DOF Presentation Protocol (V2)

DOF Protocol Specification

Technical Steering Committee

DOF Presentation Protocol (V2): DOF Protocol Specification

Technical Steering Committee

Document Information

Document Tracking Number	dof-2009-12
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 Presentation Protocol (V2), OpenDOF TSC, [dof-2009-12] (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. Specification 9
    2.1. Sealing and Unsealing PDUs 13
    2.2. Operation Identifiers 14
    2.3. Loop Detection and Prevention 15
    2.4. Operation Lifecycle Management 16
        2.4.1. Operation State: Initial and Retry 19
        2.4.2. Operation State: Lost 19
        2.4.3. Operation State: New Source 19
        2.4.4. Operation State: Further Source 19
        2.4.5. Operation State: Reuse 19
        2.4.6. Operation State: Closer Source 20
        2.4.7. Operation State: Duplicate 20
    2.5. Operation Management 20
    2.6. Common DPP Capabilities 20
        2.6.1. SID Security 21
        2.6.2. IAM 21
        2.6.3. ACTAS 22
        2.6.4. Tunnel Domain 23
        2.6.5. Commands for Common Capabilities 24
        2.6.6. Cancel All 24
        2.6.7. Node Down 25
        2.6.8. Operation Source Lost and Source Found 25
        2.6.9. Operation Rename 27
        2.6.10. Ping and Ping Response 27
        2.6.11. SID Query and SID Query Response 28
        2.6.12. Heartbeat 29
3. External PDU Reference 31
    3.1. DOF Common Types, OpenDOF TSC, [dof-2009-1] (7.0.1, 12 January 2018) 31
    3.2. DOF Protocol Specification, OpenDOF TSC, [dof-2008-1] (7.0.1, 12 January 2018) 33
    3.3. DOF Security Specification, OpenDOF TSC, [dof-2008-16] (7.0.1, 12 January 2018) 35

Specifications and PDUs

DPPv2 Command Header `dof-2009-12-pdu-1` 9
DPPv2 Response Header `dof-2009-12-pdu-2` 10
DPPv2 Security Header `dof-2009-12-pdu-3` 12
DPPv2 Trailer `dof-2009-12-pdu-4` 13
DPP must check valid use of cached SIDs. `dof-2009-12-spec-1` 14
DPP datagrams with a **Sequence** of zero (0) must use the **SID** of the sending node. `dof-2009-12-spec-2` 15
The sequence of a forwarded datagram with a global operation identifier must use a greater sequence than when received. `dof-2009-12-spec-3` 15
Correctly manage operations that use local and global operation identifiers during forwarding. `dof-2009-12-spec-4` 15
Correctly initiate global operations. `dof-2009-12-spec-5` 16
Correctly initiate local operations. `dof-2009-12-spec-6` 16
DPP datagrams with Duration of zero (0) must cause the operation to be cancelled. `dof-2009-12-spec-7` 16
DPP durations must be extended by the grace period before termination. `dof-2009-12-spec-8` 17
DPP durations must be transmitted so that they are received correctly. `dof-2009-12-spec-9` 17
Handle duplicate operations and retries as required. `dof-2009-12-spec-10` 18
DPPv2 Application Format `dof-2009-12-pdu-5` 20
DPPv2 Command/Response Format `dof-2009-12-pdu-6` 21
Compare AS Assigned SIDs using only the base identity. `dof-2009-12-spec-11` 21
IAM Permission `dof-2009-12-pdu-7` 21
ACTAS Permission `dof-2009-12-pdu-8` 22
Tunnel Domain Permission `dof-2009-12-pdu-9` 23
Cancel All (command) `dof-2009-12-pdu-10` 24
Node Down (command) `dof-2009-12-pdu-11` 25
Source Lost (command) `dof-2009-12-pdu-12` 26
Source Found (command) `dof-2009-12-pdu-13` 26
Operation Rename (command) `dof-2009-12-pdu-14` 27
DPP Ping (command) `dof-2009-12-pdu-15` 27
DPP Ping (response) `dof-2009-12-pdu-16` 28
Respond correctly to Query PDUs. `dof-2009-12-spec-12` 28
SID Query (command) `dof-2009-12-pdu-17` 29
SID Query (response) `dof-2009-12-pdu-18` 29
DPP Heartbeat (command) `dof-2009-12-pdu-19` 29
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1` 31
Compressed Unsigned 32-Bit `dof-2009-1-pdu-3` 31
Object Identifier (OID) `dof-2009-1-pdu-4` 32
Object Identifier Attribute `dof-2009-1-pdu-5` 32
Source Identifier `dof-2009-1-pdu-19` 33
Operation Identifier `dof-2009-1-pdu-20` 33
General DPP Header `dof-2008-1-pdu-7` 34
General DPP Trailer `dof-2008-1-pdu-8` 34
Security Mode of Operation Header `dof-2008-1-pdu-9` 34
Security Mode of Operation Trailer `dof-2008-1-pdu-10` 34
General Application Header `dof-2008-1-pdu-13` 35
Security Scope `dof-2008-16-pdu-6` 35
Permission `dof-2008-16-pdu-7` 35

1. About This Document

This document contains information about DOF protocols. This is critical information for anyone who needs to work with the protocols themselves, and contains useful background information for those interested in the architecture and inner workings of DOF implementations.

This information is highly technical. This document serves as a specification and reference against which someone could implement networked software that interoperated with DOF products without using existing libraries.

The Technical Steering Committee of the OpenDOF Project, Inc. (ODP-TSC) manages this document.

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.

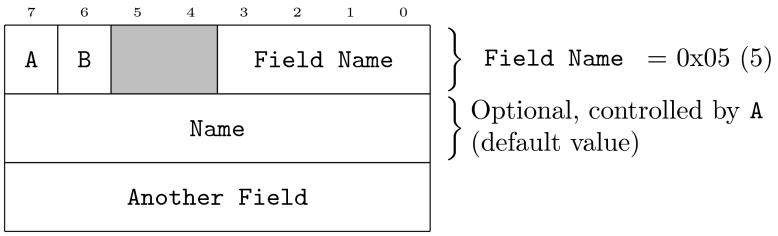
1.3.4. Fields

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. Specification

This section describes version two (2) of the DOF Presentation Protocol. The syntax of the PDUs is described first, followed by sections that discuss the meaning and use of the fields.

The protocol begins with the required one-byte header. This byte contains the version (2) and a bit determining if a flag byte is present. The flag byte is described later.

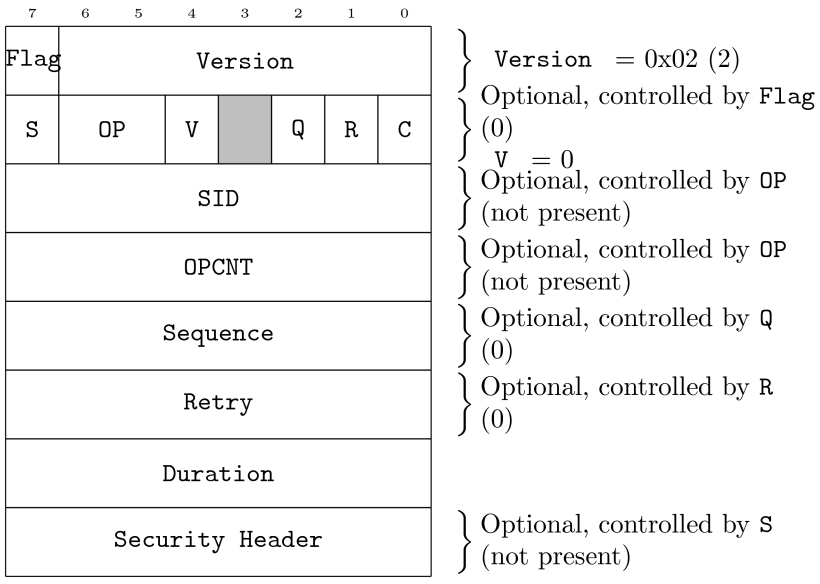
If the bit is set, then a single flag byte follows (potentially followed by additional fields). The default value of the flags byte is zero (0).

For secure datagrams, there are two boundaries. The first begins with the DPP version byte, and is the point at which the datagram is secured from tampering. This is called the security boundary. The second, which begins within or at the end of the security mode of operation header, is the point at which the datagram is encrypted. This is the encryption boundary.

There are two primary variants of the DPPv2 header: commands and responses. The meaning of the flag byte itself is dependent on the variant, and so the first determination needs to be made based on the 5th bit of the flags byte (**V**). Assuming the flag value is FLAG, if FLAG & 0x10 is 0x00 then the PDU is a command (**V** is 0). Otherwise the PDU is a response.

⇄ DPPv2 Command Header `dof-2009-12-pdu-1`

Instance of General DPP Header`dof-2008-1-pdu-7`



`Flag` **One bit.** Controls the presence of the flag fields, which are present if set.

`Version` **Seven bits.** Specifies the DPP version.

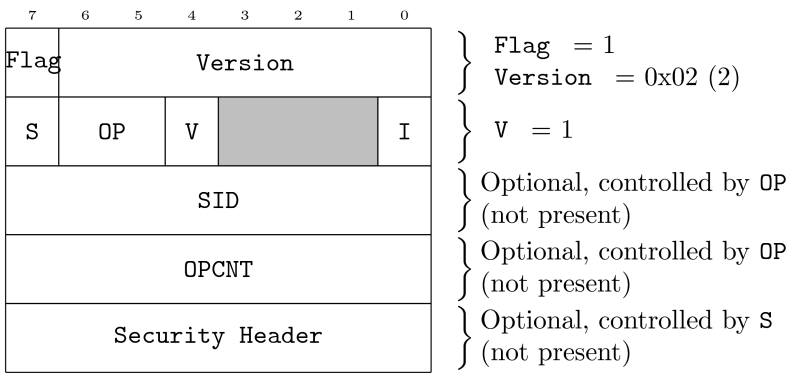
`S` **One bit. Optional, controlled by** `Flag`. Controls the **Security Header**.

OP	**Two bits. Optional, controlled by** Flag. Controls the presence of the **SID** and **OPCNT** fields. Possible values are zero (0), one (1), and three (3).
V	**One bit. Optional, controlled by** Flag. The indicated value (0) indicates the PDU is a command.
Q	**One bit. Optional, controlled by** Flag. Controls the presence of the **Sequence** field.
R	**One bit. Optional, controlled by** Flag. Controls the presence of the **Retry** field.
C	**One bit. Optional, controlled by** Flag. When set, indicates that the command completes on any FINAL response. This means that no state is required to be kept once a response has been processed. This bit can be used by intermediate nodes to remove state once a response is received and forwarded.
SID	**Instance of** Source Identifier$_{\texttt{dof-2009-1-pdu-19}}$.**Optional, controlled by** OP. If **OP** is one (1), then the SID is implied to be that of the sending node. This must either be known from a cache or queried (if required). The field is present if the **OP** field is three (3).
OPCNT	**Instance of** Compressed Unsigned 32-Bit$_{\texttt{dof-2009-1-pdu-3}}$.**Optional, controlled by** OP. Present if the **OP** field is not zero (0).
Sequence	**One byte. Optional, controlled by** Q. Present if the **Q** field is one. This field is described in the section on Loop Detection.
Retry	**One byte. Optional, controlled by** R. Present if the **R** field is one. This field is used as a floating window. A retry is defined by a **Retry** value is that is "greater" than the previous value, ignoring wrapping of the value.
Duration	**One byte.** This field contains the desired lifetime of the command in seconds, logically starting at the time of reception. Values from 1 to 128 mean to add that number of seconds to the datagram reception time. Values from 129 to 255 mean to add 128 + (value - 128) * 32 seconds to the datagram reception time. The value of zero is special and must be flagged as such.
Security Header	**Instance of** DPPv2 Security Header$_{\texttt{dof-2009-12-pdu-3}}$.**Optional, controlled by** S. Present if the **S** field is one.



DPPv2 Response Header `dof-2009-12-pdu-2`

Instance of General DPP Header$_{\texttt{dof-2008-1-pdu-7}}$



`Flag` **One bit.**

`Version` **Seven bits.** Specifies the DPP version.

`S` **One bit.** Controls the **Security Header**.

`OP` **Two bits.** Controls the presence of the **SID** field and **OPCNT** field. Valid values are zero (0), two (2), and three (3). The value of one (1) indicates that a sender SID is used, and is not valid for responses. Note that the value (3) is mandatory if there could be confusion regarding the SID (as an example, if the packet were sent multicast or broadcast).

`V` **One bit.** The indicated value (0x01) indicates the PDU is a response.

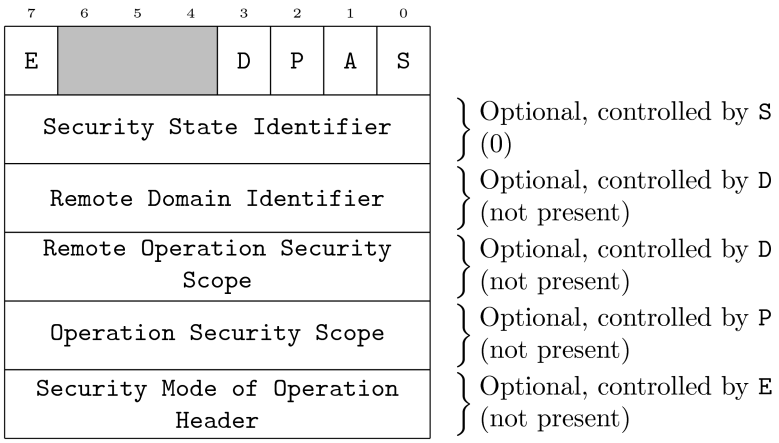
`I` **One bit.** Determines whether the response is INTERMEDIATE or FINAL. If clear, then the response is FINAL. If set, the response is INTERMEDIATE.

`SID` **Instance of** Source Identifier`dof-2009-1-pdu-19`.**Optional, controlled by** `OP`. If **OP** is one (1), then the SID is implied to be that of the sending node. This must either be known from a cache or queried (if required). The field is present if the **OP** field is three (3).

`OPCNT` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `OP`. Present if the **OP** field is not zero (0).

`Security Header` **Instance of** DPPv2 Security Header`dof-2009-12-pdu-3`.**Optional, controlled by** `S`. Present if the **S** field is one.

DPPv2 Security Header `dof-2009-12-pdu-3`



`E` **One bit.** Controls the presence of the **Security Mode of Operation Header** field.

`D` **One bit.** Controls the presence of the **Remote Domain Identifier** field.

`P` **One bit.** Controls the presence of the **Operation Security Scope** field.

`A` **One bit.** Set to one (1) if the **Security State Identifier** was assigned by the authentication server, cleared (0) if the **Security State Identifier** was assigned by locally.

`S` **One bit.** Controls the presence of the **Security State Identifier** field.

`Security State Identifier` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `S`. Present if the **S** field is one.

`Remote Domain Identifier` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `D`. Present if **D** is one. Represents the domain identifier for tunneled domain traffic.

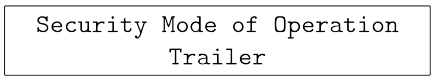
`Remote Operation Security Scope` **Instance of** Security Scope`dof-2008-16-pdu-6`.**Optional, controlled by** `D`. Present if **D** is one. The function of the operation security scope is described in the *DOF Security Specification* manual. This field represents the original security scope of the operation in the remote domain, and is required whenever a **Remote Domain Identifier** is present.

`Operation Security Scope` **Instance of** Security Scope`dof-2008-16-pdu-6`.**Optional, controlled by** `P`. Present if **P** is one. The function of the

	operation security scope is described in the *DOF Security Specification* manual.
`Security Mode of Operation Header`	**Instance of** Security Mode of Operation Header`dof-2008-1-pdu-9`.**Optional, controlled by** `E`. Present if **E** is one.

⇄ **DPPv2 Trailer `dof-2009-12-pdu-4`**

Instance of General DPP Trailer`dof-2008-1-pdu-8`



`Security Mode of Operation Trailer`	**Instance of** Security Mode of Operation Trailer`dof-2008-1-pdu-10`.Controlled by the **E** field of DPP.2.3: DPPv2 Security Header and the **S** field from DPP.2.2: DPPv2 Response Header or DPP.2.1: DPPv2 Command Header.): Present if **S** is one, and if so that **E** is one.

2.1. Sealing and Unsealing PDUs

Sending a secure PDU requires that the PDU be sealed. This process includes protecting the contents beginning with the DPP version byte and possibly encrypting it beginning after the last DPP header byte. The opposite of sealing is unsealing, and each received secure PDU must be unsealed. The processes of sealing and unsealing require some amount of shared state between the sender and receiver.

The process of sealing a datagram includes:

- Obtaining the datagram, including the DPP fields and application data. This includes requesting the **Security Mode of Operation Header** from the security mode of operation. This also requires obtaining the fields for the DPP **Security Header**.
- Once the DPP fields and application data are determined and the PDU is formulated, requesting that the security mode encrypt or otherwise authenticate the data, using as input the formulated datagram, the security state and the security mode. This will likely result in the generation of a **Security Mode of Operation Trailer**.
- Take the result, including the new trailer, and passing it to the network layer.

The process of unsealing a datagram includes:

- Obtaining the datagram, and parsing it as far as the **Security Mode of Operation Header**.
- Determining the security state, if unknown then dropping the datagram or closing the session. Requesting that the security mode read its header.
- Based on the security mode and the security state, decrypting or validating the datagram.
- If successful, continuing to process the datagram.

There is the question of how to identify the state for a given communication. There are four cases for determining the state involved for a datagram:

- 2-point lossless session. In this case the state is associated with a session. It is assumed that the session is known for each datagram either accepted or written. No state identifier is required.
- 2-point lossy session. In this case a state identifier is present. This state identifier is returned as part of establishing the security information. Note that if the lossy relationship is unique (perhaps due to transport knowledge, like port numbers) then the situation is the same as for lossless session and no state identifier is required.
- Point-to-multipoint lossy session. This is the same as 2-point lossy session.
- *N*-node sessions. In this case there must be a security state identifier present. It is assigned as part of the process of joining the session.

2.2. Operation Identifiers

DPP manages command state based on the operation identifier contained in the PDU. If there is no operation identifier present in a PDU then there is no opportunity for DPP state to be managed for that operation. This is true even in the case where the duration is greater than zero. In that case the operation state contains the requested duration, but there is no way for the DPP to manage any state and so retry and other DPP functionality is not available.

In order to optimize bytes on the wire, it is possible for a PDU to use compression for OPIDs. A compressed SID is called an "implied" SID. If the SID is present in the header then it is "explicit".

An implied SID is beneficial because it can reduce datagram size significantly. Its use can cause problems, however, when the receiver needs to know the actual SID (needs to "expand" it).

One method for implied SID expansion is to cache knowledge of SIDs based on source information. The rules that cover this behavior are discussed later.

The other method is to ask the sender. This is done with SID Query and SID Query Response PDUs.



DPP must check valid use of cached SIDs. `dof-2009-12-spec-1`

Only valid implied SID methods are allowed. DPP **OP** of 1 is only valid on commands, and an **OP** of 2 is only valid on responses.

In general, the following apply to operation identifiers:

- If the receiver does not know the SID of the sender and it is required, it must query the SID (described below).
- Cache SIDs based on the sending transport address (including DNP logical address). This means that knowledge about the SID used by a source should be remembered once it is known. This cache should persist for as long as there are active operations with the same source.
- Learn SIDs when possible. When this can be done depends on the definitions for the particular DPP version.
- Use explicit SIDs when there is doubt about whether the receivers will have a cached value. For example, if a sender has no outstanding operations on a session then the receivers have most likely cleared their caches of his SID. In this case the use of an explicit SID will re-seed the caches. A retry of the operation can then use implied, assuming that the cache exists.

2.3. Loop Detection and Prevention

The **Sequence** field represents a number that increases as a command propagates. It is guaranteed that a command's **Sequence** will be greater than that of its parent, but it is not guaranteed that the difference between parent and child will be one (1). Operations with a **Sequence** of 255 cannot be forwarded. Any command with a **Sequence** of zero (0) that includes an operation identifier must use the SID of the sender (the source). If the **SID** is explicit, then in the case of a **Sequence** of zero it is known to be the SID of the source.

Commands that are forwarded using the same operation identifier must increase the **Sequence**. This means that if the same command is received again by the same node it can be detected as a loop because the **Sequence** would be greater than that already associated with the operation.



DPP datagrams with a Sequence of zero (0) must use the SID of the sending node. `dof-2009-12-spec-2`

This allows caching the SID in the case of **Sequence** zero PDUs, without requiring a SID Query.



The sequence of a forwarded datagram with a global operation identifier must use a greater sequence than when received. `dof-2009-12-spec-3`

The typical use is to increment the sequence. This requirement implies that an operation with a sequence of 255 (the maximum) must not be forwarded.

There are two fundamental types of commands: DIRECTED and FLOODED. Directed commands will follow a known path that prevents looping, while flooded commands have no defined path and therefore may loop back on the same node.

Directed commands are not required to use global operation identifiers, and they should not. This means that a forwarding node is free to use its own "local" operation identifier in the forwarded PDU, including using a **Sequence** of zero (0). This is desirable because it minimizes datagram size. Flooded commands are required to use a global operation identifier, meaning that it must not be changed as the PDU is forwarded. The use of a global operation identifier requires the use and increasing of the **Sequence** field. The **Sequence** field must also be greater than zero (0).

It is possible for a node to receive a "local" operation identifier that it determines should be flooded. In this case it may be beneficial to assign a global operation identifier based on the forwarding node's SID. This can be done, but in this case the initial **Sequence** value must be one (1) and not zero (0). This is typically only a benefit if the SID of the local operation identifier that was received is not known. If the local SID is known then it can be used in the global operation identifier (still increasing the **Sequence**).

However, once an operation is global (meaning it has a **Sequence** greater than zero (0)) then it must not be converted back to a local operation even if the forwarding node can make the operation directed. This requirement prevents a type of ringing in the network.



Correctly manage operations that use local and global operation identifiers during forwarding. `dof-2009-12-spec-4`

This includes using the correct operation type based on directed and flooded commands, and correctly mapping global to global and local to global during forwarding.

Correctly initiate global operations. `dof-2009-12-spec-5`

Nodes must use global operation identifiers for all flooded operations, with a **Sequence** greater than zero (0).

Correctly initiate local operations. `dof-2009-12-spec-6`

Nodes should use local operation identifiers for all directed operations, with a **Sequence** equal to zero (0).

On reception, there is no sure way to know if an operation was DIRECTED or FLOODED. It is known that all operations with a **Sequence** of zero (0) are directed. It can be assumed that operations with a **Sequence** greater than zero (0) are FLOODED, although there are some exceptions. It is possible that the system will imply that a DIRECTED operation is actually FLOODED.

2.4. Operation Lifecycle Management

Each command includes an associated duration, which may be zero (a special case), or some number of seconds that extends from the receipt of the command (including any delay before initial processing). Commands that are retried, as discussed next, continue to be extended into the future as each retry is received.

A duration of zero means that the operation should end immediately and that any existing DPP state for the operation should be immediately removed (it also means that no DPP state should be *created* based on the operation). However, as mentioned in the *DOF Protocol Stack*, these operations should be presented to the appropriate application for processing.

The ability to extend an operation means that another datagram must be received before the expiration time specified. In order to overcome issues of clock skew (primarily), the actual expiration time of an operation is extended by a grace period that is dependent on the duration value. The length of this grace period is zero seconds for a duration of zero (which means cancel), four seconds for durations up to 128 seconds, and then increasing by one second for every 128 seconds past that. This means up to 35 seconds over the maximum one hour duration.

DPP datagrams with Duration of zero (0) must cause the operation to be cancelled. `dof-2009-12-spec-7`

The PDU is still presented to the application for further processing, but must be flagged as being a cancel instead of a retry. Further, existing DPP state should be removed, and no new DPP state created. This lack of state means that operations should usually not attempt to send responses to these types of operations, although the state of the operation is known to the operation itself (but not being kept by DPP) and that could be used to send a response. It is also typically invalid to forward these operations to other nodes.

DPP durations must be extended by the grace period before termination. `dof-2009-12-spec-8`

Care must be taken not only to include the grace period calculation but also to ensure that values round up correctly. The goal is to ensure that the operation does not time out too early, before the sender has an opportunity to extend the operation.

DPP durations must be transmitted so that they are received correctly. `dof-2009-12-spec-9`

When operations are forwarded partially through their lifetime, the receiving node may compute a different grace period, and hence a different timeout. Senders must adjust the duration so that the duration plus the (receiver's) computed grace period will match the actual timeout on the sender.

The following sections detail the DPPv2 requirements for operation lifecycle management. This section covers rules for three separate cases: bridge nodes, terminal nodes and gateways. The rules for bridge nodes assume that no application is present. The rules for terminal nodes and gateways assume that an application is present.

DPPv2 requires that certain state be maintained if operations are going to be properly managed. This state is defined in this section. Other versions may require different state as discussed in their respective sections.

The following is assumed to be known for each operation:

- The operation identifier. This one piece of information is common for all DPP versions.
- The operation source. This is an DNP logical address, which includes a transport address (and possibly some DNP logical address information). It also has two special values: *LOCAL*, meaning that the node itself is the source; and *LOST*, meaning that there is no current source. Part of the source information is the transport type, either *LOSSY* or *LOSSLESS*. Priority is given to *LOSSLESS* sources over *LOSSY* sources.
- The current sequence. This is the sequence received for the current source. It has no meaning for source *LOST*, and must be zero (0) for source *LOCAL*.
- The current retry. This is a sliding-window value, not absolute. Its value is not dependent on the source information.
- A flag, *FORWARDED*, that indicates that the operation has been forwarded using the same operation identifier.

One of the most fundamental requirements of the DPP is to prevent duplicates. This rejection of duplicates is *version independent*. This means that each version of DPP must define not only how it handles loops and duplicates with its own version, it must also define how inter-version comparisons are made. As DPPv2 is the first DPP version defined that manages operations, all future versions must define comparisons against DPPv2.

Terminal nodes may elect not to maintain operation state. This lack of state means that duplicates will not be noticed and that operations may execute multiple times if a duplicate is received. This optimization *should not* apply to transport retransmissions, which should always be tracked and retransmissions ignored. It also *should not* apply to cases where the sender explicitly asks the application to not respond in the future through an acknowledgement.

Duplicate operations can be seen because either:

- A loop exists in the network, and a proxy is echoing the operation.
- A node retransmitted a PDU to overcome datagram loss on a transport.
- A node retried an operation in order to obtain new results.
- A node reused an operation identifier. This is not normal, but the ability to prevent the reuse of an operation identifier depends on persistent state. If the node is reinitialized (reboots, for example) then the state may be lost and a duplicate created.

The appropriate response in each of these cases is different, and so it is critical, to the extent possible, that a node be able to distinguish which case is causing a duplicate when it receives one.

There are three DPPv2 fields which are directly related to this determination:

- First, the OPID field itself.
- Second, the Sequence field.
- Third, the Retry field.



Handle duplicate operations and retries as required. `dof-2009-12-spec-10`

A node must correctly manage duplicate operations, retransmissions and retries. The specific requirements are listed below.

The general lifecycle information for operations was presented earlier in this section. The following steps indicate the order in which fields are compared and the required result:

- Compare the operation identifier with outstanding operations that are in process. If it doesn't exist, and the application validates the request then the operation is new and needs to be processed as *initial*. If the operation identifier already has current state, continue with the next step.
- If the operation has no current source then process the PDU as a *new source*. Otherwise continue with the next step.
- Compare the Sequence fields. If the received operation has a greater sequence number and came from a different transport address, then it is caused by a longer path and must be handled as *further*. If it came from the same source transport address and has a greater sequence, then it signals a loss of state and may indicate a self-sustaining operation. In that case the operation is treated as in the *reuse* case. If the sequence is less than the current value then a closer source has been identified. Follow the *closer* case as described below. Otherwise, continue with the next step.
- At this point the sequence number must be the same in both new and current operations and the transport address may be different. If the current transport is *LOSSY* with session type *NONE*, and the new operation is the same then process as *new source* unless the new retry is less than the current retry in which case the new operation is dropped.
- If the new operation is *LOSSY* with session type *NONE* then it is dropped (note that the current operation must not be *LOSSY/NONE* or the previous rule would have applied).
- If the new operation is *LOSSLESS* and the current operation is *LOSSY* or the new operation is *LOSSY* and the current operation is *LOSSY/NONE* then process according to *new source*. Otherwise, continue to the next step.
- If the new and current operations use different sessions then drop the new operation.

- At this point the sessions are the same. Compare the Retry field. If the retry is greater, then the operation is a *retry* and should be processed as such. Otherwise, continue to the next step.
- Process the PDU as a *duplicate*.

So, out of the preceding steps there are several actions that can take place. The following sections describe these actions.

2.4.1. Operation State: Initial and Retry

Initial and Retry states are determined directly from DPP once the operation has been validated by the application. This may mean that the operation is passed to the application twice: once for validation and again for execution. However, in both the Initial and Retry cases the operation will be executed.

In the case of a terminal node the application may signal the DPP that no state is to be kept. It may also be that the operation is signaled as complete but doesn't generate a response. Operations that do not have an operation identifier are immediately complete, although the application is still told of their creation (and current completed state). Operations that arrive and are already timed out (or cancelled, with a duration of zero) are also immediately complete but the application is told of the creation (and completion).

The application may also forward the operation, either using the same operation identifier or using a new operation identifier. When this occurs the DPP should track that the operation has been forwarded. This will allow more optimized handling of certain state transitions later on.

As a special case, a node may choose not to maintain state for individual operations. In this case each operation will appear to be an "initial" operation.

Usually an initial operation will result in at least the following state being kept:

- The source transport address of the request.
- The OPID, Retry and Sequence fields.

In the case of a retry the state is updated to the retry field present in the received PDU and the lifespan of the operation is also updated. The application is told of the retry.

2.4.2. Operation State: Lost

This is the state when an operation has lost its source (described as *LOST*, above).

2.4.3. Operation State: New Source

This state exists to handle the case of a lost source due to communication failure. Any operation is better than a lost operation, so any new source is accepted. The state of the operation is updated to that provided in the new PDU. The application is told of the new source.

2.4.4. Operation State: Further Source

This case is handled by silently dropping the PDU. No state changes are made to the operation.

2.4.5. Operation State: Reuse

This situation is handled the same as the loss of the source for the operation. This is described above in the section on Operation State: Lost. The reason for this is that the longer Sequence may depend on the original shorter Sequence, and the Operation State: Lost will validate that the operation has a correct source.

2.4.6. Operation State: Closer Source

In this case the state is updated to the fields present in the received PDU and the source transport address of the request is also updated. The application is told of the new source.

2.4.7. Operation State: Duplicate

In this case the PDU is dropped with no response, and no changes are made to the existing operation.

2.5. Operation Management

In general the only required management for terminal nodes is the handling of lost parents and timeouts. This is identified earlier in the general discussion on operations in the : Lost and : Timeout sections.

When an existing operation loses its source then it is the responsibility of the node to search for an alternate source. This applies even to terminal nodes. The handling of the loss of a source includes reducing the lifespan of the operation, so if a new source is not identified the operation will terminate due to a timeout.

Until a new source is identified, responses to the operation cannot be sent. These responses should be dropped if attempted.

To search for a new source the common DPP Operation Source Lost PDU is used. This PDU can be sent to all transports and over all sessions. Note that once sent the operation will either be reset (because of a retry) or timeout.

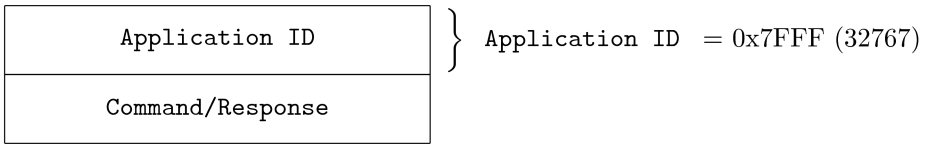
However, note that the state stored for the operation must be modified so that any retry operation will become the new source.

2.6. Common DPP Capabilities

The following capabilities are provided, which match the required common DPP capabilities. Each of these PDUs is treated as if sent by an application using APPID 0x7FFF.

⇄ **DPPv2 Application Format** `dof-2009-12-pdu-5`

Instance of General Application Header`dof-2008-1-pdu-13`

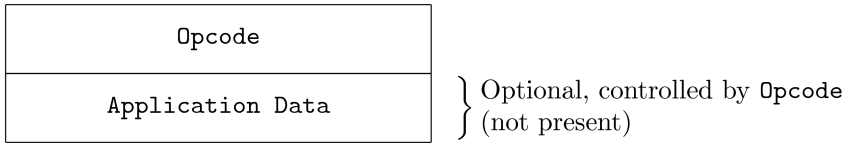


`Application ID` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Contains the application identifier for the enclosed data.

`Command/Response` **Instance of** DPPv2 Command/Response Format`dof-2009-12-pdu-6`.Contains the command or response data.

When used to send responses, the following format is used.

DPPv2 Command/Response Format `dof-2009-12-pdu-6`



`Opcode` **One byte.** The value will indicate the particular operation.

`Application Data` **Variable length. Optional, controlled by** `Opcode`. Each **Opcode** will define the required **Application Data**.

2.6.1. SID Security

DPPv2 defines two permissions that govern the use of a source identifier (SID) in an operation identifier. These permissions must be negotiated and understood whenever this version of DPP is used.

When an operation is sent from a node using a particular SID, then ACTAS is required. However, IAM implies ACTAS and so either permission may be granted. Explicitly negotiating IAM provides a benefit that the other node knows the SID of the sender.

However, operations that use a sequence of zero must use the SID of the sender, and so they require the IAM permission.

Compare AS Assigned SIDs using only the base identity. `dof-2009-12-spec-11`

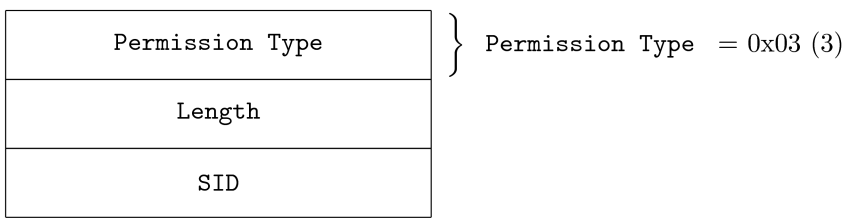
The uniqueness of AS Assigned SIDs is to prevent collisions, but the permission is granted only using the base identity. The remaining fields are not part of the comparison when used in permission validation.

2.6.2. IAM

The IAM permission is required for a node to use a particular SID to represent itself. This permission must only be granted in all permission group identifiers.

IAM Permission `dof-2009-12-pdu-7`

Instance of Permission`dof-2008-16-pdu-7`



`Permission Type`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.
`Length`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Contains the length of the **SID**.
`SID`	**Instance of** Source Identifier`dof-2009-1-pdu-19`.The SID of the requesting node.

It is an error to request the IAM permission using multiple SIDs in the same session. To do otherwise would be a violation of a specification, as a session is always associated with a single transport source address.

Note that failures to correctly request and obtain a valid IAM permission results in a "dead" session since no further operations (which must be secure) can be initiated. Failures to negotiate IAM result from misconfiguration (either in the node's request or in the AS storage).

2.6.2.1. Normalization

Normalization includes removing all duplicates.

2.6.2.2. Comparison

IAM is a terminal permission and permission A $\leq$ permission B only if the two SIDs are equal (see note).



There is a special case for a SID of class 0x08 (AS Assigned) where the SID comparison only compares the node identifier portion of the OID. This comparison requires that both permissions have the same OID class.

2.6.2.3. Complement

There is no complement.

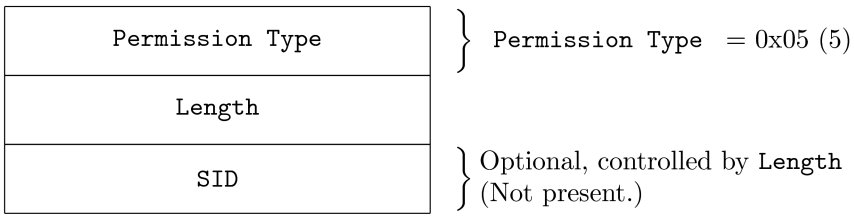
2.6.3. ACTAS

The ACTAS permission is required for a node to represent another node as proxy (meaning to forward operations that use another node's SID). This permission must only be granted in all permission group identifiers.



ACTAS Permission **`dof-2009-12-pdu-8`**

Instance of Permission`dof-2008-16-pdu-7`



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

`Length`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Contains the length of the **SID**.
`SID`	**Instance of** Source Identifier`dof-2009-1-pdu-19`.Present if **Length** is not zero. The SID that the node requests to act as the requesting node.

2.6.3.1. Normalization

Normalization includes removing all duplicates. Further, if any ACTAS permission does not include the SID (Length is zero) then all other ACTS permissions can be removed.

2.6.3.2. Comparison

ACTAS is a terminal permission because it does not utilize permission group identifiers (it must be granted in all permission groups). ACTAS permission A $\leq$ permission B (is allowed by) if any of the following is true:

- Permission B is an ACTAS permission with the same SID (see below).
- Permission B is an ACTAS permission with no SID (indicating match all).
- Permission B is an IAM permission with the same SID (see below).



There is a special case for a SID of class 0x08 (AS Assigned) where the SID comparison only compares the node identifier portion of the OID. This comparison requires that both permissions have the same OID class.

2.6.3.3. Complement

There is no complement.

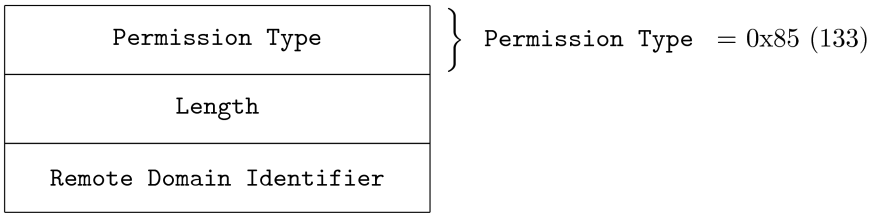
2.6.4. Tunnel Domain

The Tunnel Domain permission is required to send any operation that includes a remote domain identifier. This permission must only be granted in all permission group identifiers.



Tunnel Domain Permission **`dof-2009-12-pdu-9`**

Instance of Permission`dof-2008-16-pdu-7`



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

`Length`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Contains the length of the **SID**.
`Remote Domain Identifier`	**Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.Represents the remote domain identifier for tunneled domain traffic. This can be the wildcard remote domain identifier (0x3FFFFFFF) to indicate permission to tunnel all remote domains.

2.6.4.1. Normalization

Normalization occurs in each security scope and includes removing all duplicates. Further, if any Tunnel Domain permission includes the wildcard remote domain identifier then all other Tunnel Domain permissions can be removed. All comparisons must occur in the same security scope.

2.6.4.2. Comparison

Tunnel Domain is a terminal permission because it does not utilize permission group identifiers (it must be granted in all permission groups). Tunnel Domain permission A $\leq$ permission B (is allowed by) if any of the following is true:

- Permission B is a Tunnel Domain permission with the same remote domain identifier.
- Permission B is a Tunnel Domain permission with the wildcard remote domain identifier. # 1. Complement

There is no complement.

2.6.5. Commands for Common Capabilities

The following commands and responses are defined by DPPv2 in order to satisfy the general DPP common capability requirements. The specific behaviors of these operations are discussed in the general DPP section of the *DOF Protocol Stack* document.

The behavior of these operations is consistent with the general requirements – only the formatting of the specific PDUs is left to the specific DPP version, and they are defined here.

2.6.6. Cancel All

Cancel all operations from a given SID.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): IAM
Permission (response): N/A



Cancel All (command) `dof-2009-12-pdu-10`

COMMAND/DIRECTED
NOOPID

Instance of DPPv2 Command/Response Format`dof-2009-12-pdu-6`



`Opcode` **One byte.**

`SID` **Instance of** Source Identifier`dof-2009-1-pdu-19`.The SID of the operations that should be cancelled.

See the *DOF Protocol Stack* for details on this command.

2.6.7. Node Down

Cancel all operations from a given SID and note that the sending node stopping.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): IAM
Permission (response): N/A

Node Down (command) `dof-2009-12-pdu-11`

COMMAND/DIRECTED
NOOPID
Instance of DPPv2 Command/Response Format`dof-2009-12-pdu-6`



`Opcode` **One byte.**

`SID` **Instance of** Source Identifier`dof-2009-1-pdu-19`.The SID of the operations that should be cancelled.

See the *DOF Protocol Stack* for details on this command.

2.6.8. Operation Source Lost and Source Found

Declare that the source of an operation is lost.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): None

Permission (response): N/A

⇄ Source Lost (command) `dof-2009-12-pdu-12`

COMMAND/DIRECTED
NOOPID
Instance of DPPv2 Command/Response Format`dof-2009-12-pdu-6`



`Opcode` **One byte.**

`OPID` **Instance of** Operation Identifier`dof-2009-1-pdu-20`.The OPID of the operation with the lost source.

The Source Lost PDU does not require any DPP headers unless it is sent securely, in which case it requires the security headers. It does not require an OPID, Retry Count, etc.

Note that both of these commands include the operation identifier in the payload of the command. This means that a full (non-implicit) identifier must be known. This can cause a problem if the SID of the operation identifier is not known. In this case there can be no assumptions about the state of the operation – the parent node may be operating correctly and only the session had a problem.

This problem is more difficult because almost every node must, at some point, use the Source Lost command. This effectively requires that all nodes obtain the actual SID of all operations. In particular they must determine the SID of all distance zero sources based on distinct transport addresses.

Declare that the source of an operation is found.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): IAM
Permission (response): N/A

⇄ Source Found (command) `dof-2009-12-pdu-13`

COMMAND/DIRECTED
NOOPID
Instance of DPPv2 Command/Response Format`dof-2009-12-pdu-6`



`Opcode` **One byte.**

`OPID` **Instance of** Operation Identifier`dof-2009-1-pdu-20`.The OPID of the operation that has a new source. Other fields come from the DPP headers. ACTAS permission is required on this OPID for secure sessions.

See the *DOF Protocol Stack* for details on these commands. In particular, note that these are listed as 'directed' commands although they are potentially sent to multiple nodes. They are also not fully 'flooded'. The full description of their behavior is found in the protocol stack reference.

2.6.9. Operation Rename

Rename an operation.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): ACTAS
Permission (response): N/A

⇄ Operation Rename (command) `dof-2009-12-pdu-14`

COMMAND/FLOODED
OPID
Instance of DPPv2 Command/Response Format`dof-2009-12-pdu-6`



`Opcode` **One byte.**

`OPID` **Instance of** Operation Identifier`dof-2009-1-pdu-20`.The OPID of the operation that should be renamed.

See the *DOF Protocol Stack* for details on this command.

2.6.10. Ping and Ping Response

Ping a node, requesting a response.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast
Security: Session.
Permission (command): None
Permission (response): None

⇄ DPP Ping (command) `dof-2009-12-pdu-15`

COMMAND/DIRECTED

OPID
Instance of DPPv2 Command/Response Format`dof-2009-12-pdu-6`



`Opcode` **One byte.**

DPP Ping (response) `dof-2009-12-pdu-16`

RESPONSE
Instance of DPPv2 Command/Response Format`dof-2009-12-pdu-6`



`Opcode` **One byte.**

See the *DOF Protocol Stack* for details on this command and response.

2.6.11. SID Query and SID Query Response

This command and response are not DPP common behaviors, as they deal with SID compression. SID compression is a feature of DPPv2, not of the general DPP, and so commands that deal with compression are version specific.

The query and query response PDUs are the way to query the implied SID of another node. The SID learned from this method should be cached for at least ten (10) seconds, or longer if state is available (as for a session).

In the case of an unsecured session the SID of a node must be consistent, and within a single Security Domain the SID of a node must be consistent and have the required permissions. However, a single transport address may have different SIDs in different Security Domains (including unsecured operations as being a single "Unsecured Security Domain".

Respond correctly to Query PDUs. `dof-2009-12-spec-12`

When a transport unicast query PDU is received then a response must be sent. Multicast and broadcast queries are silently ignored. A query response should be received within ten (10) seconds of the query. The response is scoped to the Security Domain it is received in.

The PDU formats are described here.

Request the receiver to send its SID.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast
Security: Session.
Permission (command): None

Permission (response): IAM

SID Query (command) `dof-2009-12-pdu-17`

COMMAND/DIRECTED
NOOPID
Instance of DPPv2 Command/Response Format`dof-2009-12-pdu-6`



`Opcode` **One byte.**

SID Query (response) `dof-2009-12-pdu-18`

RESPONSE
Instance of DPPv2 Command/Response Format`dof-2009-12-pdu-6`



`Opcode` **One byte.**

`SID` **Instance of** Source Identifier`dof-2009-1-pdu-19`.The SID of the responding node.

2.6.12. Heartbeat

Send a packet that does nothing, but can indicate the session is active.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast
Security: Session.
Permission (command): None
Permission (response): N/A

DPP Heartbeat (command) `dof-2009-12-pdu-19`

COMMAND/DIRECTED
NOOPID
Instance of DPPv2 Command/Response Format`dof-2009-12-pdu-6`



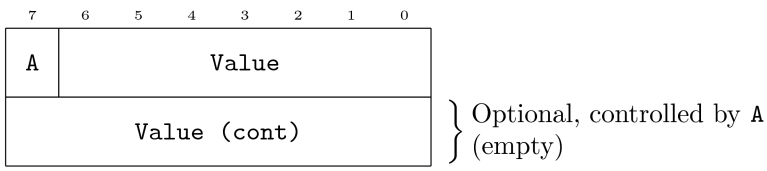
| Opcode **One byte.**

See the *DOF Protocol Stack* for details on this command.

3. External PDU Reference

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

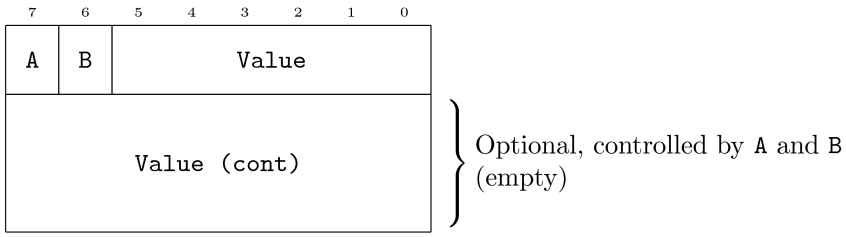
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1`



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

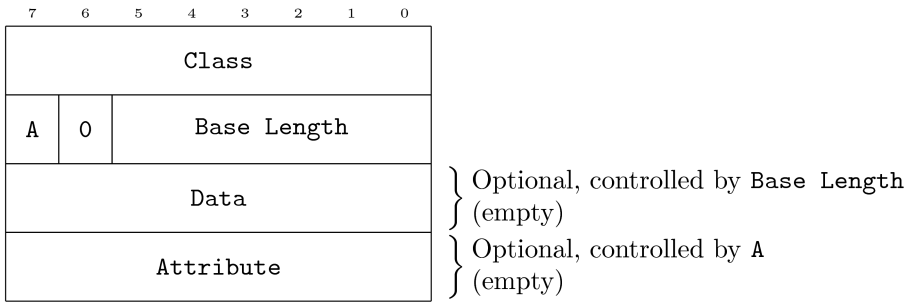


`A` **One bit.** Controls the size of `Value` and the presence of the `B` field.

`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.

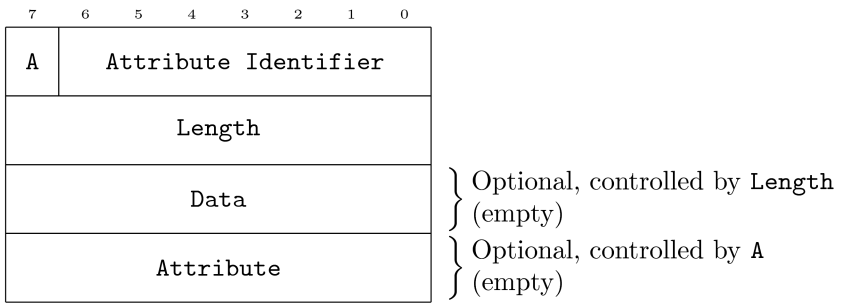
`A` **One bit.** Controls the presence of `Attribute`.

`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.

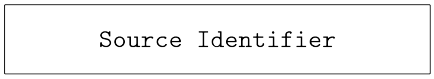
`Length` **One byte.** This field contains the length of the `Data`.

`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.

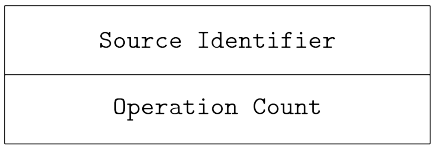
⇄ **Source Identifier `dof-2009-1-pdu-19`**

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



`Source Identifier` **Instance of** Object Identifier (OID)`dof-2009-1-pdu-4`.The maximum `Base Length` is 0x14 (20) bytes, with a minimum of one byte. The `A` field is zero (0).

⇄ **Operation Identifier `dof-2009-1-pdu-20`**

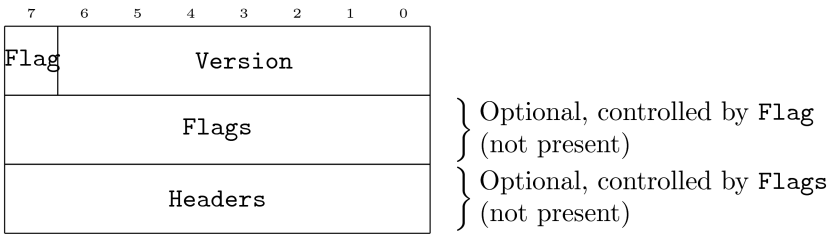


`Source Identifier` **Instance of** Source Identifier`dof-2009-1-pdu-19`.

`Operation Count` **Instance of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.The unique identifier of the operation in the context of the source.

3.2. DOF Protocol Specification, OpenDOF TSC, [dof-2008-1] (7.0.1, 12 January 2018)

General DPP Header `dof-2008-1-pdu-7`



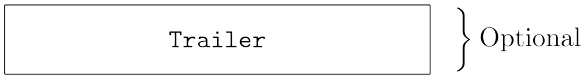
`Flag` **One bit.** Controls the presence of the `Flags` field, which is present if set.

`Version` **Seven bits.** Specifies the DPP version.

`Flags` **Variable length. Optional, controlled by** `Flag`. Contains options defined by the version.

`Headers` **Variable length. Optional, controlled by** `Flags`. Header fields required by the `Version` or controlled by `Flags`.

General DPP Trailer `dof-2008-1-pdu-8`



`Trailer` **Variable length. Optional, as needed.** Trailing data as defined by the `Version`.

Security Mode of Operation Header `dof-2008-1-pdu-9`



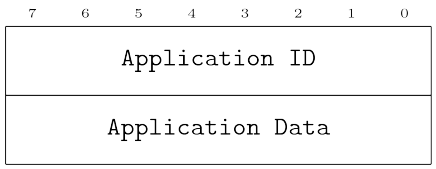
`Fields` **Variable.** The format of the data is controlle dby the mode of operation, which is always known in the context.

Security Mode of Operation Trailer `dof-2008-1-pdu-10`



`Fields` **Variable.** The format of the data is controlle dby the mode of operation, which is always known in the context.

General Application Header `dof-2008-1-pdu-13`

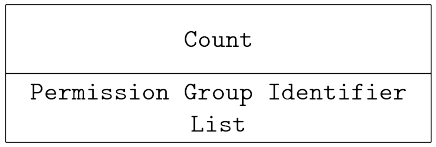


`Application ID` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Contains the application identifier for the enclosed data.

`Application Data` **Optional, variable length.** Contents are defined by the application.

3.3. DOF Security Specification, OpenDOF TSC, [dof-2008-16] (7.0.1, 12 January 2018)

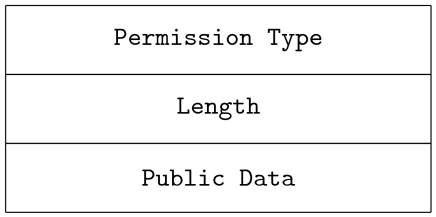
Security Scope `dof-2008-16-pdu-6`



`Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Specifies the length of the **Permission Group Identifiers**).

`Permission Group Identifier List` **Array of** Compressed Unsigned 32-Bit`dof-2009-1-pdu-3`.**Optional, controlled by** `Count`.

Permission `dof-2008-16-pdu-7`



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

`Length`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.**Optional, controlled by** `Permission Type`. Controls presence of **Data**. Present only if the **Permission Type** is odd. This field controls the size of **Data**.
`Public Data`	**Variable length. Optional, controlled by** `Length`. Present only if **Length** is greater than zero. The permission data depends on the **Permission Type**.