INTERNATIONAL STANDARD

ISO/IEC 15961

First edition
2004-10-15

Information technology — Radio frequency identification (RFID) for item management — Data protocol: application interface

Technologies de l'information — Identification par radiofréquence (RFID) pour la gestion d'objets — Protocole de données: interface d'application

Reference number
ISO/IEC 15961:2004(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 15961 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 31, *Automatic identification and data capture techniques*.

Introduction

The technology of Radio Frequency Identification (RFID) is based on non-contact electronic communication across an air interface. The structure of the bits stored on the memory of the RF tag is invisible and accessible between the RF tag and the interrogator only by the use of the appropriate air interface protocol, as specified in the appropriate part of ISO/IEC 18000. The transfer of data between the application and the interrogator in open systems requires data to be presented in a consistent manner on any RF tag that is part of that open system. Application commands from the application and responses from the interrogator also require being processed in a standard way. This is not only to allow equipment to be interoperable, but in the special case of data carrier, for the data to be encoded on the RF tag in one systems implementation for it to be read at a later time in a completely different and unknown systems implementation. The data bits stored on each RF tag must be formatted in such a way as to be reliably read at the point of use if the RF tag is to fulfil its basic objective. The integrity of this is achieved through the use of a data protocol as specified in this International Standard and ISO/IEC 15962.

Manufacturers of radio frequency identification equipment (interrogators, RF tags, etc) and the users of RFID technology require a publicly available data protocol for RFID for item management. This International Standard and ISO/IEC 15962 specify this data protocol, which is independent of any of the air interface standards defined in ISO/IEC 18000. As such, the data protocol is a consistent component in the RFID system that may independently evolve to include additional air interface protocols.

The transfer of data to and from the application, supported by appropriate application commands is the subject of this International Standard. The companion International Standard, ISO/IEC 15962, specifies the overall process and the methodologies developed to format the application data into a structure to store on the RF tag.

Information technology — Radio frequency identification (RFID) for item management — Data protocol: application interface

1 Scope

The data protocol used to exchange information in an RFID system for item management is specified in this International Standard and in ISO/IEC 15962. Both International Standards are required for a complete understanding of the data protocol in its entirety; but each focuses on one particular interface:

- This International Standard addresses the information interface with the application system.
- ISO/IEC 15962 deals with the processing of data and its presentation to the RF tag, and the initial processing of data captured from the RF tag.

This International Standard focuses on the interface between the application and the data protocol processor, and includes the specification of the transfer syntax and definition of application commands and responses. It allows data and commands to be specified in a standardised way, independent of the particular air interface of ISO/IEC 18000.

This International Standard

- provides guidelines on how data shall be presented as objects;
- defines the structure of object identifiers, based on ISO/IEC 9834-1;
- specifies the commands that are supported for transferring data between the application and the RF tag;
- specifies the responses that are supported for transferring data between the RF tag and the application;
- provides a formal description of all the processes using ASN.1, as specified in ISO/IEC 8824-1;
- specifies the transfer syntax, based on the Basic Encoding Rules of ISO/IEC 8825-1, for data to be transferred from and to the application.

It is expected that this International Standard will be used as a reference to develop software appropriate for particular applications, or for particular RF equipment.

NOTE Conventionally in International Standards, long numbers are separated by a space character as a "thousands separator". This convention has not been followed in this International Standard, because the arcs of an object identifier are defined by a space separator (according to ISO/IEC 8824 and ISO/IEC 8825). As the correct representation of these arcs is vital to this International Standard, all numeric values have no space separators except to denote a node between two arcs of an object identifier.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 8824-1, *Information technology — Abstract Syntax Notation One (ASN.1) — Specification of basic notation* (equivalent to ITU-T Recommendation X.680)

ISO/IEC 8825-1, *Information technology — ASN.1 encoding rules — Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)* (equivalent to ITU-T Recommendation X.690)

ISO/IEC 9834-1, *Information technology — Open Systems Interconnection — Procedures for the operation of OSI Registration Authorities: General procedures* (equivalent to ITU-T Recommendation X.660)

ISO/IEC 15962:2004, *Information technology — Radio frequency identification (RFID) for item management — Data protocol: data encoding rules and logical memory functions*

ISO/IEC 18000 (all parts), *Information technology — Radio frequency identification for item management*

ISO/IEC 19762-1, *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary — Part 1: General terms relating to AIDC*[1)]

ISO/IEC 19762-3, *Information technology — Automatic identification and data capture (AIDC) techniques — Harmonized vocabulary — Part 3: Radio frequency identification (RFID)*[1)]

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19762-1, 19762-3 and the following apply.

NOTE: For terms defined below and in ISO/IEC 19762-1 or ISO/IEC 19762-3, the definitions given below apply.

3.1.1 Application command
The instruction issued from the application to the Data Protocol Processor in order to initiate an action or operation with the RF tag(s) via the interrogator.

3.1.2 Application memory
The area on the RF tag available for storing data written to it. Sometimes known as *user memory*.

3.1.3 Arc
A specific branch of an object identifier tree, with new arcs added as required to define a particular object. The top three arcs of all object identifiers compliant with ISO/IEC 9834-1 are defined in Annex A.

3.1.4 BER = Basic Encoding Rules
An ASN.1 encoding method.

3.1.5 Block
The minimum number of bytes on an RF tag that can be in a write transaction, or read transaction, across the air interface.

1) To be published.

3.1.6 Command / Response Unit
That part of the Data Protocol Processor that processes application commands and sends responses to control encoding, decoding, structuring of the Logical Memory and transfer to the Tag Driver.

3.1.7 Data carrier
A device or medium used to store data as a relay mechanism in an AIDC system, e.g. bar code, OCR character string, RF tag.

3.1.8 Data compaction
A mechanism, or algorithm, to process the original data so that it is represented efficiently in fewer octets in a data carrier than in the original presentation.

3.1.9 Data Compactor
The implementation of the data compaction process defined in ISO/IEC 15962.

3.1.10 Data Protocol Processor
The implementation of the processes defined in ISO/IEC 15962, including the Data Compactor, Formatter, Logical Memory, and Command / Response Unit.

3.1.11 Element name
A component of a Reference Type or enumerated list in ASN.1 syntax.

3.1.12 Formatter
The implementation of the data formatting process defined in ISO/IEC 15962.

3.1.13 Logical Memory
A software analogue on the Data Protocol Processor of the Logical Memory Map.

3.1.14 Logical Memory Map
An array of contiguous octets of memory on the RF tag, representing the application (or user) memory to be used exclusively for the encoding of objects, objectIds and their associated Precursor on the RF tag. The system information (see 7.1.2) shall be defined by different means, or stored in a separate area on the RF tag. This can be achieved by partitioning memory, partly for system information and mainly for the Logical Memory Map purpose.

3.1.15 Object
A well-defined piece of information, definition, or specification which requires a name in order to identify its use in an instance of communication.

3.1.16 Object identifier
A value (distinguishable from all other such values) which is associated with an object.

3.1.17 OBJECT IDENTIFIER type
A simple ASN.1 type whose distinguished values are the set of all object identifiers allocated in accordance with the rules of ISO/IEC 8824-1 (ITU-T X.680).

3.1.18 Octet
An ordered sequence of eight bits considered as a unit, equivalent to an 8-bit byte.

NOTE: The term is used in preference to "byte" in this International Standard and in the ASN.1 standards to avoid confusion in cases where there is a hardware association e.g. 7-bit byte, 16-bit byte.

3.1.19 RELATIVE-OID type
A particular object identifier where a common root-OID (for the first and subsequent arcs) is implied, and remaining arcs after the root-OID are defined by the RELATIVE-OID.

3.1.20 Response
The feedback received by the application from an application command sent to the Data Protocol Processor.

3.1.21 System information
Information held on the RF tag, or generated by unique features of the air interface, that specify Data Protocol parameters to establish the Logical Memory and other formatting rules.

3.1.22 Tag Driver
The implementation of the process to transfer data between the Data Protocol Processor and the RF tag.

3.1.23 Transfer syntax
The abstract syntax and concrete syntax used in the transfer of data between open systems.

NOTE: The term "transfer syntax" is sometimes used to mean encoding rules, and sometimes used to mean the representation of bits in data while in transit.

3.1.24 Type reference
A name, in ASN.1 syntax, associated uniquely with a characteristic e.g. ObjectId.

3.1.25 Unique item identifier
A code assigned to an item (for example a product, transport unit, returnable asset) that is unique within the domain and scope of a code system. When used with this Data Protocol, the particular ObjectId that defines the unique item identifier shall rely on the fact that each instance of its object shall be unique and unambiguous with all others related objects. As the object is unique, its use in the RF tag confers uniqueness to the RF tag itself.

3.2 Abbreviated terms

BER	Basic Encoding Rules (of ASN.1)
EAN.UCC	EAN International & Uniform Code Council, Inc
IATA	International Air Transport Association
UPU	Universal Postal Union

4 Protocol model

4.1 Overview

RFID supports bit encodation in the RF tag memory. Unlike other data carrier standards prepared by ISO/IEC JTC1 SC31 which require encodation schemes that are specific to the individual data carrier technology, ISO/IEC 18000 does not specify the interpretation of bits or octets encoded on the RF tag memory. However, as an RF tag is a relay in a communication system, each tag used for open systems item management needs to have data encoded in a consistent manner. The prime function of this International Standard is to specify a common interface between the application programs and the RF interrogator. The prime function of ISO/IEC 15962 is to specify the common encoding rules and logical memory functions.

RF tags utilise electronic memory, which is typically capable of increasing data capacity as new generations of product are introduced. Differences in data capacity of each RF tag type, whether similar or dissimilar, are recognised by the data protocol defined in these two International Standards.

Different application standards may have their own particular data sets or data dictionaries. Each major application standard for item management needs to have its data treated in an unambiguous manner, avoiding confusion with data from other applications and even with data from closed systems. The data protocol specified in these International Standards ensures the unambiguous identification of data.

4.2 Layered protocol

The protocol layers of an implementation of RFID for item management are illustrated schematically in Figure 1.

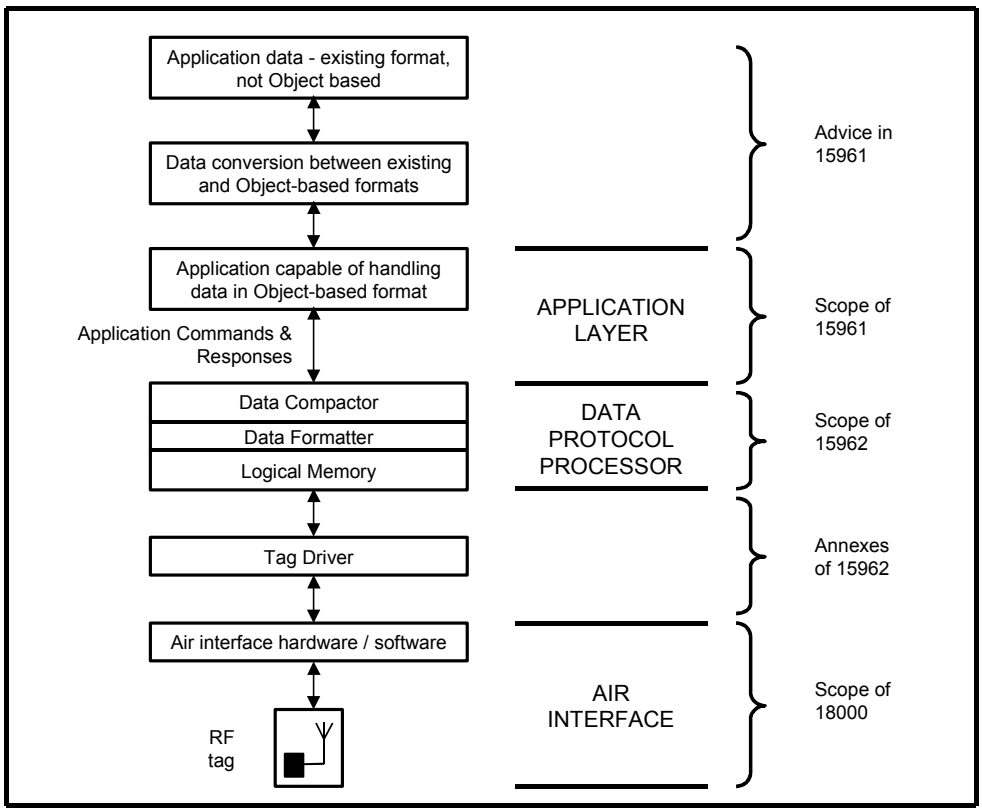


Figure 1 — Schematic of Protocol Layers for an Implementation of RFID for Item Management

The data protocol specified in this International Standard is independent of the different RF tag technologies specified in ISO/IEC 18000, which is concerned with different air interface protocols that function between the interrogator and the RF tag. This independence is achieved by implementing the standards at different levels in the protocol hierarchy. The RFID data protocol defined in this International Standard is primarily concerned with the upper layers as described below:

Application layer - as defined in this International Standard

- The RFID data protocol specifies how data is presented as objects, each uniquely identified with an object identifier, which are meaningful to the application and can be encoded on the RF tag.
- This RFID data protocol defines application commands and responses so that application programs can specify what data to transfer to and from the RF tag and to append, update or delete data on the RF tag.
- This RFID data protocol also defines error messages as responses to the application.

The application interface of this RFID data protocol is based on ASN.1, which:

- provides a means of defining the protocol which is independent of the host application, operating system, and programming language and also independent of the specific command structures between the interrogator and tag driver.
- identifies any data object distinctly from all others using object identifiers, even to enable different data formats to be intermixed on the same RF tag.

- defines unambiguous commands and responses, so that they can be intermixed with data on the same wired or wireless network.

- provides the abstract syntax for defining the commands and responses in a structured and consistent and verifiable manner, and provides the transfer syntax that defines the byte stream transferred between the processes of this International Standard and those of ISO/IEC 15962.

- enables implementation in a variety of computer languages through the use of compilers, alternatively programs can be written from the specification. In either case there is a vital need for the transfer syntax to be fully consistent and compliant to function in open systems where the sender and recipient can be unknown to one another.

Data Protocol Processing - as defined in ISO/IEC 15962

- The RFID data protocol specifies how data is encoded, compacted and formatted on the RF tag and how this data is retrieved from the RF tag to be meaningful to the application.

- This RFID data protocol provides for a set of schemes that compact the data to make more use of the memory space.

- This RFID data protocol also supports various storage formats to enable efficient use of memory and efficient access procedures.

All these features are described and specified later in this International Standard and its companion standard. Figure 1, and the outline description above, applies to a general process. Different rules may apply to RF tags that are capable of executing commands (see 7.3.3 of ISO/IEC 15962).

This RFID data protocol specifies the application level communication and the RF tag interrogator level rules for data encoding, compaction and storage formats. This protocol may be implemented:

- on the same platform as the application.

- on a separate platform linked to the application platform e.g. linked via a serial link, LAN or internet connection.

- on an embedded platform e.g. in a bar code printer/RFID encoder, in a bar code/RFID scanner, or a dedicated RFID interrogator.

This RFID data protocol has been designed such that the actual platform on which it is implemented is transparent to the application. It is also independent of the programming language used by the application. If both standards are not implemented, care will need to be taken to maintain the functionality between the two standards. The compliance clauses of both standards address these points in greater detail.

The rules specified in these International Standards create a complete independence between the application and the technology of the air interface and RF tag. The type of RF tag used in an implementation can be changed without requiring the application to change.

4.3 Functional processes

There are various functional processes that need to take place to write data to an RF tag and to read data from it. Figure 2 shows a schematic of an implementation where the processing of the data protocol resides in the interrogator. This illustration is provided to help with the understanding of the processes, and although a typical implementation, many others are possibly compliant with this data protocol.

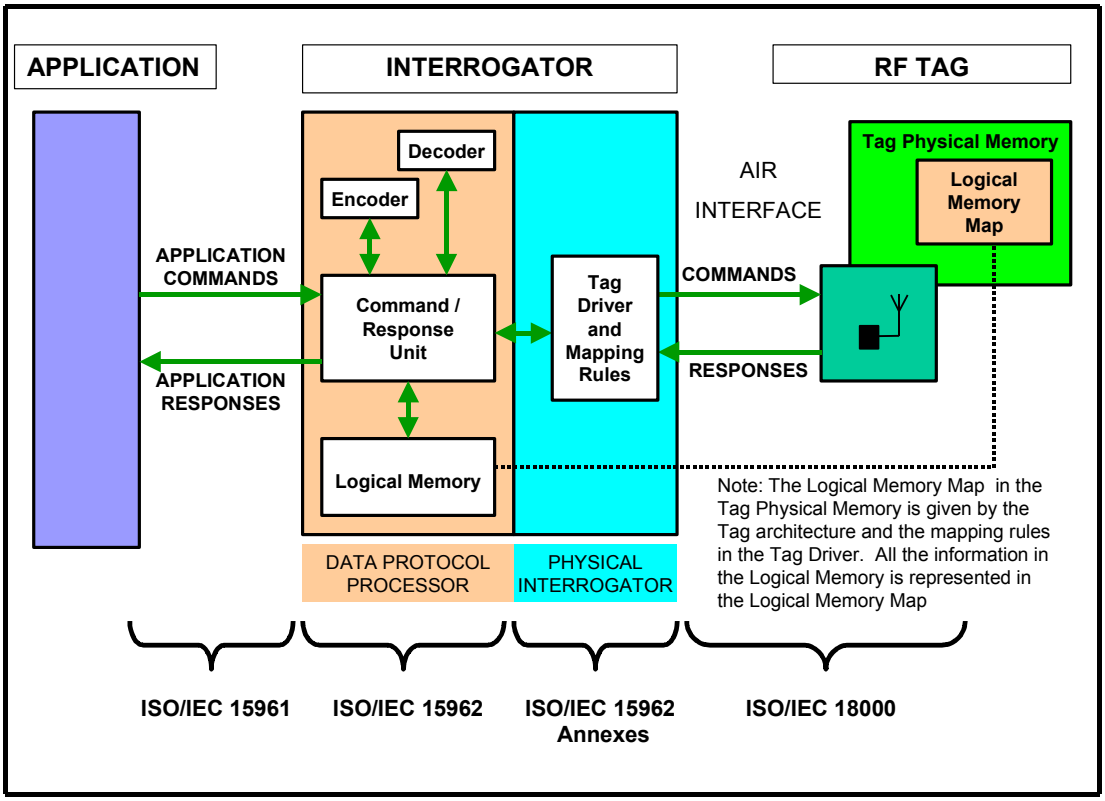


Figure 2 — Logical Functions and Interfaces

Application is the user application database and software.

The data flows between the application and the Data Protocol Processor are formatted according to this International Standard and are uncompacted. However, there are numerous established systems where data is formatted to be compliant, for example, with a bar code related syntax. It is therefore reasonable to insert interface modules in the data flow to convert from and to existing application formats.

NOTE: Careful consideration should be given to the extent that established systems need to be supported relative to the potential benefits to be gained from adopting the data protocol specified in this International Standard and ISO/IEC 15962. This is because this protocol has been developed around the features of RFID, such as selective read/write and the ability to lock data. Older protocols are unlikely to support such features.

Interrogator is the module in which all the basic processing of the data protocol takes place and there is an interface to the RF tag.

Data Protocol Processor provides all the processing, which is as specified in ISO/IEC 15962 and is required for handling application data. It consists of the following components, all of which are described more fully below: Command/Response Unit, Logical Memory, Encoder (which supports a Data Compactor and Formatter function) and Decoder (which supports the inverse functions of the Encoder). The Data Protocol Processor can physically reside anywhere between the application software and the tag driver but shall contain all the components.

Command/Response Unit for receiving the application commands from the application in a format specified in this International Standard, acting upon these commands where appropriate and converting to the specific RF tag lower level command codes.

EXAMPLE:

An application command of *write Data Object {name}* is application related. The data protocol recognises this and can format the data onto the Logical Memory in the Data Protocol Processor. The information from the particular RF tag is required to set the parameters of the Logical Memory Map (e.g. number of octets, whether a directory is in use, etc) on the RF Tag. The Tag Driver converts the application command into a tag-specific command.

It can be seen from this example that there is a distinct boundary between the Data Protocol Processor and the Tag Driver.

Logical Memory. This is an array of contiguous octets (or bytes) of memory acting as a common representation of the Logical Memory Map in the user memory of the RF tag to which the object identifiers and data objects are mapped in octets. The Logical Memory takes into account some parameters of the real RF tag, for example the block size, the number of blocks and the storage format. The Logical Memory ignores any detailed tag architecture.

The use of the Logical Memory means that an application can interface with an application-compliant RF tag, but that individual RF tags can have completely different memory capacities and architectures. This enables an implementation to benefit from new technological developments permitted within the framework of ISO/IEC 18000, such as larger capacity or faster access RF tags, without changing the application.

Encoder controls the process of writing data through the functional processes performed by the Data Compactor Module and Formatter Module.

Data Compactor provides the standard compaction rules to reduce the number of octets stored on the RF tag and transferred across the air interface. Numeric data, for example, is octet based to some coded character set for the application, but can be encoded in a compact form on the RF tag memory.

Formatter provides the processes to place the object identifier and object (data) into an appropriate and efficient format for storing on the Logical Memory.

NOTE: The physical mapping of bits to comply with the RF tag architecture is performed by the Tag Driver.

Decoder controls the process of reading and interpreting data through the functional processes performed by the Data De-compactor Module and De-formatter Module.

Tag Driver provides two main functions:

- It maps the contents of the Logical Memory to the Logical Memory Map of the RF tag in use.
- It provides facilities that accept the application commands of this data protocol, and converts them to a format that results in calls to command codes supported by the particular RF Tag. For example, an application command **write Data Object {name}** could result in the RF tag related command of write (block #, data).

The description of the tag driver for particular RF tags is provided in annexes of ISO/IEC 15962. For the purpose of ISO/IEC 15962, a tag driver is unique to a particular air interface type of RF tag as specified in the appropriate part of ISO/IEC 18000. This is a logical representation; physical implementation could combine features of different logical tag drivers. An interrogator may support one or many tag drivers.

RF Tag, although beyond the scope of these International Standards, is shown to complete the flow of data and commands.

Logical Memory Map represents all the data in the Logical Memory of the Data Protocol Processor converted (or mapped) to a location structure determined by the mapping rules in the Tag Driver and the architecture of the RF tag.

5 Data structure

5.1 Notation

5.1.1 The octet: the basic unit for 8-bit coding

This International Standard supports binary, 7-bit, 8-bit and user data that may exceed 8-bits per character. The common unit of coding is the octet (also known as the 8-bit byte). Binary data shall be padded with leading zero bits until the binary value is octet aligned; 7-bit data shall be represented as octets with bit 8 (see 5.1.2) set to a zero value. Data exceeding 8-bits shall be encoded in multiple octets.

An octet is represented as 2 hexadecimal characters with the values 0-9, A-F.

5.1.2 Bit ordering

Within each octet, the most significant bit is bit 8 and the least significant is bit 1. Accordingly, the weight allocated to each bit is:

Bit Value	bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
Weight	128	64	32	16	8	4	2	1

5.1.3 Octet conversion

The 8-bit value is converted into the two hexadecimal characters with bit 8, bit 7, bit 6 and bit 5 having the weights 8, 4, 2 and 1 respectively to define the first hexadecimal character. Bit 4, bit 3, bit 2 and bit 1 retain the weights 8, 4, 2 and 1 respectively to define the second hexadecimal character.

5.2 Structure of the transfer between ISO/IEC 15961 and ISO/IEC 15962

All object identifiers, data, commands, and responses transferred between the application and the Data Protocol Processor shall be octet aligned. This simplifies the construction of the transfer and aids parsing the octets. It has no significance on the encoding efficiency on the RF tag itself, because the process by the Data Protocol Processor (as specified in ISO/IEC 15962) controls the final encodation.

6 Abstract and transfer syntax

ASN.1 defines:

- an **abstract syntax**, which is effectively a data definition language used to define repetitive and optional structures using a number of primitive data types. It is equivalent to high level programming languages, but is independent (hence "abstract") from any of these.
- a **transfer syntax** from which the encoding rules are derived. These rules determine the bit pattern representation during the transfer of data structures created using the abstract syntax.

6.1 Abstract syntax

The abstract syntax for this International Standard shall be ASN.1 as defined in ISO/IEC 8824-1. The notation shall be as specified in that standard.

The syntax required for RFID for item management are specified in Clause 7.

6.1.1 Character set

The character set used to define an ASN.1 item shall consist of:

A to Z
a to z
0 to 9
: = , { } < . @ ( ) [ ] - ' " | & ^ * ; !

This character set is identical to that defined in ISO/IEC 8824-1.

6.1.2 Universal Types

ASN.1 supports a number of universal types that are fundamental to the syntax; sometimes called "built-in types". Each has been assigned a class tag in ISO/IEC 8824-1 to unambiguously identify each type of data. Universal types are shown in capital (uppercase) letters e.g. **UNIVERSAL**. The Universal Types used in this International Standard, together with their Class Tags, are shown in Table 1.

Table 1 — Universal Types Used in this Standard

Universal Type	Class Tag
BOOLEAN	1
INTEGER	2
OBJECT IDENTIFIER	6
OCTET STRING	4
RELATIVE-OID (reserved for future commands)	13
SEQUENCE & SEQUENCE OF	16

6.1.3 Type references

In addition to the Universal Types, ASN.1 enables application specific types to be defined. When a type is defined, it is given a name to reference it in another type assignment. Type references begin with an uppercase letter and the complete name is shown without spaces. There are a few variants to the presentation of the subsequent characters. This International Standard uses the convention of mixed upper and lowercase characters e.g. **TypeReference**.

The **TypeReference** name is followed by the three character sequence ": : =" to separate it from its definition.

Examples of **TypeReference** names that are used in this International Standard are:

ApplicationFamilyId
ObjectId
StorageFormat
TagId

All the **TypeReference** names used in this International Standard are defined in the appropriate sub-clause.

6.1.4 Element names

The components or elements of a TypeReference or enumerated list are named using a lowercase letter at the beginning, e.g. **elementName**. For some elements, further typing is required either to a TypeReference or a Universal Type.

Examples of **elementNames** that are used in this International Standard are:

accessMethod
applicationFamilyId
applicationSubFamily
commandCode
compactParameter
object
objectId
tagId

NOTE: In this International Standard, some **elementNames** and **TypeReference** names are often the same with the exception that the first letter is lowercase for the **elementName** and uppercase for the **TypeReference**.

All the **elementNames** used in this International Standard are defined in the appropriate sub-clauses.

6.1.5 Other ASN.1 conventions illustrated

By using a simple example of ASN.1 syntax, unrelated to the purpose of this International Standard, it is possible to illustrate the aspects covered in 6.1.2 to 6.1.4. Other aspects will also be identified.

EXAMPLE:

```
CustomerOrder : : = SEQUENCE {
    orderNumber         INTEGER
    name                OCTET STRING
    address             CustomerAddress
    productDetails SEQUENCE OF SEQUENCE {
                        productCode         OBJECT IDENTIFIER
                        quantity            INTEGER (1..999)
                        } ,
    urgency             ENUMERATED {
                        nextDay (0),
                        -- excludes Saturday and Sunday
                        firstClass (1),
                        roadTransport (2),
                        -- typically three days
                        }
}
```

NOTE:

- In this example, : : = separates the named TypeReference **CustomerOrder** from the definition.
- The curly brackets { } following **SEQUENCE** and the end, specify that **orderNumber, name, address, productDetails** and **urgency** are all elements of the CustomerOrder type reference.
- The element name **address** is further specified in the **CustomerAddress** type reference (excluded from the example for brevity).
- The element **productDetails** consists of two further elements **productCode** and **quantity**. This pair of elements is repeated n times, based on the **SEQUENCE OF SEQUENCE**. The { } define the boundary of the Type.
- The element **urgency** offers one of three codes: 0, 1 or 2 by the use of the **ENUMERATED** type. The { } define the boundary of the Type.
- Comments in this example "excludes Saturday and Sunday" and "typically three days" are preceded by the double dash "--".

- The **INTEGER** value for the element **quantity** is constrained by the "(1..999)" to be any value in the range 1 to 999, making it an error to order 1000 or more items of a **productCode**.

6.1.6 Modular structure of ASN.1 syntax

In keeping with the ASN.1 standards, the syntax in this International Standard has been presented in a modular format. Separate modules are used for the commands and for the responses. Each module contains:

- A unique name.
- A unique object identifier that refers to this particular standard (in accordance with ISO/IEC 8824-1). The penultimate arc is either "commandModules (126)" or "responseModules (127)" to distinguish the data flows. The final arc of each command/response pair has the same name and value to link these together.
- The use of the key words DEFINITIONS, BEGIN and END to be compliant with ISO/IEC 8824-1 and to allow the modules to be processed through compiler tools.
- A statement that this standard uses "EXPLICIT TAGS", which indicates that all of the elements ultimately encoded as UNIVERSAL TYPES.

The structure of a command module follows the following common format:

```
Module Name
  {ISO(1) standard(0) rfid-data-protocol (15961) commandModules (126) moduleName(n)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

CommandName
-- assignments

END
```

The responseModule follows a similar format.

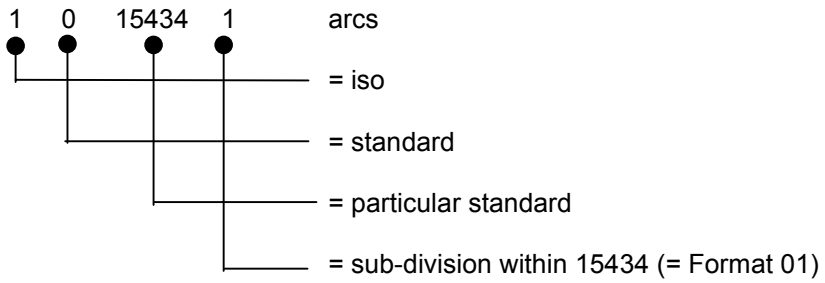
Within each module, all of the elements are defined in such a way as to reduce these to UNIVERSAL TYPES. This avoids the need to implement any import function within the module.

6.1.7 Object identifiers

This International Standard shall use the OBJECT IDENTIFIER type as defined in ISO/IEC 8824-1 and with identifiers assigned as specified in ISO/IEC 9834-1. This uses a registration tree with a common implied root node (ISO/IEC 9834-1), a series of arcs from each node, with new arcs added as required to define a particular object. Thus, the body responsible for a particular node:

- has a defined set of arcs to identify itself
- can manage the allocation of arcs under its node, independently of other bodies
- is assured of uniqueness from all other arcs in the registration tree

EXAMPLE:



The only top arcs permitted for all object identifiers are shown in Table 2.

Table 2 — Object Identifier Top Arcs

Identifier Arc Name	Numeric Value
itu-t	0
iso	1
joint-iso-itu-t	2

The second arc is administered by the relevant organisation named for the top arc. The current list of top and second arcs is given in Annex A.

The third arc is controlled by the system or body defined for the second arc (see Annex A); sometimes this is a Registration Authority. The hierarchical structure continues until the object is identified uniquely. The procedure of naming object identifiers ensures that each object is unique within its "parent" arc and that each parent arc is unique within its previous level, right back to the top 3 arcs.

NOTE: This structure enables object identifiers from different domains (e.g. open and closed systems) to be encoded unambiguously on an RF tag memory.

Two forms of object identifier can be used with the RFID Data Protocol:

- OBJECT IDENTIFIER: This full structure is used for communications between the application and the Data Protocol Processor defined by the scope of this International Standard. The OBJECT IDENTIFIER type is fully specified in 6.2.2. The full structure is suitable for use in ISO/IEC 15962 where the set of object identifiers to be encoded on the RF tag have different higher level arcs. In such cases all the arcs of the OBJECT IDENTIFIER shall be encoded in the RF tag.

- RELATIVE-OID: This structure is not used for communication between the application and the Data Protocol Processor defined by the scope of this International Standard. However, the RELATIVE-OID structure is applied in ISO/IEC 15962 in situations where a common root applies to the set of object identifiers to be encoded on the RF tag. For example, if all the object identifiers have the common root 1 0 15434 1 (as in the example in above) encoding space can be saved on the RF tag if this common root does not have to be encoded for each object identifier. The RELATIVE-OID assumes a common root-OID, which is either encoded or declared in some other way.

  NOTE: The specification of the RELATIVE-OID is included in this International Standard to illustrate for those mainly concerned with the application interface how it is used for encoding by the Data Protocol Processor of ISO/IEC 15962.

6.2 Transfer syntax

The transfer syntax of this International Standard is based on the Basic Encoding Rules (BER) of ASN.1, as specified in ISO/IEC 8825-1.

6.2.1 Structure of the transfer encoding

The structure of the transfer encoding for the data protocol for RFID for Item Management is described below:

1. type identifier octet(s) that encode the ASN.1 tag (class and number) of the Type used to qualify the data value (see 6.2.2);

2. length octets that define a count of the octets that make up the contents (see 6.2.3);

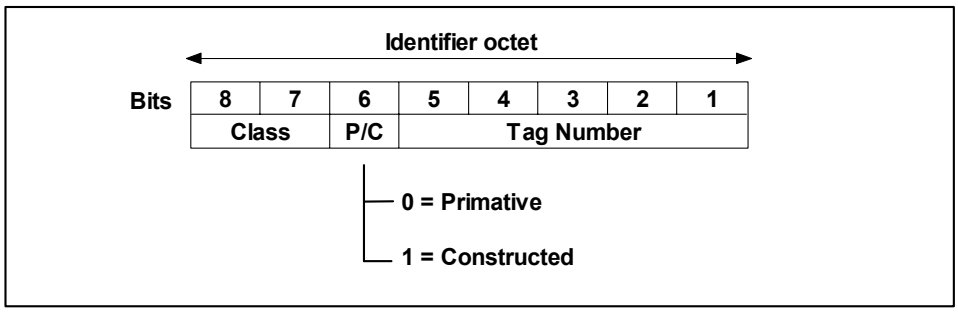
3. content (or value) octets (see 6.2.4).

This is sometimes known as Type, Length, Value (**TLV**). When the encoding is based on a sequence of **TLV**, it is known as Primitive encoding. The value V can be a triplet of **TLV**, and when this structure is used it is known as Constructed encoding, for example **TL TLV TLV TLV**. The choice between Primitive and Constructed encoding is largely determined by the Basic Encoding Rules of ISO/IEC 8825-1. The constructor types of SEQUENCE and SEQUENCE OF that shall use a Constructed encoding to be fully compliant the ASN.1 standards. Otherwise, the rules of ISO/IEC 8825-1 require that Primitive encoding is used; or offer a choice, in which case only Primitive encoding shall be used in this International Standard. The option is clearly defined for each of the Universal Types (see 6.1.2) that have their BER encoding rules defined in subsequent clauses.

The module OBJECT IDENTIFIER shall be encoded in a TLV structure at the beginning of the transfer.

6.2.2 Encoding the ASN.1 type identifier

The ASN.1 type identifier shall be encoded as a single octet for the Types defined in this International Standard, as illustrated in Figure 3.



where: bits 8 and 7 shall encode the class of ASN.1 tag as defined in Table 3

bit 6 shall encode whether this is a Primitive or Constucted tag (see below)

bits 5 to 1 shall encode the number of the ASN.1 tag

Figure 3 — Type Identifier Octet Structure

The 2-bit value for Class shall be one of the values defined in Table 3. For the Universal Types defined in this International Standard, the value shall be '00_2'.

Table 3 — Encoding of ASN.1 Class of Tag

Class	Bit 8	Bit 7
Universal	0	0
Application	0	1
Context-specific	1	0
Private	1	1

The single bit value for the 'P/C' component shall be set to '0_2' to indicate Primitive encoding structures, or shall be set to '1_2' to indicate constructed encoding structures.

The 5-bit value for the ASN.1 tag shall encode the Class tag number as a binary integer with bit 5 as the most significant bit. The Class tags specified for this International Standard are defined in Table 1.

EXAMPLE:

Universal Type = OBJECT IDENTIFIER

ASN.1 Type identifier = 00 0 00110

6.2.3 Length encoding

Although ISO/IEC 8825-1 allows other forms of length encoding, only the definite form shall be used in this International Standard because this applies to both primitive and constructed encoding. For the definite form, the length octets shall consist of one or more octets, depending on the number of octets in the contents. If the number of octets in the contents is less than, or equal to, 127, then a single length octet shall be used. Bit 8 shall be '0_2' and bits 7 to 1 shall encode the number of octets in the content (which may be zero) as an unsigned binary integer with bit 7 as the most significant bit.

EXAMPLE:

L = 38 is encoded as 00100110_2

If the number of octets in the contents is more than 127, then two or more length octets shall be used. The length shall be converted to an octet aligned value, for example, a length of 201 octets converts to $C9_{16}$ (or 11001001). This value is encoded in the second and subsequent octets. The first octet shall be encoded as follows:

a. Bit 8 shall be 1_2.

b. Bits 7 to 1 shall encode the number of subsequent octets in the length octets, as an unsigned binary integer with bit 7 as the most significant bit.

c. The value 11111111_2 shall not be used to allow for future extension:

EXAMPLE:

Length of content = 357
Convert to HEX = 01 65_{16}
= 00000001_2 01100101_2
As this is 2 octets, the first octet = 10000010_2

The complete length encoding is:

10000010 00000001 01100101_2
= 82 01 65_{16}

6.2.4 Contents Octets

The contents octets encode the data value, which can be zero, one or more octets, depending on the Universal Type as specified in subsequent subclauses.

6.2.5 Encoding of a BOOLEAN value

The encoding of a BOOLEAN value shall be primitive to comply with ISO/IEC 8825-1. The Boolean value shall be encoded in a single octet. If the Boolean value is FALSE, the octet shall be zero. If the Boolean value is TRUE, the octet shall have any non-zero value, at the sender's option.

6.2.6 Encoding an INTEGER value

The encoding of an INTEGER value shall be primitive to comply with ISO/IEC 8825-1. The integer shall be encoded in one or more octets using the following procedures.

For positive integers and zero:

1. The whole number is converted to a binary integer number in a bit field with the most significant bit first.
2. The bit field is aligned to octet boundaries by adding leading zero bits.
3. If the high order bit is 1, add a pad octet 00_{16} as a prefix.

   NOTE: The high order bit of 0 denotes that the encodation is of a positive integer value.

   EXAMPLE:

   Integer 128
   Step 1: 10000000
   Step 2: 10000000
   Step 3: 00000000 10000000

For negative integers, the encodation is to a twos-complement rule:

1. The whole number is converted to a binary integer number in a bit field with the most significant bit first.
2. The bit field is aligned to octet boundaries by adding leading zero bits.
3. The binary value from Step 2 is bit complemented (ie 0 to 1, 1 to 0).
4. The twos-complement rule is applied adding 1_2 to the bit string of Step 2.
5. If the high order bit is 0, add a pad octet FF_{16} as a prefix.

   NOTE: The high order bit of 1 denotes that the encodation is of a negative integer value.

   EXAMPLE:

   Integer -27066
   Step 1: 1101001 10111010
   Step 2: 01101001 10111010
   Step 3: 10010110 01000101
   Step 4: 10010110 01000110

For decoding, the lead bit of the encoded integer value identifies whether the value is positive or negative.

If it is a positive value, conversion takes place on the remaining bits with the least significant bit being in position 0. The decimal integer value is the sum of the values 2^n, where n is the position number, or:

$$\sum_{n=0}^{p-1} 2^n$$

If it is a negative value, conversion takes place on the remaining bits with the least significant bit being in position 0. The first stage is to create a decimal integer value as the sum of the values of 2^n. The second stage takes this as a positive decimal integer from which is subtracted the value 2^p, where p is the position number of the lead bit that identifies this as a negative integer. As an equation, this is:

$$\sum_{n=0}^{p-1} 2^n - 2^p$$

EXAMPLE:

10010110	01000110		
1			indicates -ve
0010110	01000110	=	5702
$2^p = 2^{15} =$			32768
5702 - 32768 = -27066			

6.2.7 Encoding the OBJECT IDENTIFIER value

The encoding of an OBJECT IDENTIFIER value shall be primitive to comply with ISO/IEC 8825-1. The object identifier value is encoded as a series of octet aligned values as follows:

1. The first two arcs of the registration tree are encoded as a single integer using the formula:

   40f + s
   where f = the value of the first arc
   s = the value of the second arc

2. The value "n" of each additional arc is encoded into an octet-aligned-bit-field. This is done as follows for values of "n":

   a. For n < 128:

      the decimal value is converted to binary and encoded in a single octet; thus bit 8 is set to zero

   b. For : 128 ≤ n < 16384:

      the decimal value is converted to binary and subdivided into two 7-bit strings: bit 7 to bit 1, bit 14 to bit 8. Each of these new bit strings is encoded in an octet; with bit 8 of the first octet set to 1, bit 8 of the last octet set to 0.

   c. For n ≥ 16384:

      the decimal value is converted to binary and subdivided into 7-bit strings: bit 7 to bit 1, bit 14 to bit 8, bit 21 to bit 15, and so on. Each of these new bit strings is encoded in an octet; with bit 8 of the first octet set to 1, bit 8 of the last octet set to 0 and bit 8 of the intervening octet(s) set to 1. The example below shows the process.

EXAMPLE:

1. Value = 91234_{10}

   = $1\ 01100100\ 01100010_2$

2. Split into 7-bit strings

   0000101 1001000 1100010

3. Add prefix bits 0 for last octet

   1 for preceding octet(s)

   $1_{0000101}\ 1_{1001000}\ 0_{1100010}$

Using this technique, the length of each component arc of the OBJECT IDENTIFIER is self declaring. The first octet always defines the first two arcs. Each subsequent arc is defined by one octet if the lead bit of the next octet is 0; and multiple octets if the lead bit is 1, the group of octets ends with the octet with its lead bit equal 0. The arc value is encoded in the sequence of 7-bit values.

EXAMPLE:

[00101000]	1[1111000]	0[1001010]	0[0000001]
(1 x 40) + 0	15434		1
1 0	15434		1

Although the number of arcs allows for an OBJECT IDENTIFIER of any length, this International Standard limits the length of the encoded value to be no more than 127 octets. This is a constraint placed to meet encoding requirements on the RF tag and the structure of the Logical Memory.

NOTE: The constraint is on the encoded length of the OBJECT IDENTIFIER and not the number of arcs. It should also be understood that an OBJECT IDENTIFIER encoded in 127 octets is highly unlikely.

6.2.8 Encoding an OCTET STRING value

Although the Basic Encoding Rules of ISO/IEC 8825-1 permit both forms of encoding, this International Standard only supports primitive encoding of an OCTET STRING value.

The primitive encoding contains zero, one or more octets equal in value to the octets in the application data value. The encoded octets appear in the same order as they appear in the data value and with the most significant bit of an octet being aligned in both the encoded and data presentations.

NOTE: This means that, for open systems, the octet sequence and bit ordering shall be output by the receiving system exactly as input by the sending system. It is the responsibility of the application standards to define the requirements for the sequence.

6.2.9 Encoding a SEQUENCE value

The encoding of a SEQUENCE value shall be constructed to comply with ISO/IEC 8825-1. The contents octets shall consist of the complete TLV encoding of one data value from each of the Types listed in the ASN.1 definition of the particular SEQUENCE Type. The data values shall be in the order of their appearance in the definition. Although ISO/IEC 8825-1 allows optional rules for Types with the keywords 'OPTIONAL' or 'DEFAULT' in the ASN.1 definition, this International Standard requires all Types in the SEQUENCE to appear in the constructed encoding.

EXAMPLE:

ASN.1 definition
SEQUENCE {orderNumber OCTET STRING, product OCTET STRING, quantity INTEGER}

with the values:
{orderNumber "ABC1234", product "widget", quantity "12"}

is encoded as:

T = SEQUENCE	L
30_{16}	14_{16}

T = OCTET STRING	L	V
04_{16}	07_{16}	"ABC1234"
T = OCTET STRING	L	V
04_{16}	06_{16}	"widget"
T = INTEGER	L	V
02_{16}	01_{16}	$0C_{16}$

6.2.10 Encoding a SEQUENCE OF value

The SEQUENCE OF type has the same ASN.1 tag (UNIVERSAL 16) as the SEQUENCE type; therefore, adopting the same encoding rules. The encoding of a SEQUENCE OF value shall be constructed to comply with ISO/IEC 8825-1. The contents octets shall consist of the complete TLV encoding of each value, including the encoding of the repeated UNIVERSAL class tag of the encoded elements.

EXAMPLE:

ASN.1 definition
sequence of {productCode OCTET STRING}

with the three values:
{productCode "ABC1234", "X6789Y", "PQR12345"}

is encoded as:

T = SEQUENCE	L
30_{16}	$1B_{16}$

T = OCTET STRING	L	V
04_{16}	07_{16}	"ABC1234"
T = OCTET STRING	L	V
04_{16}	06_{16}	"X6789Y"
T = OCTET STRING	L	V
04_{16}	08_{16}	"PQR12345"

7 Data flows and processes

Various processes are required to format the RF tag, to write data to it, to read from it, to modify data etc. These are defined in the sub-clauses that follow. All the processes to write and add data will be described. Where the read process is the inverse of the write process, this will be described briefly, otherwise a further description will be provided. Within this clause and subsequent clauses, the appropriate transfer encoding values will be shown.

7.1 Establishing communications between the application and the RF tag

The Data Protocol Processor does not communicate directly with the RF tag (see Figures 1 and 2), but does this through the Tag Driver. The Data Protocol Processor requires specific system information based on the configuration of the RF tag (see 7.1.2). This is to set parameters of the Logical Memory to represent correctly the

RF tag memory and to enable communication to and from the Tag Driver. To achieve this, air interface services shall be provided to the Data Protocol Processor via the Tag Driver to establish communications (see 7.1.1).

Various of these parameters need to be known to, or requested by, the Data Protocol Processor. Effectively, this procedure is used to configure the RF tag initially, to re-configure it if required, and to establish a communications link while the data transaction is open.

7.1.1 Air interface services

This International Standard is open-ended with respect to the fact that new types of RF tag may be added to ISO/IEC 18000 and leave the Data Protocol Processor unaltered. To achieve this, some basic presumptions are made about the types of RF tag in ISO/IEC 18000.

- Application memory is an integer number of octets.

  NOTE: the term application memory is used in this sub-clause as a generic name for the area of RF tag memory available for user data; this is sometimes called user memory in ISO/IEC 18000.

- Application memory shall be organised in blocks. These shall be fixed size and be of one or more octets.

- The individual blocks shall each be accessible by read and/or write.

  NOTE: This applies to the basic function, additional features may be used to restrict access to authorised users.

- In addition to the requirements (above) relating to the memory, there shall be a reliable mechanism for writing and reading to and from the application memory.

The RF tag shall have a mechanism for storing the system information (see 7.1.2), including the ability to write and read the component elements.

The technical details of the air interface services are provided in ISO/IEC 15962.

7.1.2 System information

The system information shall consist of the following elements that need to be transferred across the application interface and air interface, and are therefore part of this International Standard and ISO/IEC 15962:

tagId
applicationFamilyId
storageFormat, which itself consists of:
    accessMethod
    dataFormat

The terms and functions are described in the sub-clauses below.

The system information shall also consist of the following elements that only need to transfer between the RF tag and the Data Protocol Processor to configure the Logical Memory Map, and are therefore only part of ISO/IEC 15962:

physical block size
number of blocks

7.1.2.1 TagId (TypeReference and elementName)

TagId shall be provided by the Tag Driver for the purposes of identifying the RF tag unambiguously for at least the period of a data transaction. **TagId** acts as a file reference for the Logical Memory, and in turn provides a one-to-one link to the Logical Memory Map of the RF tag itself. **TagId** can be based upon one of the following:

a. A completely unique ID programmed in the RF tag, as specified in the ISO/IEC 18000 series).

b. A data related identifier (e.g. like the unique identifier of transport units as specified in ISO/IEC 15459-1), that provides for uniqueness within the particular domain of item management or logistics. This requires the data to be read to establish the **TagId**.

c. A virtual or session ID based on a time slot or other feature managed by the air interface protocol.

d. Combinations of (b) and (c), e.g. a virtual ID across the air interface, but requiring the data related identifier to be returned as a response.

The TypeReference **TagId** specifies its format as up to 255 octets long.

7.1.2.2 ApplicationFamilyId (TypeReference and elementName)

The value of TypeReference **ApplicationFamilyId** refers to specific identifiers that enable selective addressing of RF tags. This may be supported by a mechanism at the air interface. Because **ApplicationFamilyId** is supported by standards for smart card, the structure that follows is non-conflicting with those standard. The value of the TypeReference **ApplicationFamilyId** for RFID for Item Management can be stored on the RF tag in some form, or can be determined by the air interface services if these are sufficiently specific. The value of **ApplicationFamilyId** shall consist of two integer values concatenated into a single octet when stored within system information on the RF tag.

The first integer refers to the elementName **applicationFamily** and the following codes shall apply:

0	addresses all families
1 to 8	as assigned by SC17 and defined in ISO/IEC 15693-3 and 14443-3
9 to 12	as assigned by SC31 and defined in this International Standard below
13 to 15	reserved for future definition by ISO/IEC

The second integer of the TypeReference **ApplicationFamilyId,** for the **applicationFamilies** 9 to 12, refers to the elementName **applicationSubFamily** and the following codes shall apply:

0	reserved for SC17 purposes to address all sub families within the selected family; and therefore shall only be used in command arguments and shall not be encoded in the RF tag
1 to 15	assigned for RFID for Item Management as defined in this International Standard

This feature currently enables up to 60 specific selection criteria to be defined for managing major types of data for RFID for Item Management. The detailed assignments are under the direct control of ISO/IEC JTC1/SC31/WG4 and these are defined in Annex B.

If **ApplicationFamilyId** is not supported by a class of RF tag, and the services from the air interface do not provide this by other means, the value of the octet in the system information for the RF tag shall be 00.

7.1.2.3 StorageFormat (TypeReference and elementName)

The TypeReference and elementName **StorageFormat** contains a sequence of elementNames **accessMethod** and **dataFormat**, both of which are defined below.

7.1.2.4 accessMethod (elementName)

The elementName **accessMethod** shall define the manner in which data can be mapped on the RF tag and be accessed from the RF tag. The value of elementName **accessMethod** should be stored on the RF tag or may be defined by the air interface services, if this can be done unambiguously. **AccessMethod** shall be an integer value and the following codes shall apply:

0 **noDirectory** - This is a structure that supports the contiguous abutting of all the data, related object identifiers, and other data protocol overhead. Generally, all the data needs to be transferred across the air interface to abstract relevant data.

NOTE: The command defined in 8.20 may be used to reduce the amount of data transferred.

1 **directory** - The data is encoded exactly as for **noDirectory** but the RF tag supports an additional directory which is first read to point to the address of the relevant object identifier.

2 **selfMappingTag** - This can be selected if the RF tag has on-board processes to organise the data mapping. Such RF tags shall be capable of performing similar functions to that of the Formatter module of the Data Protocol Processor.

3 reserved for future definition by SC31.

Where a choice can be made, the following guidelines can be of assistance:

- The **noDirectory** structure is better suited to RF tags with small memory capacity, because the directory itself is an overhead that needs to be encoded. It also is better suited where there are few **objectIds** to be encoded, so that a continuous read function will transfer all the encoding (or at least sufficient of the encoded octets) to enable the octets to be parsed to find the required **objectId** and associated **object.**

- The **directory** structure is obviously better suited where the conditions differ from those suited to the **noDirectory** structure. In addition, it is better suited to applications that call for selective reading, writing, or modifying one or few **objectIds** from among many. In this type of situation, the extra read process to transfer the directory to the Data Protocol Processor are likely to be balanced be the shorter read time for the selected **objectId.**

- As the **accessMethod** has to be specified by the application, it should be possible to measure transfers of data that simulate a directory plus data structure with that of a **noDirectory** structure. Such a test can be used to determine a reasonable breakeven point between the two **accessMethods**, taking into account the variety and length of data and the typical read/write implementations expected for an application.

Once an **accessMethod** has been specified for the RF tag, the application does not need to qualify how reading, writing, or the organisation of the octets on the Logical Memory is to be achieved. This is the function of the Data Protocol Processor and the Tag Driver.

7.1.2.5 dataFormat (elementName)

The elementName **dataFormat** shall define the types of object identifier being stored on the RF tag. The **dataFormat** is used to make more efficient use of encoding space or to restrict data to one class. Mechanisms include:

- linking the **dataFormat** with the **applicationFamilyId**, which enables higher level and lower level selection processes to be linked.

- making use of a RELATIVE-OID where a common sub-tree applies to all object identifiers. This requires the arcs of the common sub-tree, or root, to be encoded only once in a root-OID (see 8.3.2 of ISO/IEC 15962). It is even possible for the root-OID to be signalled without it being actually encoded on the RF tag.

- enabling RF tags that have a **dataFormat** compliant with one particular root-OID to support other OBJECT IDENTIFIERS (see 8.3.2 of ISO/IEC 15962).

The value of **dataFormat** may be stored on the RF tag or may be defined by the air interface services if this can be done unambiguously. When defined by the air interface protocol, there shall be a one-to-one relationship with the type of RF tag and the value of **dataFormat**.

The text that follows below describes the formats; particular details about encoding rules follow in 7.2. The value of **dataFormat** shall be an integer value and the following codes shall apply:

0 **notFormatted** - This is used for RF tags not formatted to this International Standard, e.g. for closed system applications. It is also the default for an RF tag yet to be formatted.

1 **fullFeatured** - This data format supports any type of data format where the OBJECT IDENTIFIER type is used. Its prime purpose is to enable heterogeneous data (i.e. from different data dictionaries) to be encoded on the one RF tag. For example it could be used to encode data from different closed system applications unambiguously, using the ISO/IEC 9834-1 registered object identifiers for each application.

2 **rootOidEncoded** - This data format is used when all the data on the RF tag has a common root-OID, but where this does not comply with one of the specific root-OID formats described below.

NOTE: The reason for using the specific root-OIDs (see below) where they apply, and not use **dataFormat** 2, is because the RF tags and other aspects of the system support particular features of those **dataFormats** not available with **dataFormat** 2. In addition to the root-OID of **dataFormats** 3, 4, and 8 to 12 not having to be encoded, linking the **dataFormat** to the **ApplicationFamilyId** provides further systems features.

The root-OID for **dataFormat** 2 shall be directly encoded on the RF tag using the appropriate root arcs. Each object is encoded on the RF tag using a RELATIVE-OID, representing the remaining lower order arcs.

3 **iso15434** - This is used when data is from one or more formats of ISO/IEC 15434. The root-OID shall be {1 0 15434}, and shall not be encoded on the RF tag because **dataFormat** 3 signals this. Each object is encoded on the RF tag using a RELATIVE-OID, representing the ISO/IEC15454 format number and any further lower order arcs.

4 **iso6523** - This data format is used when all the data on the RF tag is from a set belonging to one or more International Code Designators (ICD) compliant with ISO/IEC 6523-1. The first three arcs of the root-OID shall be {1 0 6523}, and shall not be encoded on the RF tag because **dataFormat** 4 signals this. Where only a single ICD defines the data on the RF tag, the ICD code (1...9999) shall be encoded once as the root-OID. Each object is encoded on the RF tag using a RELATIVE-OID, representing the next three arcs according to ISO/IEC 6523-1.

NOTE: If, unusually, the RF tag needs to encode data from more than one ICD, the main or original ICD shall be encoded as a root-OID**.** Additional objects shall be encoded as full OBJECT IDENTIFIERS.

5 **iso15459** - This is used when data is from one or more formats of ISO/IEC 15459. The root-OID shall be {1 0 15459}, and shall not be encoded on the RF tag because **dataFormat** 5 signals this. Each object is encoded on the RF tag using a RELATIVE-OID, representing the ISO/IEC15459 class number and any further lower order arcs.

6 reserved.

7 reserved

8 **iso15961Combined** - This data format is used when all the data on the RF tag complies with two or more root-OIDs of ISO/IEC 15961 (currently **dataFormat** 8 to 12). The root-OID shall be {1 0 15961}

and shall not be encoded on the RF tag because **dataFormat** 8 signals this. Each object is encoded on the RF tag using a RELATIVE-OID, representing the remaining lower order arcs.

9 **ean-ucc** - This data format is used when all the data on the RF tag complies with the EAN.UCC system (as referred to in ISO/IEC 15418). The root-OID shall be {1 0 15961 9} and shall not be encoded on the RF tag because **dataFormat** 9 signals this. Each object is encoded on the RF tag using a RELATIVE-OID, representing a data element according to the rules of EAN.UCC any further lower order arcs.

10 **di** - This data format is used when all the data on the RF tag complies with the Data Identifier standard (as referred to in ISO/IEC 15418). The root-OID shall be {1 0 15961 10} and shall not be encoded on the RF tag because **dataFormat** 10 signals this. Each object is encoded on the RF tag using a RELATIVE-OID, representing a data element according to the rules of the Data Identifier standard and any further lower order arcs.

NOTE: As the Data Identifiers are alphanumeric in structure a conversion algorithm as defined in 7.2.3 shall be used to create a numeric RELATIVE-OID.

11 **upu** - This data format is used when all the data on the RF tag complies with data elements defined by UPU. The root-OID shall be set to {1 0 15961 11} and shall not be encoded on the RF tag because **dataFormat** 11 signals this. Each object is encoded on the RF tag using a RELATIVE-OID, representing a data element from appropriate UPU standards and any further lower order arcs.

12 **iata** - This data format is used when all the data on the RF tag complies with data elements defined by IATA. The root-OID shall be set to {1 0 15961 12} and shall not be encoded on the RF tag because **dataFormat** 12 signals this. Each object is encoded on the RF tag using a RELATIVE-OID, representing a data element from appropriate IATA standards and any further lower order arcs.

13 to 31 reserved.

7.2 Preparing the basic objects

This is an initial process to ensure that the data objects are prepared in a format compatible with this International Standard. It is recognised that there are message-based protocols and syntax for existing AIDC application standards that differ from the object-based protocol of this International Standard. It is possible to achieve the benefits of the object-based protocol and maintain compatibility with message based systems by converting between the two formats (see Annex C).

The outputs described in the sub-clauses that follow shall be the format of inputs to the Data Protocol Processor. However, the mechanisms to achieve them on input and output are not. The requirement is to assign an **objectId** to each **object** using the data dictionary relevant to the application standard. Specific rules are covered in 7.2.1 to 7.2.8.

7.2.1 General model

Figure 4 provides a data flow model for preparing each object identifier and associated data object. Each type of application data, and particularly those covered by this International Standard, has a data dictionary, or list of data objects or data elements.

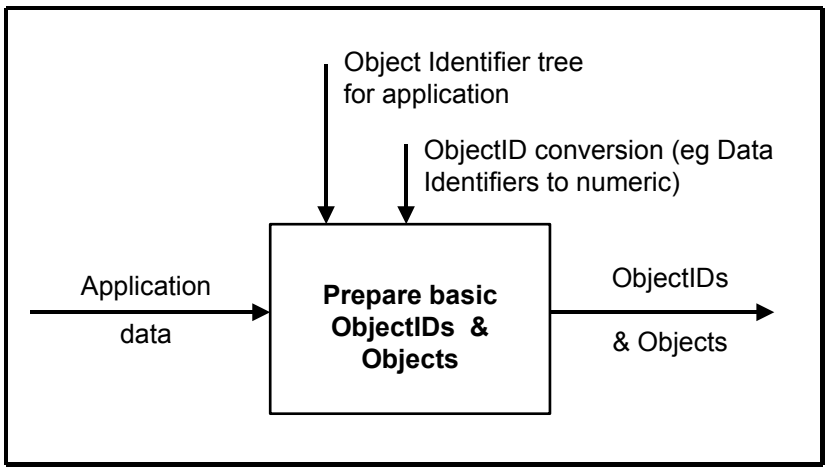


Figure 4 — Data Flow Model: Prepare Basic Objects

Any such data dictionary may evolve over a period of time and maintenance of such a data dictionary shall be independent of this International Standard. The data dictionary usually consists of a coded list (numeric, alphabetic, alphanumeric, etc) of data objects and their specification for use within the domain of the application standard. This International Standard shall not validate whether a particular coded object is valid to the externally maintained data dictionary. Only the most fundamental rules of these external data dictionaries will be presumed to be permanently defined, and these will be clearly declared below.

7.2.2 EAN.UCC system

The EAN.UCC system is used extensively in business communications, especially for retail supply chain management. For the purposes of this International Standard the relevant data syntax defined in the General EAN.UCC Specifications and are updated periodically (see Annex D for details of the maintenance body).

The common components of the object identifier for EAN.UCC system are:

1 0 15961 9

The final component of the object identifier for the EAN.UCC system shall be as defined in the General EAN.UCC Specifications.

7.2.3 Data Identifiers

The Data Identifiers (DIs) are used extensively in logistics communications, especially in primary manufacturing sectors (e.g. automotive, chemical, electronics, paper, etc).

The DIs are coded one to four characters in an alphabetic or alphanumeric structure as follows:

A

nA

nnA

nnnA

This International Standard shall presume that the DI format will remain as above and treat the DI as a 4 character code with three leading numeric values 000 to 999 plus one trailing alphabetic character A to Z.

The Data Identifiers are pre-defined in the ANSI MH 10.8.2 standard, and are updated periodically (see Annex D for details of the maintenance body).

The common components of the object identifier for Data Identifiers are:

1 0 15961 10

The final component of the object identifier for DIs is derived by converting the alphanumeric DI to a binary value and then converting this to a numeric value (see Annex E)

On input to the Data Protocol Processor, Data Identifiers shall have an object identifier which consists of the common components {1 0 15961 10} followed by a single arc which represents the specific alphanumeric DI converted to an all numeric value using the rules defined in Annex E.

7.2.4 IATA Data Format

IATA uses a data element structure for various purposes of item management, including baggage handling, for which application standards are in development for using RF tags. For the purposes of this International Standard, the relevant data syntax is in documents published by IATA (see Annex D for contact details).

The common components of the object identifier for IATA data elements are:

1 0 15961 12

The final component of the object identifier shall be as defined by IATA.

7.2.5 UPU Data Format

UPU uses a data element structure for various purposes of item management, for which application standards are in development for using RF tags. For the purposes of this International Standard the relevant data syntax is in documents published by UPU (see Annex D for contact details).

The common components of the object identifier for UPU data elements are:

1 0 15961 11

The final component of the object identifier shall be as defined by UPU.

7.2.6 Combining 15961 dataFormats

At the date of publication, four **dataFormats** have been defined with the sub-tree {1 0 15961}. These are:

9 EAN.UCC system

10 Data Identifiers

11 UPU data elements

12 IATA data elements

There may be occasions where data compliant with two, or more, of the application standards need to be encoded on an RF tag. In this case, the common components of the object identifier for such combined data objects are:

1 0 15961

On input to the Data Protocol Processor, combined 15961 data objects shall have an object identifier which consists of:

- the common components { 1 0 15961 }
- followed by a single arc (9, 10, 11 or 12) identifying the particular data dictionary

- followed by the appropriate arc(s) for the particular data (refer to 7.2.2 to 7.2.5 for details).

The data objects shall be presumed to be valid and of the correct format.

7.2.7 Other formats defined in ISO/IEC 15434

In principle, the formats defined in ISO/IEC 15434 may have their data elements defined as object identifiers. For the purposes of this International Standard the relevant data syntax for any particular format of ISO/IEC 15434 is in documents referred to in that International Standard.

The common components of the object identifier for a particular format of ISO/IEC 15434 data elements are:

1 0 15434

The final component(s) of the object identifier shall be as defined in ISO/IEC 15434. All formats can be treated as a complete object with the final arc having the value of the format number; except that Formats 01 to 09 shall be represented in the object identifier as 1 to 9 respectively. Any additional lower arcs, if appropriate, shall be as defined in ISO/IEC 15434.

NOTE: The granularity of the data object shall be as defined in ISO/IEC 15434, for example a data element, data segment, entire message. The object identifier structure does not take account of the practicalities of encoding long data objects; e.g. a complete EDI message becomes a single object.

7.2.8 ISO/IEC 15459 unique identifiers and associated attributes

ISO/IEC 15459 provides a mechanism to create various classes of unique identifier for traceability purposes. The common components of the object identifier for ISO/IEC 15459 unique identifiers are:

1 0 15459

The final component of the object identifier shall be the class of the unique identifier as defined in ISO/IEC 15459.

7.2.9 Relating Object Identifiers

Message based syntax can use recursive or looping techniques to created repeated sequences of related data (e.g. individual quantity and batch numbers linked to different product codes). When the complete message is parsed, the syntax identifies boundary points so that the attributes are correctly linked to the primary code.

With an object-based system (such as the Data Protocol of this International Standard and ISO/IEC 15962) operating at a base level, there is a risk of creating false links (i.e. product code A could be linked to quantity of product B). The problem can be overcome using one of the techniques described in Annex F.

7.3 Application system services

The application system shall provide the following:

- **ObjectId** - (see 7.3.1)
- **object** - (see 7.3.2)
- **compactParameter** - (see 7.3.3)
- **objectLock** - (see 7.3.4)

These are incorporated into the definitions of particular application commands and responses, and never transferred without the supporting commands.

7.3.1 ObjectId (TypeReference and elementName)

ObjectId shall be provided by the application system as a series of arcs, which comply with ISO/IEC 9834-1, including all the object identifiers associated with the established data formats (see 7.1.2.5). The **ObjectId** shall be presented as a (full) OBJECT IDENTIFIER in the application commands. Whenever possible, the Data Protocol Processor converts this to a RELATIVE-OID for more efficient encoding on the RF tag, based on the system information received from the Tag Driver.

7.3.2 object (elementName)

The application shall provide the value (i.e. data) of the elementName **object.** This value is octet based, the interpretation of which is fully comprehensible from the definition of the **ObjectId** provided by the application data dictionary. Specific advice about particular interpretation is given in the following sub-clauses.

7.3.2.1 Basic 8-bit character set information

Most of the business applications defined in 7.2 make use of ISO/IEC 8859-1 or its subset ISO 646. As such, the interpretation (or presentation) of the data is known to all participants in the system. This includes the specific sender and recipient even, if as is common in an open AIDC system, they are not known directly to each other. If text file and browser application software are set up to handle the ISO/IEC 8859-1 character set, most of the data content can be displayed as intended. If the content has a specific interpretation, the RFID application needs to establish between the participants exactly how particular data is to be interpreted. This can be achieved by publishing a data dictionary using the **ObjectIds** as a reference.

7.3.2.2 Support for ISO/IEC 10646

ISO/IEC 10646 supports the character glyphs of all character sets. One option to support data in any other specific character set is to pre-process this by converting to ISO/IEC 10646 using mapping tools, and then compacting this to UTF-8 encoding (as defined in ISO/IEC 10646). Any data directly defined by ISO/IEC 10646 should also be UTF-8 encoded, to reduce the number of octets required to store the **object** in the Logical Memory Map.

7.3.2.3 Support for secure or encrypted data

All forms of secure data, including encrypted data, shall be created by the application prior to being passed as an **object** to the Data Protocol Processor. This is done for two reasons:

- It requires security of data to reside with the application and to be changed independently of the Data Protocol Processor.
- It allows the Data Protocol Processor to handle all data, whether encrypted or not, in a similar manner.

The fact that data is encrypted may need to be made known to the entire set of systems users, but the actual method of encryption can be restricted to the sender and intended recipient.

Further details are provided in 7.4.

7.3.3 compactParameter (elementName)

The elementName **compactParameter** in a command shall determine whether the application data shall be compacted by the Data Protocol Processor. **compactParameter** shall be an integer value and the following codes apply.

0 **applicationDefined** - The **object** shall not be processed through the data compaction rules of ISO/IEC 15962 and remains unaltered when stored in the Logical Memory Map of the RF tag.

1 **compact** - This requires using the ISO/IEC 15962 compaction rules to compact the **object** as efficiently as possible to reduce the number of octets required on the Logical Memory Map.

2 **utf8Data** - This identifies that the **object** has been externally transformed from the ISO/IEC 10646 coded character set to UTF-8 encoding. The **object** shall not be processed through the data compaction rules of ISO/IEC 15962 and remains unaltered for transfer to the Logical Memory Map.

3 to 14 reserved for future definition

15 **de-compactedData** – This identifies that the object in a response has been de-compacted using rules in ISO/IEC15962 and restored to its original application input format

Generally, compaction should be applied because it increases the encodation efficiency on the RF tag. **Objects** already encoded to UTF-8 encoding rules should be qualified with the **compactParameter** value (2) so that subsequent reading of the **object** will clearly indicate that it shall be processed through a UTF-8 decoder for final presentation to the application receiving the data. The **compactParameter** value (0) should only be used if there is an over-riding reason not to compact the data. Reasons for this include prior compaction to application rules, or if the **object** has been encrypted. In both these cases, although compaction could be possible and the decode process would fully restore the **object**, the originally defined parameter value 0 or 2 would be lost and the receiving application could fail to undertake subsequent processing.

When **compactParameter** is returned as a response, the value (1) to indicate "compact" cannot be returned as this is an input argument. All de-compacted **objects** shall be defined as de-compactedData (15) on output from a read process.

7.3.4 objectLock (elementName)

The elementName **objectLock** is a command argument that requires the Data Protocol Processor to arrange the **object** and **objectId** and precursor in such a way that all of the associated octets can be stored and locked in a block aligned manner in the Logical Memory Map. The locking process shall have the effect of making the octets so processed permanently encoded on the RF tag. No commands are expected to be able to unlock the octets.

7.4 Data security

The data **(object)** may be made secure by the use of some form of encryption. This shall be applied prior to the **object** being transferred to the Data Protocol Processor. The decryption process shall also be applied to the **object** after it has been transferred to the application. As such, all the processes are transparent to this International Standard and ISO/IEC 15962.

The following features can be provided:

- Data security - This allows only authorised users the ability to "see" the true data. The sender will need to provide authorised users with the decryption algorithm and key. Other users can see the raw octet string, but this will be meaningless.

- Data integrity - This uses the decrypted data to identify whether the data has been modified by others prior to being read by an authorised user.

- Data validity - This uses encryption processes to reduce the risk of data from a legitimate source being copied by an unauthorised agent into another RF tag, and being passed off as the original source RF tag.

Additional advice is provided in Annex G.

8 Application commands and responses

Application commands are used to instruct the Data Protocol Processor and the interrogator to execute specific functions. They are also applied to the processing of the application data to achieve efficient encoding. The responses from the Data Protocol Processor include requested data and also information about actions undertaken and errors found. Each command/ response pair has its abstract syntax presented as modules. Each module is defined in a manner that enables the command to be invoked independently of any other command.

So that the commands and responses can easily be incorporated into the transfer syntax, code values have been assigned in this International Standard to the final arc of the command and response modules (see 8.1). Also, **completionCodes** (see 8.2) and **executionCodes** (see 8.3) are assigned to the responses. The source of definitions of elementNames and TypeReferences that apply to each command are given, including those that only apply to the commands and modules (see 8.4).

In the processing of a command, an error might be detected. The command shall be aborted and no data is transferred to or from the RF tag in which the error is detected. This is possible because processing is done in the Logical Memory. Although other error conditions might be present, the first problem identified is the only one reported. The appropriate completionCode or executionCode is returned. In the case where the command addresses multiple RF tags, all RF tags processed prior to the one with the detected error should be processed. The detection of the error aborts all subsequent processing.

The following sub-clauses define all the application commands and responses. In addition to the basic syntax, these sub-clauses will also describe the function and purpose of specific command arguments and responses. The commands and responses are grouped logically together. Within this clause, the appropriate abstract syntax is shown. Annex H shows an example of the complete transfer encoding of a command and response.

8.1 Final arc values of the command and response modules

Each command and response module shall be identified by an OBJECT IDENTIFIER. The common root for commands is **ISO (1) standard (0) rfid-data-protocol (15961) commandModules (126)**. The common root for responses is **ISO (1) standard (0) rfid-data-protocol (15961) responseModules (127)**. The final arc of each pair of command and response modules shall have the same value, effectively a RELATIVE-OID. The final arc shall be specific to the command/response pair and the following final arcs are specified:

1	**configureAfi**
2	**configureStorageFormat**
3	**inventoryTags**
4	**addSingleObject**
5	**deleteObject**
6	**modifyObject**
7	**readSingleObject**
8	**readObjectIds**
9	**readAllObjects**
10	**readLogicalMemoryMap**
11	**inventoryAndReadObjects**
12	**eraseMemory**
13	**getApp-basedSystemInfo**
14	**addMultipleObjects**
15	**readMultipleObjects**
16	**readFirstObject**

Additional command and response modules, and their final arc values, will be added in numeric sequence, as required, to this International Standard.

The complete module specifies the function that the interrogator shall perform. Each command specifies, as appropriate, particular processes to be undertaken by the Data Protocol Processor, the Tag Driver, and the interrogator in communications across the air interface. Each response specifies, as appropriate, particular processes to be undertaken by the Tag Driver, the Data Protocol Processor, and the communications across the application interface.

8.2 completionCode (elementName)

The elementName **completionCode** is part of the response to each command. The **completionCode** is an INTEGER value that reports specifically on how the particular command was processed and executed, successfully or not. It is returned in each response. If its value is 0 (00_{16}), the command has been successfully executed. If its value is 255 (FF_{16}), the command could not be executed by the system for the reason specified in the **executionCode** (see 8.3). If its value is different from 0 and 255, it indicates that the command was not be executed as instructed by the application for the reason mentioned.

NOTE: The **completionCode** provides information on the basis that the command can be invoked for the particular RF tag in the communication chain, whereas the **executionCode** indicates a systems error or success.

The following **completionCodes** apply:

0 **noError:** The command was successfully executed

1 **afiNotConfigured:** The command could not be completed, possibly because of a prior configure action

2 **afiNotConfiguredLocked:** The applicationFamilyId was found to be locked (by a previous command execution), so applicationFamilyId could not be configured, as requested by the command, on this occasion

3 **afiConfiguredLockFailed:** The applicationFamilyId was correctly configured, but the lock function could not be completed

4 **storageFormatNotConfigured:** The command could not be completed, possibly because of a prior configure action

5 **storageFormatNotConfiguredLocked:** The storageFormat was found to be locked (by a previous command execution), so storageFormat could not be configured, as requested by the command, on this occasion

6 **storageFormatConfiguredLockFailed:** The storageFormat was correctly configured, but the lock function could not be completed

7 **objectLockedCouldNotModify:** The existing encodation on the RF tag is locked, and as a result this attempt to update the **object** value could not be completed

8 **tagIdNotFound:** The particular RF tag, specified by the tagId, could not be found in the operating area

9 **objectNotAdded:** The data set of Precursor, object and objectId could not be added to the Logical Memory Map (e.g. because there was insufficient memory space). This response also applies if the RF tag supports a lock function that failed

10 **duplicateObject:** An objectId with the same value as the one to be processed (added, modified, or deleted) was found. The process was aborted.

11 **objectAddedButNotLocked:** The data set was added to a RF tag that did not support a lock feature in the memory.

12 **objectNotDeleted:** The data set of Precursor, objectId and object could not be deleted from the Logical Memory Map (e.g. because the delete function is not supported by the particular RF tag).

13 **objectIdNotFound:** The intended process could not be executed because the defined objectId, is not actually encoded on the RF tag.

14 **objectLockedCouldNotDelete:** The object is already locked and so cannot be deleted

15 **objectNotRead:** The intention to read the specified object failed

16 **objectsNotRead:** The intention to read one or more of the specified objects failed

17 **blocksLocked:** The intended action to erase encoded bytes from one or more blocks could not be actioned because blocks were previously locked.

18 **eraseIncomplete:** The intended action to erase encoded bytes from one or more blocks was interrupted and that not all blocks have been processed. The command can be re-invoked to complete the process.

19 **readIncomplete:** The intention to read the complete contents of the Logical Memory Map failed, because not all blocks from the RF tag were transferred to the Logical Memory.

20 **systemInfoNotRead:** The intention to read the system information failed.

21 **objectNotModified:** The data set of Precursor, objectId and object could not be modified on the Logical Memory Map (e.g. because the modify function is not supported by the particular RF tag).

22 **objectModifiedButNotLocked:** The object was modified in a RF tag that did not support a lock feature in the memory.

23 **failedToReadMinimumNumberOfTags:** The minimum number of RF tags were not identified with the specified selection criterion, possibly because of a time-out.

254 **undefinedCommandError:** An error occurred in a command,not defined by another code.

255 **executionError:** A system error occurred which made it impossible to action the command. The appropriate executionCode is returned in the response.

8.3 executionCode (elementName)

The elementName **executionCode** is part of the response to each command. The **executionCode** is an INTEGER value that reports on the way the command was processed and executed by the system, successfully or not. It is returned in each response. If its value is 0 (00_{16}), the command has been successfully processed, i.e. the protocol was executed. Other values indicate that a system error has occurred.

The following **executionCodes** apply:

0 **noError**: The command was executed without error

1 **noResponseFromTag**: No response was received from the RF tag

2 **tagCommunicationError**: The response(s) from the RF tag(s) was corrupted (e.g. There was an aborted frame)

3 **tagCRCError**: A CRC error was detected in the RF tag's response

4 **commandNotSupported**: The command code is not supported by the interrogator, or RF tag

5 **invalidParameter**: The command parameter(s) are invalid

6 **interrogatorCommunicationError**: An error occurred in the communication between the application and in the interrogator

7 **internalError**: An error occurred in the application software

255 **undefinedError**: An error occurred, not defined by another code

8.4 Command-related elementNames

8.4.1 addObjectsList (elementName)

The elementName **addObjectsList** is a command argument that represents the list of **objectIds, objects,** and other parameters that are to be written to the RF tag Logical Memory Map in a multiple write object function.

8.4.2 afiLock (elementName)

The elementName **afiLock** is a command argument used to determine whether the **applicationFamilyId** is to be locked or not. This is a Boolean argument. If set to TRUE, the interrogator shall lock the **applicationFamilyId** to ensure that the particular RF tag can only be used in the way prescribed. The locking process shall have the effect of making the octets so processed permanently encoded on the RF tag. No commands are expected to be able to unlock the **applicationFamilyId**.

8.4.3 avoidDuplicate (elementName)

The elementName **avoidDuplicate** is a command argument used to ensure that the **objectId** is not already encoded in the Logical Memory Map. This is a Boolean argument. If set to TRUE, the interrogator shall verify all **objectIds** in the Logical Memory Map. If the **objectId** already exists, the write command is aborted and the **completionCode** value duplicateObject (10) is returned. If set to FALSE, the interrogator shall write the **object** without any verification.

8.4.4 checkDuplicate (elementName)

The elementName **checkDuplicate** is a command argument used to invoke a particular process defined by the command (e.g. read, or delete). This is a Boolean argument.

If set to TRUE, the interrogator shall verify all **objectIds** in the Logical Memory Map. The command shall only be completed if there is no duplicate present. Otherwise, the **completionCode** value duplicateObject (10) is returned.

If the **checkDuplicate** flag is set to FALSE, the interrogator shall act upon the first occurrence of the **objectId**, associated **object** and precursor. In this case, a duplicate **objectId** and associated **object** (possibly with a different value) and precursor could be present.

8.4.5 identifyMethod (elementName)

The elementName **identifyMethod** is a command argument used to define whether all, or some, of the RF tags belonging to the particular selected **applicationFamilyId** in the operating area shall be identified. This elementName is an integer value and the following codes apply:

0	**inventoryAllTags**
1	**inventoryAtLeast**
2	**inventoryNoMoreThan**
3	**inventoryExactly**
4 to 15	**reserved for future definition**

For every argument including **inventoryAllTags**, it is necessary to specify the number of RF tags to be read using the elementName **numberOfTags** (see 8.4.9). More precise advice is provided in InventoryTagsModule (see 8.7).

8.4.6 identities (elementName)

The elementName **identities** is a command response that indicates that it represents a list of the identified **tagIds.**

8.4.7 lockStatus (elementName)

The elementName **lockStatus** is a response argument that identifies whether the **objectId** and associated **object** and precursor are locked. This is a Boolean argument. If set to TRUE, it confirms that these are locked.

8.4.8 logicalMemoryMap (elementName)

The elementName **logicalMemoryMap** is a response that represents the encodation on the RF tag in an unstructured format, i.e. the byte values as stored, but not expanded into **objectId, object, compactParameter** and **lockStatus.** It may be used for diagnostic purposes, or for other purposes where the total transfers are required.

8.4.9 maxAppLength (elementName)

The elementName **maxAppLength** is a command argument that defines the maximum compacted length of an object.

8.4.10 numberOfTags (elementName)

The elementName **numberOfTags** is a command argument that defines a limit on the particular **identifyMethod** specified. It is an integer value in the range 0 to 65535.

8.4.11 numberOfTagsFound (elementName)

The elementName **numberOfTagsFound** is a response argument that returns the actual number of RF tags observed that met the criteria. If the **identifyMethod** argument is set to **inventoryNoMoreThan**, then the response value **numberOfTagsFound** can be a lower number.

8.4.12 objectIdList (elementName)

The elementName **objectIdList** is a command argument that defines the set of **objectIds** that are required to be read from the Logical Memory Map of the RF tag. It is also a response that indicates that it represents the list of **objectIds** on the particular RF tag.

8.4.13 objectIdsFound (elementName)

The elementName **objectIdsFound** is a response argument that defines the set of **objectIds** that are encoded on the Logical Memory Map of the RF tag.

8.4.14 objects (elementName)

The elementName **objects** is a command response that indicates that it represents a related list of each instance of all the following encodation on the RF tag: **objectId, object, compactParameter,** and **lockStatus.**

8.4.15 readObjectList (elementName)

The elementName **readObjectsList** is a command argument that represents the list of **objectIds** (and associated **checkDuplicate** argument) used to be read a set of **objectIds** from the RF tag Logical Memory Map in a multiple read object function.

8.4.16 storageFormatLock (elementName)

The elementName **storageFormatLock** is a command argument used to determine whether the **storageFormat** is to be locked or not. This is a Boolean argument. If set to TRUE, the interrogator shall lock the **storageFormat** to ensure that the particular RF tag can only be used in the way prescribed. The locking process shall have the effect of making the octets so processed permanently encoded on the RF tag. No commands are expected to be able to unlock the **storageFormat**.

8.4.17 tagAndObjects (elementName)

The elementName **tagAndObjects** is a command response that represents a list of **tagIds,** and for each instance of **tagId,** represents the list of selected **objectIds, objects, compactParameters** and **lockStatus** that were read from each RF tag.

8.4.18 tagReadResponse (elementName)

The elementName **tagReadResponse** is a command response that represents a list of **objectId,** (and for each instance of this the **object, compactParameter, lockStatus** and **completionCode**) that were read from an RF tag.

8.4.19 tagWriteResponse (elementName)

The elementName **tagWriteResponse** is a command response that represents a list of **objectIds** and **completionCode** (for each **objectId**) that were written to an RF tag.

8.5 ConfigureAfiModules

The ConfigureAfiModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to write the **applicationFamilyId** (including the sub-family) into the RF tag. A fundamental requirement of this command is that only one RF tag shall be programmed per command. This is to ensure that the configuration process is robust, particularly in environments where more than one type of RF tag can be present.

The abstract syntax for ConfigureAfiModules is given in Table 4

Table 4 — ConfigureAfiModules

```
-- Configure AFI
-- The ConfigureAfiCommand instructs the interrogator to write the AFI (Application
-- Family Identifier, including the sub-family) into the RF tag.  The interrogator shall
-- lock the AFI if the Lock flag is set to true.


ConfigureAfiCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) configureAfi(1)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ConfigureAfiCommand ::= SEQUENCE {
    tagId                 OCTET STRING(SIZE(0..255)),
                          -- See Clause 7.1.2.1 for detailed specification TagId shall
                          -- be provided by the Tag Driver for the purposes of
                          -- identifying the RF Tag unambiguously for at least the
                          -- period of a transaction.
    applicationFamilyId   ApplicationFamilyId,
    afiLock               BOOLEAN
                          -- If set to TRUE, the interrogator shall lock the AFI
    }


ApplicationFamilyId ::= SEQUENCE {
    applicationFamily     INTEGER {
                          all(0), -- address all families
                          -- values 1 - 8 reserved for definition by SC17
                          afiBlock9(9),
                          afiBlockA(10),
                          afiBlockB(11),
                          afiBlockC(12)
                          -- values 9 to 12 defined as per Annex B of this
                          -- International standard
                          -- values 13 to 15 reserved for definition by ISO/IEC
                          } (0..15),
    applicationSubFamily  INTEGER {
```

```
                        all(0),-- This value shall not be encoded in the RF tag, and
                        -- shall only be used in a command to signal that the
                        -- interrogator shall address all subfamilies within the
                        -- selected family.
                        -- NOTE: This has little utility for this Data Protocol, but
                        -- is retained for compatibility with SC17 smart card commands
                        asf1-annex (1), -- values 1 to 15, for applicationFamily 9
                        -- to 12,defined as per Annex B of this International standard
                        asf2-annex (2),
                        asf3-annex (3),
                        asf4-annex (4),
                        asf5-annex (5),
                        asf6-annex (6),
                        asf7-annex (7),
                        asf8-annex (8),
                        asf9-annex (9),
                        asfA-annex (10),
                        asfB-annex (11),
                        asfC-annex (12),
                        asfD-annex (13),
                        asfE-annex (14),
                        asfF-annex (15)
                        } (0..15)



-- ApplicationFamilyId is stored as a single OCTET within system information on the tag.
-- ApplicationFamilyId allows tags to be grouped according to specific families and
-- allows any such family of tags to be selectively addressed by the application. RF Tag
-- vendors may implement mechanisms in the Tag Driver and air interface specifically for
-- selective addressing of RF Tags by ApplicationFamilyId.

}
END


ConfigureAfiResponse
    {iso(1) standard(0) rfid-data-protocol(15961)responseModules(127) configureAfi(1)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ConfigureAfiResponse ::= SEQUENCE {
    completionCode     INTEGER {
                       noError(0),
                       afiNotConfigured(1),
                       afiNotConfiguredLocked(2),
                       afiConfiguredLockFailed(3),
                       tagIdNotFound(8),
                       executionError(255)
                       },
    executionCode      INTEGER
                       -- See Clause 8.3 and notes in this syntax for a full list of
                       -- executionCodes
}
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard, as detailed:

afiLock (see 8.4.1)
applicationFamily (see 7.1.2.2)
applicationFamilyId (see 7.1.2.2)
applicationSubFamily (see 7.1.2.2)
completionCode (see 8.2)
executionCode (see 8.3)
tagId (see 7.1.2.1)

8.6 ConfigureStorageFormatModules

The ConfigureStorageFormatModule consists of a commandModule, and the associated responseModule, that instruct the interrogator to write the **storageFormat** (**accessMethod** and **dataFormat**) into the RF tag. The command also instructs the interrogator to initialise the RF tag Logical Memory Map by erasing any data already stored there. A fundamental requirement of this command is that only one RF tag shall be programmed per command. This is to ensure that the configuration process is robust, particularly in environments where more than one type of RF tag can be present.

If the **accessMethod** (incorporated in **storageFormat**) is specified as directory, the interrogator shall create the initial directory structure.

The ASN.1 Abstract Syntax for ConfigureStorageFormatModules is given in Table 5.

Table 5 — ConfigureStorageFormatModules

```
-- Configure StorageFormat
-- The ConfigureStorageFormatCommand instructs the interrogator to write the
-- StorageFormat into the RF tag, and to initialise the tag logical memory
-- map. The interrogator shall erase all the application memory, and if the
-- directory format is specified by the StorageFormat, it shall create the
-- initial directory structure. The interrogator shall lock the
-- StorageFormat if the Lock flag is set to true

ConfigureStorageFormatCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) configureStorageFormat(2)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ConfigureStorageFormatCommand ::= SEQUENCE {
    tagId              OCTET STRING(SIZE(0..255)),
                       -- See Clause 7.1.2.1 for detailed specification TagId
                       -- shall be provided by the Tag Driver for the
                       -- purposes of identifying the RF Tag unambiguously
                       -- for at least the period of a transaction.
    storageFormat      StorageFormat,
    storageFormatLock  BOOLEAN
                       -- If set to TRUE, the interrogator shall lock the
                       -- StorageFormat
    }

StorageFormat ::= SEQUENCE {
    accessMethod   INTEGER {
                       noDirectory(0),
                       directory(1),
                       selfMappingTag(2)   -- Access to objects is via high
                       -- level commands to the RF Tag and the internal
```

```
                        -- structure of the memory inside the RF Tag is not
                        -- defined
                        } (0..3),
    dataFormat      INTEGER {
                        notFormatted(0),   -- Not formatted according
                            -- to this standard
                        fullFeatured(1),   -- Supports any type of
                            -- data based on full OID
                        rootOidEncoded(2),   -- Supports any type of
                            -- data with a common root-OID
                        iso15434(3),   -- root-OID is defined as
                            -- {1 0 15434}
                        iso6523(4),   -- Supports data belonging to one or
                            -- more International Code Designators compliant
                            -- with ISO/IEC 6523-1, root-OID is defined as
                            -- (1 0 6523)
                        iso15459(5),   -- Supports unique item identifiers
                            -- compliant with ISO/IEC 15459, root-OID is
                            -- defined as (1 0 15459)
                        iso15961Combined(8),   -- Supports combinations of
                            -- formats of ISO/IEC 15961, root-OID is defined
                            -- as {1 0 15961}
                        ean-ucc(9),   -- Supports data of the EAN-UCC system,
                            -- root-OID is defined as {1 0 15961 9}
                        di(10),   -- Supports Data Identifiers (as referred to
                            -- in ISO/IEC 15418), root-OID is implied to be
                            -- {1 0 15961 10}
                        upu(11), -- Supports UPU data elements, root-OID
                            -- is defined as { 1 0 15961 11}
                        iata(12)   -- Supports IATA data elements,
                            -- root-OID is defined as {1 0 15961 11}
                        } (0..31)
}
END

ConfigureStorageFormatResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) configureStorageFormat(2)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ConfigureStorageFormatResponse ::= SEQUENCE {
    completionCode INTEGER {
                        noError(0),
                        storageFormatNotConfigured(4),
                        storageFormatNotConfiguredLocked(5),
                        storageFormatConfiguredLockFailed(6),
                        tagIdNotFound(8),
                        erasureIncomplete(18),
                        executionError(255)
                        },
    executionCode  INTEGER
                        -- See Clause 8.3 and notes in this syntax
                        -- for a full list of executionCodes
}
END
```

The following elementNames used in these modules are defined elsewhere in this international standard, as detailed:

accessMethod (see 7.1.2.4)
completionCode (see 8.2)
dataFormat (see 7.1.2.5)
executionCode (see 8.3)
storageFormat (see 7.1.2.3)
storageFormatLock (see 8.4.13)
tagId (see 7.1.2.1)

8.7 InventoryTagsModules

The InventoryTagsModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to identify a particular set of RF tags present in its operating field.

The ASN.1 Abstract Syntax for InventoryTagsModules is given in Table 6.

The command requires that the value of the **applicationFamilyId** is specified to select RF tags belonging to a particular class, typically containing data belonging to a defined domain and / or containing a defined **objectId**. The selection criteria are defined in Annex B.

The second, additional, selection criterion (**identifyMethod**) determines how many RF tags, complying with the specified **applicationFamilyId** selection criterion, need to be identified before the response can be provided. A mechanism that can be used to detect any RF tag entering the operating area, is to set the inventoryAtLeast argument to 1. Particular conditions can be confirmed by only undertaking a partial inventory, i.e. by using either the inventoryAtLeast, or the inventoryNoMoreThan arguments. A reconciliation of a known quantity of previous transactions (e.g. to identify that all items intended to be in a container are actually there) can be achieved by using the inventoryExactly argument. Details of the options are given in the module in Table 6.

The response consists of the **numberOfTagsFound** and the identities of each RF tag by its **tagId**.

Table 6 — InventoryTagsModules

```
-- InventoryTags
-- The InventoryTagsCommand instructs the interrogator to inventory and to
-- identify all tags present in its operating area. Each tag is uniquely
-- identified by its TagId.


InventoryTagsCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) inventoryTags(3)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

InventoryTagsCommand ::= SEQUENCE {
    applicationFamilyId   ApplicationFamilyId,
    identifyMethod        INTEGER {
                              inventoryAllTags (0),
                              inventoryAtLeast (1),
                              inventoryNoMoreThan (2),
                              inventoryExactly (3)
                              }(0..15),
    numberOfTags          INTEGER (0..65535)
    }

-- The ApplicationFamilyId separates fundamentally different types of application
-- data (see 7.1.2.2 and Annex B), and possibly particular objectIds. Specifying a
```

```
-- hex value xx (where x is a non-zero value) selects only the RF tags that have
-- the required data content.
-- Specifying a hex value 00 selects all the RF tags; this may be an appropriate
-- action to undertake a full inventory.
-- Specifying a hex value 0x, or x0, (where x is a non-zero value) might not be
-- logically sound because the RF tags are from different applications and the x
-- value has different meaning.

-- If the identifyMethod is set to inventoryAllTags, the interrogator shall perform
-- a complete inventory of all tags present in its field of operation. The value of
-- numberOfTags is irrelevant and should be set to zero by the application.

-- If the identifyMethod is set to inventoryAtLeast, the interrogator shall perform
-- an inventory of the tags present in its field of operation and (possibly)
-- continue waiting until it has identified a number of tags equal to numberOfTags.
-- If the numberOfTags is set to 1, the Interrogator will wait until the first tag
-- has been detected.  This is a mechanism to wait for a tag to enter the
-- interrogator field.
-- If the numberOfTags is set to more than 1, the Interrogator will wait
-- until the specified number of tags has been detected.

-- If the identifyMethod is set to inventoryNoMoreThan, the interrogator shall
-- initiate an inventory of the tags present in its field of operation and shall
-- return a response with a number of tags lower or equal to numberOfTags.
-- The interrogator may interrupt the inventory process when the numberOfTags has
-- been reached or may continue the inventory process till all tags have been read.
--      Note: This may be constrained by the air interface and anticollision
--      mechanism.

-- If the identifyMethod is set to inventoryExactly, the interrogator shall
-- initiate an inventory of the tags present in its field of operation and shall
-- return a response with the number of tags equal to numberOfTags.  This command
-- parameter could be used to confirm the actual number of tagged items in a
-- container.  The Interrogator will wait until the specified number of tags has
-- been detected.  The interrogator may interrupt the inventory process when the
-- numberOfTags has been reached or may continue the inventory process till
-- all tags have been read.
--      Note: This may be constrained by the air interface and anticollision
--      mechanism.

-- Execution of this command with the arguments inventoryAtLeast and
-- inventoryExactly can cause the interrogator to wait until sufficient RF tags
-- enter its field of operation; also the command response cannot be initiated
-- until after this delay.  It is the responsibility of the application to
-- accommodate this potentiality.

ApplicationFamilyId ::= SEQUENCE {
    applicationFamily     INTEGER(0..15),
    applicationSubFamily  INTEGER(0..15)
    }
-- The code values are defined in the ConfigureAfiCommand.

TagId ::=  OCTET STRING(SIZE(0..255))
           -- See Clause 7.1.2.1 for detailed specification

END


InventoryTagsResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) inventoryTags(3)}
```

```
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

InventoryTagsResponse ::= SEQUENCE {
    completionCode     INTEGER {
                           noError(0),
                           failedToReadMinimumNumberOfTags(23),
                           -- for example, this could be due to a time-out
                           failedToReadExactNumberOfTags(24),
                           -- for example, this could be due to a time-out
                           executionError(255)
                           },
    executionCode      INTEGER,
                       -- See Clause 8.3 and notes in this syntax
                       -- for a full list of executionCodes
    numberOfTagsFound  INTEGER (1..65535),
    identities         SEQUENCE OF TagId
    }


TagId ::=  OCTET STRING(SIZE(0..255))
           -- See Clause 7.1.2.1 for detailed specification

END
```

The following elementNames used in these modules are defined elsewhere in this International Standard, as defined:

applicationFamily (see 7.1.2.2)
applicationFamilyId (see 7.1.2.2)
applicationSubFamily (see 7.1.2.2)
completionCode (see 8.2)
executionCode (see 8.3)
identifyMethod (see 8.4.5)
identities (see 8.4.6)
numberOfTags (see 8.4.9)
numberOfTagsFound (see 8.4.10)
tagId (see 7.1.2.1)

8.8 AddSingleObjectModules

The AddSingleObjectModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to write an **object**, its **objectId**, and associated parameters into the RF tag Logical Memory Map. The command arguments can be used to lock the **objectId**, the **object** and associated parameters; and to check that the **objectId** is not already encoded on the RF tag. Only one RF tag shall be programmed per command to ensure that the writing process is robust.

The ASN.1 Abstract Syntax for AddSingleObjectModules is given in Table 7.

Table 7 — AddSingleObjectModules

```
-- Add Single Object
-- The AddSingleObjectCommand instructs the interrogator to write an object, its
-- OID and associated parameters into the tag logical memory map.
--     NOTE: There is also an AddMultipleObjectsCommand.

-- If the checkDuplicate flag is set to TRUE, the interrogator shall verify, before
-- adding the object, that no object with the same OID already exists. If such
-- object exists, the interrogator shall not perform the Add Object function and
-- shall return the appropriate Completion Code.

-- If the Lock flag is set to TRUE, the interrogator shall lock the ObjectId, the
-- Object, its compaction scheme and associated parameters into the tag Logical
-- Memory Map


AddSingleObjectCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) addSingleObject(4)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

AddSingleObjectCommand ::= SEQUENCE {
    tagId              OCTET STRING(SIZE(0..255)),
                       -- See Clause 7.1.2.1 for detailed specification
    objectId           OBJECT IDENTIFIER,-- Full OID value
    avoidDuplicate     BOOLEAN,
                       -- If set to TRUE, check for duplicate objectId
    object             OCTET STRING,
    compactParameter   INTEGER {
                           applicationDefined(0),
                           -- The object shall not be processed through the
                           -- data compaction rules of 15962 and remains unaltered
                           compact(1),
                           -- Compact object as efficiently as possible
                           -- using 15962 compaction rules
                           utf8Data(2)
                           -- Data has been externally transformed from a 16-bit
                           -- coded character set to a UTF-8 string.  The object
                           -- shall not be processed through the data compaction
                           -- rules of 15962 and remains unaltered
                           }(0..15),
    objectLock         BOOLEAN
                       -- If TRUE the interrogator shall lock the ObjectId, the
                       -- Object, its compaction scheme and other features in the
                       -- Logical Memory Map
    }

END


AddSingleObjectResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) addSingleObject(4)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

AddSingleObjectResponse ::= SEQUENCE {
    completionCode     INTEGER {
```

```
                        noError(0),
                        tagIdNotFound(8),
                        objectNotAdded(9),
                        duplicateObject(10),
                        objectAddedButNotLocked(11),
                        executionError(255)
                        },
    executionCode      INTEGER
                       -- See Clause 8.3 and notes in this syntax for a full list of
                       -- executionCodes
    }
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

avoidDuplicate (see 8.4.3)
compactParameter (see 7.3.3)
completionCode (see 8.2)
executionCode (see 8.3)
object (see 7.3.2)
objectId (see 7.3.1)
objectLock (see 7.3.4)
tagId (see 7.1.2.1)

8.9 DeleteObjectModules

The DeleteObjectModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to delete a defined **objectId** and its **object** and associated parameters. Only one RF tag and only one **objectId** shall be programmed per command to ensure that the deletion process is robust. The delete function requires the removal of the **objectId**, the associated **object**, and precursor from the Logical Memory Map.

The ASN.1 Abstract Syntax for DeleteObjectModules is given in Table 8.

Table 8 — DeleteObjectModules

```
-- Delete Object
-- The DeleteObjectCommand instructs the interrogator to delete the object
-- specified by its OID, from the tag Logical Memory Map.  This means that a
-- subsequent command to read the object will return objectNotFound.  This
-- procedure might not succeed if the object is locked, if this is found to be the
-- case, the response will return the appropriate completionCode.  If the
-- checkDuplicate flag is set to TRUE, the interrogator shall verify, before
-- deleting the requested object, that there is only a single object with the
-- requested OID. If the interrogator detects that several objects have the same
-- OID it shall not perform the DeleteObject function and shall return the
-- appropriate completionCode

-- If the checkDuplicate flag is set to FALSE, the interrogator shall delete the
-- first occurrence of the object specified by its OID.
--      NOTE: This is an argument that effectively provides no protection against
--      duplicate OIDs.  It should only be used when there is a high expectation of
--      no duplicates.

DeleteObjectCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) deleteObject(5)}
DEFINITIONS
```

```
EXPLICIT TAGS ::=
BEGIN

DeleteObjectCommand ::= SEQUENCE {
    TagId             OCTET STRING(SIZE(0..255)),
                      -- See Clause 7.1.2.1 for detailed specification
    objectId          OBJECT IDENTIFIER, -- Full OID value
                      -- This initiates the deletion of the ObjectId and the
                      -- associated Object
    checkDuplicate    BOOLEAN
                      -- If set to TRUE, the interrogator shall check that there is
                      -- only one occurrence of the ObjectId
    }

END


DeleteObjectResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) deleteObject(5)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

DeleteObjectResponse ::= SEQUENCE {
    completionCode INTEGER {
                      noError(0),
                      tagIdNotFound(8),
                      duplicateObject(10),
                      objectNotDeleted(12),
                      objectIdNotFound(13),
                      objectLockedCouldNotDelete(14),
                      executionError(255)
                      },
    executionCode  INTEGER
                   -- See Clause 8.3 and notes in this syntax for a full list of
                   -- executionCodes
    }
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

checkDuplicate (see 8.4.4)
completionCode (see 8.2)
executionCode (see 8.3)
objectId (see 7.3.1)
tagId (see 7.1.2.1)

8.10 ModifyObjectModules

The ModifyObjectModule consist of a commandModule, and the associated responseModule, that instruct the interrogator to carry out three related processes:

1. Read the complete Logical Memory Map from the RF tag.
2. Delete the specified **objectId, object** and associated precursor. If duplicated instances are found, the process is aborted.
3. Write the **objectId**, modified **object** and re-structured precursor.

If the byte string representing the modified **object** (data) is longer than the previous encodation, then the complete **object** needs to be located in a different area of the Logical Memory Map. A similar situation might arise if the **objectLock** argument is set to TRUE in the command. In both these cases, the Data Protocol Processor of ISO/IEC 15962 will control the relocation process. Only one RF tag and only one **objectId** shall be programmed per command to ensure that the modify process is robust.

The ASN.1 Abstract Syntax for ModifyObjectModules is given in Table 9.

Table 9 — ModifyObjectModules

```
-- Modify Object
-- The ModifyObjectCommand instructs the interrogator to modify an object, its OID
-- and associated parameters already on the tag Logical Memory Map. It does this by
-- deleting the specified object and associated parameters and writing the new
-- values.  To achieve this, the complete Logical Memory Map shall be read from the
-- tag.  Any instances of duplicate ObjectIds shall result in the process being
-- aborted.

-- If the object is already locked, it cannot be modified and the appropriate
-- completion code shall be returned.

-- If the byte string, that represents the modified data when prepared for encoding
-- in the Logical Memory Map, is the same length as its previous encodation, the
-- modified value is generally written to the same positions.

-- If the byte string is shorter than the previous encodation, then an offset shall
-- be encoded.

-- If the byte string is longer than the previous encodation, then it needs to be
-- located in a different area of the Logical Memory Map with this process
-- controlled by the Data Protocol Processor of ISO/IEC 15962.


ModifyObjectCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) modifyObject(6)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ModifyObjectCommand   ::= SEQUENCE {
    TagId             OCTET STRING(SIZE(0..255)),
                      -- See Clause 7.1.2.1 for detailed specification
    objectId          OBJECT IDENTIFIER,-- Full OID value
    object            OCTET STRING,
    compactParameter  INTEGER {
                      applicationDefined(0),
                      -- The object shall not be processed through the data
                      -- compaction rules of 15962 and remains unaltered
                      compact(1),
                      -- Compact object as efficiently as possible using 15962
                      -- compaction rules
                      utf8Data(2)
                      -- Data has been externally transformed from a 16-bit coded
                      -- character set to a UTF-8 string.  The object shall not be
                      -- processed through the data compaction rules of 15962 and
                      -- remains unaltered
                      }(0..15),
    objectLock        BOOLEAN
                      -- If TRUE the interrogator shall lock the ObjectId, the
```

```
                        -- Object, its compaction scheme and other features in the
                        -- Logical Memory Map
    }
END


ModifyObjectResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) modifyObject(6)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ModifyObjectResponse ::= SEQUENCE {
    completionCode     INTEGER {
                           noError(0),
                           objectLockedCouldNotModify(7),
                           tagIdNotFound(8),
                           duplicateObject(10),
                           objectNotModified(21),
                           objectModifiedButNotLocked(22),
                           executionError(255)
                           },
    executionCode      INTEGER
                       -- See Clause 8.3 and notes in this syntax for a full list of
                       -- executionCodes
    }
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

compactParameter (see 7.3.3)
completionCode (see 8.2)
executionCode (see 8.3)
object (see 7.3.2)
objectId (see 7.3.1)
objectLock (see 7.3.4)
tagId (see 7.1.2.1)

8.11 ReadSingleObjectModules

The ReadSingleObjectModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to read an **objectId**, its **object**, and associated parameters from the RF tag Logical Memory Map. Command arguments can be used to check that the **objectId** is not duplicated on the RF tag. Only one RF tag shall be programmed per command to ensure that the reading process is robust.

The ASN.1 Abstract Syntax for ReadSingleObjectModules is given in Table 10.

Table 10 — ReadSingleObjectModules

```
-- Read Single Object
-- The ReadSingleObjectCommand instructs the interrogator to read the Object
-- specified by its Object Identifier from the tag specified by its TagId.
--     NOTE: There is a ReadMultipleObjectsCommand.

-- If the checkDuplicate flag is set to FALSE, the interrogator shall return the
-- first Object found having the requested OID without checking for duplicates.

-- If the checkDuplicate flag is set to TRUE, the interrogator shall check for
-- duplicate objects having the requested OID. If more than one object with the
-- requested OID is found, the interrogator shall return the first found Object
-- having the requested OID and indicate the presence of duplicates with
-- appropriate Completion code.


ReadSingleObjectCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) readSingleObject(7)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ReadSingleObjectCommand ::= SEQUENCE {
    tagId              OCTET STRING(SIZE(0..255)),
                       -- See Clause 7.1.2.1 for detailed specification
    objectId           OBJECT IDENTIFIER,-- Full OID value
    checkDuplicate     BOOLEAN
                       -- If set to TRUE, the interrogator shall check that there is
                       -- only one occurrence of the ObjectId
    }

END


ReadSingleObjectResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) readSingleObject(7)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ReadSingleObjectResponse ::=  SEQUENCE {
    completionCode     INTEGER {
                           noError(0),
                           tagIdNotFound(8),
                           duplicateObject(10),
                           objectIdNotFound(13),
                           objectNotRead(15),
                           executionError(255)
                           },
    executionCode      INTEGER ,
                       -- See Clause 8.3 and notes in this syntax for a full list of
                       -- executionCodes
    object             OCTET STRING,
    compactParameter   INTEGER {
                           applicationDefined(0),
                           -- The object was not originally encoded through the data
                           -- compaction rules of 15962, and is as sent from the
                           -- source application and might require additional
                           -- processing by the receiving application.
```

```
                        utf8Data(2),
                        -- Data has been externally transformed from a 16-bit
                        -- coded character set to a UTF-8 string. The object
                        -- needs to be processed through an external UTF-8
                        -- decoder.
                        de-compactedData(15)
                        -- The object was originally encoded through the data
                        -- compaction rules of 15962 and de-compacted on this
                        -- read operation and restored to its original format.
                        }(0..15),
    lockStatus        BOOLEAN
                      -- If TRUE, object is locked
    }
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

checkDuplicate (see 8.4.4)
compactParameter (see 7.3.3)
completionCode (see 8.2)
executionCode (see 8.3)
lockStatus (see 7.3.4)
object (see 7.3.2)
objectId (see 7.3.1)
tagId (see 7.1.2.1)

8.12 ReadObjectIdsModules

The ReadObjectIdsModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to read all the **objectIds** from the RF tag. This module can be used in advance of a more selective command to read a specific **object**, or to identify duplicate **objectIds** so that a housekeeping procedure can be invoked. A valid response, if the RF tag Logical Memory Map has no **objectIds** stored, is to return an empty objectId list. Only one RF tag shall be programmed per command to ensure that the read process is robust.

The ASN.1 Abstract Syntax for ReadObjectIdsModules is given in Table 11.

Table 11 — ReadObjectIdsModules

```
-- Read Object Ids
-- The ReadObjectIdsCommand instructs the interrogator to read all Object Ids from
-- the tag specified by its TagId.  Objects can then be read individually by the
-- ReadObjectCommand.

-- If there are duplicate OIDs, the interrogator shall return them as multiple
-- occurrences of the OID in the objectIdsFound list.

-- If a tag has no Object IDs, the objectIdsFound will be returned empty; and
-- as such the command will be executed without error.

ReadObjectIdsCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) readObjectIds(8)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

    TagId::=   OCTET STRING(SIZE(0..255))
```

```
               -- See Clause 7.1.2.1 for detailed specification

END


ReadObjectIdsResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) readObjectIds(8)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ReadObjectIdsResponse ::=  SEQUENCE {
    completionCode     INTEGER {
                           noError(0),
                           tagIdNotFound(8),
                           executionError(255)
                           },
    executionCode      INTEGER ,
                       -- See Clause 8.3 and notes in this syntax for a full list of
                       -- executionCodes
    objectIdsFound     SEQUENCE OF ObjectId
    }

ObjectId ::= OBJECT IDENTIFIER-- Full OID value

END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

completionCode (see 8.2)
executionCode (see 8.3)
objectIdsFound (see 8.4.12)
objectId (see 7.3.1)
tagId (see 7.1.2.1)

8.13 ReadAllObjectsModules

The ReadAllObjectsModules consist of a commandmodule, and the associated responseModule, that instruct the interrogator to read the entire Logical Memory Map of the RF tag and respond with this in a completely structured way that identifies each instance of the following: **objectId, object, compactParameter, lockStatus.** Only one RF tag shall be programmed per command to ensure that the reading process is robust.

The ASN.1 Abstract Syntax for ReadAllObjectsModules is given in Table 12.

Table 12 — ReadAllObjectsModules

```
-- Read All Objects
-- The ReadAllObjectsCommand instructs the interrogator to read all objects, their
-- ObjectIds and associated parameters from the tag specified by its TagId. The
-- interrogator shall return all objects and their parameters of the tag specified
-- by its TagId.  If there are duplicate OIDs, the interrogator shall return them
-- as multiple occurrences of the OID, objects, and associated parameters in the
-- Objects list.


ReadAllObjectsCommand
```

```
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) readAllObjects(9)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

    TagId::=       OCTET STRING(SIZE(0..255))
                   -- See Clause 7.1.2.1 for detailed specification

END


ReadAllObjectsResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) readAllObjects(9)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ReadAllObjectsResponse ::=  SEQUENCE {
    completionCode INTEGER {
                       noError(0),
                       tagIdNotFound(8),
                       objectsNotRead(16),
                       executionError(255)
                       },
    executionCode  INTEGER,
                   -- See Clause 8.3 and notes in this syntax for a full list of
                   -- executionCodes
    objects    SEQUENCE OFSEQUENCE {
        objectId           OBJECT IDENTIFIER,
        object             OCTET STRING,
        compactParameter   INTEGER {
                               applicationDefined(0),
                               -- The object was not originally encoded through the
                               -- data compaction rules of 15962, and is as sent
                               -- from the source application and might require
                               -- additional processing by the receiving
                               --application.
                               utf8Data(2),
                               -- Data has been externally transformed from a 16-bit
                               -- coded character set to a UTF-8 string. The object
                               -- needs to be processed through an external UTF-8
                               -- decoder.
                               de-compactedData(15)
                               -- The object was originally encoded through the data
                               -- compaction rules of 15962 and de-compacted on this
                               -- read operation and restored to its original
                               --format.
                           }(0..15),
        lockStatus         BOOLEAN
                           -- If TRUE, object is locked
        }
    }
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

compactParameter (see 7.3.3)

completionCode (see 8.2)
executionCode (see 8.3)
lockStatus (see 7.3.4)
object (see 7.3.2)
objectId (see 7.3.1)
objects (see 8.4.11)
tagId (see 7.1.2.1)

8.14 ReadLogicalMemoryMapModules

The ReadLogicalMemoryMapModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to read the entire Logical Memory Map of the RF tag and respond with this in a completely unstructured way (i.e. by returning the encoded byte values). No processing takes place through the Data Protocol Processor as part of this read command, so that individual **objectIds, objects, compactParameter** and **lockStatus** cannot be directly identified. Also, if a directory structure has been defined by the **accessMethod,** this shall be included in the response, but shall not be distinguished from other bytes in the Logical Memory Map. The main function of this command is for diagnostic purposes, but it could also be used for other functions where reading the complete content of the Logical Memory Map is required. Only one RF tag should be programmed per command to ensure that the reading process is robust.

The ASN.1 Abstract Syntax for ReadLogicalMemoryMapModules is given in Table 13.

Table 13 — ReadLogicalMemoryMapModules

```
-- Read Logical Memory Map
-- The ReadLogicalMemoryMapCommand instructs the interrogator to return the whole
-- contents of Logical Memory Map of the tag specified by its TagId. It differs
-- from the ReadAllObjectsCommand in that this transfers the raw byte string to the
-- application, without processing through the Data Protocol Processor.  This could
-- be used for diagnostic purposes.


ReadLogicalMemoryMapCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) readLogicalMemoryMap(10)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN


    TagId::=       OCTET STRING(SIZE(0..255))
                   -- See Clause 7.1.2.1 for detailed specification

END


ReadLogicalMemoryMapResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) readLogicalMemoryMap(10)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ReadLogicalMemoryMapResponse ::=  SEQUENCE {
    completionCode     INTEGER {
                           noError(0),
                           tagIdNotFound(8),
                           readIncomplete(19),
                           executionError(255)
                           },
    executionCode      INTEGER,
```

```
                        -- See Clause 8.3 and notes in this syntax for a full list of
                        -- executionCodes
    logicalMemoryMap   OCTET STRING
}
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

completionCode (see 8.2)
executionCode (see 8.3)
logicalMemoryMap (see 8.4.8)
tagId (see 7.1.2.1)

8.15 InventoryAndReadObjectsModules

The InventoryAndReadObjectsModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to identify a particular set of RF tags present in its operating field, and further to return specific data from each individual RF tag. The inventory function is constrained by using the **applicationFamilyId** and **identifyMethod**, exactly as described in 8.7.

The command argument **objectIdList** consists of a common list of **objectIds** to be read from each RF tag defined by the selection criteria. If the **objectIdList** is null, then the interrogator shall retrieve all **objectIds** and associated data from each RF tag. The response for each **tagId** will consist of a sequence of: **objectId, object, compactParameter, lockStatus**.

The ASN.1 Abstract Syntax for InventoryAndReadObjectsModules is given in Table 14.

Table 14 — InventoryAndReadObjectsModules

```
-- Inventory and Read Objects
-- The InventoryAndReadObjectsCommand instructs the interrogator to inventory (i.e.
-- to identify) all tags present in its operating area and to read the objects
-- specified by the OID list for each inventoried tag.  Each tag is uniquely
-- identified by its TagId.

-- If the objectIdList is NULL, then the interrogator shall retrieve all objects of
-- all inventoried tags.

-- If there are duplicate OIDs, the interrogator shall return them as multiple
-- occurrences of the OID, objects, and associated parameters in the Objects list.


InventoryAndReadObjectsCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) inventoryAndReadObjects(11)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

InventoryAndReadObjectsCommand ::= SEQUENCE {
    applicationFamilyId   ApplicationFamilyId,
    identifyMethod        INTEGER {
                              inventoryAllTags(0),
                              inventoryAtLeast(1),
                              inventoryNoMoreThan(2),
                              inventoryExactly (3)
                              }(0..15),
```

```
    numberOfTags          INTEGER (0..65535),
    objectIdList          SEQUENCE OF ObjectId
    }

-- The ApplicationFamilyId separates fundamentally different types of application
-- data (see 7.1.2.2 and Annex B), and possibly particular objectIds.
-- Specifying a hex value xx (where x is a non-zero value) selects only the RF tags
-- that have the required data content.
-- Specifying a hex value 00 selects all the RF tags; this may be an appropriate
-- action to undertake a full inventory.
-- Specifying a hex value 0x, or x0, (where x is a non-zero value) might not be
-- logically sound because the RF tags are from different applications and the x
-- value has different meaning.

-- If the identifyMethod is set to inventoryAllTags, the interrogator shall perform
-- a complete inventory of all tags present in its field of operation. The value of
-- numberOfTags is irrelevant and should be set to zero by the application.

-- If the identifyMethod is set to inventoryAtLeast, the interrogator shall perform
-- an inventory of the tags present in its field of operation and shall return a
-- response only once it has identified a number of tags equal to numberOfTags.
-- If the numberOfTags is set to 1, the Interrogator will wait until the first tag
-- has been detected.  This is a mechanism to wait for a tag to enter the
-- interrogator field.
-- If the numberOfTags is set to more than 1, the Interrogator will wait until the
-- specified number of tags has been detected.  This may lead to an indefinite wait
-- for the response from the interrogator.  It is the responsibility of the
-- application to accommodate this potentiality.

-- If the identifyMethod is set to inventoryNoMoreThan, the interrogator shall
-- initiate an inventory of the tags present in its field of operation and shall
-- return a response with a number of tags lower or equal to numberOfTags.
-- The interrogator may interrupt the inventory process when the numberOfTags has
-- been reached or may continue the inventory process till all tags have been read.
--     Note: This may be constrained by the air interface and anticollision
--     mechanism.


ApplicationFamilyId ::= SEQUENCE {
    applicationFamily     INTEGER(0..15),
    applicationSubFamily  INTEGER(0..15)
    }
    -- The code values are defined in the ConfigureAfiCommand.

ObjectId ::= OBJECT IDENTIFIER-- Full OID value

END


InventoryAndReadObjectsResponse
{iso(1)          standard(0)          rfid-data-protocol(15961)          responseModules(127)
inventoryAndReadObjects(11)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

InventoryAndReadObjectsResponse ::= SEQUENCE {
    completionCode     INTEGER {
                           noError(0),
                           objectsNotRead(16),
```

```
                        failedToReadMinimumNumberOfTags(23),
                        -- for example, this could be due to a time-out
                        failedToReadExactNumberOfTags(24),
                        -- for example, this could be due to a time-out
                        executionError(255)
                        },
    executionCode      INTEGER ,
                       -- See Clause 8.3 and notes in this syntax for a full list of
                       -- executionCodes
    numberOfTagsFound  INTEGER (1..65535),
    tagIdAndObjects    SEQUENCE {
        tagId          OCTET STRING(SIZE(0..255)),
                       -- See Clause 7.1.2.1 for detailed specification
        objects        SEQUENCE OFSEQUENCE {
            objectId   OBJECT IDENTIFIER,
            object     OCTET STRING,
            compactParameter  INTEGER {
                              applicatioDefined(0),
                              -- The object was not originally encoded through the
                              -- data compaction rules of 15962, and is as sent
                              -- from the source application and might require
                              - additional processing by the receiving application.
                              utf8Data(2),
                              -- Data has been externally transformed from a 16-bit
                              -- coded character set to a UTF-8 string.The object
                              -- needs to be processed through an external UTF-8
                              -- decoder.
                              de-compactedData(15)
                              -- The object was originally encoded through the data
                              -- compaction rules of 15962 and de-compacted on this
                              -- read operation and restored to its original
                              -- format.
                              }(0..15),
            lockStatus        BOOLEAN
                              -- If TRUE, object is locked
            }
        }
    }
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

applicationFamily (see 7.1.2.2)
applicationFamilyId (see 7.1.2.2)
applicationSubFamily (see 7.1.2.2)
compactParameter (see 7.3.3)
completionCode (see 8.2)
executionCode (see 8.3)
identifyMethod (see 8.4.5)
lockStatus (see 7.3.4)
numberOfTags (see 8.4.9)
numberOfTagsFound (see 8.4.10)
object (see 7.3.2)
objectId (see 7.3.1)
objectIdList (see 8.4.10)
tagId (see 7.1.2.1)
tagIdAndObjects (see 8.4.14)

8.16 EraseMemoryModules

The EraseMemoryModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to re-set to zero the entire Logical Memory Map of the specified RF tag. This includes the directory, if this is defined as the **accessMethod.** If none of the blocks is locked, this should result in a deletion of all **objectIds, objects** and associated precursors. If any block is locked, then the **completionCode**: blocksLocked will be returned. The **objects** that remain on the RF tag can be identified by the subsequent use of the ReadObjectIdsModule (see 8.12). Only one RF tag shall be programmed per command to ensure that the erasure process is robust.

The ASN.1 Abstract Syntax for EraseMemoryModules is given in Table 15.

Table 15 — EraseMemoryModules

```
-- Erase Memory
-- The EraseMemoryCommand instructs the interrogator to reset to zero the whole
-- Logical Memory Map of the tag specified by its TagId. This results in the
-- deletion of all ObjectIds, Objects and associated parameters. The objects that
-- are locked cannot be deleted. If some objects cannot be deleted because they are
-- locked, the interrogator shall return the appropriate completionCode. Remaining
-- Objects can then be identified by the ReadObjectIdsCommand.


EraseMemoryCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) eraseMemory(12)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

    TagId::=       OCTET STRING(SIZE(0..255))
                   -- See Clause 7.1.2.1 for detailed specification

END


EraseMemoryResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) eraseMemory(12)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

EraseMemoryResponse ::=  SEQUENCE {
    completionCode     INTEGER {
                           noError(0),
                           tagIdNotFound(8),
                           blocksLocked(17),
                           eraseIncomplete(18),
                           -- process incomplete for some undefined reason, but it
                           -- is possible to re-invoke the command to complete
                           executionError(255)
                           },
    executionCode      INTEGER
                       -- See Clause 8.3 and notes in this syntax for a full list of
                       -- executionCodes
    }
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

completionCode (see 8.2)
executionCode (see 8.3)
tagId (see 7.1.2.1)

8.17 GetApplication-basedSystemInformationModules

The GetApplication-basedSystemInformationModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to read the system information and return those arguments that are relevant to the application, namely the applicationFamilyId and storageFormat.

The ASN.1 Abstract Syntax for GetApplication-basedSystemInformationModules is given in Table 16.

Table 16 — GetApplication-basedSystemInformationModules

```
-- Get Application-based System information
-- The GetApp-basedSystemInfoCommand instructs the interrogator to read the System
-- information from the tag specified by its TagId, and abstract the parameters
-- relevant to the application data: the applicationFamilyId and the storageFormat.


GetApp-basedSystemInfoCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) getApp-basedSystemInfo(13)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

    TagId::=       OCTET STRING(SIZE(0..255))
                   -- See Clause 7.1.2.1 for detailed specification

END


GetApp-basedSystemInfoResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) getApp-basedSystemInfo(13)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

GetApp-basedSystemInfoResponse ::=  SEQUENCE {
    completionCode        INTEGER {
                              noError(0),
                              tagIdNotFound(8),
                              systemInfoNotRead(20),
                              executionError(255)
                              },
    executionCode         INTEGER,
                          -- See Clause 8.3 and notes in this syntax for a full
                          -- list of executionCodes
    applicationFamilyId   ApplicationFamilyId,
    storageFormat         StorageFormat
    }

ApplicationFamilyId ::= SEQUENCE {
    applicationFamily     INTEGER(0..15),
    applicationSubFamily  INTEGER(0..15)
    }
```

```
    -- The code values are defined in the ConfigureAfiCommand.

StorageFormat ::= SEQUENCE {
    accessMethod      INTEGER (0..3),
    dataFormat        INTEGER (0..31)
    }
    -- The code values are defined in the ConfigureStorageFormatCommand.

END
```

The following elementNames used in these modules are defined elsewhere in this International Standard, as detailed:

accessMethod (see 7.1.2.4)
applicationFamily (see 7.1.2.2)
applicationFamilyId (see 7.1.2.2)
applicationSubFamily (see 7.1.2.2)
completionCode (see 8.2)
dataFormat (see 7.1.2.5)
executionCode (see 8.3)
storageFormat (see 7.1.2.3)
tagId (see 7.1.2.1)

8.18 AddMultipleObjectsModules

The AddMultipleObjectsModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to write a set of **objects**, their **objectIds**, and associated parameters into the RF tag Logical Memory Map. The command arguments can be used to lock individually the **objectId**, the **object** and associated parameters; and to check that the **objectId** is not already encoded on the RF tag. Only one RF tag shall be programmed per command to ensure that the writing process is robust.

The ASN.1 Abstract Syntax for AddMultipleObjectsModules is given in Table 17.

Table 17 — AddMultipleObjectsModules

```
-- Add Multiple Objects
-- The AddMultipleObjectsCommand instructs the interrogator to write a sequence of
-- objects, their ObjectIds and associated parameters into the tag logical memory
-- map.
--      NOTE: There is also an AddSingleObjectCommand.

-- If the checkDuplicate flag is set to TRUE, the interrogator shall verify, before
-- adding the objects, that no object with the same OID already exists. If such an
-- object exists, the interrogator shall not perform the Add Object function and
-- shall return the appropriate Completion Code.

-- If the Lock flag is set to TRUE, the interrogator shall lock the ObjectId, the
-- Object, its compaction scheme and associated parameters into the tag Logical
-- Memory Map


AddMultipleObjectsCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) addMultipleObjects(14)}
DEFINITIONS
EXPLICIT TAGS ::=
```

```
BEGIN


AddMultipleObjectsCommand ::= SEQUENCE {
    tagId              OCTET STRING(SIZE(0..255)),
                       -- See Clause 7.1.2.1 for detailed specification
    addObjectsList     SEQUENCE OF SEQUENCE{
        objectId           OBJECT IDENTIFIER,-- Full OID value
        avoidDuplicate     BOOLEAN,
                           -- If set to TRUE, check for duplicate objectId
        object             OCTET STRING,
        compactParameter   INTEGER {
                               applicationDefined(0),
                               -- The object shall not be processed through the data
                               -- compaction rules of 15962 and remains unaltered
                               compact(1),
                               -- Compact object as efficiently as possible using
                               -- 15962 compaction rules
                               utf8Data(2)
                               -- Data has been externally transformed from a 16-bit
                               -- coded character set to a UTF-8 string. The object
                               -- shall not be processed through the data compaction
                               -- rules of 15962 and remains unaltered
                               }(0..15),
        objectLock         BOOLEAN
                           -- If TRUE the interrogator shall lock the ObjectId, the
                           -- Object, its compaction scheme and other features in
                           -- the Logical Memory Map
        }
    }

END


AddMultipleObjectsResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) addMultipleObjects(14)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

AddMultipleObjectsResponse ::= SEQUENCE {
    tagWriteResponse   SEQUENCE OF SEQUENCE{
        objectId           OBJECT IDENTIFIER,-- Full OID value
        completionCode     INTEGER{
                               noError(0),
                               tagIdNotFound(8),
                               objectNotAdded(9),
                               duplicateObject(10),
                               objectAddedButNotLocked(11),
                               executionError(255)
                               }
        },
    executionCode      INTEGER
                       -- See Clause 8.3 and notes in this syntax for a full list of
                       -- executionCodes
    }
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

addObjectsList (see 8.4.1)
avoidDuplicate (see 8.4.3)
compactParameter (see 7.3.3)
completionCode (see 8.2)
executionCode (see 8.3)
object (see 7.3.2)
objectId (see 7.3.1)
objectLock (see 7.3.4)
tagId (see 7.1.2.1)
tagWriteResponse (see 8.4.16)

8.19 ReadMultipleObjectsModules

The ReadMultipleObjectsModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to read a set of **objectIds**, their **objects**, and associated parameters from the RF tag Logical Memory Map. Command arguments can be used to check that the **objectId** is not duplicated on the RF tag. Only one RF tag shall be programmed per command to ensure that the reading process is robust.

The ASN.1 Abstract Syntax for ReadMultipleObjectsModules is given in Table 18.

Table 18 — ReadMultipleObjectsModules

```
-- Read Multiple Objects
-- The ReadMultipleObjectsCommand instructs the interrogator to read the Objects
-- specified by their Object Identifiers from the tag specified by its TagId.
--     NOTE: There is a ReadSingleObjectCommand.

-- If the checkDuplicate flag is set to FALSE, the interrogator shall return the
-- first Object found having the requested OID without checking for duplicates.

-- If the checkDuplicate flag is set to TRUE, the interrogator shall check for
-- duplicate objects having the requested OID. If more than one object with the
-- requested OID is found, the interrogator shall return the first found Object
-- having the requested OID and indicate the presence of duplicates with
-- appropriate Completion code.


ReadMultipleObjectsCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) readMultipleObjects(15)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ReadMultipleObjectsCommand ::= SEQUENCE {
    tagId                 OCTET STRING(SIZE(0..255)),
                          -- See Clause 7.1.2.1 for detailed specification
    readObjectList        SEQUENCE OF SEQUENCE{
        objectId              OBJECT IDENTIFIER,-- Full OID value
        checkDuplicate        BOOLEAN
                              -- If set to TRUE, the interrogator shall check that
                              -- there is only one occurrence of the ObjectId
        }
    }

END
```

```
ReadMultipleObjectsResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) readMultipleObjects(15)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ReadMultipleObjectsResponse ::=  SEQUENCE {
    tagReadResponse    SEQUENCE OF SEQUENCE{
        objectId           OBJECT IDENTIFIER,
        object             OCTET STRING,
        compactParameter   INTEGER {
                               applicationDefined(0),
                               -- The object was not originally encoded through the
                               -- data compaction rules of 15962, and is as sent
                               -- from the source application and might require
                               -- additional processing by the receiving
                               -- application.
                               utf8Data(2),
                               -- Data has been externally transformed from a 16-bit
                               -- coded character set to a UTF-8 string. The object
                               -- needs to be processed through an external UTF-8
                               -- decoder.
                               de-compactedData(15)
                               -- The object was originally encoded through the data
                               -- compaction rules of 15962 and de-compacted on this
                               -- read operation and restored to its original
                               -- format.
                               }(0..15),
        lockStatus         BOOLEAN,
                           -- If TRUE, object is locked
        completionCode     INTEGER {
                                   noError(0),
                                   tagIdNotFound(8),
                                   duplicateObject(10),
                                   objectIdNotFound(13),
                                   objectNotRead(15),
                                   executionError(255)
                                   }
        },
    executionCode      INTEGER
                       -- See Clause 8.3 and notes in this syntax for a full list of
                       -- executionCodes
    }
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

checkDuplicate (see 8.4.4)
compactParameter (see 7.3.3)
completionCode (see 8.2)
executionCode (see 8.3)
lockStatus (see 7.3.4)
object (see 7.3.2)
objectId (see 7.3.1)
readObjectList (see 8.4.12)
tagId (see 7.1.2.1)
tagReadResponse (see 8.4.15)

8.20 ReadFirstObjectModules

The ReadFirstObjectModules consist of a commandModule, and the associated responseModule, that instruct the interrogator to read an object in the first position on the RF tag returning its **objectId**, its **object**, and associated parameters from the RF tag Logical Memory Map. Only one RF tag shall be programmed per command to ensure that the reading process is robust.

An efficient process could be achieved by being able to transfer only the relevant blocks of data across the air interface. This requires the application to provide the following:

- Some degree of certainty that the expected **object** is encoded in the particular position, which can be achieved by rules in the application standard.
- The known, or maximum, compacted length of the expected **object**. This is used to establish the number of octets to be transferred and result in the expected **object** being returned. The value of the compacted length can be determined by the procedure described below.
- The value of the expected **objectId** to enable the Data Protocol Processor to convert this to encoded octets and add this and other encoded parameters to the compacted length.

The relevant compaction schemes and the characters that can be compacted are defined in Table 19.

Table 19 — Compaction Schemes and characters

Compaction Scheme	Number of Characters	Characters Supported
Numeric (4-bit)	10	0..9
5-bit	31	A..Z [\ ] ^ _
6-bit	64	As Numeric + 5-bit and the following: SPACE ! “ # $ % & ‘ ( ) * + , - . / : ; < = > ? @
7-bit	128	All ISO/IEC 646 characters including the control characters
8-bit	256	All ISO/IEC 8859-1 characters including the control characters

EXAMPLES
As an example of its use, consider an application standard requiring that the unique item identifier for ISO/IEC 15459 transport units is placed in the first position. The maximum length of this identifier is 35 alphanumeric characters. Assume that the particular data capture point handles transport units from various sources and has no prior knowledge of the length of particular codes, so the worst case has to be catered for. Table 19 provides a list of the characters capable of being encoded by the compaction schemes of ISO/IEC 15962. The alphanumeric characters specified by ISO/IEC 15459 can all be compacted by the 6-bit scheme. So the maximum length of the encoded object is (35 X 6/8 = 26.25) 27 octets, because the value has to be rounded up to be properly encoded. The Data Protocol Processor adds to this the encoded length of the **objectId** and associated parameters to determine the number of blocks to be transferred.

If the object is of known length and structure, say a 20 digit code value, Table 19 shows that this can be compacted using the numeric compaction scheme at 4 bits per digit. So the total compacted length of the **object** is 10 octets. Although integer compaction may be invoked to produce a shorter encoded **object**, calculating the length based on the simpler numeric compaction covers all cases for numeric unique item identifiers.

The basic objective is to minimise the transfer across the air interface with a high degree of certainty that the object is returned. It is possible to refine the process to determine number of octets. The length of the

expected **object** can be calculated more precisely by simulating the compaction process using samples of actual returned objects.

The ASN.1 Abstract Syntax for ReadSingleObjectModules is given in Table 20.

Table 20 — ReadFirstObjectModules

```
-- Read First Object
-- The ReadFirstCommand instructs the interrogator to read the object specified
-- by its position stored in the tag.

-- This command might have features across the air interface that could be faster
-- than reading a single named object. The application may therefore select to have
-- the most often accessed object stored first in the tag.

ReadFirstObjectcommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) readFirstObject(16)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ReadFirstObjectCommand ::= SEQUENCE {
    TagId          OCTET STRING(SIZE(0..255)),
                   -- See Clause 7.1.2.1 for detailed specification
    objectId       OBJECT IDENTIFIER, -- This is the expected objectId.
                   -- Its value is used by the processes of ISO/IEC 15962 to
                   -- determine the number of blocks that need to be transferred. It
                   -- is not used as part of the search process itself
    maxAppLength   INTEGER(1..65535)
                   -- This value is specified by the application to represent the
                   -- maximum compacted length of the expected object
    }

END


ReadFirstObjectResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) readFirstObject(16)}
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

ReadFirstObjectResponse ::=  SEQUENCE {
    ObjectId           OBJECT IDENTIFIER, -- This is the actual ObjectId as found in
                       -- the first position
    object             OCTET STRING,
    compactParameter   INTEGER {
                           applicationDefined(0),
                           -- The object was not originally encoded through the data
                           -- compaction rules of 15962, and is as sent from the
                           -- source application and might require additional
                           -- processing by the receiving application.
                           utf8Data(2),
                           -- Data has been externally transformed from a 16-bit
                           -- coded character set to a UTF-8 string. The object
                           -- needs to be processed through an external UTF-8
                           -- decoder.
                           de-compactedData(15)
                           -- The object was originally encoded through the data
```

```
                        -- compaction rules of 15962 and de-compacted on this
                        -- read operation and restored to its original format.
                        }(0..15),
    lockStatus          BOOLEAN,
                        -- If TRUE, object is locked
    completionCode      INTEGER {
                            noError(0),
                            tagIdNotFound(8),
                            objectIdNotFound(13),
                            objectNotRead(15),
                            executionError(255)
                            },
    executionCode       INTEGER
                        -- See Clause 8.3 and notes in this syntax for a full list of
                        -- executionCodes
    }
END
```

The following elementNames used in these modules are defined elsewhere in this International Standard as defined:

compactParameter (see 7.3.3)
completionCode (see 8.2)
executionCode (see 8.3)
lockStatus (see 7.3.4)
maxAppLength (see 8.4.9)
object (see 7.3.2)
objectId (see 7.3.1)
tagId (see 7.1.2.1)

8.21 Development commands

Development commands may be created using the syntax defined in this International Standard and following the modular format, with a separate module used for the command and for the response. Annex I provides detailed advice for development commands not to be in conflict with this International Standard and ISO/IEC 15962.

9 Compliance, or classes of compliance, to this standard

9.1 Application compliance

An application is only expected to support the commands that are meaningful to the application. For every command specified for an application, all the constituent components shall be supported.

9.2 Compliance of the Data Protocol Processor

For a Data Protocol Processor to claim compliance with this International Standard, it shall support all of the commands and their constituent parts.

9.3 Compliance of the RF tag and RF interrogator

It is unlikely that every class of tag compliant with the 18000 series of air interface standards will have the functionality to support all of the command and response modules. Partial compliance by a particular class of RF tag shall be defined by reference to the command and response modules that are supported. For whatever reason, if a device does not support particular command and response modules, then compliance to this International Standard can be claimed by responding with the **executionCode = commandNotSupported (4)** for these unsupported modules. If the claim is made to support a command, then every constituent feature of that command shall be supported.

Annex A
(normative)

First, Second and Third Arcs of Object Identifier Tree

The top arcs of the object identifier tree are specified in ISO/IEC 9834-1. The current list of arcs is defined in Table A.1. These ensure a uniqueness of all object identifiers through the administration by the Registration Authority or other mechanism defined in the column "Value of n".

Table A.1 — Object Identifiers: Top Arcs

Registration Hierarchy Name Tree	Object Identifier	Value of n	Reference in ISO/IEC 9834-1 ITU-T Rec X660
itu-t (0) recommendation (0) n	{0 0 n}	1 to 26 assigned to letters a to z. Arcs below these have numbers of ITU-T and CCITT Recommendation	Amd 2 B.4
itu-t (0) question (1) n	{0 1 n}	ITU-T Study Group, qualified by study period	Amd 2 B.5
itu-t (0) administration (2) n	{0 2 n}	X-121 DCC	Amd 2 B.6
itu-t (0) network-operator (3) n	{0 3 n}	X-121 DNIC for a distinct code for packet switching applications	Amd 2 B.7
itu-t (0) identified organisation (4) n	{0 4 n}	Assigned by ITU Telecommunications Standardisation Bureau Procedures specified in ITU-T X.669	Amd 2 B.8
iso (1) standard (0) n	{1 0 n}	ISO standard number	Amd 2 A.4
iso (1) registration-authority (1) n **(withdrawn 1998 Amd 2)**	{1 1 n}		
iso (1) member-body (2) n	{1 2 n}	3 digit country code of ISO 3166	Amd 2 A.5
iso (1) identified-organisation (3) n	{1 3 n}	ICD allocated by Registration Authority (ISO 6523)	Amd 2 A.6
joint-iso-itu-t (2) n	{2 n}	In accordance with ITU-T Rec X662.1 (ISO/IEC 9834.3)	Amd 2 C.3
joint-iso-itu-t (2) country (16) country-name (n)	{2 16 n}		(1992) Anx A.5
joint-iso-itu-t (2) registration-procedure (17) specific-procedure (n)	{2 17 n}		(1992) Anx A.4
joint-iso-itu-t (2) international RA (23) n	{2 23 n}	To ITU-T X.666	

Examples of object identifiers:

0 0 24 660: itu-t (0) recommendation (0) x (24) rec.x 660 (660)

1 0 8571 2 1: iso (1) standard (0) FTAM (8571) abstract-syntax (2) pci (1)

1 2 250 1 6325 316: ISO (1) member-body (2) France (250) type-org (1) abc (6325) marketing agreement (316)

NOTE: This is not a real example but an illustration from ISO/IEC 9834-1.

2 16 840 46 3125: joint-iso-itu-t (2) country (16) country-name (840) state-or-province (46) organisation abc (3125)

NOTE: This is not a real example but an illustration from ISO/IEC 9834-1

2 17 2 3: joint-iso-itu-t (2) registration-procedures (17) document-types (2) binary (3)

NOTE: This is not a real example but an illustration from ISO/IEC 9834-1

Annex B
(normative)

Code Assignments for ApplicationFamilyId

AFI Code (Decimal)	ASF Code (Decimal)	SystemInfo Encodation (Hexadecimal)	Assigned to:
9	1	91	EAN.UCC system (to be defined before publication)
9	2	92	EAN.UCC system (to be defined before publication)
9	3	93	EAN.UCC system (to be defined before publication)
9	4	94	EAN.UCC system (to be defined before publication)
9	5	95	EAN.UCC system (to be defined before publication)
9	6	96	EAN.UCC system (to be defined before publication)
9	7	97	EAN.UCC system (to be defined before publication)
9	8	98	RESERVED
9	9	99	RESERVED
9	10	9A	RESERVED
9	11	9B	RESERVED
9	12	9C	RESERVED
9	13	9D	RESERVED
9	14	9E	RESERVED
9	15	9F	RESERVED
10	1	A1	ANS MH10.8.2 Data Identifiers for unique identifier for items
10	2	A2	ANS MH10.8.2 Data Identifiers for unique identifier for transport units
10	3	A3	ANS MH10.8.2 Data Identifiers for unique identifier for returnable transport items
10	4	A4	RESERVED
10	5	A5	RESERVED
10	6	A6	RESERVED
10	7	A7	RESERVED
10	8	A8	RESERVED
10	9	A9	RESERVED
10	10	AA	RESERVED
10	11	AB	RESERVED
10	12	AC	RESERVED
10	13	AD	RESERVED
10	14	AE	RESERVED
10	15	AF	RESERVED
11	1	B1	ISO/IEC 15459 unique identifier for items
11	2	B2	ISO/IEC 15459 unique identifier for transport units
11	3	B3	ISO/IEC 15459 unique identifier for returnable transport items
11	4	B4	RESERVED
11	5	B5	RESERVED
11	6	B6	RESERVED
11	7	B7	RESERVED
11	8	B8	RESERVED
11	9	B9	RESERVED
11	10	BA	RESERVED
11	11	BB	RESERVED
11	12	BC	RESERVED

AFI Code (Decimal)	ASF Code (Decimal)	SystemInfo Encodation (Hexadecimal)	Assigned to:
11	13	BD	RESERVED
11	14	BE	RESERVED
11	15	BF	RESERVED
12	1	C1	IATA Baggage Tag
12	2	C2	RESERVED
12	3	C3	RESERVED
12	4	C4	RESERVED
12	5	C5	RESERVED
12	6	C6	RESERVED
12	7	C7	RESERVED
12	8	C8	RESERVED
12	9	C9	RESERVED
12	10	CA	RESERVED
12	11	CB	RESERVED
12	12	CC	RESERVED
12	13	CD	RESERVED
12	14	CE	RESERVED
12	15	CF	RESERVED

NOTES: AFI = applicationFamilyId; the codes are shown as decimal values to be compatible with the requirements of this International Standard

ASF = applicationSubFamily: the codes are shown as decimal values to be compatible with the requirements of this International Standard

The codes for SystemInfo are shown as hexadecimal values to be compatible with the requirements of ISO/IEC 15962

Annex C
(informative)

Accommodating established data formats

This International Standard has been prepared on the basis that its object based protocol differs from the message based protocols and syntax of some AIDC application standards. Therefore, basic data objects need to be presented in a manner relevant to the application standard, for example in terms of:

- the data object being supported in the data dictionary
- the format of the data (e.g. numeric, alphanumeric), including its length
- combinations of data objects which are valid or illegal

These features are outside the scope of this International Standard and are the responsibility of the application.

Some conversion process is necessary until the application systems can handle data and identifiers in the format specified in this International Standard. It is possible to have two independent implementation paths: one to write data, and one to read data.

For some major applications, some rigorous rules exist of what constitutes legitimate data. Software exists to ensure compliance to this format when using the existing message based syntax. Users need to ensure that, as they implement an object based method of writing data, the data itself follows the basic rules. Figure C.1 illustrates this schematically.

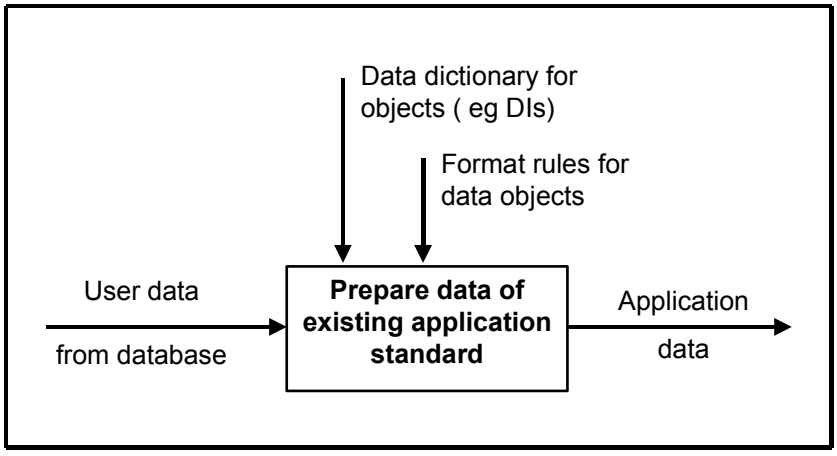


Figure C.1 — Data Flow Model : Prepare Data of Existing Application Standards

A similar process is required when data is read from an RF tag.

AIDC technologies based on write-once-read-many-times (WORM) technology can rely on the fact that data, as written, is what is read. This means the message syntax is encoded in the data carrier. The read-write capabilities of RFID, and the object based nature of the data protocol of this International Standard, means that an established syntax has to be constructed based on the object identifier structure. While there is a requirement to output data with a particular data syntax (e.g. that of ISO/IEC 15434), then a conversion module is required to correctly map the set of object identifiers and data objects to the message format required for the application of this International Standard. This shall require the inversion of the conversion rules for some of the application data.

In addition, the message syntax of the established application standard needs to be created. This process generally requires all but the final arc of the object identifier to be discarded and for data separators (compliant with the application standard) to be inserted correctly. Refer to the appropriate application standards for the precise rules.

A logical development step is for the application standard to develop procedures to accept output based on transfer syntax.

Annex D
(informative)

Contact Addresses for Managers of Main Application Data Dictionaries

Organisations responsible for the maintenance of different codes and established data elements are listed below:

D.1 EAN.UCC System

EAN•UCC Application Identifiers Secretariat:

c/o EAN International
Rue Royale, 145
B-1000 Brussels
BELGIUM

Tel: +32 2/227 10 20
Fax: +32 2/227 10 21
E-Mail: info@ean.be

D.2 Data Identifiers

ANSI MH 10.8.2 Data Identifiers maintained by:

ANSI MH10/SC 8/WG2
c/o ANSI MH10.8.2 chair
Material Handling Industry Association (Secretariat)
8720 Red Oak Blvd - Suite 201
Charlotte, NC 28217
UNITED STATES OF AMERICA

Tel: +1 (704) 676 1190
Fax: +1 (704) 676 1199
E-Mail: MH10.8.2.chair@mhia.org (Requestors should submit via E-Mail)

D.3 IATA data elements

International Air Transport Association (IATA)
IATA Centre
Route de l'Aeroport 33
PO Box 416
CH-1215 Geneva 15 Airport
SWITZERLAND

Tel: +41 (22) 799 2723

D.4 UPU data elements

TBD

Annex E
(normative)

Converting alphanumeric Data Identifiers to the final arc of the Object Identifier

The final component of the object identifier for Data Identifiers (as defined in the ANSI MH 10.8.2 standard) is derived by converting the alphanumeric DI to a binary value and then converting this to a numeric value.

The procedure is as follows:

1. Convert the numeric component (if any) of the DI to a binary value (up to 10 bits):

   EXAMPLE: 14 = 1110

2. Convert the alphabetic component of the DI to an ordinal binary value of 5 bits length (where A = 00001 and Z = 11010).

   EXAMPLE: K (the 11th letter) = 01011

3. Prefix the binary numeric component (from Step 1) to the fixed length ordinal binary value (from Step 2).

   EXAMPLES: 14K = 1110 01011

   Q = 10001

4. Convert the binary value to a decimal value. Examples are shown in Table C.1.

Table E.1 — Sample of DI Conversion to Object ID

DI	Binary Conversion	Object ID Final Component (Decimal)	Complete Object ID (Decimal)
J	01010	10	1 0 15961 10 10
12K	1100 01011	395	1 0 15961 10 395
14K	1110 01011	459	1 0 15961 10 459
1P	1 10000	48	1 0 15961 10 48
Q	10001	17	1 0 15961 10 17

NOTE: The procedure converts DIs that are only alphabetic, and those with low value numeric components, to a low value final component object identifier. This reduces the encoding space required for the more common DIs.

Annex F
(informative)

Relating data objects

Message based syntax can use recursive or looping techniques to created repeated sequences of related data (e.g. individual quantity and batch numbers linked to different product codes). When the complete message is parsed, the syntax identifies boundary points so that the attributes are correctly linked to the primary code.

With an object-based system (such as the Data Protocol of this International Standard and ISO/IEC 15962) operating at a base level, there is a risk of creating false links (i.e. product code A could be linked to quantity of product B). The problem can be overcome using one of the techniques described below. The methods should only be adopted if incorporated into the application standard associated with the item being managed. The illustrations limit the number of constructed data elements to 255 per RF tag, but different rules could be developed if a greater number of constructions is required. Either rule is transparent to the complete Data Protocol of this International Standard and ISO/IEC 15962, and so requires the processing to be implemented as part of the application. The options are included in this International Standard to describe robust ways to preserve an object-based data capture process using the object identifier tree structure.

F.1 Concatenation technique

Specific object identifiers can be created that link a defined set of attributes in a concatenated manner.

EXAMPLE:

lowest arc 245 =

- sequence number 1 octet
- product code 8 octets
- quantity 1 octet
- batch number 4 octets

In this case the object identifier whose lowest arc is 245 is encoded as the hexadecimal value 81 75 (using the rules in 6.2.2). The first octet of the Object, the sequence number, distinguishes one similar object data from another.

Using this technique, each different arrangement of basic elements to create the concatenated construction would be given a different final node. So, the concatenation of product + quantity + expiry date would have its lowest arc value different from that for product + quantity + batch.

This method is more suitable when fixed combinations of element have to be created and the length of each objects is fixed.

F.2 Object identifier extension technique

The basic object identifier can be extended by the addition of a new final arc, with this as a 'linking' value.

EXAMPLE:

The following three elements are to be linked:

- product code - final arc 48
- quantity - final arc 17
- batch - final arc 20

Assume that there are two different products whose details are encoded on the RF tag. So the linking extensions 1 and 2 apply. Six individual ObjectIds would be encoded:

... 48 1
.... 48 2
.... 17 1
.... 17 2
.... 20 1
.... 20 2

The extension value is used to link the different objects as a logical combination.

This method is more suitable when many different combinations of element have to be created and the length of, at least, one object can vary between occurrences.

The Extension technique for relating objects and their associated physical entities is similar to Scheme B for applying data security (see Annex G.1). Therefore, for any one **objectId**, the technique shall only be applied to data security or to linking physical entities.

Annex G
(informative)

Data security issues

Although data security is beyond the scope of this International Standard and ISO/IEC 15962, the following advice is provided to show how features of the Data Protocol may be used to achieve more secure data.

G.1 Object identifier issues

Encrypted data shall be associated with its own unique OBJECT IDENTIFIER. This ensures that authorised users can recognise encrypted data, but does not declare to other users this fact. The **object** itself simply appears with the **compactParameter** set as userDefined (see 7.3.3).

One method, called Scheme A for later reference, of creating the OBJECT IDENTIFIER is for this to have a final arc at the same level of all other final arcs in the application system. This is a systems level adoption of data security and requires all authorised users to know that the data is encrypted; however, the rules do not need to be publicly declared.

Another method¸ called Scheme B for later reference, for creating a unique OBJECT IDENTIFIER to identify the encrypted data is to extend the OBJECT IDENTIFIER of the plain (unencrypted) data and add an additional lower arc. This technique can be adopted bi-laterally between sender and authorised user(s), or at the systems levels for all authorised users. This technique can also be used to define the encryption type, selected keys and so on.

EXAMPLE:

0 1 15961 nn nn	Plain object
0 1 15961 nn nn 1	Encrypted object
0 1 15961 nn nn 2	Encryption type
0 1 15961 nn nn 3	Key

Scheme B is similar to that proposed for relating objects and their associated physical entities (see Annex F.2). Therefore, for any one **objectId**, the technique shall only be applied to data security or to linking physical entities.

G.2 The data object

The **object** containing the application data shall have its **compactParameter** set to userDefined after encryption.

The basic **object** should be expanded to include a pre-defined data field or signature of the authorised writing party. This will help ensure data integrity as any unauthorised modification of the encrypted **object** without access to the private key will most probably destroy the authorised signature. This will help identify that the **object** has been changed without authority.

If a completely different OBJECT IDENTIFIER is assigned to the encrypted data (Scheme A above), then it may need to be expanded to also contain additional unencrypted octets that define the encryption scheme and/or selection from a set of keys.

G.3 Using the TagId

The **TagId** is intended to be unique to the RF tag, distinguishing it from all others. It is usually created using more robust techniques than can be used to write data into the Logical Memory Map. As such, it can be used to enhance data validity.

One method is to concatenate the **TagId** value to the basic **object** data and encrypt the entire expanded **object**. When decrypted by an authorised user, the **TagId** within the expanded **object** can be compared with the real **TagId** (part of the system information) to verify that they are identical. This may be applied to either Scheme A or B for creating the OBJECT IDENTIFIER, described above.

Another method is to use the **TagId** to modify the original key for encryption and decryption. This can be made to work for binary keys such as DES, where the **TagId** can be *exclusive or-ed* with the original key.

Before this approach is used, the implementers should verify that this type of modification of the key does not undermine the encryption method.

G.4 Advice on public key methods of encryption

Public key algorithms require longer keys to provide the same strength as symmetric keys. For example, a 512-bit public-key encryption cipher would have the equivalent strength of a 64-bit symmetric key cipher. This cipher length could preclude the use of public key ciphers in smaller capacity tags.

If a public key method of encryption is used, the data security will be compromised if the private key is used to encrypt the data and a public key is used to decrypt the data. Likewise, data integrity will be compromised if the public key is used to encrypt the data and the private key is used to decrypt the data. Double encryption or other means must be used to ensure data security and integrity.

The public key should not be encoded in the tag unless it is locked as an unauthorised party could violate the data integrity by overwriting the public key with another key and using another corresponding private key to encrypt altered data in the object.

Annex H
(informative)

Example of a transfer encoding

This annex illustrates the basic encoding rules specified in this International Standard by showing the representation in octets of a (hypothetical) AddMultipleObjects command and response.

H.1 Functional description of the command

The function of the command being illustrated is that of writing two data objects to the RF tag, that has the unique identifier C7 37 79 C2 B7 A3 DB EF. Other details are as follows:

- 1st object identifier "1 0 15961 10 30", with the object "ABC123456", that is to be locked.
- 2nd object identifier "1 0 15961 10 17", with the object "50", that is not to be locked.
- There is no requirement to avoid duplicating data that is already encoded on the RF tag.
- Both objects are to be compacted.

H.2 The abstract syntax for the AddMultipleObjects command

```
AddMultipleObjectsCommand
{iso(1) standard(0) rfid-data-protocol(15961) commandModules(126) addMultipleObjects(14)}

DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

AddMultipleObjectsCommand ::= SEQUENCE {
    tagId                   OCTET STRING(SIZE(0..255)),
                            -- See Clause 7.1.2.1 for detailed specification
    addObjectsList          SEQUENCE OF SEQUENCE{
        objectId                OBJECT IDENTIFIER,-- Full OID value
        avoidDuplicate          BOOLEAN,
                                -- If set to TRUE, check for duplicate objectId
        object                  OCTET STRING,
        compactParameter        INTEGER {
                                    applicationDefined(0),
                                    -- The object shall not be processed through the data compaction rules of
                                    -- 15962 and remains unaltered
                                    compact(1),
                                    -- Compact object as efficiently as possible using 15962 compaction rules
                                    utf8Data(2)
                                    -- Data has been externally transformed from a 16-bit coded character set to a
                                    -- UTF-8 string. The object shall not be processed through the data
                                     -- compaction  rules of 15962 and remains unaltered
                                    }(0..15),
        objectLock              BOOLEAN
                                -- If TRUE the interrogator shall lock the ObjectId, the Object, its compaction
                                 -- scheme and other features in the Logical Memory Map
        }
    }
END
```

H.3 The AddMultipleObjects command with the data values

```
AddMultipleObjectsCommand       -- {1 0 15961 126 14}
        ::=                     {
         tagId                  C7 37 79 C2 B7 A3 DB EF HEX
         addObjectsList         {

         -- 1st object
         {
         objectId               {1 0 15961 10 30},
         avoidDuplicate         FALSE,
         object                 "ABC123456",
         compactParameter       compact(1),
         objectLock             TRUE
         },

         -- 2nd object
         {
         objectId               {1 0 15961 10 17},
         avoidDuplicate         FALSE,
         object                 "50",
         compactParameter       compact(1),
         objectLock             FALSE
         }
        }
}
```

Note: objectId values are presented as OBJECT IDENTIFIERS. Object values are presented in " " as printable characters.

H.4 The transfer encoding for the example command

The transfer encoding in octets of the command value given above (after applying the transfer encoding rules defined in this International Standard) is shown below in tabular form, with each row representing one of the lines of Annex H.3. The values of the Type identifiers, of lengths, and of the contents are shown in hexadecimal, two hexadecimal digits per octet.

Abstract Syntax		**Type**[1]	**Length**	**Value**
AddMultipleObjectsCommand[2]		06	05	28 FC 59 7E 0E
SEQUENCE[3]		30	3F	
TagId		04	08	C7 37 79 C2 B7 A3 DB EF
SEQUENCE (AddObjectsList)[4, 5]	OF	30	33	
SEQUENCE [3]		30	1B	
ObjectId		06	05	28 FC 59 0A 1E
AvoidDuplicate[6]		01	01	00
Object		04	09	41 42 43 31 32 33 34 35 36
CompactParameter[7]		02	01	01
ObjectLock[6]		01	01	FF
SEQUENCE [3]		30	14	
ObjectId		06	05	28 FC 59 0A 11
AvoidDuplicate[6]		01	01	00
Object		04	02	35 30
CompactParameter		02	01	01
ObjectLock[6]		01	01	00

Further explanation of particular encoded values are given below, where the number refers to the number in the tabular list.

1, The values in the **Type** column are derived from the rules in 6.2.2
2. The module OBJECT IDENTIFIER is encoded using the same rules as for any OBJECT IDENTIFIER.
3. As defined in 6.2.9, the SEQUENCE has no associated value.
4. As defined in 6.2.10 the **addObjectsList** argument specifies a SEQUENCE OF that has no associated value.
5. As there are two sets of objectId and associated data, the length of all nested encoding has to be encoded on the line.
6. The arguments **avoidDuplicate** and **objectLock** are BOOLEAN. The encoded value for BOOLEAN = FALSE shall be **00**. The encoded value for BOOLEAN = TRUE, may be any value other than **00.**
7. As defined in 7.3.3, the **compactParameter** argument is encoded as an INTEGER.

The complete transfer encoding of the command as an octet stream is:

06 05 28 FC 59 7E 0E 30 3F 04 08 C7 37 79 C2 B7 A3 DB EF 30 33 30 1B 06 05 28 FC 59 0A 1E 01 01 00 04 09 41 42 43 31 32 33 34 35 36 02 01 01 01 01 FF 30 14 06 05 28 FC 59 0A 11 01 01 00 04 02 35 30 02 01 01 01 01 00

H.5 Functional description of the response

For this example, assume that the RF tag was found and both sets of object identifier and object were correctly added to the RF tag, using all the correct processes of ISO/IEC 15962, with the one significant exception the the first object could not be locked as requested by the command. As the command was successfully executed, in terms of the systems communications, the executionCode to be returned is “0 noError”.

H.6 The abstract syntax for the AddMultipleObjects response

AddMultipleObjectsResponse
{iso(1) standard(0) rfid-data-protocol(15961) responseModules(127) addMultipleObjects(14)}

```
DEFINITIONS
EXPLICIT TAGS ::=
BEGIN

AddMultipleObjectsResponse ::= SEQUENCE {
    tagWriteResponse     SEQUENCE OF SEQUENCE{
        objectId             OBJECT IDENTIFIER,-- Full OID value
        completionCode       INTEGER{
                                 noError(0),
                                 tagIdNotFound(8),
                                 objectNotAdded(9),
                                 duplicateObject(10),
                                 objectAddedButNotLocked(11),
                                 executionError(255)
                                 }
        },
    executionCode        INTEGER
                         -- See Clause 8.3 and notes in this syntax for a full list of executionCodes
    }
END
```

H.7 The AddMultipleObjects response with the data values

```
AddMultipleObjectsResponse   -- { 1 0 15961 127 14 }
        ::=                  {
    tagWriteResponse     {

         -- object 1 add response
         {
         objectId            { 1 0 15961 10 30 },
        completionCode       11
        },

         -- object 2 add response
        {
        objectId              { 1 0 15961 10 17 },
        completionCode        0
         }

    executionCode            0
    }
```

H.8 The transfer encoding for the example response

The transfer encoding in octets of the response value given above (after applying the basic encoding rules defined in this International Standard) is shown below in tabular form, with each row representing one of the lines of Annex H.7. The values of the Type identifiers, of lengths, and of the contents are shown in hexadecimal, two hexadecimal digits per octet.

Abstract Syntax		**Type**	**Length**	**Value**
AddMultipleObjectsResponse		06	05	28 FC 59 7F 0F
SEQUENCE		30	1D	
SEQUENCE	OF	30	18	
(TagWriteResponse)				
SEQUENCE		30	0A	
ObjectId		06	05	28 FC 59 0A 1E
CompletionCode		02	01	0B
SEQUENCE		30	0A	
ObjectId		06	05	28 FC 59 0A 11
CompletionCode		02	01	00
ExecutionCode		02	01	00

The complete transfer encoding of the response as an octet stream is:

06 05 28 FC 59 7F 0F 30 1D 30 18 30 0A 06 05 28 FC 59 0A 1E 02 01 0B 30 0A 06 05 28 FC 59 0A 11 02 01 00 02 01 00

Annex I
(informative)

Guidance to implementers of development commands

When implementing development commands, the following rules should be followed to ensure non-interference with the processes defined in ISO/IEC 15962.

- Each command/response pair should have a unique name, which should differ from any of the unique names defined in 8.1.

- It should have a unique object identifier compliant with any of the appropriate rules of ISO/IEC 9834-1. An additional condition is that the root-OID should not be **{iso (1) standards (0) rfid-data-protocol (15961)}**. This ensures separate definitions of any future standardised commands and development commands.

- Irrespective of the number of preceding arcs, the final arc for commands should be **commandModules (m), moduleName (n)**; while the final arcs for responses should be **responseModules (m+1), moduleName (n)**. Thus, the command and response modules should have their penultimate arc value that differs by 1, but the final arc identifying the module should have the same value for each command/response pair.

- It should use the key words DEFINITIONS, BEGIN and END to be compliant with abstract syntax of this International Standard.

- It should contain a statement that the modules use "EXPLICIT TAGS", which indicates that all of the elements are ultimately encoded as UNIVERSAL TYPES.

- The module should only use elementNames associated with system information exactly as specified in 7.1.2 and its sub-clauses. These elementNames are: **accessMethod, applicationFamily, applicationFamilyId, applicationSubFamily, dataFormat, storageFormat,** and **tagId**

- The module should only use elementNames associated with the data, that is part of the application system services, exactly as specified in 7.3 and its sub-clauses. These elementNames are: **compactParameter, object, objectId,** and **objectLock.**

- The module should only contain processing arguments that are defined in clause 9.2 of ISO/IEC 15962. This means that the only processing arguments that should be used are:

  afiLock (see this International Standard 8.4.2)
  avoidDuplicate (see 8.4.3)
  checkDuplicate (see 8.4.4)
  identifyMethod (see 8.4.5)
  lockStatus (see 8.4.7)
  maxAppLength (see 8.4.9)
  numberOfTags (see 8.4.10)
  storageFormatLock (see 8.4.16)

  If any of these elementNames are used in a development module, they should invoke exactly the same processing as is described in this International Standard and ISO/IEC 15962.

- Any of the other command-related element names specified in 8.4 may be used in a manner as described in this International Standard. In addition, new elementNames may be created that provide construction to the modular syntax, but should not imply any new processing by the ISO/IEC 15962 data protocol processor, or the Tag Driver, or any new air interface commands.

- Completion codes used in the response should be selected from 8.2 of this International Standard.

- Execution codes used in the response should be selected from 8.3 of this International Standard.

ICS 35.040

Price based on 82 pages