 7 (i.e., bytes are transmitted in incrementing address order).

There is no requirement that word (8-byte) addresses must be on an 8-word boundary.

A.4.1.10 Command and Response Codes (hex)

Commands and Command Sequences

WAKEUPMH10	1st command packet: 0x0a
	2nd command packet, in hex: 00 0a 20 00 00 0b 00 00 00 00 00
	3rd command packet, in hex: 02 00 (00 through ff) XX XX XX XX XX XX XX XX
	4th command packet, in hex: 03 00 (00 through ff) XX XX XX XX XX XX XX XX
FAIL	0x08
SUCCESS	0x09
DATA_READ	0x0b plus applicable parameters
WRITE	0x0d plus applicable parameters

Responses

ACKNOWLEDGE	0x00
ERROR	0xff

Note: The 3rd and 4th command packets are optional. The value defined XX is user defined and may be used to activate a group of tags based on the contents of memory locations 0.00 through 0.07 (i.e. addresses 0 through 7). As noted in section A.4.1.3, these addresses are hard coded with a value that is unique to each tag (unique ID). As such, tag activation may be selected based on a range of tag ID's through the use of the optional 3rd and 4th command packets as part of the WAKEUPMH10 command. The 3rd command packet sets the lower limit (i.e. greater than) while the 4th command packet sets the upper limit (i.e. less than).

The third byte of the optional command packets (packets 3 and 4) includes a mask that defines which bytes are to be used in tag activation. Each bit of the third byte corresponds to one of the bytes to be compared. Thus, the most significant bit (0x08) corresponds to address 0 and will use the 4th byte of the command packet. In the same manner the least significant bit (0x01) corresponds to address 7 and will use the 11th byte of the command packet.

A.4.2 RFID Tag Interface Definition

A.4.2.1 Introduction

This section discusses lower-level details of tag operation:

- Framing information that must accompany command data and tag responses, including preambles, start delimiters, and CRC definitions;
- Bit-level specifications for the device, including data encoding techniques and timing constraints; and
- General specifications, including carrier frequency ranges and appropriate data rates for the RFID chip.

A.4.2.2 Field Sequences for Tag Commands

Each tag command and tag response is composed of several fields. For example, the tag functional specification described two of the fields required in sending the command DATA_READ to a tag: the value for the DATA_READ command, ID, and ADDRESS. This section discusses the framing fields for commands sent from base station to tag, as well as the structure of the tag response, if any.

The framing fields for base station to tag commands enable the tag to properly decode the signal it receives and to validate the data it recovers from that signal. There are three framing fields that precede the COMMAND field: PREAMBLE_DETECT, PREAMBLE, AND START_DELIMITER. An additional framing field, CRC, is always present as the last data sent from the base station to the tag. A complete base station to tag command, then, uses the following sequence (see Figure A-2):

PREAMBLE_DETECT

PREAMBLE

START_DELIMITER

COMMAND

(ID) (ADDRESS) (BYTE_DATA) (WORD_DATA)

CRC

The items in parentheses are not included in all commands.

There are also framing fields that accompany responses from the tag to the base station. All tag responses include two framing fields, QUIET and RETURN_PREAMBLE, which precede the data or acknowledgment signal being sent from the tag. As in the forward (base station-to-tag) link, a CRC field is always present as the last field in the communication. A complete tag-to-base station response has the following field sequence:

QUIET

RETURN_PREAMBLE

(ACKNOWLEDGE) (ERROR) (WORD_DATA)

CRC

The items in parentheses are not included in all responses; in fact, exactly one of these items will be included. The RETURN_PREAMBLE includes both bit synchronization and frame synchronization components.

Example: In the case where a DATA_READ command was sent and a tag responded to that command, the complete field sequence (including forward and return link fields) would be as follows (see Figure A-2):

Field	**Link Direction**
PREAMBLE_DETECT	forward
PREAMBLE	forward
START_DELIMITER	forward
COMMAND	forward
ID	forward
ADDRESS	forward
CRC	forward
QUIET	return
RETURN_PREAMBLE	return
WORD_DATA	return
CRC	return

The tag uses a backscatter technique to communicate data to the reader; thus, the reader must be steadily powering the tag as well as listening to the tag response throughout the tag-to-reader (backscatter) communication. This applies to all fields in the return link.

When a tag receives a write command, it may execute a write operation. (The details of the conditions under which a write will occur are described in section A.4.2.) If a write operation is executed, the final field in the overall field sequence will always be WAIT.

During the WAIT field, when the tag is writing data into the EEPROM, the reader must steadily power the tag. On-off key data should not be sent during this time.

For example, a successful WRITE command would include the following fields:

Field	**Link Direction**
PREAMBLE_DETECT	forward
PREAMBLE	forward
START_DELIMITER	forward
COMMAND	forward
ID	forward
ADDRESS	forward
BYTE_DATA	forward
CRC	forward
QUIET	return
RETURN_PREAMBLE	return
ACKNOWLEDGE	return
CRC	return
WAIT	(powering during write)

A.4.2.3 Data Encoding/Bit Pattern Definitions for All Fields

Data is encoded and presented in slightly different ways in different fields. This subsection provides specifications for the component fields.

For reader-to-tag communication (forward link), data is sent using an on-off key format. The radio frequency field being on corresponds to 1, while the radio frequency field being off corresponds to 0. The on-off ratio specification is 40 dBc.

For tag-to-reader communication (return link), data is sent using backscatter techniques. This requires that the reader provide steady power to the tag during the return link. While the reader powers the tag, the tag alternately opens and shorts its antenna connection, changing the effective impedance of the tag front end and thus changing the overall radio frequency reflectively of the tag as seen by the base station. No on-off key modulation

data may be sent during this time. During the WAIT field (when tags write data into their memory), the reader also must provide steady power to the tag. No on-off key modulation data may be sent during this time.

A.4.2.3.1 Preamble Detect Field

Minimum of 2 bytes of field on.

A.4.2.3.2 Preamble (in NRZ format; equivalent to 9 bits of Manchester 0)

010101010101010101

A.4.2.3.3 Start Delimiter (in NRZ format; includes Manchester errors; spaces ignored)

11 00 11 10 10

A.4.2.3.4 COMMAND, ID, ADDRESS, BYTE_DATA, WORD_DATA (forward link fields)

Data is Manchester encoded as follows:

- 01 = field off/field on = logic 0
- 10 = field on/field off = logic 1

A.4.2.3.5 CRC (forward and return link)

Command and response packets include a trailing 16-bit CRC. The CRC is generated and checked on the command and response bytes. The preamble and start delimiter are not included in calculations. The CRC uses the CRC-CCITT polynomial $X^{16} + X^{12} + x^5 + 1$. The accumulator is initialized to all ones (ffff hex). The 16-bit calculated CRC is transmitted inverted over the radio frequency link. It is uninverted at the receiver before being used. The result of the CRC check is all zeroes (0000) if there is no error.

A.4.2.3.6 QUIET (return link)

Tag does not modulate impedance for 2 bytes. Antenna is left open.

A.4.2.3.7 Return_Preamble

00 00 01 01 01 01 01 01 01 01 00 01 10 11 00 01
When the tag executes backscatter, a half-bit 0 and half-bit 1 sent by the tag are defined as follows:

0 = antenna open
1 = antenna short

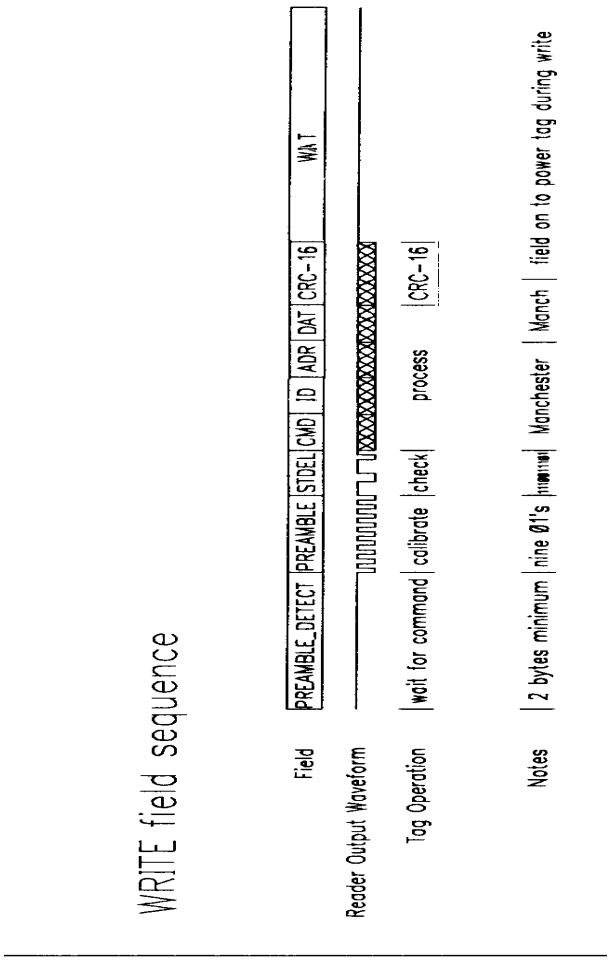


Figure A-2: Sample Command/Response Packets (Write Command)

A.4.2.3.8 Acknowledge, Error, Word_Data (valid tag responses)

For the return link, FM0 data encoding is used. In FM0 encoding, data transitions occur at all bit boundaries. In addition, data transitions occur at the mid-bit of logic 0 being sent. Thus, the logic sequence 0 1 1 0 would be transmitted as follows:

10 11 00 10 (FM0 encoding)
0 1 1 0 (logic sequence)

Note that the starting output half-bit must be known to define the FM0 sequence (the sequence 01 00 11 01 is a valid FM0 encoding of 0 1 1 0 as well). Because the return preamble ends with a 1, it resolves the ambiguity: the first FM0 half-bit is always 0.

A.4.2.3.9 CRC

The CRC algorithm used is defined in the previous forward link CRC section. Encoding of the return CRC data follows the same rules as the encoding for the valid tag responses.

A.4.2.3.10 Wait

During the WAIT field, the base station provides steady power to the tag. No on-off key data may be sent during the write operation.

A.4.3 Communication Sequences at Packet Level

Figure A-3 shows several examples of communication sequences at the packet level. Sequence 1 depicts a packet sequence that includes a write command. The sequence includes a wait for write time, which provides the necessary time for the chip to complete its write operation. In addition, following the wait for write time, the base station issues a tag resync signal. This signal is composed of 10 consecutive 01 signals. The purpose of the tag resync signal is to initialize the tag data recovery circuitry. It is required after a write because the base station may output spurious edges during the wait for write time. Without the tag resync, tags may miscalibrate as a result of the spurious signals that may be generated.

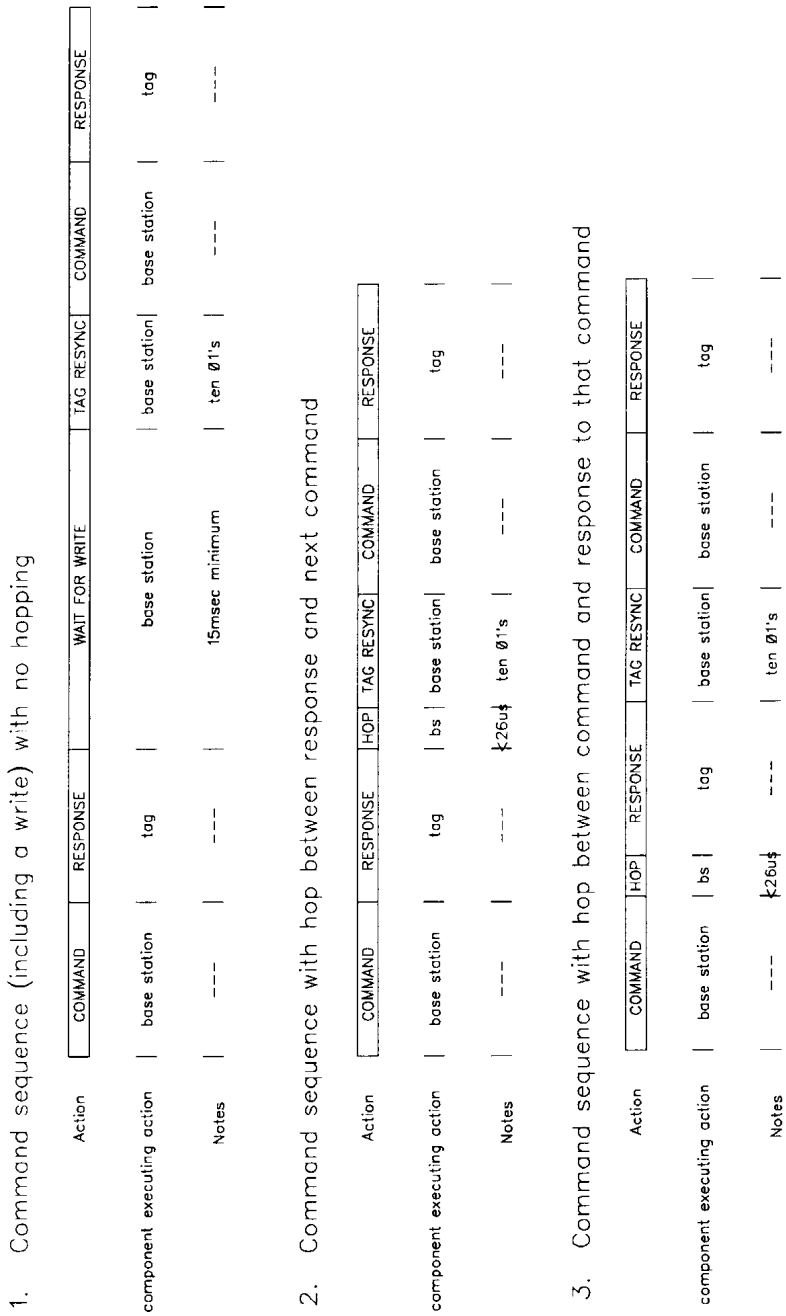


Figure A-3: Examples of Communications Sequences at the Packet Level

ANNEX B

APPLICATION PROGRAMMERS INTERFACE (API)

B.1 Introduction

B.1.1 Purpose

The Application Program Interface (API) provides a standard mechanism for accessing the device driver of a T6-compliant RFID interrogator. The driver supported by the API provides the following features:

- Read and write support including
  - multiple tags in one command from the application
  - multiple fields in one command from the application
  - constant data (all tags) or variable data (per tag) options
  - lock option after write or verify
- Support of many data types, such as
  - Integer (char, short, long)
  - Strings
- Name resolution
  - For user-supplied text data field names, the API determines physical address, data size, and data type
  - Handling of field-name to physical address mapping completely hidden from application
  - Name resolution can be bypassed if desired

B.1.2 Nomenclature and Conventions

The following conventions are used in this API with regard to function, constant, type and variable names:

- C-language constants start with the characters "G4" and will be all uppercase, with underscores between words or abbreviations (for example, "G4_CONST_DEC"). Constants beginning with the characters "G4E" indicate error conditions. Status and return codes beginning with "G4" (not "G4E") report non-error conditions and statuses.
- C-language functions and macros start with the characters "G4", but are in mixed case, with no underscores. Textual references to C-language functions will include parentheses after the name (for example "G4FunctionName()").

- C-language data types start with the characters "g4" and are in all lowercase, with underscores between words or abbreviations. Textual references to C-language functions will be italicized (for example "*g4_struct_type*").

- C-language variables and parameters to function calls do not start with the characters "G4" and are in mixed case, with no underscores. Textual references to C variables and parameters will be italicized (for example "VariableName").

- The word "null" has multiple uses in this document, depending on spelling and case. The term "NULL" refers to zero-value C-language pointers. The term "NUL" is a zero-value ASCII character or a zero-value byte. The term "null-terminated string" refers to strings of printable ASCII characters, with a zero-value byte placed in memory directly after the last printable character of the string.

- This API uses a C-language structure of type *g4_control_t*. The description of each routine contains checklist of the fields in the structure that are used by the routine. The following terms are used with respect to the checklist:

  - require: The function uses this member and the value is typically set by the user (caller).

  - once: The function uses this member, but the value is typically set up once per application execution and is not changed during the execution of the application.

  - default: The function uses this member, but the value is typically the default.

  - previous: The function uses this member, but the value is typically set by a previous function.

  - not used: The function does not use this member.

  - output: The function sets this member.

- In addition to the status and error codes defined in this standard, a block of vendor-specific codes for both error and non-error conditions is reserved. This range of codes is defined in the header file, and is bounded by the constants G4_VENDOR_FIRST_CODE, G4_VENDOR_LAST_CODE, G4E_VENDOR_FIRST_ERROR_CODE, and G4E_VENDOR_LAST_ERROR_CODE.

B.2 Subroutine Calls

B.2.1 Subroutine G4Open()

B.2.1.1 Purpose

This subroutine opens the interface to the interrogator.

B.2.1.2 Synopsis

```
#include <stdio.h>

#include <g4rfid.h>

g4_interr_t *G4Open(InterrogatorName, Status, ErrorString, CustomInfoTable,
CustomInfoFileName)

const char *InterrogatorName;

signed long *Status;

char *ErrorString;

void *CustomInfoTable;

char *CustomInfoFileName;
```

B.2.1.3 Description

The G4Open() subroutine establishes a connection between the workstation and an interrogator or set of interrogators. The opened interrogator descriptor is used by subsequent subroutines, such as G4Read() and G4Write(), to access the interrogator. The version of the API supported by the driver is indicated in the *version_major* and *version_minor* fields of the *g4_interr_t* structure and is set by the driver of the G4Open() call. The *version_major* field shall contain 1 and the *version_minor* field shall contain 0 for drivers compliant with this standard. Subsequent releases of Part II will be identified in numerical sequence.

B.2.1.4 Parameters

InterrogatorName	String indicating the interrogator name
Status	Pointer to a long that can contain error status after the call completes
ErrorString	Pointer to a string that may contain textual error information after the call. If this pointer is non-NULL and the call fails, a string containing printable error information is returned. The maximum length is G4_ERROR_STRING_MAX-1.
CustomInfoTable	If this pointer is non-NULL, it points to a memory block that contains vendor specific data.
CustomInfoFileName	If this pointer is non-NULL, it points to a fully-qualified (including path) filename of a file containing vendor specific configuration data.

B.2.1.5 Return Values

Upon successful completion, a pointer to the opened interrogator descriptor structure is returned; otherwise, a NULL pointer is returned, the status is set to indicate the error.

B.2.1.6 Status Codes

Any one of the following status codes may be placed in the memory location pointed to by *Status* during a G4Open() call

G4E_DESC_NO_INTERR_MEMORY	No memory could be allocated for the *g4_interr_t* descriptor structure.
G4E_DESC_NO_ATTR_MEMORY	No memory could be allocated for the *g4_attr_*t descriptor structure.
G4E_CONFIG_OPEN	The configuration file could not be opened.
G4E_CONFIG_FORMAT	The configuration file format is incorrect.
G4E_CONFIG_CLOSE	The configuration file could not be closed.
G4E_OPEN_DEBUG_LOG	The debug log file could not be opened.
G4E_SET_DEBUG_LOG	The debug log could not be set to line buffered.
G4E_INTERR_NAME_NOT_FOUND	The interrogator name was not found in the configuration file.
G4E_PORT_LOCKED	The I/O port is already open by another process.
G4E_PORT_OPEN	The I/O port could not be opened.
G4E_PORT_GETATTR	Old attributes could not be obtained for the I/O port.
G4E_PORT_SETATTR	New attributes could not be set for the I/O port.
G4E_PORT_BAUD	The I/O port baud rate could not be set.
G4E_PORT_FLUSH	The I/O port could not be flushed.
G4E_PORT_FLOW	Hardware RTS/CTS flow control could not be set for the I/O port.
G4E_PORT_BLOCK_ERROR	I/O port read blocking could not be changed for the port.
G4E_PORT_BREAK_ERROR	A serial line break could not be processed for the I/O port.
G4E_INTERR_BAUD	The interrogator baud rate could not be set.
G4E_INTERR_NOSYNC	Initial contact could not be made with the interrogator.
G4E_INTERR_START	The interrogator application did not start.
G4E_INTERR_ATTRIBUTE_REJECT	An attribute was rejected by the interrogator.
G4E_ATTRIBUTE_REJECT	An attribute was rejected by the API.

B.2.1.7 Related Information

The G4Close(), G4GetAttr(), G4SetAttr() subroutines

B.2.2 Subroutine G4Close()

B.2.2.1 Purpose

Closes the interrogator associated with an interrogator descriptor.

B.2.2.2 Synopsis

```
#include <stdio.h>

#include <g4rfid.h>

signed long G4Close(InterrogatorDescriptor, Status)

g4_interr_t *InterrogatorDescriptor;

signed long *Status;
```

B.2.2.3 Description

The G4Close() subroutine closes a connection between the workstation and an interrogator or set of interrogators associated with the *InterrogatorDescriptor* parameter. It also closes the debug log associated with the descriptor and frees other internally allocated resources.

The G4Close() subroutine attempts to cancel outstanding asynchronous I/O requests on this interrogator descriptor. If the asynchronous I/O requests cannot be canceled, the application is blocked until the requests have completed. All open interrogator descriptors are closed when a process exits.

B.2.2.4 Parameters

InterrogatorDescriptor	Specifies a valid open interrogator descriptor
Status	Pointer to a long that can contain error status after the call completes

B.2.2.5 Return Values

Upon successful completion, a value of 0 is returned; otherwise, a value of -1 is returned, and the memory location pointed to by *Status* is set to indicate the error.

B.2.2.6 Status Codes

G4E_SET_ATTR	The interrogator attribute could not be set.
G4E_PORT_GETATTR	Old attributes could not be obtained for the I/O port.
G4E_PORT_SETATTR	New attributes could not be set for the I/O port.
G4E_PORT_BAUD	The I/O port baud rate could not be set.
G4E_INTERR_BAUD	The interrogator baud rate could not be set.
G4E_INTERR_NOSYNC	Initial contact could not be made with the interrogator.
G4E_PORT_UNLOCK	The I/O port could not be unlocked.
G4E_PORT_FLUSH	The I/O port could not be flushed.
G4E_PORT_CLOSE	The I/O port could not be closed.

B.2.2.7 Related Information

The G4Open() subroutine

B.2.3 Subroutine G4Log()

B.2.3.1 Purpose

Write data to the debug log associated with an interrogator descriptor.

B.2.3.2 Synopsis

```
#include <stdio.h>

#include <g4rfid.h>

signed long G4Log(InterrogatorDescriptor, LogString, Status)

g4_interr_t *InterrogatorDescriptor;

char *LogString;

signed long *Status;
```

B.2.3.3 Description

If logging is supported by the driver, the G4Log() subroutine writes the informational null-terminated string pointed to by the *LogString* parameter to the debug log file. The debug log file is opened for the *InterrogatorDescriptor* as part of the G4Open() function. The G4Log() function does not append a new-line character to the file but rather assumes that any desired new-line character are included in the string pointed to by *LogString*.

If logging is not supported by the driver, the G4Log() subroutine must still be present in the library, but it will take no action other than to return a status code indicating that logging is not supported.

B.2.3.4 Parameters

InterrogatorDescriptor	Specifies a valid open interrogator descriptor
LogString	Points to the null-terminated string to be written to the log file
Status	Points to a long that will contain error status after the call completes (if a non-zero value is returned).

B.2.3.5 Return Values

Upon successful completion, a value of 0 is returned; otherwise, a value of -1 is returned, and the status is set to indicate the error.

B.2.3.6 Status Codes

G4E_LOGGING_NOT_SUPPORTED	The driver does not support logging.
G4E_LOGGING_MEDIA FULL	Log media is full or otherwise not writable.

B.2.3.7 Related Information

The G4Open() and G4SetLogLevel() subroutines

B.2.4 Subrountine G4SetLogLevel()

B.2.4.1 Purpose

Set the logging level for diagnostics in the driver.

B.2.4.2 Synopsis

```
#include <stdio.h>

#include <g4rfid.h>

signed long G4SetLogLevel(InterrogatorDescriptor, LogLevel, Status)

g4_interr_t *InterrogatorDescriptor;

int LogLevel;

signed long *Status;
```

B.2.4.3 Description

If logging is supported by the driver, the G4SetLogLevel() subroutine set the logging level for internal diagnostics used by the driver. The debug log file is opened for the *InterrogatorDescriptor* as part of the G4Open() function. A value of 0 for *LogLevel* indicates that the driver should produce no diagnostic logging (including those log entries generated by G4Log()). A value of 1 (one) indicates that the driver should only log those messages generated by G4Log(). A value of 2 (two) indicates that the driver should log messages generated by G4Log and produce a minimal amount of internally generated log messages. Other positive values (manufacturer-specific) indicate increased detail in the log.

If logging is not supported by the driver, the G4SetLogLevel() subroutine must still be present in the library, but it will take no action other than to return a status code indicating that logging is not supported.

B.2.4.4 Parameters

InterrogatorDescriptor	Specifies a valid open interrogator descriptor
LogString	Points to the (null-terminated) string to be written to the log file.
Status	Points to a long that will contain error status after the call completes (if a non-zero value is returned).

B.2.4.5 Return Values

Upon successful completion, a value of 0 is returned; otherwise, a value of -1 is returned, and the status is set to indicate the error.

B.2.4.6 Status Codes

G4_LOGGING_NOT_SUPPORTED	The driver does not support logging.
G4E_LOGGING_LEVEL_ERROR	The logging level specified by the application is out of range.

B.2.4.7 Related Information

The G4Open() and the G4Log() subroutines

B.2.5 Subroutine G4Identify()

B.2.5.1 Purpose

Identify RFID tags.

B.2.5.2 Synopsis

#include <stdio.h>

#include <g4rfid.h>

signed long G4Identify(InterrogatorDescriptor, ControlPointer)

g4_interr_t *InterrogatorDescriptor;

g4_control_t *ControlPointer;

B.2.5.3 Description

The G4Identify() subroutine identifies RFID tags in the field of view of the interrogator referred to by the *InterrogatorDescriptor* parameter using rules in the *g4_control_t* structure and stores results in the *g4_control_t* structure referenced by the *ControlPrinter* parameter.

B.2.5.4 Parameters

InterrogatorDescriptor	Specifies an open interrogator descriptor
ControlPrinter	Points to a *g4_control_t* structure

B.2.5.5 Control Structure Checklist

flag	require
ostatus	require
select_equation	require (if G4_STORE_SELECTION_LIST flag is set)
taglist	once
max_tags	once
timeout_msec	default
start_tag	default
select_list_number	default
valid_tags	output/previous
end_tag	output
status	output
istatus	not used
fields	not used
field_status	not used
field_status[n]	not used
field_name	not used
field_name[n]	not used
tag_field_value	not used
tag_field_value[n]	not used
field_value	not used
field_value[n]	not used
physadrs	not used
physadrs[n]	not used

B.2.5.6 Applicable Control Flags

The following control flags can be bit-wise OR'ed together and are placed in the flag field of the *g4_control_t* structure.

G4_STORE_SELECTION_LIST

G4_IDENTIFY_NO_OP

G4_IDENTIFY_STOP_AFTER_ALL_TRIES

G4_IDENTIFY_STOP_AFTER_ONE_IDENTIFIED

G4_IDENTIFY_STOP_AFTER_ONE_DETECTED

G4_IDENTIFY_ALL_TAGS

G4_IDENTIFY_INCREMENTAL

B.2.5.7 Return Values

Upon successful completion, a value of 0 is returned; otherwise, a value of -1 is returned, and the *g4_control_t* structure status is set to indicate the error.

B.2.5.8 Status Codes

G4E_ID_FULL_TAGLIST	The API tag list is already full.
G4E_ID_NO_TAGLIST	No tag list array was specified.
G4E_ID_NO_OSTATUS	No ostatus array was specified.
G4E_ILLEGAL_FLAG	An illegal control flag was specified.
G4E_SELECT_NO_EQUATION	No select equation was specified and G4_STORE_SELECTION_LIST was specified.
G4E_SELECT_EQUATION	A format error was found in the selection equation.
G4E_ILLEGAL_NAME_ENTRY	A format error was found in the name resolution process.
G4E_NAME_NOT_FOUND	A name matching the selection equation was not found.
G4E_NAME_ADRS_ERROR	An illegal value was detected for the address associated with the name.
G4E_NAME_SIZE_ERROR	An illegal value was detected for the size associated with the name.
G4E_NAME_TYPE_ERROR	An illegal value was detected for the type associated with the name.
G4E_SELECT_REJECT	An attempt was made to alter an interrogator select list that is in use.
G4E_SELECT_LIST_FULL	An attempt was made to add a selection command to a full interrogator select list.
G4E_SELECT_ILLEGAL_LIST	An invalid interrogator select list number was specified during the selection.
G4E_SELECT_ILLEGAL_COMMAND	An attempt was made to add an illegal command to an interrogator select list.
G4E_EMPTY_SELECT_LIST_INTERR	An identification was started with an empty interrogator select list.
G4E_ID_ILLEGAL_LIST	An invalid interrogator select list number was specified during the identification.
G4E_ID_LIST_FULL_INTERR	The interrogator ID list holding identified tags is full.
G4E_COMMAND_REJECT	The identification command was rejected by the interrogator.
G4E_ILLEGAL_BYTE_COUNT	An illegal byte count was specified to the interrogator.

B.2.5.9 Related Information

- The G4GetControl(), G4Read(), and G4Write() subroutines
- The g4rfid.h file, which contains the *g4_control_t* structure and values

B.2.6 Subroutine G4Read()

B.2.6.1 Purpose

Read RFID tags.

B.2.6.2 Synopsis

```
#include <stdio.h>

#include <g4rfid.h>

signed long G4Read(InterrogatorDescriptor, ControlPointer)

g4_interr_t *InterrogatorDescriptor;

g4_control_t *ControlPointer;
```

B.2.6.3 Description

The G4Read() subroutine reads RFID tags in the field of view of the interrogator referred to by the *InterrogatorDescriptor* parameter using rules in the *g4_control_t* structure and stores results in the *g4_control_t* structure referenced by the *ControlPrinter* parameter.

Note that if an access ID is specified by the application for a read operation but is not needed (i.e., there is no access ID protection on the field(s)), the read will succeed, but a status code of G4_READ_ACCESS_ID_NOT_NEEDED will be returned.

Also note that the unique tag ID must always be readable (and therefore can never be read-protected via the access ID mechanism).

B.2.6.4 Parameters

InterrogatorDescriptor	Specifies an open interrogator descriptor
ControlPrinter	Points to a *g4_control_t* structure

B.2.6.5 Control Structure Checklist

flag	require
istatus	require
ostatus	require
fields	require
field_name[n]	require
tag_field_value[n]	require (array data)
field_value[n]	require (constant data)
taglist	once
max_tags	once
field_name	once
tag_field_value	once (array data)
field_value	once (constant data)
timeout_msec	default
start_tag	default
field_status	default
field_status[n]	default
physadrs	default
physadrs[n]	default
valid_tags	previous
end_tag	previous
status	output
select_equation	not used
select_list_number	not used

B.2.6.6 Applicable Control Flags

The following flags can be bit-wise OR'ed together and are placed in the *flag* field of the *g4_control_t* structure.

G4_READ_DATA

G4_READ_AND_VERIFY

G4_RW_CONSTANT

G4_RW_ARRAY

G4_LOCK_DATA

G4_UNLOCK_DATA

B.2.6.7 Return Values

Upon successful completion, a value of 0 is returned; otherwise, a value of -1 is returned, and the *g4_control_t* structure is set to indicate the error.

B.2.6.8 Status Codes

Any one of the following status codes may be placed in the memory pointed to by the *ostatus* field of the *g4_control_t* structure.

G4E_ILLEGAL_FLAG	An illegal control flag was specified.
G4E_RW_VALUE_NULL	A NULL *field_value* or *tag_field_value* entry was specified.
G4E_RW_START_PAST_END	The *start_tag* entry is greater than the *end_tag* entry.
G4E_RW_END_PAST_VALID	The *end_tag* entry is greater than the *valid_tags* entry.
G4E_RW_END_PAST_MAX	The *end_tag* entry is greater than the *max_tags* entry.
G4E_RW_NO_FIELDS	The *fields* entry is zero.
G4E_RW_ISTATUS_NULL	The *istatus* pointer is NULL.
G4E_RW_OSTATUS_NULL	The *ostatus* pointer is NULL, and one of the *field_status* pointers is NULL.
G4E_RW_NAME_RESOLUTION	Name resolution is bypassed (a *field_name* pointer is NULL), and one of the *physadrs* pointers is also NULL.
G4E_RW_NAME_NUL	A field name is a zero length string.
G4E_NO_MEMORY	Memory could not be allocated for temporary status or name resolution results.
G4E_SELECT_NO_EQUATION	No select equation was specified, and G4_STORE_SELECTION_LIST was specified.
G4E_SELECT_EQUATION	A format error was found in the selection equation.
G4E_ILLEGAL_NAME_ENTRY	A format error was found in the name resolution process.
G4E_NAME_NOT_FOUND	A name matching the selection equation was not found.
G4E_NAME_ADRS_ERROR	An illegal value was detected for the address associated with the name.
G4E_NAME_SIZE_ERROR	An illegal value was detected for the size associated with the name.
G4E_NAME_TYPE_ERROR	An illegal value was detected for the type associated with the name.
G4E_EMPTY_SELECT_LIST_INTERR	The interrogator select list was empty.
G4E_INVALID_SELECT_LIST	An invalid interrogator select list number was specified.
G4E_ID_LIST_FULL_INTERR	The interrogator ID list holding tags failing the write multiple is full.
G4E_COMMAND_REJECT	The command was rejected by the interrogator.
G4E_READ_BAD_ACCESS_ID	Read attempt failed due to bad access ID.
G4_READ_ACCESS_ID_NOT_NEEDED	Access ID specified but not needed for read operation.

B.2.6.9 Related Information

- The G4GetControl() and the G4Identify() subroutines
- The g4rfid.h file, which contains the *g4_control_t* structures and values

B.2.7 Subroutine G4Write()

B.2.7.1 Purpose

Write RFID tags.

B.2.7.2 Synopsis

```
#include <stdio.h>

#include <g4rfid.h>

signed long G4Write(InterrogatorDescriptor, ControlPointer)

g4_interr_t *InterrogatorDescriptor;

g4_control_t *ControlPointer;
```

B.2.7.3 Description

The G4Write() subroutine writes RFID tags in the field of view of the interrogator referred to by the *InterrogatorDescriptor* parameter using rules in the *g4_control_t* structure and stores results in the *g4_control_t* structure referenced by the *ControlPointer* parameter. Calls to G4Write() for read-only tags (or for read/write tags in a read-only interrogator system) will fail, with the error status code G4E_WRITE_ERROR_RD_ONLY.

Note that if an access ID is specified by the application for a write operation but is not needed (i.e., there is no access ID protection on the field(s)), the write will succeed, but a status code of G4_READ_ACCESS_ID_NOT_NEEDED will be returned.

B.2.7.4 Parameters

InterrogatorDescriptor	Specifies an open interrogator descriptor
ControlPointer	Points to a *g4_control_t* structure

B.2.7.5 Control Structure Checklist

The following is the checklist for G4Write() non-multiple:

flag	require
istatus	require
ostatus	require
fields	require
field_name[n]	require
tag_field_value[n]	require (array data)
field_value[n]	require (constant data)
taglist	once
max_tags	once
field_name	once
tag_field_value	once (array data)
field_value	once (constant data)
timeout_msec	default
start_tag	default
field_status	default
field_status[n]	default
physadrs	default
physadrs[n]	default
valid_tags	previous
end_tag	previous
status	output
select_equation	not used
select_list_number	not used

The following is the checklist for G4Write() with Multiple Tags:

flag	require
istatus	require
ostatus	require
select_equation	require
fields	require
field_name[n]	require
field_value[n]	require
taglist	once
max_tags	once
field_name	once
field_value	once
timeout_msec	default
start_tag	default
select_list_number	default
field_status	not used
field_status[n]	not used
physadrs	not used
physadrs[n]	not used
tag_field_value	not used
tag_field_value[n]	not used
valid_tags	output
end_tag	output
status	output

B.2.7.6 Applicable Control Flags

The following control flags can be bit-wise OR'ed together and are placed in the flag field of the *g4_control_t* structure.

G4_STORE_SELECTION_LIST	G4Write() multiple tags
G4_RW_CONSTANT	G4Write() non-multiple, G4Write() multiple
G4_RW_ARRAY	G4Write() non-multiple
G4_LOCK_DATA	G4Write() non-multiple
G4_UNLOCK_DATA	G4Write() non-multiple
G4_WRITE_TAGLIST	G4Write() non-multiple
G4_WRITE_MULTIPLE	G4Write() multiple
G4_WRITE_MULTIPLE_START	G4Write() multiple

B.2.7.7 Return Values

Upon successful completion, a value of 0 is returned; otherwise a value of -1 is returned, and the *g4_control_t* structure is set to indicate the error.

B.2.7.8 Status Codes

Any one of the following status codes may be placed in the memory pointed to by the *ostatus* field of the *g4_control_t* structure.

G4E_ILLEGAL_FLAG	An illegal control flag was specified.
G4E_RW_VALUE_NULL	A NULL *field_value* or *tag_field_value* entry was specified.
G4E_RW_START_PAST_END	The *start_tag* entry is greater than the *end_tag* entry.
G4E_RW_END_PAST_VALID	The *end_tag* entry is greater than the *valid_tags* entry.
G4E_RW_END_PAST_MAX	The end_tag entry is greater than the max_tags entry.
G4E_RW_NO_FIELDS	The *fields* entry is zero.
G4E_RW_ISTATUS_NULL	The input status pointer is NULL.
G4E_RW_OSTATUS_NULL	The output status pointer is NULL, and one of the *field_status* pointers is NULL.
G4E_RW_NAME_RESOLUTION	Name resolution is bypassed (a *field_name* pointer is null), and one of the *physadrs* pointers is also NULL.
G4E_RW_NAME_NUL	A field name is a zero-length string.
G4E_NO_MEMORY	Memory could not be allocated for temporary status or name resolution results.
G4E_SELECT_NO_EQUATION	No select equation was specified, and G4_STORE_SELECTION_LIST was specified.
G4E_SELECT_EQUATION	A format error was found in the selection equation.
G4E_ILLEGAL_NAME_ENTRY	A format error was found in the name resolution process.
G4E_NAME_NOT_FOUND	A name matching the selection equation was not found.
G4E_NAME_ADRS_ERROR	An illegal value was detected for the address associated with the name.
G4E_NAME_SIZE_ERROR	An illegal value was detected for the size associated with the name.
G4E_NAME_TYPE_ERROR	An illegal value was detected for the type associated with the name.
G4E_EMPTY_SELECT_LIST_INT ERR	The interrogator select list was empty.
G4E_INVALID_SELECT_LIST	An invalid interrogator select list number was specified.

G4E_ID_LIST_FULL_INTERR	The interrogator ID list holding tags failing the write multiple is full.
G4E_COMMAND_REJECT	The command was rejected by the interrogator.
G4E_WRITE_ERROR_RD_ONLY	Attempt to write a read-only tag or to write a read/write tag with a read-only interrogator.
G4E_WRITE_BAD_ACCESS_ID	Write attempt failed due to bad access ID.
G4_READ_ACCESS_ID_NOT_NEEDED	Access ID specified but not needed for write operation.

B.2.7.9 Related Information

- The G4GetControl(), G4Read() and G4Identify() subroutines
- The g4rfid.h file, which contains the *g4_control_t* structures and values

B.2.8 Subroutine G4SetAttr()

B.2.8.1 Purpose

Sets interrogator state or parameters.

B.2.8.2 Synopsis

#include <stdio.h>

#include <g4rfid.h>

signed long G4SetAttr(InterrogatorDescriptor, AttributePointer, Status)

g4_interr_t *InterrogatorDescriptor;

g4_attr_t *AttributePointer;

signed long *Status;

B.2.8.3 Description

The G4SetATtr() subroutine set attributes that are associated with the interrogator that is identified by the *InterrogatorDescriptor* parameter. The G4SetAttr() subroutine uses the *g4_attr_t* structure reference by the *AttributePointer* parameter to determine the number of attributes to set, the names of each attribute(s) to set, and the value or list of values to set for each attribute.

The *num_attributes* structure member of the *AttributePointer* parameter is a list of null-terminated ASCII strings containing the names of each of the attributes that the subroutine should attempt to set. For each attribute name in the *attr_name* list, the *attr_value* structure member of the *AttributePointer* parameter contains a list of one or more attribute values that the subroutine should attempt to set for the corresponding attribute name.

Most attributes are non-volatile. Once set, they are never altered by the interrogator. Therefore, only attributes that are being changed need to be transmitted to the interrogator.

B.2.8.4 Parameters

InterrogatorDescriptor	Specifies an open interrogator descriptor
AttributePointer	Points to a *g4_attr_t* structure

B.2.8.5 Return Values

Upon successful completion, a value of 0 is returned; otherwise, a value of -1 is returned, and the status is set to indicate the error.

B.2.8.6 Status Codes

G4E_INTERR_ATTRIBUTE_REJECT	An attribute was rejected by the interrogator.
G4E_ATTRIBUTE_REJECT	An attribute was rejected by the driver.

B.2.8.7 Related Information

- The G4Open() an dG4GetAttr() subroutines
- The g4rfid.h file, which contains the *g4_attr_t* structures and values

B.2.9 Subroutine G4GetAttr()

B.2.9.1 Purpose

Retrieves state information or parameters from the interrogator.

B.2.9.2 Synopsis

```
#include <stdio.h>

#include <g4rfid.h>

signed long G4GetAttr(InterrogatorDescriptor, AttributePointer, Status)

g4_interr_t *InterrogatorDescriptor;

g4_attr_t *AttributePointer;

signed long *Status;
```

B.2.9.3 Description

The G4GetAttr() subroutine gets the current values of attributes that are associated with the interrogator that is identified by the *InterrogatorDescriptor* parameter. The G4GetAttr() subroutine gets a list of the current attribute values for each of the attribute names identified in the *attr_name* list member of the *g4_attr_t* structure.

B.2.9.4 Parameters

InterrogatorDescriptor	Specifies an open interrogator descriptor

AttributePointer Points to a *g4_attr_t* structure

B.2.9.5 Return Values

Upon successful completion, a value of 0 is returned; otherwise, a value of -1 is returned, and the status is set to indicate the error.

B.2.9.6 Status Codes

G4E_INTERR_ATTRIBUTE_UNKNOWN An attribute was unknown by the interrogator.

G4E_ATTRIBUTE_UNKNOWN An attribute was unknown by the driver.

B.2.9.7 Related Information

- The G4Open() and G4SetAttr() subroutines
- The g4rfid.h file, which contains the *g4_attr_t* structure and values

B.2.10 Subroutine G4GetControl()

B.2.10.1 Purpose

This routine initializes the *g4_control_t* structure with default values.

B.2.10.2 Synopsis

```
#include <stdio.h>

#include <g4rfid.h>

signed long G4GetControl(ControlPointer)

g4_control_t *ControlPointer;
```

B.2.10.3 Description

The G4GetControl() subroutine initializes the *g4_control_t* structure referenced by the *ControlPointer* parameter. This subroutine may be used to initialize the structures to default values before setting members to application-specific values, which ensures source code compatibility with future software releases.

B.2.10.4 Parameters

ControlPointer Points to a *g4_control_t* structure.

B.2.10.5 Return Values

Upon successful completion, a value of 0 is returned; otherwise, a value of -1 is returned.

B.2.10.6 Status Codes

No error codes have been defined for this routine at this time.

B.2.10.7 Related Information

- The G4Identify(), G4Read(), and G4Write() subroutines
- The g4rfid.h file, which contains the *g4_control_t* structure and values

B.2.11 Subroutine G4GetFieldNames()

B.2.11.1 Purpose

This routine returns a list of names of the fields that are available for a given tag or tags.

B.2.11.2 Synopsis

#include <stdio.h>

#include <g4rfid.h>

signed long G4GetFieldNames(InterrogatorDescriptor, ControlPointer, Status)

g4_interr_t *InterrogatorDescriptor;

g4_control_t *ControlPointer;

signed long *Status;

B.2.11.3 Description

The G4GetFieldNames() subroutine fills in the control structure's *field_names* member with lists of the tag fields that are available for each tag listed in the control structure's *taglist* member.

B.2.11.4 Parameters

ControlPrinter	Points to a *g4_control_t* structure
InterrogatorDescriptor	Specifies an open interrogator descriptor
Status	Pointer to a long that can contain error status after the call completes

B.2.11.5 Control Structure Checklist

flag	not used
ostatus	not used
select_equation	not used
taglist	required
max_tags	once
timeout_msec	not used
start_tag	default
select_list_number	not used
valid_tags	output/previous
end_tag	output
status	output
istatus	not used
fields	not used
field_status	not used
field_status[n]	not used
field_name	not used
field_name[n]	not used
tag_field_value	not used
tag_field_value[n]	not used
field_value	not used
field_value[n]	not used
physadrs	not used
physadrs[n]	not used

B.2.11.6 Return Values

Upon successful completion, this function will return a value of 0. All other return values are defined by the manufacturer of the underlying driver.

B.2.11.7 Status Codes

The error codes for this routine are defined by the manufacturer of the underlying driver and may be any value between -110000 and -110050. Please consult the vendor manuals for specific values and their associated meanings.

B.2.11.8 Related Information

The g4rfid.h file

B.3 Data Structures and Data Types

B.3.1 Data Structure and Data Types Overview

Arguments are generally passed to functions in the form of API-specific structure pointers rather than as a list of values or value pointers. Advantages to this approach include:

- Backward compatibility as the API evolves. By including helper functions in the API that set the structures to default values, members can be added to support new features without breaking existing application code.

- Performance is improved by passing one pointer to a structure rather than a long parameter list.

- Since fields are referenced by name, the application code is more self-commenting and less error prone. With a long parameter list, it would be easy to reverse the order of or omit a parameter.

- The same main control structure is used for the routines G4Identify(), G4Read(), G4Write(), and so forth. Once the application programmer understands the structure, all routines that employ the structure should be easy to use.

- Depending on driver implementation, the same structures may be passed down through the layers of API and driver code. Each layer uses input fields and updates output fields as appropriate. This can lead to simpler maintenance of the API and driver.

B.3.2 Structure g4_interr_t

The *g4_interr_t* structure is used to maintain a handle between the application and the driver/interrogator. It also indicates the version of the API supported by the driver.

The structure is defined as follows:

```
#ifdef WIN32
  #define g4_portid_t  HANDLE
#else
  #define g4_portid_t  int
#endif

typedef struct {
unsigned long version_major;                /* API Major Version Number*/
unsigned long version_minor;                /* API Minor Version Number*/
void     *current_attribute;                /*current attribs for the interrogator */
unsigned long  port_type;                   /* port type */
char          port_name[G4_PORT_NAME_MAX]; /* ascii port name */
unsigned short port_number;                 /* socket  port number */
unsigned short port_connection_tries;       /* sckt cntct tries before fail */
g4_portfd_t    port_fd;                     /* open port file descriptor */
#ifdef WIN32
SOCKET         port_socket;   /* WinSock uses different type from com port */
#endif
int           receive_block;                /* boolean, blocking read on or off */
char           debug_name[G4_DEBUG_NAME_MAX];       /* ascii debug file name */
FILE           *debug_log;                  /* debug log file pointer */
unsigned char  wsdebug;  /* workstation debug level */
char           error_string[G4_ERROR_STRING_MAX];  /* descript.of err code */
} g4_interr_t;
```

B.3.2.1 Structure Member version_major

The member *version_major* contains the major version number of the API specification supported by the driver (e.g., if the version is 1.2, *version_major* is 1) and is set by the driver during the G4Open() call.

B.3.2.2 Structure Member version_minor

The member *version_minor* contains the minor version number of the API specification supported by the driver (e.g., if the version is 1.2, *version_minor* is 2) and is set by the driver during the G4Open() call.

B.3.2.3 Structure Member current_attribute

The member *current_attribute* is a pointer to a void and is used to hold attribute information for an open interrogator. The type is void (to permit flexibility in implementation of the driver by each manufacturer).

B.3.2.4 Structure Member port_type

The member *port_type* is used to numerically indicate the type of port to which the associated basestation is connected. This member may have a value of either G4_PORT_TYPE_SERIAL in the case of a serial port connection, or G4_PORT_TYPE_TCPIP in the case of a TCP/IP port connection.

B.3.2.5 Structure Member port_name

The member *port_name* contains the name of the specific “port_type” port to which the associated basestation is connected. For example, a connection to a port of type G4_PORT_TYPE_SERIAL might have a *port_name* value of “/dev/tty0” or “COM1”.

B.3.2.6 Structure Member port_number

The member *port_number* identifies the specific port number for connections of type G4_PORT_TYPE_TCPIP.

B.3.2.7 Structure Member port_connection_tries

The member *port_connection_tries* identifies the number of times connections to a port will be attempted before a failure condition is declared.

B.3.2.8 Structure Member port_id

The member *port_fd* contains the file descriptor for the interrogator connection when the connection is of *port_type* G4_PORT_TYPE_SERIAL.

B.3.2.9 Structure Member port_socket

The member *port_socket* contains the socket descriptor for the interrogator connection when connection is of *port_type* G4_PORT_TYPE_TCPIP.

B.3.2.10 Structure Member receive_block

The member *receive_block* indicates whether read operations are blocking or non-blocking for the associated interrogator. A value of ON indicates that read operations will be blocking, and OFF indicates that read operations will be non-blocking.

B.3.2.11 Structure Member debug_name

The member *debug_name* contains the name of a file to which debug information will be written. If this member is NULL, debug information will not be logged to disk.

B.3.2.12 Structure Member debug_log

The member *debug_log* contains a file descriptor of the file to which debug information will be logged. If the *debug_name* member is NULL or contained the name of a file that could not be opened, the *debug_log* member will be set to NULL.

B.3.2.13 Structure Member wsdebug

The member *wsdebug* is a numeric value that indicates the level of workstation debug logging that will be recorded for the associated interrogator connection. Valid values for this member include any number from the set of positive integers.

B.3.2.14 Structure Member error_string

The member *error_string* contains a null-terminated text string message pertaining to the last workstation error condition that was detected. This string has a maximum length of G4_ERROR_STRING_MAX including the terminating NULL character.

B.3.3 Structure g4_control_t

The structure *g4_control_t* is the main control structure that the application uses with the G4Identify(), G4Read(), and G4Write() functions. The structure is defined as follows:

```
typedef struct {
unsigned long flag;                    /* function rules */
unsigned long timeout_msec;            /* maximum allowed time */
unsigned long version_major;           /* API Major Version Number*/
unsigned long version_minor;           /* API Minor Version Number*/
g4_tag_list_t *taglist;                /* pointer to tag list */
size_t      start_tag;                 /* first tag to be processed */
size_t      end_tag;                   /* last tag to be processed */
size_t      valid_tags;                /* number of valid taglist entries */
size_t      max_tags;                  /* maximum size of taglist */
signed long status;                    /* overall function status */
g4_status_t   *istatus;                /* pointer to input status list */
g4_status_t   *ostatus;                /* pointer to output status list */
g4_status_t   **field_status;          /* pointer to array of arrays of status */
g4_status_t   **field_status_save;     /* to saved field_status */
unsigned long select_list_number;      /* the selection list number */
char         *select_equation;         /* pointer to select list */
size_t       fields;                   /* number of fields names */
char         **field_name;             /* pointer to arrays of names */
void         **tag_field_value;        /* pointer to array of arrays of values*/
void         **field_value;            /* pointer to array of constant values */
g4_physadrs_t  **physadrs;             /* pointer to array of arrays of address*/
g4_physadrs_t  **physadrs_save;        /* to saved physadrs */
g4_access_id_t  **access_ids;          /* ptr to array of lists of (access IDs)*/
void         *custom_table;            /* ptr to vendor-specific config data */
char         *cust_file_name;          /* ptr to name of vendor-specific cfg file */
} g4_control_t;
```

The *g4_control_t* structure is used to pass identification, read, or write rules to their functions and to return results. The structure members are described in the following sections.

B.3.3.1 Structure Member flag

The member *flag* describes the rules to be followed by G4Identify(), G4Read(), and G4Write(). It is unchanged by the API or driver.

The values for *flag* are as follows:

G4_STORE_SELECTION_LIST	Indicates that a new selection list should be stored in the interrogator. If used in combination with an identification operation or with a write multiple operation, the new list is stored before the identification operation or write operation is performed. This value should not be used in conjunction with G4_IDENTIFY_INCREMENTAL.
	When a write multiple is issued, name resolution determines the physical addresses (write address and possibly selection address) of the fields determined by the selection equation. The result is downloaded to the interrogator. This downloaded result can be used multiple times by issuing a write multiple without storing a selection list.

	Since the write data size and type is tied criteria determined by the selection equation, the equation must be present even if it is not being stored.
The following G4Identify() flags are mutually exclusive.	
G4_IDENTIFY _NO_OP	No identification is performed, but the select list may be stored.
G4_IDENTIFY _AFTER_ALL_TRIES	Runs all identification tries.
G4_IDENTIFY _AFTER_ONE_IDENTIFIED	Skips unused tries after at least one tag is identified.
G4_IDENTIFY _AFTER_ONE_DETECTED	Stops after a tag is detected without actually identifying it.
The following G4Identify() flags are mutually exclusive and are active when G4_IDENTIFY_STOP_AFTER_ALL_TRIES is set.	
G4_IDENTIFY_ALL	Identifies and reports all selected tags in the field-of-view.
G4_IDENTIFY_INCREMENTAL	Reports only tags not previously reported.
The following G4Read() flags are mutually exclusive.	
G4_READ_DATA	Performs reads and fills in the *tag_field_value* arrays.
G4_READ_AND_VERIFY	Performs reads and verifies the data against one of two sources.
These G4Read() and G4Write() flags are mutually exclusive.	
G4_RW_CONSTANT (for G4Read())	Verifies read data against constants in the *field_value* member. G4_RW_ARRAY verifies read data against values in the *tag_field_value* array.
G4_RW_CONSTANT (for G4Write())	Writes constant data specified in the *field_value* member. G4_RW_ARRAY writes data specified in the *tag_field_value* array.
The following G4Read() and G4Write() flags are mutually exclusive and are active for G4Write() or G4Read() if G4_READ_AND_VERIFY is set.	
G4_LOCK_DATA	Locks a data field that has been verified correctly. Fields that are not verified are not locked.
G4_UNLOCK_DATA	Unlocks a data field. The unlock function may be disabled by hardware or software for security reasons. In these cases, when a lock is irreversible, an error status will be returned.
The following G4Write() flags are mutually exclusive.	
G4_WRITE_TAGLIST (non-multiple)	writes to tags based on *taglist*.
G4_WRITE_MULTIPLE	writes to tags based on selection criteria. G4_WRITE_MULTIPLE requires the *fields* member to be 1 and the G4_RW_CONSTANT flag to be set. If G4_STORE_SELECTION_LIST is set, the write is based on the specified selection criteria and field name. If G4_STORE_SELECTION_LIST is clear, the write is based on the previously set up selection criteria and field name.

G4_WRITE_MULTIPLE_START	This G4Write() flag is active when G4_WRITE_MULTIPLE is set. If set, the write multiple is started. If clear, the write multiple is not started, but the selection list may be loaded.
G4_SET_ACCESS_ID	This G4Read() and G4Write() flag is used to indicate that access IDs for specified tag fields are to be set to the new values provided in the *access_ids* member of the *g4_control_t* structure.

B.3.3.2 Structure Member timeout_msec

The member *timeout_msec* holds the maximum allowed time (in milliseconds) for the command. A value of zero disables the timeout. Zero should typically be used unless the application wants to abort infinite identification tries after a specified time. It is unchanged by the API or driver.

Application programmers and system integrators may use the timeout mechanism for non-blocking implementations. Further discussion of non-blocking implementations will be addressed as either a release or as an errata to this standard.

B.3.3.3 Structure Member taglist

The member *taglist* is initialized to point to the array of *g4_tag_list_t* structures. The pointer is unchanged by the API or driver.

For calls to G4Identify(), the array is filled with the tag identification data. The array size should be at least equal to max_tags. For G4Read() or G4Write() (non-multiple), it points to the array of target tags for the G4Read() or G4Write(). The array elements are unchanged by the API or driver. For write multiple, the array is filled with failing tag identifiers.

B.3.3.4 Structure Member start_tag

The member *start_tag* indexes the first *g4_tag_list_t* entry to be processed. It is unchanged by the API or driver. For non-incremental G4Identify() or G4Write() multiple, it is initialized to the next taglist element to be written. It is typically initialized to zero. For incremental G4Identify(), it is not used. For G4Read() and G4Write() (non-multiple), it is initialized to the first taglist element to be processed. To read or write the entire list, it is typically zero.

B.3.3.5 Structure Member end_tag

The member *end_tag* is initialized to index the last *g4_tag_list_t* entry to be processed plus 1.

For G4Identify() or G4Write() (multiple), it is not initialized. It is written by the API or driver to *valid_tags*. For G4Read() and G4Write() (non-multiple), it is initialized to the last *taglist* element to be processed plus 1. To read or write the entire list, it is typically *valid_tags*. It must be less than or equal to *valid_tags*. It is unchanged by the API or driver.

B.3.3.6 Structure Member valid_tags

The member *valid_tags* is an index to the last valid *taglist* element plus 1 (assuming that *taglist* is a zero-based array). Equivalently, it holds the number of valid entries in *taglist*, assuming that the application started at element 0.

For calls to non-incremental G4Identify() and G4Write() (multiple), *valid_tags* is not initialized. For incremental G4Identify(), it indicates the next taglist element to be written. It is updated by the API/driver. For G4Read() and G4Write() (non-multiple), it is initialized to provide a check against *end_tag*. It is typically the *valid_tags* output of the G4Identify(). It is unchanged by the API/driver.

B.3.3.7 Structure Member max_tags

The member *max_tags* is initialized to the maximum number of elements in *taglist*. It is also related to *istatus* and *ostatus*, so the length of those arrays must match. It is unchanged by the API/driver.

For the routines G4Identify() or G4Write() (multiple), *max_tags* determines the last tag to be returned from the identification process, which should be no greater than the size of the space allocated for *taglist*. A zero value indicates that only overall status is returned.

B.3.3.8 Structure Member status

The member *status*, written by the API, holds the overall status of the G4Identify(), G4Read(), or G4Write() process. These routines return a value of 0 to indicate overall success.

For functions using the flag G4_IDENTIFY_STOP_AFTER_ONE_DETECTED, *status* describes the result. Zero indicates none detected, and one indicates at least one detected. A function return code of -1 indicates an overall error.

The status describes the error.

Error status includes the following:

G4E_BADF	The *InterrogatorDescriptor* parameter does not specify a valid interrogator descriptor.
G4E_NO_MEMORY	Temporary memory cannot be allocated.
G4E_ILLEGAL_FLAG	An illegal flag or combination of flags was specified.
G4E_COMMAND_REJECT	A command to the interrogator was rejected. This may occur if the interrogator is not ready to accept commands.
G4E_NO_SELECT_LIST	The *select_list* member is NULL, and the G4_STORE_SELECT_LIST flag is set.
G4E_ID_INVALID_SELECT	The selection string cannot be parsed.
G4E_INVALID_SELECT_LIST	The interrogator cannot support the specified selection list number.
G4E_ILLEGAL_BYTE_COUNT	The interrogator cannot support the specified number of read or write bytes.
G4E_ID_EMPTY_SELECT_LIST_INTERR	The interrogator select list is empty. It must be stored before it is used.
G4E_ID_NO_TAGLIST	The *taglist* member is NULL, and the identified tags are to be returned. There is no place to store the returning tag data.
G4E_ID_NO_OSTATUS	The *ostatus* member is NULL, and the identified tags are to be returned. There is no place to store the returning status.
G4E_ID_FULL_TAGLIST	The value of the *start_tag* member is not less than the value of the *max_tags* member.
G4E_ID_LIST_FULL	The interrogator memory allocated for identification became full before all tags were identified. This error is most likely the result of multiple incremental identifications.
G4E_RW_END_PAST_VALID	The *end_tag* member is greater than the *valid_tags* member.
G4E_RW_END_PAST_MAX	The value of the *end_tag* member is greater than the value of the *max_tags* member.
G4E_RW_NO_FIELDS	The *fields* member is zero.
G4E_RW_NAME_RESOLUTION	Neither absolute tag field information nor field name resolution members are provided.
G4E_RW_NAME_NUL	A field name provided for name resolution is a zero-length string.
G4E_RW_ISTATUS_NULL	The input status pointer *istatus* is NULL. For G4Read() and G4Write(), input status must be available.
G4E_RW_OSTATUS_NULL	For G4Read() and G4Write(), output status must be reported for each field. If the summary status pointer *ostatus* is NULL and one element of *field_status* is NULL, there is no array allocated to hold the status.
G4E_WRITE_NOT_CONSTANT	The G4_WRITE_MULTIPLE flag requires the G4_RW_CONSTANT flag, since write multiple requires constant data.
G4E_WRITE_FIELDS	For a write multiple, *fields* must be one. The API does not support write multiple to more than one field at a time.

B.3.3.9 Structure Member istatus

For G4Read() and G4Write() (non-multiple), the member *istatus* holds the input status array. Input status is not altered by the API. A zero in *istatus* indicates that the target tag should be processed. A non-zero value inhibits processing. When processing is inhibited, the input status is copied to the output status.

B.3.3.10 Structure Member ostatus

The member *ostatus* points to the array of *g4_status_t* entries that will be filled in with the status of the G4Identify(), G4Read(), or G4Write() for each tag. A zero value indicates normal completion.

Errors include:

G4E_ID_READ _ERROR	The tag was identified, but associated data may be incorrect.
G4E_FIELD_NOT_FOUND	One of the field names could not be resolved into a physical address, length, and type.
G4E_FIELD_ADRS_ERROR	The field address determined by name resolution is beyond the range handled by the interrogator.
G4E_FIELD_SIZE_ERROR	The field size determined by name resolution is beyond the range handled by the interrogator.
G4E_FIELD_TYPE_ERROR	The field type determined by name resolution is not one of the types handled by the API (integer or string).
G4E_READ_ERROR	One of the fields could not be successfully read.
G4E_VERIFY_ERROR	One of the fields did not data verify correctly.
G4E_WRITE_ERROR	One of the fields could not be successfully written.
G4E_WRITE_TAG_ERROR	One of the fields had the write rejected by the tag. For example, the field might be locked.
G4E_LOCK_ERROR	One of the fields could not be locked successfully.
G4E_LOCK_VERIFY_ERROR	One of the fields did not lock verify correctly.
G4E_WRITE_LOCKED	One of the fields is locked.
G4E_ACCESS_ID_MISSING	One of the fields identified in the operation is access ID protected but no access ID was provided for the field. Whether the missing access ID was a read access ID or a write access ID depends on the operation that resulted in this condition.
G4E_ACCESS_ID_INCORRECT	One of the fields identified in the operation was access ID protected but the access ID provided the field was incorrect. Whether the incorrect access ID was a read access ID or a write access ID depends on the operation that resulted in this condition.
G4E_READ_BAD_ACCESS_ID	Read attempt failed due to bad access ID.
G4E_WRITE_BAD_ACCESS_ID	Write attempt failed due to bad access ID.
G4_READ_ACCESS_ID_NOT_NEEDED	Access ID specified but not needed for read or write operation.

B.3.3.11 Structure Member field_status

The member *field_status* is a pointer to an array. Each array element corresponds to one field and holds a pointer to an array. Each element of this final array of *g4_status_t* entries corresponds to one tag. The entries are filled with the status of the G4Read() or G4Write() for each field of each tag. A zero value indicates normal completion. For other status codes, see *ostatus*.

Output status can be obtained for each tag (*ostatus*) or each field of each tag (*field_status*).

If the *ostatus* entry is a non-NULL pointer, its array is filled in with tag summary status. If the *ostatus* entry is NULL (G4Read() or G4Write()), all elements of *field_status* must be non-NULL, i.e., status for a field cannot be completely ignored.

If the *field_status* pointer is non-NULL, its elements are examined. If an element is non-NULL, its array is filled with status for the field.

B.3.3.12 Structure Member field_status_save

The member *field_status_save* is used internally by the API or driver. It does not need to be initialized, and its value on exit is unspecified.

B.3.3.13 Structure Member select_list_number

The member *select_list_number* holds the select list number. On the interrogator, select lists hold the tag selection criteria. A select list can be used multiple times if the selection criteria are not changed.

Selection criteria are applicable to G4Identify() and G4Write() multiple.

The API supports select lists 0-255. A given interrogator will typically support fewer.

B.3.3.14 Structure Member select_equation

The member *select_equation* points to the selection criteria character string. An empty string indicates that all tags should be selected.

Section **Error! Reference source not found.** of this document describes the selection equation.

B.3.3.15 Structure Member fields

The member *fields* contains the number of data fields to be processed for each tag by G4Read() and G4Write() (non-multiple) calls. It is not changed by the API/ driver.

One function call can process multiple data fields.

This member is used by the API or driver in accessing the *field_name*, *tag_field_value*, *field_value*, *field_status*, and *physadrs* structure members. Therefore, the size of those arrays must match.

For write multiple, *fields* must be set to 1.

B.3.3.16 Structure Member field_name

A field name is the textual name for a tag data field. The API resolves the textual name into a physical tag address, size, and type.

When used as a parameter to G4Read() and G4Write(), the *field_name* member contains a pointer to an array. Each array element is a pointer to a null-terminated character string. The string describes a field name. Neither the pointers nor the names are altered by the API or driver.

A zero length string is considered an error.

B.3.3.17 Structure Member field_name

A constant data value for each field can be used for a G4Read() verify or a G4Write() call. It must be used for write multiple, where the data written to each tag must be the same.

The member *field_value* contains a pointer to an array. Each element of that array (one for each field) is a pointer to a memory location holding a data value. Neither the pointers nor the values are altered by the API or driver.

The size of the data value in memory must match or be smaller than the size of the tag data field allocated for the *field_value* entry. For integers, sizes of 1, 2, 3, and 4 bytes are supported. The API assumes the following array element size based on the size of the data field.

1: byte
2: short
3: long
4: long

For short and long elements, unused bytes are ignored on write and verify, but a warning is issued to the debug log if ignored bytes have non-zero bits.

B.3.3.18 Structure Member tag_field_value

The member *tag_field_value* is used as an input to G4Read() and holds data to be verified against data read from the tag. The data is not altered by the API or driver.

The member *tag_field_value,* when used as an input to G4Write(), holds data to be written to the tag. It is only used for non-multiple writes, where the data written to each tag may be different. It is not altered by the API or driver.

The member *tag_field_value* is a pointer to an array. Each array element corresponds to one field and holds a pointer to an array. Each element of this final array corresponds to one tag. The pointers are not altered by the API or driver.

Since the array type varies with the data type, it cannot be passed to the API. It is determined from the name resolution size and type.

Integer type Sizes of 1, 2, 3, and 4 bytes are supported. The API assumes the following array element size based on the size of the data field:

1: byte
2: short
3: long
4: long

For short and long elements, unused bytes are set to zero on read. Unused bytes are ignored on write and verify, but a warning is issued to the debug log if ignored bytes have non-zero bits.

String type To support null terminated strings, the size of each element must be the size of the data field plus 1.

For operations in which the array is an input to the API (e.g. write), bytes are read by the API up to the size of the data field. The extra byte (typically a NUL character, but not checked) is ignored. NUL bytes within the string are also ignored.

For operations in which the array is an output from the API (e.g. read), bytes are written by the API up to the size of the data field, and a NUL character is written to the extra byte.

B.3.3.19 Structure Member access_ids

The member *access_ids* is a pointer to an array. Each element in the array corresponds to one tag field and holds a pointer to an array of *g4_access_id_t* structures. Each element of this final array corresponds to one tag.

The *g4_access_id_t* structure holds a pointer to a current access ID and a pointer to a new access ID. These two pointers are void pointers that point to an appropriate access ID type for the tag being addressed.

Whether the access IDs contained in the *access_ids* member of the *g4_control_t* structure are read access IDs or write access IDs depends on the tag operation being performed.

The *current_access_id* element of the *g4_access_id_t* structure contains a pointer to the current access ID for the specific tag field being referenced. The *current_access_id* field should be supplied for reads to and writes from tag fields that have been access ID protected.

The *new_access_id* element of the *g4_access_id_t* structure contains a pointer to the new access ID for a specific tag field. If *new_access_id* points to a data value equal to the data value pointed to by *current_access_id*, then the access ID is not changed. If *new_access_id* field points to a data value different from the one pointed to by *current_access_id*, the driver will attempt to change the access ID to the new value (if the G4_SET_ACCESS_ID flag is set for the read or write operation).

The *new_access_id* element of the *g4_access_id_t* structure is also used to remove access ID protection from a specific tag field. This is accomplished by setting the *new_access_id* element equal to NULL for the desired tag field.

Note that if an access ID is specified by the application for a read operation but is not needed (i.e., there is no access ID protection on the field(s)), the read will succeed, but a status code of G4_READ_ACCESS_ID_NOT_NEEDED will be returned.

B.3.3.20 Structure Member physadrs

The member *physadrs* is a pointer to an array of structures of type *g4_physadrs_t*. Each element of *physadrs* corresponds to one field of a tag. The *g4_physadrs_t* structure holds the field address, size, and type that result from name resolution. The element *phys_address* of the *g4_physadrs_t* structure is the byte address of the field in tag memory. The element *phys_bit_offset* of the *g4_physadrs_t* structure is the offset in bits (from the beginning of the byte) of the start of the field. (For tags in which all fields are placed on byte boundaries, *phys_bit_offset* will always be 0.) The element *phys_size* of the structure is the size in bits of the field. The element *phys_type* describes the data format.

If fields are accessed through the *field_name* member, the API translates this name into an absolute tag address, size, and type. This process is called name resolution and requires processing in the driver.

There are times that the application may wish to bypass name resolution, possibly to improve performance. One example might be a captive application in which the implementation details of the tag and interrogator are known to the application. Another example is a read-modify-write operation in which the name resolution during the read can be reused during the write.

If the *field_name* pointer is NULL, name resolution is bypassed. If *field_name* is non-NULL, but one of the pointer entries in the array (for one field) is NULL, name resolution is bypassed for that field.

If both *field_name* and the *field_name* array entry are non-NULL, name resolution is performed for that field. If not, the *physadrs* member is checked. If the *physadrs* pointer is NULL, the API must perform name resolution for all fields. If name resolution is bypassed for any field, an error is reported.

If the *physadrs* pointer is non-NULL, the array is checked. If an element of the array (one element for each field) is NULL, the API or driver cannot use that field. If name resolution is bypassed for that field, an error is reported.

If all elements for a field are non-NULL, the API can use the *physadrs* array for that field. The API uses the array in one of two ways:

- If name resolution is enabled for that field, the contents of the array (the *physadrs_t* structures) are ignored by the API. Instead, they are written with the results of the name resolution.

- If name resolution is bypassed for that field, the address, offset, size, and type are used for that field. If an individual tag element is invalid, an error is reported for that tag only.

B.3.3.21 Structure Member cust_table

The member *cust_table* is a pointer to a table that contains vendor-specific configuration data. This member is a void pointer, that allows applications to pass through the driver pointers to vendor-specific (and vendor-defined) data structures, without altering the definition of the API for each vendor.

B.3.3.22 Structure Member cust_file_name

The member *cust_file_name* is a pointer to a string containing the name of a file that contains custom (manufacturer-specific) configuration data.

B.3.4 Structure g4_physadrs_t

The structure *g4_physadrs_t* contains name resolution information, as described in the synopsis of the *physadrs* member of the *g4_control_t* structure. The *g4_physadrs_t* structure is defined as follows:

```
typedef struct {

unsigned long phys_address;

unsigned long phys_bit_offset;

unsigned long phys_size;

unsigned long phys_type;

} g4_physadrs_t;
```

B.3.4.1 Structure Member phys_address

The member *phys_address* is the byte address of the field in tag memory.

B.3.4.2 Structure Member phys_bit_offset

The member *phys_bit_offset* is the offset in bits (from the beginning of the byte) of the start of the field. (For tags in which all fields are placed on byte boundaries, *phys_bit_offset* will always be 0.)

B.3.4.3 Structure Member phys_size

The member *phys_size* describes the number of bits of tag memory required to store the tag field.

B.3.4.4 Structure Member phys_type

The member *phys_type* describes the data format.

B.3.5 Structure g4_access_id_t

The structure type *g4_access_id_t* is described in the synopsis of the *access_ids* member of the *g4_control_t* structure type. The definition of the *g4_access_id_t* structures type is as follows:

```
typedef struct{

void *current_access_id;

void *new_access_id;

} g4_access_ids_t;
```

B.3.5.1 Structure Member current_access_id

The member *current_access_id* contains a pointer to the current access ID for the specific access ID for the specific tag field being referenced. It is defined as a pointer to the type void, permitting the API to accept different data types for different manufacturers' access ID mechanisms. The *current_access_id* field should be supplied for reads to and writes from tag fields that have been access ID protected.

B.3.5.2 Structure Member new_access_id

The member *new_access_id* structure contains a pointer to the new access ID for the specific tag field being referenced. It is defined as a pointer to the type void, permitting the API to accept different data types for different manufacturer's access ID mechanisms. The *new_access_id* element of the *g4_access_id_t* structure contains a pointer to the new access ID for a specific tag field. If *new_access_id* points to a data value equal to the data value pointed to by *current_access_id*, then the access ID is not changed. If the *new_access_id* field points to a data value different from the one pointed to by *current_access_id*, then the driver will attempt to change the access ID to the new value (if the G4_SET_ACCESS_ID flag is set for the read or write operation).

The *new_access_id* element of the *g4_access_id_t* structure is also used to remove access ID protection from a specific tag field. This is accomplished by setting the *new_access_id* element equal to NULL for the desired tag field.

B.3.6 Structure g4_attr_t

The structure type *g4_attr_t* is used as a parameter to the functions G4SetAttr() and G4GetAttr() functions. Each driver is responsible for internally initializing and maintaining the data passed to it via structures of *g4_attr_t*.

The definition of the *g4_attr_t* structure is as follows:

```
typedef struct {
long    num_attributes          /* number of attributes in the list */
char    **attr_name;            /* pointer to arrays of names */
void    **attr_value;           /* pointer to array of arrays of values*/
} g4_attr_t;
```

B.3.6.1 Structure Member attr_name

The member *attr_name* is a pointer to a table that contains a pointer to an array. Each array element is a pointer to a null-terminated character string. The string describes an interrogator or driver attribute to be set or retrieved. Neither the pointers nor the names are altered by the API or driver. A zero-length string is considered an error.

B.3.6.2 Structure Member attr_value

The member *attr_name* is a pointer to a table that contains a pointer to an array. Each array element is a pointer to a null-terminated character string. The string describes an interrogator/driver attribute to be set or retrieved. Neither the pointers nor the names are altered by the API or driver. A zero-length string is considered an error.

B.3.7 Type g4_status_t

The type *g4_status_t* is used for reporting function status, and is defined as follows:

```
typedef signed long g4_status_t ;
```

Negative numbers are reserved for error conditions set by the API or driver. Positive numbers are used to indicate non-error status. In addition to the status and error codes define by this standard, a block of vendor-specific codes for both error and non-error conditions is reserved. This range of codes is defined in the header file, and is bounded by the constants: G4_VENDOR_FIRST_CODE, G4_VENDOR_LAST_CODE, G4E_VENDOR_FIRST_ERROR_CODE and G4E_VENDOR_LAST_ERROR_CODE.

B.3.8 Structure g4_tag_list_t

The structure type *g4_tag_list_t* is used as a parameter to the G4SetAttr() and G4GetAttr() functions. Each driver is responsible for internally initializing and maintaining the data passed to it via structures of *g4_tag_list_t*.

The definition of the *g4_tag_list_t* structure is as follows:

```
typedef struct {

unsigned char id[G4_TAG_ID_SIZE]; /* tag id */

void        *adjunct; /* for use by vendor */

} g4_tag_list_t;
```

B.3.8.1 Structure Member id

The member *id* is a pointer to an array of characters. The maximum size of this character array is defined as G4_TAG_ID_SIZE. The data contained in this field is used to uniquely identify a specific tag and is the hexadecimal representation of the Manufacturer Tag ID as described in Annex C of this standard.

B.3.8.2 Structure Member adjunct

The member *adjunct* is a void pointer and is provided to allow vendors to associate additional information with a tag. An example of information would be a value that identifies a tag type or class.

B.4 Selection Equation

B.4.1 Selection Equation Forms

Currently, two forms are supported.

Form 1:
"" (a NULL string).
This form causes all tags in the field of view to be selected.

Form 2:
"TAGID == 0303xxxxxxxx"

This form causes all tags with a unique tag ID (expressed in hexadecimal) equal to the value stated to be selected (*xx* indicates a don't care hexadecimal digit). The value must begin with a decimal digit.

Additional selection equation forms may be defined in future revisions of this specification.

Tag selection is based on software tag type and one arithmetic operation on one field which exists for the software tag type.

B.4.2 Tokens

The forms described in Section B.4.1 are made of tokens. Equation tokens (name, logical and arithmetic operator, values) are separated by white space (space, tab).

Tokens 1 and 2:

For Form 2, the first token must be TAGID and the second token must be ==. That is, for all but Form 1, part of the expression must be a selection based on a unique tag ID.

Token 3:

For Form 2, Token 3 must be a unique tag ID in hexadecimal. The number is a hexadecimal number (lowercase), where *xx* indicates a don't care byte.

B.4.3 Select List Number

The interrogator retains selection criteria, which can be referred to by a structure member called *select_list_number*. If the selection criteria have not changed, they can be specified by number, avoiding reforming and storing the selection list.

For a new equation, it can be stored in the interrogator prior to or as part of an identification or write multiple process.

API HEADER FILE

This section contains the g4rfid.h header file.

```
/* General-purpose constants */
/* Maximum length of textual error strings */
#define G4_ERROR_STRING_MAX  80
/* Maximum length of a port name ASCII string */
#define G4_PORT_NAME_MAX  256

/* Maximum length of a debug file name ASCII string */
#define G4_DEBUG_NAME_MAX  256

/* length of the “id” field in the g4_tag_list_t structure */
#define G4_TAG_ID_SIZE        0X0040
```

```
/* Status constants */

/* The driver does not support logging. */
#define G4_LOGGING_NOT_SUPPORTED       1
/* Access ID specified but not needed for read or write operation. */
#define G4_READ_ACCESS_ID_NOT_NEEDED          2

/* Vendor-specific status codes, used for reporting vendor-specific */
/* status not addressed by the standard (but not treated as errors */
#define G4_VENDOR_FIRST_CODE       1000
#define G4_VENDOR_LAST_CODE        1999
/* Vendor-specific error codes, used for reporting internal driver */
/* or interrogator problems */
#define G4E_VENDOR_FIRST_ERROR CODE -1001
#define G4E_VENDOR_LAST_ERROR_CODE  -1999

/* The InterrogatorDescriptor parameter does not specify */
/* a valid interrogator descriptor. */
#define G4E_BADF                        -2000
/* An attribute was rejected by the driver. */
#define G4E_ATTRIBUTE_REJECT       -2001
/* An attribute was unknown by the driver. */
#define G4E_ATTRIBUTE_UNKNOWN    -2002
/* The command was rejected by the interrogator.  */
#define G4E_COMMAND_REJECT         -2003
/* The configuration file could not be closed. */
#define G4E_CONFIG_CLOSE           -2004
/* The configuration file format is incorrect. */
#define G4E_CONFIG_FORMAT          -2005
/* The configuration file could not be opened. */
#define G4E_CONFIG_OPEN            -2006
/* No memory could be allocated for the g4_attr_t descriptor structure. */
#define G4E_DESC_NO_ATTR_MEMORY                -2007
/* No memory could be allocated for the g4_interr_t descriptor structure */
#define G4E_DESC_NO_INTERR_MEMORY    -2008
/* An identification was started with an empty interrogator select list. */
#define G4E_EMPTY_SELECT_LIST_INTERR  -2009
#define G4E_FIELD_ADRS_ERROR -2010
#define G4E_FIELD_NOT_FOUND -2011
#define G4E_FIELD_SIZE_ERROR -2012
#define G4E_FIELD_TYPE_ERROR –2013

/* The API tag list is already full. */
#define G4E_ID_FULL_TAGLIST        -2014
/* An invalid interr select list number was specified during the id */
#define G4E_ID_ILLEGAL_LIST        -2015
/* The interrogator ID list holding identified tags is full.  */
#define G4E_ID_LIST_FULL_INTERR     -2016
/* No ostatus array was specified. */
#define G4E_ID_NO_OSTATUS          -2017
/* No tag list array was specified. */
#define G4E_ID_NO_TAGLIST          -2018
#define G4E_ID_READ_ERROR          -2019
/* An illegal byte count was specified to the interrogator.  */
#define G4E_ILLEGAL_BYTE_COUNT    -2020
/* An illegal control flag was specified. */
```

```
#define G4E_ILLEGAL_FLAG                -2021
/* A format error was found in the name resolution process */
#define G4E_ILLEGAL_NAME_ENTRY    -2022
/* An attribute was rejected by the interrogator. */
#define G4E_INTERR_ATTRIBUTE_REJECT    -2023
/* An attribute was unknown by the interrogator */
#define G4E_INTERR_ATTRIBUTE_UNKNOWN          -2024
/* The interrogator baud rate could not be set. */
#define G4E_INTERR_BAUD                 -2025
/* The interrogator name was not found in the configuration file. */
#define G4E_INTERR_NAME_NOT_FOUND     -2026
/* Initial contact could not be made with the interrogator. */
#define G4E_INTERR_NOSYNC               -2027
/* The interrogator application did not start or is not running */
#define G4E_INTERR_START                -2028
/* An invalid interrogator select list number was specified. */
#define G4E_INVALID_SELECT_LIST      2029
#define G4E_LOCK_ERROR -2030
#define G4E_LOCK_VERIFY_ERROR -2031
/* The logging level specified by the application is out of range */
#define G4E_LOGGING_LEVEL_ERROR -2032
/* Log media is full or otherwise not writable. */
#define G4E_LOGGING_MEDIA FULL      -2033
/* An illegal value was detected for the address associated with the name. */
#define G4E_NAME_ADRS_ERROR        -2034
/* A name matching the selection equation was not found */
#define G4E_NAME_NOT_FOUND         -2035
/* An illegal value was detected for the size, associated with the name. */
#define G4E_NAME_SIZE_ERROR        -2036
/* An illegal value was detected for the type associated with the name. */
#define G4E_NAME_TYPE_ERROR        -2037
/* Memory couldn't be allocated for temp status or name resolution results. */
#define G4E_NO_MEMORY                   -2038
/* The debug log file could not be opened. */
#define G4E_OPEN_DEBUG_LOG         -2039
/* The I/O port baud rate could not be set. */
#define G4E_PORT_BAUD                   -2040
/* I/O port read blocking could not be changed for the port. */
#define G4E_PORT_BLOCK_ERROR       -2041
/* A serial line break could not be processed for the I/O port. */
#define G4E_PORT_BREAK_ERROR       -2042
/* The I/O port could not be closed. */
#define G4E_PORT_CLOSE                  -2043

/* Hardware RTS/CTS flow control could not be set for the I/O port. */
#define G4E_PORT_FLOW                   -2044
/* The I/O port could not be flushed. */
#define G4E_PORT_FLUSH                  -2045
/* Old attributes could not be obtained for the I/O port. */
#define G4E_PORT_GETATTR                -2046
/* The I/O port is already open by another process. */
#define G4E_PORT_LOCKED                 -2047
/* The I/O port could not be opened. */
#define G4E_PORT_OPEN                   -2048
/* New attributes could not be set for the I/O port. */
#define G4E_PORT_SETATTR                -2049
```

```
/* The I/O port could not be unlocked. */
#define G4E_PORT_UNLOCK             -2050
#define G4E_PSWD_INCORRECT          -2051
#define G4E_PSWD_MISSING            -2052
#define G4E_READ_ERROR              -2053
/* The end_tag entry is greater than the max_tags entry. */
#define G4E_RW_END_PAST_MAX         -2054
/* The end_tag entry is greater than the valid_tags entry. */
#define G4E_RW_END_PAST_VALID       -2055
/* The input status pointer is null. */
#define G4E_RW_ISTATUS_NULL         -2056
/* field name is a zero length string. */
#define G4E_RW_NAME_NUL             -2057
/* Name resolution is bypassed (a field_name pointer is null)  */
/* and one of the physadrs pointers is also null. */
#define G4E_RW_NAME_RESOLUTION      -2058
/* The fields entry is zero. */
#define G4E_RW_NO_FIELDS            -2059
/* The ostatus pointer is null && 1 of the field_status pointers is null. */
#define G4E_RW_OSTATUS_NULL         -2060
/* The start_tag entry is greater than the end_tag entry. */
#define G4E_RW_START_PAST_END       -2061
/* A null field_value or tag_field_value entry was specified. */
#define G4E_RW_VALUE_NULL           -2062
/* A format error was found in the selection equation. */
#define G4E_SELECT_EQUATION         -2063
/* An attempt was made to add an illegal cmd to an interr select list. */
#define G4E_SELECT_ILLEGAL_COMMAND  -2064
/* An invalid interr select list number was specified during the selection. */
#define G4E_SELECT_ILLEGAL_LIST     -2065
/* An attempt was made to add a selection command to a */
/* full interrogator select list.  */
#define G4E_SELECT_LIST_FULL        -2066
/* No select equation was specified */
#define G4E_SELECT_NO_EQUATION      -2067
/* An attempt was made to alter an interr select list which is in use. */
#define G4E_SELECT_REJECT           -2068
/* The interrogator attribute could not be set. */
#define G4E_SET_ATTR                -2069
#define G4E_SET_DEBUG_LOG           -2070
#define G4E_VERIFY_ERROR            -2071
#define G4E_WRITE_ERROR             -2072
#define G4E_WRITE_LOCKED            -2073
#define G4E_WRITE_TAG_ERROR         -2074

/* Attempt to write a read-only tag (or to write a read/write tag with a */
/* read-only interrogator). */
#define G4E_WRITE_ERROR_RD_ONLY     -2075
/* bad access ID on write attempt */
#define G4E_WRITE_BAD_ACCESS_ID     -2076
/* bad access ID on read attempt */
#define G4E_READ_BAD_ACCESS_ID      -2077
#define G4E_ID_INVALID_SELECT       -2078
#define G4E_LIST_FULL               -2080
#define G4E_WRITE_NOT_CONSTANT      -2081
#define G4E_WRITE_FIELDS            -2082
```

```
#define G4E_ACCESS_ID_MISSING       -2083
#define G4E_ACCESS_ID_INCORREST  -2084

/* Control flags */
#define G4_IDENTIFY_NO_OP              0x0000001
#define G4_IDENTIFY_ALL_TAGS           0x0000002
#define G4_IDENTIFY_INCREMENTAL 0x00000004
#define G4_IDENTIFY_STOP_AFTER_ALL_TRIES       0x00000008
#define G4_IDENTIFY_STOP_AFTER_ONE_IDENTIFIED         0x00000010
#define G4_IDENTIFY_STOP_AFTER_ONE_DETECTED 0x00000020
#define G4_LOCK_DATA                   0x00000040
#define G4_READ_AND_VERIFY             0x00000080
#define G4_READ_DATA                   0x00000100
#define G4_RW_ARRAY                    0x00000200
#define G4_RW_CONSTANT                 0x00000400
#define G4_STORE_SELECTION_LIST    0x00000800
#define G4_UNLOCK_DATA                 0x00001000
#define G4_WRITE_MULTIPLE              0x00002000
#define G4_WRITE_MULTIPLE_START    0x00004000
#define G4_WRITE_TAGLIST               0x00008000
#define G4_PORT_TYPE_SERIAL    0x00010000
#define G4_PORT_TYPE_TCPIP     0x00020000
#define G4_IDENTIFY_AFTER_ALL_TRIES  0x00040000
#define G4_IDENTIFY_AFTER_ONE_IDENTIFIED 0x00080000
#define G4_IDENFITY_AFTER_ONE_DETECTED   0x00100000
#define G4_SET_ACCESS_ID       0x00200000
typedef signed long g4_status_t ;

typedef struct{
void *current_access_id;
void *new_access_id;
} g4_access_ids_t;

typedef struct {
long        num_attributes                  /* number of attributes in the list */
char        **attr_name;                    /* pointer to arrays of names */
void        **attr_value;                   /* pointer to array of arrays of values*/
} g4_attr_t;

typedef struct {
unsigned char id[G4_TAG_ID_SIZE];  /* tag id */
void         *adjunct;          /* for use by vendor */
} g4_tag_list_t;

typedef struct {
unsigned long phys_address;
unsigned long phys_bit_offset;
unsigned long phys_size;
unsigned long phys_type;
} g4_physadrs_t;

#ifdef WIN32
  #define g4_portid_t  HANDLE
#else
  #define g4_portid_t  int
#endif
```

```
typedef struct {
unsigned long version_major;                /* API Major Version Number*/
unsigned long version_minor;                /* API Minor Version Number*/
void     *current_attribute;                /*current attribs for the interrogator */
unsigned long  port_type;                   /* port type */
char          port_name[G4_PORT_NAME_MAX]; /* ascii port name */
unsigned short port_number;                 /* socket  port number */
unsigned short port_connection_tries;       /* sckt cntct tries before fail */
g4_portfd_t    port_fd;                     /* open port file descriptor */
#ifdef WIN32
SOCKET         port_socket;   /* WinSock uses different type from com port */
#endif
int           receive_block;                /* boolean, blocking read on or off */
char          debug_name[G4_DEBUG_NAME_MAX];      /* ascii debug file name */
FILE          *debug_log;                   /* debug log file pointer */
unsigned char  wsdebug;  /* workstation debug level */
char          error_string[G4_ERROR_STRING_MAX];  /* descript.of error code */
} g4_interr_t;

typedef struct {
unsigned long flag;                         /* function rules */
unsigned long timeout_msec;                 /* maximum allowed time */
g4_tag_list_t *taglist;                     /* pointer to tag list */
size_t      start_tag;                      /* first tag to be processed */
size_t      end_tag;                        /* last tag to be processed */
size_t      valid_tags;                     /* number of valid taglist entries */
size_t      max_tags;                       /* maximum size of taglist */
signed long status;                         /* overall function status */
g4_status_t   *istatus;                     /* pointer to input status list */
g4_status_t   *ostatus;                     /* pointer to output status list */
g4_status_t   **field_status;               /* pointer to array of arrays of status */
g4_status_t   **field_status_save;          /* to saved field_status */
unsigned long select_list_number;           /* the selection list number */
char          *select_equation;             /* pointer to select list */
size_t       fields;                        /* number of fields names */
char         **field_name;                  /* pointer to arrays of names */
void         **tag_field_value;             /* pointer to array of arrays of values*/
void         **field_value;                 /* pointer to array of constant values */
g4_physadrs_t  **physadrs;                  /* pointer to array of arrays of address*/
g4_physadrs_t  **physadrs_save;             /* to saved physadrs */
g4_access_id_t  **access_ids;               /* ptr to array of lists of (access IDs)*/
void          *custom_table;                /* ptr to vendor-specific config data */
char          *cust_file_name;              /* ptr to name of vendor-specific cfg file */
} g4_control_t;
```

API CODE EXAMPLES

These examples show the fundamental features of the API. They omit more advanced features and error checking for clarity. In the examples,

- The application never needs to manipulate the tag list.
- Incremental identification is simple.

- Identification status (failing tags) are passed to the read and write operations effortlessly. Following read and write operations are skipped with no user code required.
- It is not much harder to read or write three fields than one. Just specify how many to read, what the names are, and where the results should go. (See example 5).
- Data is returned in the application's natural format (char, short, long, string).
- Nearly all the structure members can be set up once at the top of the program. They never change through the life of the application.
- The pointer-to-pointer code is simple boilerplate which is set up once.
- Each read/write operation returns status per tag. This array is indexed identically with the data.
- The read and write operations process all identified tags with no user coding.
- The advanced features are available, but invisible if not used. Features can be added while maintaining backward-compatibility with existing user code.

The following code is the initial setup used for the eight examples.

```
#include "g4rfid.h"
#define MAX_TAGS   100
#define MAX_FIELDS   3

int main()
{
signed long status;          /* return status */
g4_interr_t *Interrogator;    /* interrogator control structure */
g4_attr_t Attribute;          /* interrogator attribute structure */
g4_control_t Control;            /* function control structure */
g4_tag_list_t Taglist[MAX_TAGS];  /* the tag list */
g4_status_t  id_status[MAX_TAGS]; /* id results */
g4_status_t  rw_status[MAX_TAGS]; /* read/write results */
unsigned char color[MAX_TAGS];    /* char array to hold */
                          /* returning color */
unsigned short size[MAX_TAGS];    /* short array to hold */
                          /* returning size */
unsigned long model[MAX_TAGS];   /* long array to hold */
                         /*returning model */
unsigned char paid;             /* write data */
char attr_names[3][60];
int #attr_values[3];
int level;
int stat2;
/*  The following is the pointer to pointer setup, done once */
/*  per program instance.  It is boilerplate code which can be */
/*  copied from program to program intact. */
char *field_name[MAX_FIELDS];
void *field_value[MAX_FIELDS];
void *tag_field_value[MAX_FIELDS];
G4GetControl(&Control);
Control.field_name = field_name;
Control.field_value = field_value;
Control.tag_field_value = tag_field_value;
```

Example 1
This example opens an interrogator for Doorway 1.

```
Interrogator = G4Open("Doorway 1",&status, NULL, NULL, NULL);
```

Example 2
This example changes an interrogator attribute.

```
/* Setup*/
attr_values[0] = &level;
attr_values[1] = $stat2;
strcpy (attr_names[0], “level”);
strcpy (attr_names[1], “stat2”);
Attribute.num_attributes = 2;
Attribute.attr_names = attr_names;
Attribute.attr_value = attr_values;
G4GetAttr(Interrogator,&Attribute,&status); /* Get attributes */
Level = 7;   /* set new level */
G4SetAttr(Interrogator,&Attribute,&status); /* store attrs */
```

Example 3
This example identifies the tags in the field-of-view.

```
Control.flag = (G4_STORE_SELECTION_LIST | G4_IDENTIFY_ALL_TAGS);
Control.select_equation = "";                    /* select all */
Control.taglist = Taglist;
Control.max_tags = MAX_TAGS;
Control.ostatus = id_status;
G4Identify(Interrogator,&Control);
```

Example 4
This example reads model numbers from all tags that were identified in Example 3.

```
Control.flag = G4_READ_DATA;      /* read command */
Control.istatus = id_status;      /* input status from identify */
Control.ostatus = rw_status;      /* resulting status per tag */
Control.fields = 1;     /* read one field */
field_name[0] = "MODEL";      /* name of the field to read */
tag_field_value[0] = model;      /* data appears here */
G4Read(Interrogator,&Control);
```

Example 5
This example reads 3 fields at once.

```
Control.fields = 3;
field_name[1] = "SIZE";
tag_field_value[1] = size;
field_name[2] = "COLOR";
tag_field_value[2] = color;
G4Read(Interrogator,&Control);
```

Example 6
This example indicates to all tags that something has been "paid."

```
Control.flag = G4_RW_CONSTANT; /* write same value to all tags */
field_name[0] = "PAID";      /* name of the field to write */
field_value[0] = &paid;
paid = 1;                /* 1 means paid */
G4Write(Interrogator,&Control);
```

Example 7
This example incrementally identifies tags.

```
Control.flag = G4_IDENTIFY_INCREMENTAL;
G4Identify(Interrogator,&Control);
```

Example 8
This example closes the interrogator interface.

```
G4Close(Interrogator,&status);
return(0);
}  /* end of example program */
```

ANNEX C

MEMORY LAYOUT

C.1 Overview

This Specification is written to document the changes and current state of memory allocations for the ANSI 10.8.4 UHF tags, particularly that of the tag serial numbering scheme. It also includes the recommended format for tag memory layout and structure. It is planned that all ANSI 10.8.4 UHF products follow a common format structure, therefore it is important that all parties involved are aware of any changes to this memory architecture and that is architecture be followed for all tags manufactured. For virtually all deployed reader systems, bytes 0-7 and bytes 10-17 should be automatically used by the reader in an identification sequence.

C.2 Tag Data

Portions of data, such as the Tag Serial Number, Manufacturing Location Code, and Hardware Tag Type, will be programmed at the factory and locked, while the Tag Memory Layout may be programmed by the end-user, and may be kept unlocked if so desired. The six bytes of data in the ASIC Chip ID, are programmed by the chip manufacturer and are retained in the tag, although these bytes are not locked and may be overwritten by the user if the space is needed for a given application.

C.3 Tag Memory Map

The following table describes the overall memory map of the tags.

Table C-1: Layout of Tag Memory Map

Bytes	Field Name	Written	Locked
0-7	Tag ID	Manufacturing	Manufacturing
8,9	Tag Manufacturer	Manufacturing	Manufacturing
10,11	Tag Hardware Type	Manufacturing	Manufacturing
12-17	Tag Memory Layout	Manufacturing or Application	As Required by Application
18-23	Temporarily used for ASIC Chip Information	Integrated Circuit Foundry	As Required by Application
18-127	User Data*	Application	As Required

* Definition and Format of User Data determined by Tag Memory Layout.

The first eight bytes of the tag memory shall be programmed with unique Tag ID numbers. This field is hard-coded in the ASIC to allow tag sort algorithms to function properly, therefore it is important that these Tag Serial Numbers be unique. This Tag Serial Number is based on data programmed into the tag in decimal address locations 18 through 23 by the integrated circuit (IC) foundry. The data is converted to a condensed format and a date code is added to generate a Tag Serial Number. See the table below.

C.4 Tag Serial Number

Table C-2: Layout of Tag Serial Number (Bytes 0-7)

MSB **LSB**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
M L	M L	M L	M L	M L	M L	M L	M L

Zero	x coordinate	y coordinate	Wafer	Lot ID	Date Code	ISO	Ck
3 bits M L	(8-bits) M L	(8-bits) M L	5 bits M L	10 bits M L	16 bits MSB LSB	12 bits MSB LSB	2 bits M L

All fields, including both the tag memory bytes, and the Tag Serial Number data fields are shown Most Significant Bit (MSB) left justified. These first three fields have been retained for the possibility that it is necessary to alter the Tag ID after programming at wafer sort. This is made possible by maintaining the first byte unlocked such that the final programming station may check for duplicate Tag IDs and use these bits, if necessary for generating unique Tag IDs.

The Tag Serial Number shall be programmed and locked at the factory with a unique number for each tag.

Die Number (bits 45-63) – This is a 19-bit hexadecimal representation of the physical location of the die on the specified wafer. This number is taken from the data written by the ASIC foundry in byte address locations 22 and 23. Maximum value is 524,287.

A modification of the Die number field is incorporated by the production programming test system when a tag with duplicate Lot, Wafer, and Die data in bytes 18 through 23 are detected within a given session. The production programming and test system will assign a pseudo Die Number in the Tag Serial Number which has the most significant bet set and a sequential number assigned for the remainder of the field starting from 0 for a given session.

Wafer Number (bits 40-44) – This is a truncated 5-bit hexadecimal representation of the wafer number in a given lot run taken from the data written by the ASIC foundry in byte address location 21. It was necessary to truncate this value to keep the number of bits used for the Tag ID to 8 bytes as required by the system architecture. This field value ranges from 1 to 24.

Lot ID (bit 30-39) – This is a 10-bit base 26 binary representation of the last two alpha characters of the full lot number. This field is now generated by taking the last 2 character of the full lot number. This is converted to a 5-bit character scheme. Decimal characters from 0 to 9 are directly converted to hexadecimal characters 0 to 9. Alpha characters A to Z are converted to hexadecimal values 0 to 25.

Data Code (bits 14-29) – This is a 16-bit field that is a hexadecimal representation of the number of days since January 1, 1998. This field provides enough combinations for up to 179 years. Maximum value for this field is 65,635.

ISO Code (bits 2-13) – This is a 12-bit hexadecimal field that has been included to meet anticipated ANSI/ISO standard requirements. At present, all tags produced will be programmed with a hexadecimal value equivalent to that of the production line starting at '001' hexadecimal.

Check Sum (bits 0, 1) – The most recent change of the Tag Serial Number involves the addition of a 2-bit check sum. This represents the truncated sum of the bits set to 1 for 62 bits preceding the check sum in the Tag Serial Number field. Valid values are 0, 1, 2, & 3.

C.5 Manufacturer ID and Tag Hardware

These bytes may be used for group select functions and/or warranty purposes. All these bytes are programmed and locked at the factory.

Table C-3: Layout of Tag Manufacturer and Tag Hardware Type (Bytes 8-11)

Bytes	Field Name	Valid Range	Format
8, 9	Manufacturer ID	00 – 99, AA – ZZ	ASCII
10, 11	Tag Hardware Type	0000 – FFFF	HEX

Manufacturer ID (Bytes 8,9) – These two bytes have been reserved for encoding the Tag Manufacturer ID in accordance with ISO 14816. These fields will initially be encoded with the following codes depending on the manufacturer of the tags.

Table C-4: Manufacturer Codes

Manufacturer	ASCII Representation	Hexadecimal Code
Reserved	“AT”	4154
Reserved	“HT”	4854
Reserved	“AA”	4141
Reserved	“AS”	4153
Reserved	“AN”	414E

Tag Hardware Type (Bytes 10,11) – This is a 2-byte hexadecimal representation of the tag hardware design. This number will be different for each type of hardware change made to the tag design that affects the function of the tag. This does not include differences in tag packaging, or color, nor does it include differences in RF operational frequency. This field will be used to distinguish differences in commands or caommand structure. It will also be used to distinguish differences in data protocol or optional features such as audio, or visual indicators. All tags manufactured with ANSI 10.8.4 UHF ASICs will be encoded with a hexadecimal "0001" for tag hardware type. Battery ASIC tags will be encoded with hexadecimal "0010" for the Tag Hardware Type.

Check Tags will be programmed and lock at the factory with an 80xx hexadecimal for the Tag Hardware Type, where "xx" represents a "don't care" for the second byte. This second "don't care" byte will initially be programmed with "00". An indication of Check Tag status for the tag is also provided in the Embedded Applications Code as described below.

C.6 Tag Memory Layout

The Tag Memory Layout is used by an application to determine the format of the subsequent User Data. Bytes 12 through 17 are programmed to "FF" from the factory unless otherwise specified.

These next two fields may be programmed either at the factory, as a custom field, or by the customer after it is shipped from the factory. The owner of the tag may optionally decide to either lock these fields or keep them unlocked so that the data format may be altered throughout the life of the tag.

Table C-5: Tag Memory Layout (Bytes 12-17)

Bytes	Field Name	Valid Range	Format
12	Embedded Application Code	00 – FF	Hexadecimal
13 - 157	Tag Memory Map Allocation	0000000000 – FFFFFFFFFF	Hexadecimal

Embedded Application Code (Byte 12) – This is the top-level hierarchy of tag memory layout. This field, in conjunction with the Tag Memory Allocation allows an application to determine the format and content of the user data. This field can be used to represent various formats of the tag data content.

Current hexadecimal assignments of the Embedded Application Code are as follows:

Table C-6: Embedded Application Codes

Embedded Application Code	Description of Embedded Application Codes
00, FF	Unformatted, programmed to “FF” at manufacturer.
01	Reserved
02	Customer Specific Memory Allocation
03	File Allocation Table (Long Directory) – TBD in future
04	Check Tag
05	RFID Reader Configuration Tag
06	Reserved
07	Reserved for Engineering Development
08	Reserved
09	Reserved
0A	ASN.1 Compliant Data Format
0B	ANSI MH10.8.4 Compliant Data Format
0C – FE	To be allocated and registered by to application based needs.

Tag Memory Allocation Map (Bytes 13-17) – This field is based on a structured hierarchy that allows for an application to determine the format and content of the user data in conjunction with the Embedded Application Code in byte 12 defined above.

C.6.1 Embedded Application Code "01" – Reserved

All other Tag Memory Allocation Map combinations for Embedded Applications Code "01" are currently considered undefined. These remaining characters may be used to further specify functions and/or applications that have been appended to the primary application.

C.6.2 Embedded Application Code "02" – Customer Specific Memory Allocation

Customer specific data and file allocation tables are not detailed in this specification. However, it is envisioned that customers must register a Customer Specific Memory Allocation Code (CSMAC) with the manufacturer’s marketing and obtain a configuration

string to write their two byte CSMAC value as well as an embedded application code of "02" in tag memory location byte 12.

The following table provides examples of specific tag memory allocation fields for Customer Specific Tag Memory applications already being used by customers. It should be noted that this table is not inclusive of all customer specific memory allocations, nor does it represent all the codes registered with marketing. It is recommended that the manufacturer's Marketing department be contacted for obtaining a current listing and registering any additional codes.

Table C-7: Customer Specific Memory Allocation Codes Bytes 13 & 14

Hexadecimal Code	ASCII Representation	Customer/Description

Other Customer Specific Tag Memory applications that may be added in the future include (but are not limited to):

- Theft Detection and Deterrence
- Emergency Warning Systems

C.6.3 Embedded Application Code "03" – File Allocation Table (Long Directory)

The Tag Memory Allocation for Embedded Applications Code "03", File Allocation Table, has not yet been defined, and is only a placeholder at current. This format will allow a user to view the tag memory similar to that of a floppy drive on a computer.

C.6.4 Embedded Application Code "04" – Check Tag

The Check Tag architecture is made available so the reader may selectively read the check tag to verify the type of antenna attached and determine duty cycle and/or output power. This check tag function may also be used for providing end-to-end system diagnostic operational verification. These bytes are programmed and locked at the factory. The definition of the first byte of the Tag Memory Allocation for the check tag architecture is as follows:

Table C-8: Tag Memory Allocation for Check Tag EAC

ASCII Code (Bytes 13 thru 17)	Valid Hexadecimal Values	Description of Data for Check Tag EAC Tag Memory Allocation
LLxxxxxxxx	0 – 255	Byte 13 is a hexadecimal representation of the clearance distance from the antenna necessary to meet safety guidelines, where "LL" is an indication of this distance in centimeters. This also provides an indication that there is a warning label on radome. This information provides an indication to the reader on RF radio functions such as limits on duty cycle and output power.
XxGGxxxxxx	0 - 255	Byte 14 is a hexadecimal value that indicates the linearly polarized gain of the antenna in dB relative to an isotropic radiator. A reader may use this data in the tag to automatically adjust output power or duty cycle based on the country code and the regulations for that region.
XxxxPPxxxx	0 – 255	Byte 15 is a hexadecimal value that indicates the recommended output power for a reader system in dBm.

The following data is programmed and locked into check tags at the time of antenna manufacture:

Table C-9: Check Tag Memory Map

Address	Length	Format	Description
24	8	ASCII	Antenna Frequency in MHz
32	16	ASCII	Antenna Manufacturer Identification
48	16	ASCII	Antenna Model Number
64	8	ASCII	Antenna Date of Manufacture
72	**8**	**ASCII**	**Cable Length in centimeters**

All fields are padded on the front with ASCII spaces or hexadecimal "20". Descriptions of these fields are provided below:

Antenna Frequency in MHz: This is an 8-byte ASCII field representing the center frequency of the antenna in MHz.

Antenna Manufacturer Identification: This is a 16-byte ASCII field representing the manufacturer of the antenna. These are represented with uppercase ASCII characters and padded with spaces at the end.

Antenna Model Number: This is a 16-byte ASCII field representing the ordering model number for the antenna. As an example, the order number for the 915 MHz circular polarized antenna is "12-0510-004 ". This field is padded with spaces, or hexadecimal code "20" to the end of the field.

Antenna Date of Manufacture: This is an 8-byte ASCII field representing the date the antenna was manufactured. The format is YYYYMMDD where YYYY represents the year, MM the month, and DD the day of the month.

Cable Length in centimeters: This is an 8-byte ASCII field representing the cable length in centimeters.

An indication that the tag is a valid Check Tag is also provided in the Tag Hardware Type byte locations 10 and 11 with a value of “80xx” as described previously.

C.6.5 Embedded Application Code "05" – RFID Reader Configuration Tag

The Tag Memory Allocation for Embedded Applications Code “05”, RFID Reader Configuration Tag, has been reserved to indicate a tag used for configuring a reader by means of a special diagnostic configuration mode. This tag can be used for setting various configuration parameters in a reader or group of readers simply by reading the tag in the configuration mode. Format of this data is to be defined by the reader specification or may be added in a future version of this document as an appendix.

C.6.6 Embedded Application Codes "06 through 09"

Reserved.

C.6.7 Embedded Applications Code "0A" – ASN.1 Compliant Data Format

The Tag Memory Allocation for Embedded Applications Code “0C”, ASN.1 Compliant Data Format, has been reserved to indicate a tag used in a fashion as defined in the AI 6 model. These data memory allocations are as follows:

Table C-10: ASN.1 Compliant Data Memory Map

Address	Length	Description
24	8	System Information (6 bytes required by ASN.1)
32	32	Directory (28 bytes required by ASN.1)
64	16	Data Object (6 + 9 bytes required by ASN.1)

C.6.8 Embedded Application Code "0B" – ANSI MH10.8.4 Compliant Data Format

Defined by this standard.

C.6.9 Embedded Application Codes "0B through FF"

Reserved.

C.7 ASIC Chip Information

The final pre-determined field of the tag is the ASIC Chip Information. The initial intent of this data was primarily for use in warranty purposes, and this information will still be used in that way, however, as described above, the majority of this data will be represented in an alternate, but permanent, format in the Tag ID field (bytes 0 through 7). This data will also prove useful during initial pilot runs and production system development for IC traceability. This data is programmed into the tag at the ASIC foundry and provides ASIC die traceability.

If this space is deemed necessary for a given application, the tag user or owner may use it. This data is only necessary to provide the data for initial programming of the Tag ID at the factory, once the Tag ID has been programmed, these fields may be overwritten.

Table C-11: Layout of ASIC Chip Information (Bytes 18-23)

Bytes	Field Name	Valid Range	Format
18,19,20	Lot ID (3 Bytes)	0AA – 9ZZ	ASCII
21	Wafer Number	00 – FF	HEX
22	x-coordinate of die on wafer	00 – FF	HEX
23	y-coordinate of die on wafer	00 – FF	HEX

The Lot ID field is a mixed-format field that represents a portion of the ASIC lot number. The first byte in address location 18, is a numeric value. The next two bytes, in address locations 19 and 20 are an alpha ASCII representation of the last two characters of the lot number. Only the data from address locations 19 and 20 are encoded into the Tag Serial number. The definition is a two one-byte fields representing the x and y coordinates of the die on the wafer. Byte 22 represents the x coordinate, byte 23 represents the y coordinate.

C.8 User Data

Bytes 18 through 127 are defined as the customer programmable portion of the tag. The owner of the tag may use these remaining data bytes of the tag as desired, although the format of these fields should be determined by the Tag Memory Allocation described above. The data programmed into the tags for these bytes will follow the format bytes 18 through 23 as described above. Bytes 24 through 127 are all programmed with hexadecimal “FF” with the following exceptions:

Table C-12: Factory Data for User Data Area (Bytes 24-127)

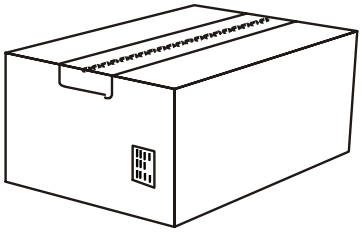
Address (Decimal)	Address (Hex)	Data (Hex)
27	1B	1C
36	24	23
45	2D	2A
54	36	31
63	3F	38
73	49	49
82	52	52
91	5B	5B
100	64	64
109	6D	6D
118	76	76
127	7F	7F

None of these bytes will be locked at the factory, unless specific order request is negotiated with the customer for an additional cost.

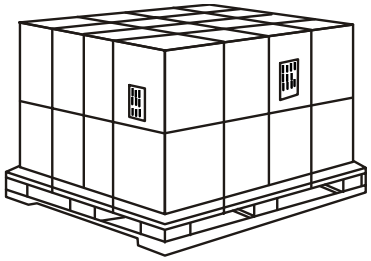
ANNEX D

RECOMMENDED LABEL LOCATIONS ON VARIOUS CONTAINERS

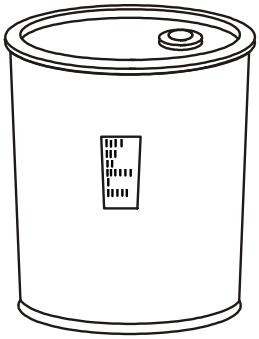
Change e), f), h), k), l) to be directly affixed to container. Add an example of b) showing on the pallet.



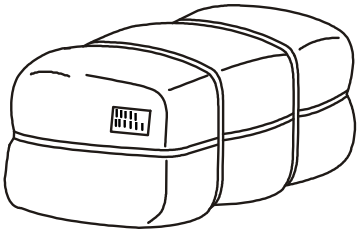
a) Box or carton with transport package label



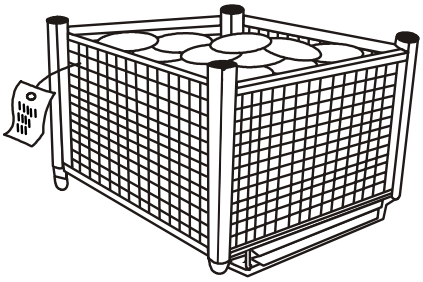
b) Pallet with two unit load labels



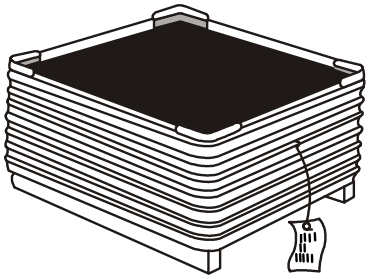
c) Drum, barrel, or cylindrical container



d) Bale

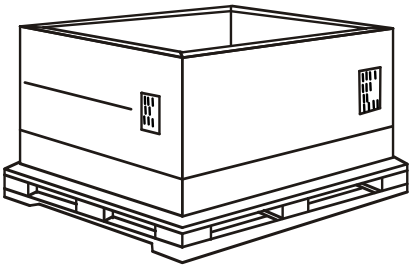


e) Basket, wire mesh container

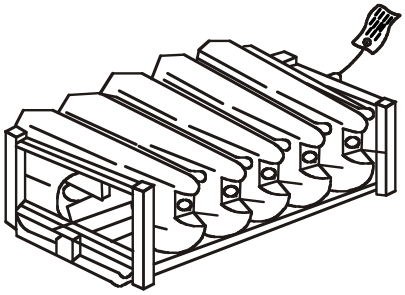


f) Metal bin or tub

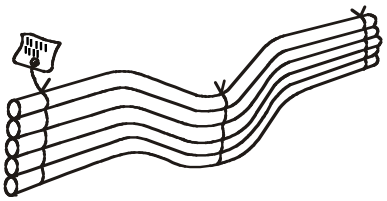
ANNEX D – LABEL PLACEMENT ON VARIOUS PACKAGES



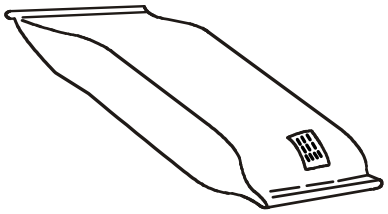
g) Pallet box



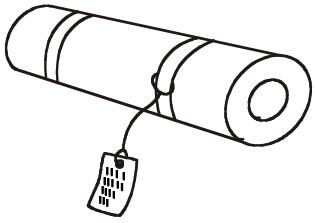
h) Rack



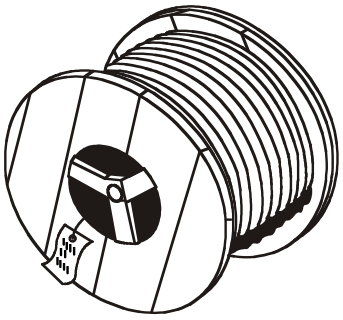
i) Bundle



j) Bag



k) Roll or coil



l) Reel of cable