DOF Object Model Types

DOF Protocol Specification

Technical Steering Committee

DOF Object Model Types: DOF Protocol Specification

Technical Steering Committee

Document Information

Document Tracking Number	dof-2010-2
Document Version	7.0.1, 12 January 2018
Build Version	7.0.1.8
Source Information	2018-01-12, "." at commit c4a21c8bf741a32d4fe20776861d03baab74639a on branch "master" of repository core-specifications
External Reference	DOF Object Model Types, OpenDOF TSC, [dof-2010-2] (7.0.1, 12 January 2018)

This document is managed by the Technical Steering Committee of the OpenDOF Project, Inc.. Contact the committee using the following information.

Contact Information

Post Mail	OpenDOF Project, Inc. Technical Steering Committee 3855 SW 153rd Drive Beaverton, OR 97003
Telephone	1-503-619-4114
Fax	1-503-644-6708
Email	tsc@member.opendof.org [mailto:tsc@member.opendof.org]

Copyright

Copyright © 2008-2018 Panasonic Corporation, Bryant Eastham

Patents

Contributors to the OpenDOF Project make a non-enforcement pledge related to any patent required to implement the protocols described in this document. For more information see https://opendof.org/pledge.

License

See https://opendof.org/license-specifications.

Table of Contents

1. About This Document 1
    1.1. Audience 1
    1.2. Authority 1
    1.3. PDU Definitions 1
        1.3.1. Context 1
        1.3.2. Generalization 2
        1.3.3. Qualifications 2
        1.3.4. Fields 4
    1.4. Specifications, Notes and Warnings 5
    1.5. Printing This Document 6
    1.6. Comparing Documents 6
        1.6.1. Text Comparison 6
        1.6.2. Graphical Comparison 6
2. Introduction 9
3. Type and Value Formats 11
    3.1. Integer Types 11
        3.1.1. Unsigned 8-bit Integer 11
        3.1.2. Unsigned 16-bit Integer 11
        3.1.3. Unsigned 32-bit Integer 12
        3.1.4. Unsigned 64-bit Integer 12
        3.1.5. Signed 8-bit Integer 12
        3.1.6. Signed 16-bit Integer 13
        3.1.7. Signed 32-bit Integer 13
        3.1.8. Signed 64-bit Integer 13
    3.2. Floating-point Types 14
        3.2.1. 32-bit Floating Point 14
        3.2.2. 64-bit Floating Point 14
    3.3. String Type 15
        3.3.1. General Encoded String 16
    3.4. Date/Time Type 16
    3.5. Boolean Type 17
    3.6. DOF IID Type 17
    3.7. DOF OID Type 17
    3.8. UUID Type 18
    3.9. BLOB Type 18
        3.9.1. BLOB Type Formatting 19
    3.10. Array Types 19
        3.10.1. Array Type Formatting 20
    3.11. Reference Type 21
    3.12. Nullable Type 21
    3.13. Structure Type 22
4. External PDU Reference 25
    4.1. DOF Common Types, OpenDOF TSC, [dof-2009-1] (7.0.1, 12 January 2018) 25

Specifications and PDUs

Integer Wire Format `dof-2010-2-pdu-1` 11
Unsigned 8-bit Integer Type Format `dof-2010-2-pdu-2` 11
Unsigned 16-bit Integer Type Format `dof-2010-2-pdu-3` 12
Unsigned 32-bit Integer Type Format `dof-2010-2-pdu-4` 12
Unsigned 64-bit Integer Type Format `dof-2010-2-pdu-5` 12
Signed 8-bit Integer Type Format `dof-2010-2-pdu-6` 13
Signed 16-bit Integer Type Format `dof-2010-2-pdu-7` 13
Signed 32-bit Integer Type Format `dof-2010-2-pdu-8` 13
Signed 64-bit Integer Type Format `dof-2010-2-pdu-9` 14
Floating Point Wire Format `dof-2010-2-pdu-10` 14
32-bit Floating Point Type Format `dof-2010-2-pdu-11` 14
64-bit Floating Point Type Format `dof-2010-2-pdu-12` 15
String Wire Format `dof-2010-2-pdu-13` 15
General Encoded String Type Format `dof-2010-2-pdu-14` 16
Date/Time Type Format `dof-2010-2-pdu-15` 16
Boolean Type Format `dof-2010-2-pdu-16` 17
IID Type Format `dof-2010-2-pdu-17` 17
OID Type Format `dof-2010-2-pdu-18` 17
UUID Type Format `dof-2010-2-pdu-19` 18
BLOB Wire Format `dof-2010-2-pdu-20` 18
BLOB Type Format `dof-2010-2-pdu-21` 19
Array Wire Format `dof-2010-2-pdu-22` 20
Array Type Format `dof-2010-2-pdu-23` 20
Reference Type Format `dof-2010-2-pdu-24` 21
Nullable Wire Format `dof-2010-2-pdu-25` 21
Nullable Type Format `dof-2010-2-pdu-26` 22
Integer Wire Format `dof-2010-2-pdu-27` 22
Nullable Type Format `dof-2010-2-pdu-28` 22
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1` 25
Compressed Unsigned 24-Bit `dof-2009-1-pdu-2` 25
Compressed Unsigned 32-Bit `dof-2009-1-pdu-3` 26
Object Identifier (OID) `dof-2009-1-pdu-4` 26
Object Identifier Attribute `dof-2009-1-pdu-5` 27
Interface Identifier (IID) `dof-2009-1-pdu-21` 27
Type Identifier `dof-2009-1-pdu-23` 28
Type Reference `dof-2009-1-pdu-40` 28

1. About This Document

This document contains reference information for DOF Object Model Type formats. DOF data types are extensible, with its base defined in the *DOF Common Types* specification.

Each Type and data format is defined by a Type Identifier, which is registered with the Technical Steering Committee of the OpenDOF Project, Inc. (ODP-TSC) before use. This document contains many of these defined formats, but not all of them. The registry of Type Identifier assignments (found at https://opendof.org/registry-types) contain links to other defining documents.

1.1. Audience

The primary audience of this document is people implementing libraries that use DOF protocols. Others that are concerned about how DOF protocols work on a network at a very low level can also benefit from this document.

Readers should be familiar with technical protocol documentation. This document is similar to an 'RFC' for DOF protocols, and familiarity with the language used in these types of documents is helpful.

This document is *not* required reading for those who need to use existing DOF libraries, although system designers may benefit from an understanding of this information. In particular, the information related to security and connections can help system designers to better understand how DOF systems work.

1.2. Authority

This document, in its English form, is the authoritative document for DOF protocols. Due to the detailed nature of protocol design, it may be difficult for a reader to implement these protocols correctly based on a translated document. When questions arise, the English version is authoritative.

All implementations that claim to be DOF compliant must satisfy the requirements set forth in this document.

This document is managed by the Technical Steering Committee of the OpenDOF Project, Inc., referred to as 'ODP-TSC'. The inside cover of this document contains contact information for the Technical Steering Committee.

The web site for the Technical Steering Committee is https://opendof.org/tsc.

1.3. PDU Definitions

This document deals with the transmission of information on a network. In order to describe this, diagrams show different fields, their network order, and positioning.

All DOF protocol documentation uses a similar diagramming style, discussed in this section. The word 'PDU', which stands for 'Protocol Data Unit', indicates these diagrams.

1.3.1. Context

Each PDU represents information sent on the network in some context. This usually means that there is information sent 'surrounding' the PDU itself. This information provides the 'context' for the PDU.

For example, the context for the DOF Protocol Stack itself is a network transport. This could be UDP/IP, TCP/IP, or any other transport. The transport will require certain headers and trailers, but they are not shown in each DOF Protocol Stack PDU. In this example the transport provide the context of the DOF Protocol Stack PDU.

In a similar way, the DOF Protocol Stack defines the context for the other DOF protocols. The Object Access Protocol, for example, can use the DOF Protocol Stack for its context. Just as the transport headers are not shown for the DOF Protocol Stack, the DOF Protocol Stack headers are not shown for the Object Access Protocol.

Understanding the context requires understanding the relative position of the different layers in the protocol stack. Each PDU defines the ordering of all of its parts, and may indicate that it is itself an instance of some other general PDU format. This means that it is always possible to start at a high-level PDU definition and understand it completely, but not always possible to identify everywhere that a particular PDU may be used (as a part of some other higher-level PDU).

Each protocol specification defines its specific contexts. The context definition includes a description of the underlying stack layer (or the containing layer), and defines the information that must be available from that layer (for reception) or passed to that layer (for transmission).

As an example, the context of the DOF Network Protocol is a transport. The transport has a requirement to provide the addresses of nodes, and so part of the context for the transport is the sender address (for reception) or the target address (for transmission). Any transport that will work with the DOF Protocol Stack must meet the context requirements.

1.3.2. Generalization

Many times a particular PDU will be a specific type of a more general PDU. In these cases, the PDU will indicate that it is an 'instance' of the more general type.

1.3.3. Qualifications

Each PDU will have qualifications placed on its use. The general qualifications include security and transport requirements. The following terms identify these qualifications. Others may be present as well.

1.3.3.1. Transport Qualifications

There are three major transport categories for DOF protocols (see the *DOF Protocol Specification*, Transport Requirements). These are None, Lossy, and Lossless. There are also three addressing types: Unicast, Multicast, and Broadcast. PDUs indicate the ability to use each of these categories as follows:

```
Session: Combinations of None, Lossy (2-node, n-node), or Lossless.
Addressing: Combinations of Unicast, Multicast, or Broadcast.
```

1.3.3.2. Security Qualifications

Each PDU categorizes security qualifications according to the major aspects of security (see the *DOF Security Specification*, Overview). These are Encryption, Data Integrity, Authentication, and Access Control. Encryption prevents viewing of the PDU, Data Integrity ensures that others do not modify the PDU, Authentication indicates knowledge of the nodes involved in the communication, and Access Control restricts those able to use the PDU.

Data Integrity is required according to the DOF Protocol Stack specification for all secure packets. Authentication is required whenever permissions are involved, or when it is important to know something

about the relationship between the sender and receiver. Encryption is required if the contents of the packet contain information that must not be visible to potential attackers.

In addition, there is the case of unsecured communication. PDUs that do not require security will list 'Unsecured' as a possible choice. Finally, certain PDUs are required to be Unsecured, and are identified as such.

Each application PDU must specify its security requirements. If Access Control is required, the PDU will indicate the permissions required for the command and for any response. However, it is typical for all of the PDUs of a given protocol to share the same requirements, and so the protocol specification may define its security qualifications in a single section. In this case, the individual PDUs should refer back to this section to avoid confusion.

It is also typical for applications to require the security of the session. A PDU indicates this as 'session security', and is its own security qualification. If the PDU allows the 'None' session type then this implies that Unsecured is allowed, as session type 'None' cannot be secure.

For PDUs that refer to session security or a common security section, one of the following formats is used:

```
Security: See the section 'Security Qualifications'.
```

or

```
Security: Session.
```

The PDU uses the following identifications to specify the security qualifications for a PDU:

```
Unsecured: (description).
Encryption: (description).
Message authentication: (description).

Permission (command): (description).
Permission (response): (description).
```

For each of these, the (description) varies. There are many different combinations of values, as described here.

Unsecured can have the values 'Required', 'Allowed', 'Not Allowed'.

Encryption can have the values 'Required', 'Allowed', 'Not Allowed'.

Message authentication can have the values 'Required', 'Allowed', 'Not Allowed'.

The specific PDU determines the permissions required (for both command and response).

There are combinations that are not valid. The following table defines all allowed combinations if a combination does not appear in the table then it is invalid.

In general, security theory links Encryption and Data Integrity. If one is 'Required' then the other must at least be 'Allowed'. Encryption and Data Integrity are the opposite of Unsecured. If either is 'Required' then unsecured must be 'Not Allowed'. Again, note that since data integrity is a common requirement for all secure packets the PDUs do not list it separately.

Unsecured / Encryption / Message Authentication	**Description**
Required / Not Allowed / Not Allowed	Indicates the PDU must be unsecured.
Allowed / Allowed / Allowed	Indicates there are no security requirements.

Unsecured / Encryption / Message Authentication	Description
Not Allowed / Allowed / Allowed	Indicates the PDU must be either encrypted or authenticated. This is the lowest level of security possible for a PDU.
Not Allowed / Required / Allowed	Indicates that encryption is required, and data integrity is optional.
Not Allowed / Allowed / Required	Indicates that encrypted is optional, but authentication is required.
Not Allowed / Required / Required	Indicates both encryption and authentication are required.

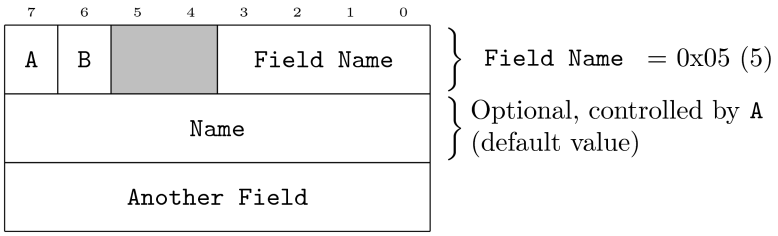
1.3.4. Fields

Each PDU shows a container, represented as a sequence of fields. The table fully describes each field.

The following is an example of a PDU description.

⇄ *Example PDU*`example-pdu-1`

PDU Context



`A`	**One bit.** Controls the presence of `Name`.
`B`	**One bit.** Field description.
`Field Name`	**Type and size description.** Field description.
`Name`	**Type and size description. Optional, controlled by** `A`. Field description.
`Another Field`	**Type and size description.** Field description.

The container represents the PDU. The diagram shows Fields first to last, left to right and top to bottom. This means that the first fields appear on the wire before the last fields. This is a general principle: things on the top and toward the left in the diagrams appear on the wire before things on the right and bottom. Note that the transport is free to use whatever ordering it must, and memory layout may different from the PDU definition. However, the transfer of a PDU from a program, over an arbitrary transport and into another program must maintain the perceived ordering between the two programs. PDU diagrams are always shown MSB first (most significant byte) and msb first (most significant bit) unless otherwise indicated.

Each field identifies its contents and provides information necessary to understand its use. In the example above, the diagram shows the first six fields of the container. Each field is named, with the exception of reserved fields which are indicated as being grayed out. Field names use `monospace text` similar to

that used in program listing. The third field shown is indicated as reserved. Reserved bits have special requirements described in the *DOF Protocol Specification*.

Several different field cases are typical, described in the following sections.

1.3.4.1. Typical Field

A typical field defines its name, its type, and its description. As shown, this field is not optional, and so it must appear in the position indicated by the diagram. The PDU uses `Field Name` throughout the related documentation to refer to the field. The type of the field is a reference to some other defined PDU. Otherwise, the PDU indicates the size of the field and any defined structure.

1.3.4.2. Optional Field

An optional field is similar to the typical case, but is indicated as optional in a note to the right. Each optional field will indicate what controls whether or not it is included as well as the default value that the field takes on when it is absent.

1.3.4.3. Fixed Values

Fields may have required values based on the specific PDU. This is common in the case of a PDU being an instance of another PDU, where the specific instance requires certain field values. In this case, it is not valid for the PDU to contain values other than that specified, if so the PDU is invalid.

The required value is shown to the right of the field as a comment.

1.3.4.4. Bit Fields

It is common for PDUs to leverage a single byte to store multiple fields (bit fields). The packing of multiple fields into a single byte is indicated as shown in the earlier example. This packing of bits may also leave space that can be indicated as reserved.

The PDU indicates reserved bits visually although the PDU does not individual name these fields.

Note that bit fields may specify default values just as other fields. However, single bit fields do not show their hexadecimal representation. In addition, bits without names that have required values may indicate their values in the place of the name.

1.4. Specifications, Notes and Warnings

This is a technical specification document. It is critical to understand how the document indicates specification items. There are also annotations for notes (implementation notes or other information that is worthy of extra notice) and warnings (pitfalls or extremely critical information). In addition, there are implementation notes, which point out optimizations that are not obvious.

Each type of callout ends either when another callout begins or when the text goes back to the normal indentation.



This is a note. The text is inset and there is a header to indicate the beginning of the text.



This is an implementation note. The text is inset and there is a header to indicate the beginning of the text.



This is a warning. The text is bold inset and there is a different header to indicate the beginning of the text.



This is an example`example-spec-1`

This is the text.

Following the summary is descriptive information about the specification item. Each specification item has a unique number assigned.

1.5. Printing This Document

The format of this document is for two-sided printing on US Letter paper.

1.6. Comparing Documents

The easiest way to compare versions of this document is to compare the corresponding source. The source format is AsciiDoctor, which is a markup-style text format. This format can be compared using any text comparison tool, including those that work with source control documents. To facilitate determining the source for each document there is information about the commit, branch, and repository along with the document version information. In addition the following guidelines are followed to make things easy to compare:

1. The input files put each sentence on a separate line, with no hard line-wrapping.

2. To the extent possible, macros and templates are used to determine formatting and structure.

3. Graphics and diagrams also use text formats.

If all you have available is a compiled document (PDF or HTML) then there are tools available to compare them. This section summarizes some of these methods with their pros and cons.

1.6.1. Text Comparison

There are several free tools that will do comparisons of the text of a PDF. One of these is the xdocdiff plugin for WinMerge. The plugin extracts the text from the PDF (including some information like page numbers), and WinMerge then compares the text.

WinMerge is available from http://winmerge.org. The xdocdiff plugin is available from http://freemind.s57.xrea.com/xdocdiffPlugin/en/index.html.

1.6.2. Graphical Comparison

In order to compare formatting, or to see the changes in the context of the document, a graphical compare is required. There are several free tools, but each has limitations.

1. diff-pdf, available from https://github.com/vslavik/diff-pdf, shows an overlay of the matching pages in different colors.

2. DiffPDF, available fromhttp://www.qtrac.eu/diffpdf.html, shows side-by-side red-lined differences of matching pages.

Both of these tools suffer from their page-by-page comparison methods. This means that when context shifts because of additions or removals that they start sensing changes at each page boundary. DiffPDF has the ability to select comparison ranges, allowing the program to synchronize at the beginning of each section, but this requires manual configuration and can be difficult for large documents.

There are also commercial tools that do an excellent job at comparing PDF documents. The first is Adobe Acrobat (http://www.adobe.com/products/acrobat.html), which in later releases has a 'Compare Document' feature. The result of the comparison is an annotated PDF, and hovering the mouse over the change will show the details.

Another commercial tool is PDF Content Comparer (http://www.inetsoftware.de/products/pdf-content-comparer) from i-net Software. This tool does side-by-side comparison, but has an intuitive auto-synchronize that keeps similar content lined up even when additions or deletions have occurred.

2. Introduction

Each Type Identifier is required to define its format and the format of the values it defines. The type format extends a standard PDU defined in *DOF Common Types*.

This document contains many of these additional type and value definitions. Each section presents a type, the structure of its data, and the structure of the data values that it defines.

All use of these formats must adhere to the requirements specified for the type. Failure to meet the requirements is a serious error that can affect interoperability of DOF systems.



The types described in this document are independent of protocol versioning. They are common among all DOF protocol versions, and are fixed. They can be added to over time, and protocols that use those new types can be negotiated if necessary according to the DOF Protocol Stack specification.

Throughout this document are references to "ODP-TSC" which is the Technical Steering Committee of the OpenDOF Project, Inc. Contact information for the ODP-TSC is at the beginning of this document (inside cover page).

3. Type and Value Formats

The following sections describe the specifications for many types supported by DOF implementations. Note that it is not possible for a receiver to extract data for types that it does not understand. This is acceptable in most cases, because the implementer would program the receiver to understand a particular interface definition including the types that the interface requires. There are exceptions to this, typically for general user applications that can discover and interact with many interfaces including those it has not previously seen. In this case, the applications should be updated whenever new types are added, or it cannot know how to use those types.

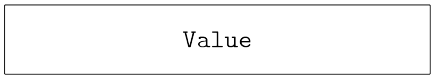
3.1. Integer Types

DOF supports signed and unsigned integers with lengths from 1 byte up to 8 bytes.

The rules for different sizes of integer types are all very similar. This document combines them, with only a mention of the differences. In addition, the section combines the specifications for unsigned vs. signed types here with only a note of the differences.

The main difference is the value format on the wire. In each case, the value on the wire contains the same number of bits as the type declares. The values are MSB first. The signed types are 2's complement.

⇄ **Integer Wire Format** `dof-2010-2-pdu-1`



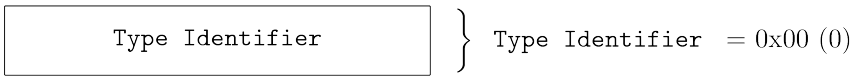
`Value` **Length determined by type.** The integer value. Signed types are 2's complement. The size on the wire corresponds to the type size exactly.

3.1.1. Unsigned 8-bit Integer

The value is one byte using Integer Wire Format.

⇄ **Unsigned 8-bit Integer Type Format** `dof-2010-2-pdu-2`

Instance of Type Identifier`dof-2009-1-pdu-23`



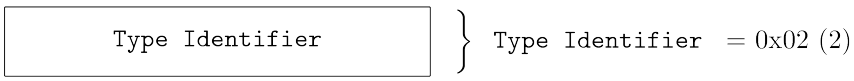
`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

3.1.2. Unsigned 16-bit Integer

The value is two bytes using Integer Wire Format.

⇄ **Unsigned 16-bit Integer Type Format** `dof-2010-2-pdu-3`

Instance of Type Identifier`dof-2009-1-pdu-23`



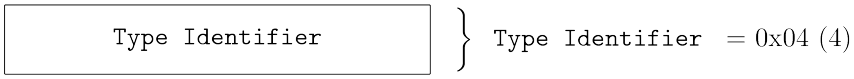
`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

3.1.3. Unsigned 32-bit Integer

The value is four bytes using Integer Wire Format.

⇄ **Unsigned 32-bit Integer Type Format** `dof-2010-2-pdu-4`

Instance of Type Identifier`dof-2009-1-pdu-23`



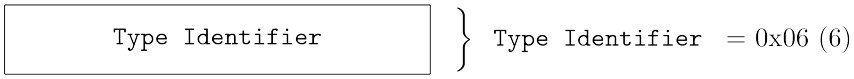
`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

3.1.4. Unsigned 64-bit Integer

The value is eight bytes using Integer Wire Format.

⇄ **Unsigned 64-bit Integer Type Format** `dof-2010-2-pdu-5`

Instance of Type Identifier`dof-2009-1-pdu-23`



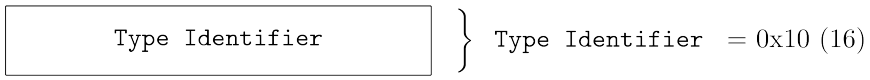
`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

3.1.5. Signed 8-bit Integer

The value is one byte using Integer Wire Format.

Signed 8-bit Integer Type Format `dof-2010-2-pdu-6`

Instance of Type Identifier`dof-2009-1-pdu-23`



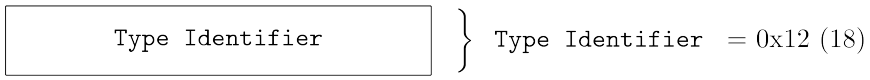
`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

3.1.6. Signed 16-bit Integer

The value is two bytes using Integer Wire Format.

Signed 16-bit Integer Type Format `dof-2010-2-pdu-7`

Instance of Type Identifier`dof-2009-1-pdu-23`



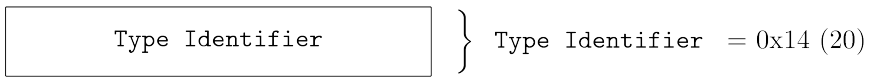
`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

3.1.7. Signed 32-bit Integer

The value is four bytes using Integer Wire Format.

Signed 32-bit Integer Type Format `dof-2010-2-pdu-8`

Instance of Type Identifier`dof-2009-1-pdu-23`



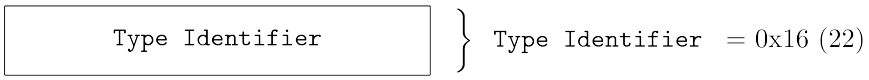
`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

3.1.8. Signed 64-bit Integer

The value is eight bytes using Integer Wire Format.

⇄ **Signed 64-bit Integer Type Format** `dof-2010-2-pdu-9`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

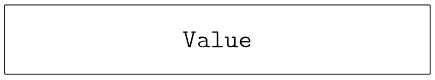
3.2. Floating-point Types

There are two types of floating-point numbers supported: 32-bit and 64-bit.

The rules for different sizes of floating point types are all very similar. This document combines them with only a mention of the differences.

The main difference is the value format on the wire. In each case, the value on the wire contains the same number of bits as the type declares. The values use IEEE 754 formatting.

⇄ **Floating Point Wire Format** `dof-2010-2-pdu-10`



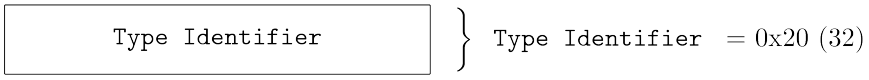
`Value` **Length determined by type.** The floating point value. Values use IEEE 754 formatting. The size on the wire corresponds to the type size exactly.

3.2.1. 32-bit Floating Point

The value is four bytes using Floating Point Wire Format.

⇄ **32-bit Floating Point Type Format** `dof-2010-2-pdu-11`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

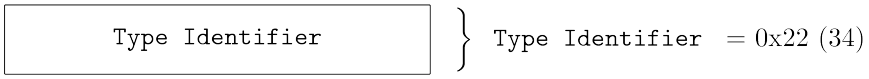
3.2.2. 64-bit Floating Point

The value is eight bytes using Floating Point Wire Format.



64-bit Floating Point Type Format `dof-2010-2-pdu-12`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

3.3. String Type

DOF includes support for strings with defined encodings. Strings are variable-length, with a maximum byte length of 2^{22} bytes (not characters). This length information is included in the data. The type definition defines the maximum number of *characters*, while on the wire it is the number of *bytes*.



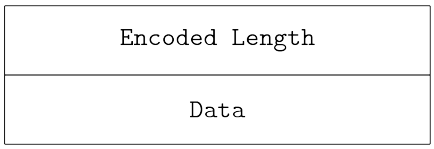
Except for very special encodings, there is only a weak relationship between the maximum character length in the specification and the maximum byte length on the wire. For example, UTF-8 can take between 1 and 6 bytes per character. A UTF-8 string with a maximum character length of 10 could take up to 60 bytes on the wire.

String types specify the encoding used in their definitions. In all cases, the encoding values uses the IANA assigned MIBenum values found at http://www.iana.org/assignments/character-sets. Use the preferred MIME name in all cases. Values less than three and greater than or equal to 3000 are reserved.

Localization and globalization of DOF applications is beyond the scope of this document. However, the design comes from the guidelines of RFC2277 and RFC2130. These RFCs provide guidelines for protocol design and best practices for the Internet community. The string type provided in DOF follow these guidelines by allowing multiple encodings.



String Wire Format `dof-2010-2-pdu-13`



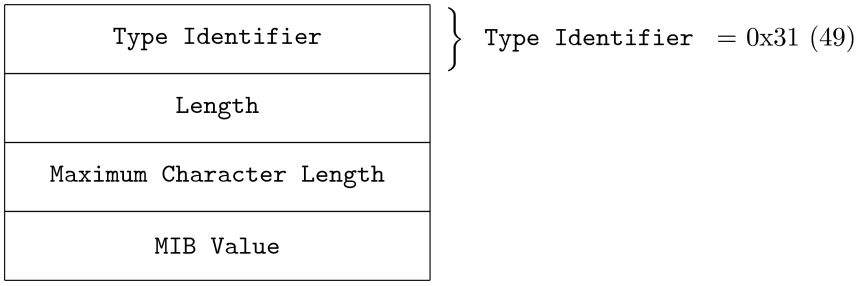
`Encoded Length` **Instance of** Compressed Unsigned 24-Bit`dof-2009-1-pdu-2`.Controls the length of `Data`.

`Data` **Variable-length bytes. Optional, controlled by** `Encoded Length`. This field contains the encoding of the string. If the encoding is not specified by the type then the Encoding field is used to define the encoding used.

3.3.1. General Encoded String

General Encoded String Type Format `dof-2010-2-pdu-14`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

`Length` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The size of the remaining fields in this PDU.

`Maximum Character Length` **Instance of** Compressed Unsigned 24-Bit`dof-2009-1-pdu-2`.The maximum character length of the string. This value may be zero, although this defines a degenerate type.

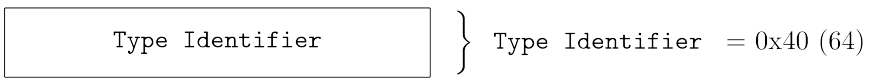
`MIB Value` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The MIB enum number from http://www.iana.org/assignments/character-sets. Values that are marked as reserved in the IANA reference must not be used.

3.4. Date/Time Type

There is a single type for passing date and time information. The format uses milliseconds since Midnight, January 1, 1970 UTC. This is the same format used in the Java VM.

Date/Time Type Format `dof-2010-2-pdu-15`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

The value is passed as eight bytes using Integer Wire Format. The contents are milliseconds since Midnight, January 1, 1970 UTC.

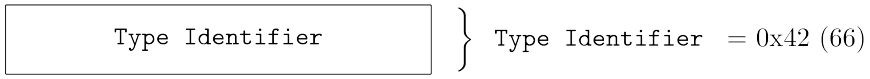
3.5. Boolean Type

Boolean values are defined similar to the C Programming Language: a value of zero (0) is False, and any other value is interchangeably defined as True. The actual value passed may be modified on the wire and only the True and False values are guaranteed.



Boolean Type Format `dof-2010-2-pdu-16`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

The value is passed as one byte using Integer Wire Format. The value of zero is defined as False, any other value as True.

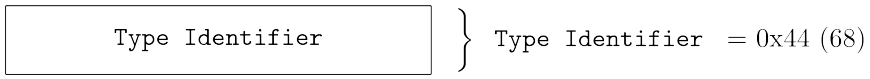
3.6. DOF IID Type

Interface Identifiers can be passed as values.



IID Type Format `dof-2010-2-pdu-17`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

The value is passed using Interface Identifier (IID).

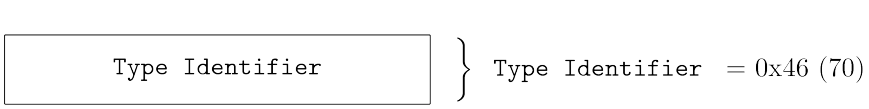
3.7. DOF OID Type

Object Identifiers can be passed as values.



OID Type Format `dof-2010-2-pdu-18`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

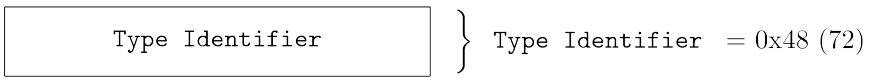
The value is passed using Object Identifier (OID).

3.8. UUID Type

UUIDs, or Universally Unique Identifiers can be used as values.

UUID Type Format `dof-2010-2-pdu-19`

Instance of Type Identifier`dof-2009-1-pdu-23`



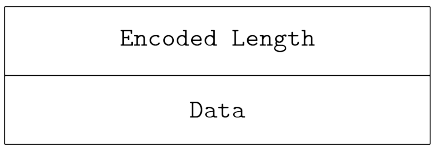
`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

The value is MSB first. See RFC 4122 for more information. Note that the rule for MSB first includes all fields. This is *not* the same as the typical UUID storage format (which is dependent on machine endianness).

3.9. BLOB Type

BLOB stands for Binary Large Object. DOF supports BLOBs for situations where a single entity (like an image) needs to be transmitted. Of course these can also be modeled as arrays of unsigned bytes, but that may confuse the semantics of the object.

BLOB Wire Format `dof-2010-2-pdu-20`



`Encoded Length` **Instance of** Compressed Unsigned 24-Bit`dof-2009-1-pdu-2`.Controls the length of `Data`.

`Data` **Variable-length bytes. Optional, controlled by** `Encoded Length`. This contains the data associated with the BLOB. The data is passed

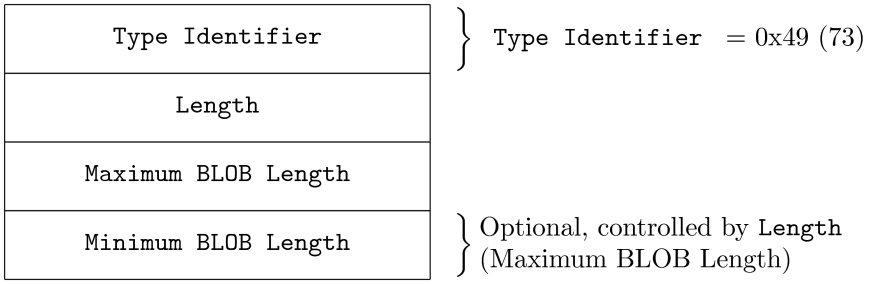
in whatever format is agreed upon, and are independent of source and destination characteristics (like endianess).

3.9.1. BLOB Type Formatting



BLOB Type Format `dof-2010-2-pdu-21`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

`Length` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The size of the remaining fields in this PDU.

`Maximum BLOB Length` **Instance of** Compressed Unsigned 24-Bit`dof-2009-1-pdu-2`.The maximum length of the BLOB. This value may be zero, although this defines a degenerate type.

`Minimum BLOB Length` **Instance of** Compressed Unsigned 24-Bit`dof-2009-1-pdu-2`.**Optional, controlled by** `Length`. The minimum length of the BLOB. If present this value must be less than or equal to `Maximum BLOB Length`.

3.10. Array Types

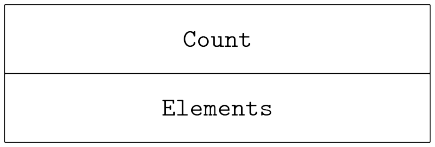
Both variable- and fixed-length arrays are supported in DOF. Arrays are simply lists where each element type is identical, and where the minimum and maximum lengths of the list can be specified as part of the type.

Variable-length arrays include both a maximum and a minimum length. This can be useful when *at least* a certain number of elements are required, although the minimum length can be zero.

Arrays include a single nested type specifier. This means that all array elements have the same type.

The full set of type definitions, including structures and arrays, are allowed as the nested types. This means that arrays of arrays, etc. are all allowed.

Array Wire Format `dof-2010-2-pdu-22`



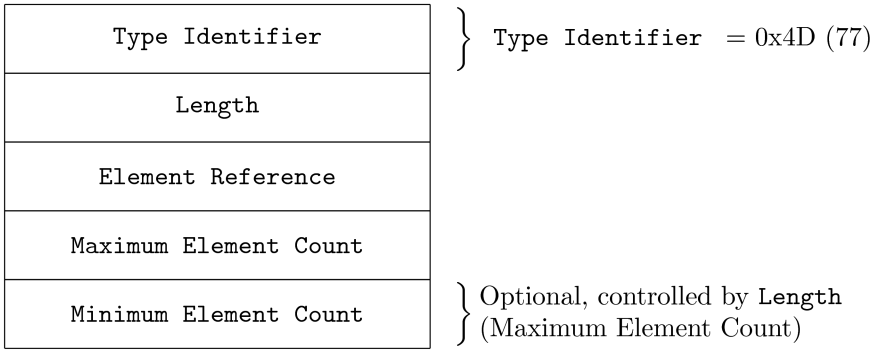
`Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Controls the number of `Elements`.

`Elements` **Sequential values. Optional, controlled by** `Count`. This contains a list of elements, each of which is formatted according to the data type of the array. Note that array types may be composed of other arrays or complex types.

3.10.1. Array Type Formatting

Array Type Format `dof-2010-2-pdu-23`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

`Length` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The size of the remaining fields in this PDU.

`Element Reference` **Instance of** Type Reference`dof-2009-1-pdu-40`.The reference to the type of the array elements.

`Maximum Element Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The maximum number of elements in the array. This value may be zero, although this defines a degenerate type.

`Minimum Element Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.**Optional, controlled by** `Length`. The minimum

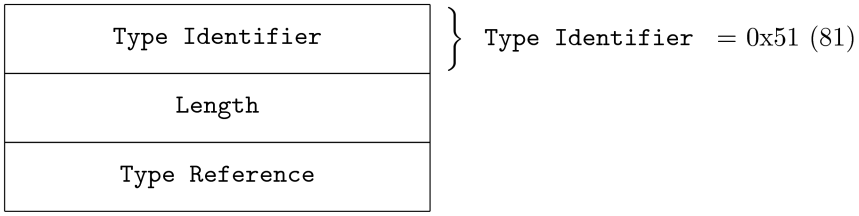
number of elements in the array. If present this value must be less than or equal to `Maximum Element Count`.

3.11. Reference Type

A reference type refers to a type in the type context by reference. It is important to note that the reference can be to any type in the context, either before or after the reference itself. However, it is impossible for a reference to refer to itself directly.

⇄ **Reference Type Format** `dof-2010-2-pdu-24`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

`Length` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The size of the remaining fields in this PDU.

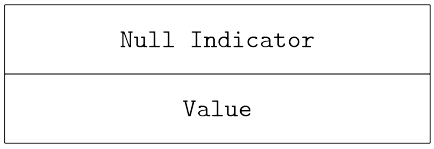
`Type Reference` **Instance of** Type Reference`dof-2009-1-pdu-40`.This identifies the referenced type in the context that is understood by its use. This usually means that the context is supplied by the application protocol.

3.12. Nullable Type

The nullable type allows for the use of a null value on the wire in addition to the base type of the nullable type. This is similar to the handling of objects in the Java and C# languages, which allow for values to include a NULL. In the DOF type system there is no use of the NULL except as allowed by this type – in other words a type must be declared to allow the use of NULL.

The nullable type has a single sub-type, which is the type that it must take if the value NULL is not used.

⇄ **Nullable Wire Format** `dof-2010-2-pdu-25`

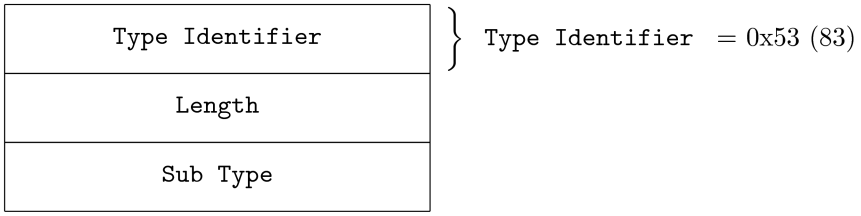


`Null Indicator` **One byte.** Controls the presence of `Value`, which is present if its value is not zero. In this case the value is not NULL. If the value of this field is zero then the overall value is NULL.

`Value` **Variable length. Optional, controlled by** `Null Indicator`. Contains the value. The length of this data is controlled by the `???` (from the type).

Nullable Type Format `dof-2010-2-pdu-26`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

`Length` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The size of the remaining fields in this PDU.

`Sub Type` **Instance of** Type Reference`dof-2009-1-pdu-40`.The reference to the type of the value when not null.

3.13. Structure Type

Structures are similar to fixed-length arrays, with the difference being that the elements can be of different types. In the same way, they are passed on the wire as a list of the corresponding component values.

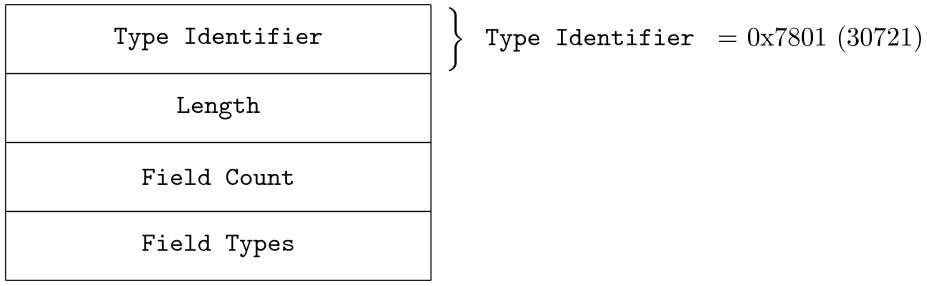
Integer Wire Format `dof-2010-2-pdu-27`



`Fields` **Variable length.** The count and types are controlled by `???` and `???` from the type definition. This contains a list of structure fields, each of which is formatted according to the data type of the field. Note that field types may be arrays or other complex types.

Nullable Type Format `dof-2010-2-pdu-28`

Instance of Type Identifier`dof-2009-1-pdu-23`



`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies each basic type, registered with ODP-TSC.

`Length` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The size of the remaining fields in this PDU.

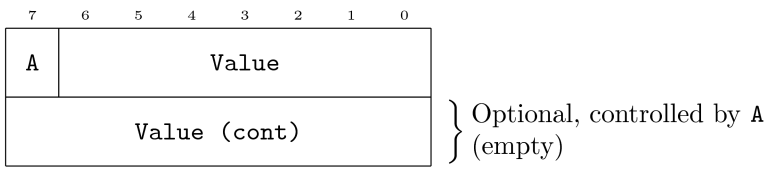
`Field Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The number of fields in the structure. This value may be zero, although this defines a degenerate type.

`Field Types` **Array of** Type Reference`dof-2009-1-pdu-40`.A reference to the type each field.

4. External PDU Reference

4.1. DOF Common Types, OpenDOF TSC, [dof-2009-1] (7.0.1, 12 January 2018)

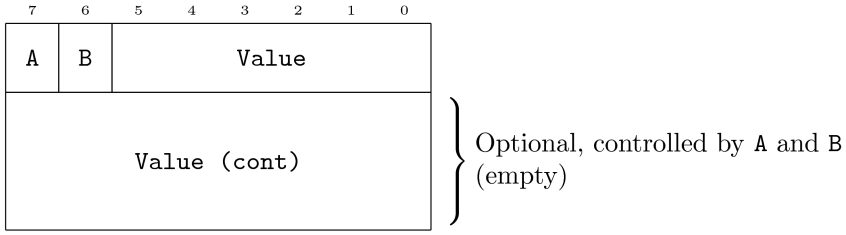
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1`



`A` **One bit.** Controls the size of `Value`.

`Value` **Either seven or fifteen bits, MSB, controlled by A.** If the `A` field is zero then the `Value` is contained in a single byte. If the `A` field is one then the `Value` is passed in fifteen bits with the least significant bits passed in a second byte.

Compressed Unsigned 24-Bit `dof-2009-1-pdu-2`

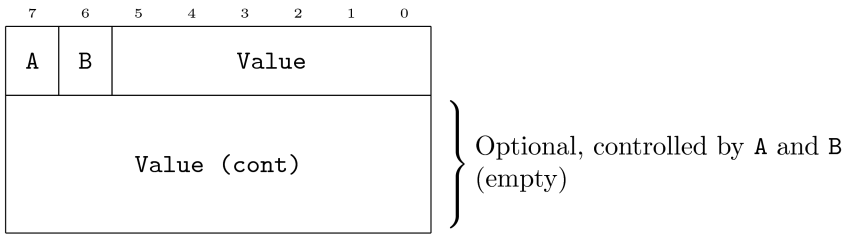


`A` **One bit.** Controls the size of `Value` and the presence of the `B` field.

`B` **One bit.** Controls the size of `Value` or becomes part of the value.

`Value` **Either seven, fourteen, or twenty-two bits, MSB, controlled by A and B.** If `A` is zero the `Value` is seven bits and one byte is used. In this case, the `B` field is part of the `Value`. If `A` is one then the `B` field determines the size of the `Value` rather than being part of it. If `B` is zero then fourteen bits (two bytes) are used, and if `B` is one then twenty-two bits (three bytes) are used.

Compressed Unsigned 32-Bit `dof-2009-1-pdu-3`

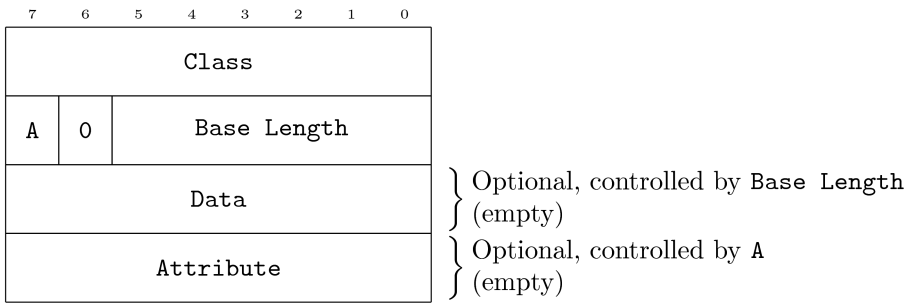


`A` **One bit.** Controls the size of `Value` and the presence of the `B` field.

`B` **One bit.** Controls the size of `Value` or becomes part of the value.

`Value` **Either seven, fourteen, or thirty bits, MSB, controlled by A and B.** If `A` is zero the `Value` is seven bits and one byte is used. In this case, the `B` field is part of the `Value`. If `A` is one then the `B` field determines the size of the `Value` rather than being part of it. If `B` is zero then fourteen bits (two bytes) are used, and if `B` is one then thirty bits (four bytes) are used.

Object Identifier (OID) `dof-2009-1-pdu-4`



`Class` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.The OID class, registered with ODP-TSC.

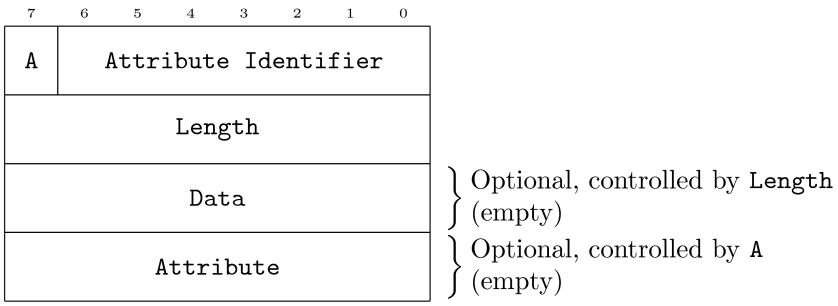
`A` **One bit.** Controls the presence of `Attribute`.

`Base Length` **Six bits.** This contains the length of the `Data`.

`Data` **Variable bytes.** The data associated with the OID. `Base Length` bytes.

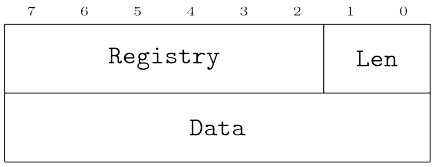
`Attribute` **Instance of** Object Identifier Attribute`dof-2009-1-pdu-5`.**Optional, controlled by** `A`. If `A` is one then this field is present. Attributes form an ordered list, with each attribute containing a flag controlling the presence of the next attribute.

⇄ Object Identifier Attribute `dof-2009-1-pdu-5`



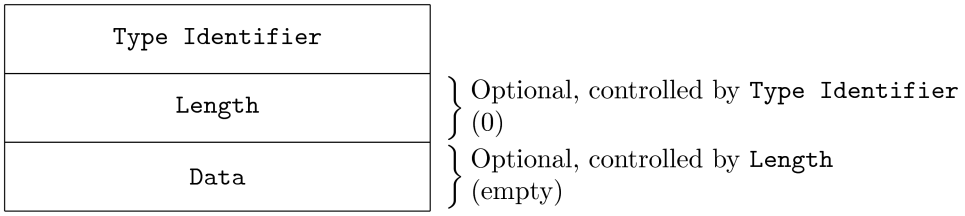
`A`	**One bit.** Controls the presence of `Attribute`.
`Attribute Identifier`	**Seven bits.** The attribute type, registered with ODP-TSC.
`Length`	**One byte.** This field contains the length of the `Data`.
`Data`	**Variable bytes.** The data associated with the attribute. `Length` bytes.
`Attribute`	**Instance of** Object Identifier Attribute`dof-2009-1-pdu-5`.**Optional, controlled by** `A`. If `A` is one then this field is present. Attributes form an ordered list, with each attribute containing a flag controlling the presence of the next attribute.

⇄ Interface Identifier (IID) `dof-2009-1-pdu-21`



`Registry`	**Six bits.** The registry of the IID, registered with the ODP-TSC.
`Len`	**Two bits.** Defines the length of the `Data`, which is $2^{(LEN-1)}$. `Len` equal to zero (0) is invalid.
`Data`	**Variable length, controlled by `Len`.** The identifier for the IID within the `Registry`.

Type Identifier `dof-2009-1-pdu-23`



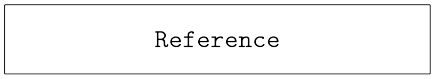
`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Controls the presence of `Length`. If this value is odd then the `Length` field is present. This identifies each basic type, registered with ODP-TSC.

`Length` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Controls the presence of `Data`.

`Data` **Variable length. Optional, controlled by** `Length`. The type data depends on the `'name': 'Type Identifier'}`. This data can provide additional information related to the type.

Type Reference `dof-2009-1-pdu-40`

Instance of Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.



`Reference` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This identifies the referenced type by index in the associated type context.