Object Access Protocol (V1)

DOF Protocol Specification

Technical Steering Committee

Object Access Protocol (V1): DOF Protocol Specification

Technical Steering Committee

Document Information

Document Tracking Number	dof-2008-2
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	Object Access Protocol (V1), OpenDOF TSC, [dof-2008-2] (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. Overview 9
3. Specification 11
    3.1. Application Identifier Number 11
    3.2. Command/Response Formats 11
    3.3. Interface Definition 13
    3.4. OAP Operation Types 14
        3.4.1. OAP Unicast 14
        3.4.2. OAP Broadcast 15
        3.4.3. OAP Multicast 15
    3.5. Operation Identity and Lifecycle 15
    3.6. Item Identifiers 16
    3.7. Bindings 16
    3.8. Aliases 17
    3.9. Sessions 18
    3.10. Lossless 2-Node Sessions 18
        3.10.1. Protocol Options 19
        3.10.2. Effects of Session Closing 19
        3.10.3. Invalid Opcodes 19
    3.11. Lossy Sessions 20
        3.11.1. Transport Addressing and OAP Addressing 20
        3.11.2. Invalid Opcodes 20
    3.12. Responses 20
    3.13. Command Control 21
        3.13.1. Large Population Operations (Heuristic Percentage, Delay, and Acknowledge List) 23
        3.13.2. Heuristic Percentage 25
        3.13.3. Delay 25
        3.13.4. Do Not Execute 26
        3.13.5. Verbosity 26
        3.13.6. Acknowledgement Lists 27
    3.14. Types and Values 28
    3.15. Permissions 28
        3.15.1. Binding Permission with Wildcards 28
        3.15.2. Binding Permission without Wildcards 32
        3.15.3. Normalization, Comparison, and Complement 33
        3.15.4. Requestor Permission 33
        3.15.5. Provider Permission 34
        3.15.6. Define Permission 34

3.16. Access Control 35
3.17. Domains and Partitions 36
3.18. General Operation State Type 36
3.19. General Routing Behavior 36
3.19.1. Operation Identification 37
3.19.2. Forwarding vs. Flooding 37
3.19.3. Responses 37
3.20. General Consolidation Behavior 38
3.21. Minimally Implemented Providers 38
3.21.1. Transports 38
3.21.2. Operation Lifecycle 38
3.21.3. Mode Switching 39
3.22. Minimally Implemented Requestors 40
4. Command Reference 41
4.1. Command Classes 41
4.1.1. Interest 41
4.1.2. Actions 43
4.1.3. Relationships 44
4.2. Activate 45
4.2.1. Small Implementation Hints 46
4.2.2. Flows 46
4.2.3. Details 46
4.3. Advertise 47
4.3.1. Small Implementation Hints 47
4.3.2. Flows 47
4.3.3. Details 48
4.3.4. Command Control 48
4.3.5. Routing Rules 48
4.4. Change 48
4.4.1. Small Implementation Hints 49
4.4.2. Flows 50
4.4.3. Details 50
4.5. Connect 50
4.5.1. Small Implementation Hints 51
4.5.2. Flows 51
4.5.3. Details 51
4.6. Define 52
4.6.1. Small Implementation Hints 56
4.6.2. Flows 56
4.6.3. Details 56
4.6.4. Routing Rules 57
4.6.5. Consolidation Rules 57
4.7. Exception 57
4.7.1. Small Implementation Hints 59
4.7.2. Details 59
4.8. Full Connect 60
4.8.1. Small Implementation Hints 61
4.8.2. Flows 61
4.8.3. Details 61
4.9. Get 62
4.9.1. Small Implementation Hints 63
4.9.2. Flows 63
4.9.3. Details 63
4.10. Invoke 64

4.10.1. Small Implementation Hints 65
4.10.2. Flows 65
4.10.3. Details 66
4.11. Open 66
4.11.1. Small Implementation Hints 67
4.11.2. Flows 67
4.11.3. Details 67
4.11.4. Command Control 68
4.12. Provide 68
4.12.1. Small Implementation Hints 69
4.12.2. Flows 70
4.12.3. Details 70
4.12.4. Primary Providers 71
4.12.5. Routing Rules 71
4.12.6. Consolidation Rules 71
4.13. Register 72
4.13.1. Small Implementation Hints 73
4.13.2. Flows 74
4.13.3. Details 74
4.13.4. Consolidation Rules 74
4.14. Set 74
4.14.1. Small Implementation Hints 76
4.14.2. Flows 76
4.14.3. Details 76
4.15. Signal 76
4.15.1. Small Implementation Hints 77
4.15.2. Flows 77
4.15.3. Details 78
4.16. Subscribe 78
4.16.1. Small Implementation Hints 80
4.16.2. Flows 80
4.16.3. Details 80
4.16.4. Consolidation Rules 81
4.17. Watch 81
4.17.1. Small Implementation Hints 82
4.17.2. Flows 82
4.17.3. Details 82
5. OAP Sequence Diagrams 83
5.1. Interest 83
5.1.1. Simple Command and Reaction 83
5.1.2. Proxied Command, Delayed Reaction 83
5.2. Object Access 84
5.2.1. Simple Command Response 84
5.2.2. Proxied Command with Immediate Response 84
5.2.3. Command Response with Retry and Response 85
5.2.4. Command Response with Retry including Acknowledge 85
5.3. Subscribe/Register 86
5.3.1. Merged Proxied Command, Delayed Reaction 86
6. External PDU Reference 89
6.1. DOF Common Types, OpenDOF TSC, [dof-2009-1] (7.0.1, 12 January 2018) 89
6.2. DOF Protocol Specification, OpenDOF TSC, [dof-2008-1] (7.0.1, 12 January 2018) 92
6.3. DOF Security Specification, OpenDOF TSC, [dof-2008-16] (7.0.1, 12 January 2018) 93

List of Figures

5.1. Sequence for Simple Command and Reaction 83
5.2. Sequence for Proxied Command and Reaction 84
5.3. Sequence for Simple Command Response 84
5.4. Sequence for Proxied Command Response 85
5.5. Sequence for Command Response with Retry Response 85
5.6. Sequence for Command Response with Acknowledge 86
5.7. Sequence for Merged Proxied Command Response with Reaction 87

List of Tables

3.1. General Verbosity Meanings 27
4.1. Exception Error Codes 59

Specifications and PDUs

OAP Binding-Oriented Command `dof-2008-2-pdu-1` 11
OAP Item-Oriented Command `dof-2008-2-pdu-2` 12
OAP Response `dof-2008-2-pdu-3` 13
Handle all OAP operation types and flooded operations. `dof-2008-2-spec-1` 14
Manage operation lifecycle. `dof-2008-2-spec-2` 15
Item Identifier `dof-2008-2-pdu-4` 16
Binding `dof-2008-2-pdu-5` 16
Valid alias values must be different from 0 (zero). `dof-2008-2-spec-3` 17
The Aliases must remain valid and not reused until the declaring Provide command would time out, not affected by cancellation, and the time out should account for clock skew. `dof-2008-2-spec-4` 17
Aliases are numbers, passed MSB first, and nodes should use the smallest space possible. `dof-2008-2-spec-5` 18
Requestors must not use aliases with ambiguous targets. `dof-2008-2-spec-6` 18
DSP Application Attribute `dof-2008-2-pdu-6` 19
Invalid opcodes cause lossless 2-node sessions to close. `dof-2008-2-spec-7` 19
Invalid opcodes on lossy sessions cause the receiver to drop the datagram. `dof-2008-2-spec-8` 20
Providers send completions, not intermediate nodes. `dof-2008-2-spec-9` 20
Responses must originate from a compatible security partition. `dof-2008-2-spec-10` 21
Use the OAP provider identifier in all responses where the responder is ambiguous. `dof-2008-2-spec-11` 21
Command Control `dof-2008-2-pdu-7` 22
Providers must support the heuristic behavior of command control. `dof-2008-2-spec-12` 25
Providers must support the delayed execution behavior of command control. `dof-2008-2-spec-13` 26
Providers must support the do not execute behavior of command control. `dof-2008-2-spec-14` 26
Providers must understand and should follow verbosity behavior of command control. `dof-2008-2-spec-28` 27
Providers should recognize acknowledgements and prevent further responses. `dof-2008-2-spec-15` 27
Providers should recognize pre-acknowledgement. `dof-2008-2-spec-16` 27
Value List `dof-2008-2-pdu-8` 28
Binding Permission with Wildcards `dof-2008-2-pdu-9` 28
Interfaces must appear in the permission in increasing sorted order. The permission does not allow the wildcard IID. `dof-2008-2-spec-17` 30
Objects must appear in the permission in increasing sorted order. The permission allows the Broadcast OID. The permission does not allow attributes on OIDs. `dof-2008-2-spec-18` 30
Attributes must appear in the permission in increasing sorted order Each Attribute Identifier may appear in the list at most once. `dof-2008-2-spec-19` 30
Binding Permission Attribute `dof-2008-2-pdu-10` 30
Binding Permission without Wildcards `dof-2008-2-pdu-11` 32
Requestor Permission `dof-2008-2-pdu-12` 33
Provider Permission `dof-2008-2-pdu-13` 34
Define Permission `dof-2008-2-pdu-14` 34
Nodes must correctly request and enforce permissions on secure sessions. `dof-2008-2-spec-20` 36
Non-routing nodes ignore DPP FLOODED operations that do not match a provided binding. `dof-2008-2-spec-21` 37
Activate (command) `dof-2008-2-pdu-15` 46
Advertise (command) `dof-2008-2-pdu-16` 47

Change (command) `dof-2008-2-pdu-17` 49
Connect `dof-2008-2-pdu-18` 50
Define `dof-2008-2-pdu-19` 52
Define `dof-2008-2-pdu-20` 52
Interface Definition `dof-2008-2-pdu-43` 53
Property `dof-2008-2-pdu-21` 54
Event `dof-2008-2-pdu-22` 54
Method `dof-2008-2-pdu-23` 55
Exception `dof-2008-2-pdu-24` 55
Each Define response item list must appear in order of increasing Item Identifier. `dof-2008-2-spec-22` 57
Duplicate Item Identifiers in a Define response are invalid. `dof-2008-2-spec-23` 57
Internal Exception (response) `dof-2008-2-pdu-25` 58
Provider Exception (response) `dof-2008-2-pdu-26` 58
The ENC-TC controls Exception error codes. `dof-2008-2-spec-24` 60
Full Connect (command) `dof-2008-2-pdu-27` 61
Get (command) `dof-2008-2-pdu-28` 62
Get (response) `dof-2008-2-pdu-29` 63
Invoke (command) `dof-2008-2-pdu-30` 64
Invoke (response) `dof-2008-2-pdu-31` 65
Open (command) `dof-2008-2-pdu-32` 66
Open (response) `dof-2008-2-pdu-33` 67
Provide (command) `dof-2008-2-pdu-34` 69
Multiple bindings expressed from the same node must use correct operation identifiers. `dof-2008-2-spec-25` 70
Multiple identical bindings provided from the same node must use the same alias. `dof-2008-2-spec-26` 70
Providers must not use the same alias for different bindings at the same time. `dof-2008-2-spec-27` 71
Register (command) `dof-2008-2-pdu-35` 72
Register (response) `dof-2008-2-pdu-36` 73
Set (command) `dof-2008-2-pdu-37` 74
Set (response) `dof-2008-2-pdu-38` 75
Signal (command) `dof-2008-2-pdu-39` 76
Subscribe (command) `dof-2008-2-pdu-40` 78
Subscribe (response) `dof-2008-2-pdu-41` 79
Watch (command) `dof-2008-2-pdu-42` 81
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1` 89
Compressed Unsigned 32-Bit `dof-2009-1-pdu-3` 89
Object Identifier (OID) `dof-2009-1-pdu-4` 90
Object Identifier Attribute `dof-2009-1-pdu-5` 90
Interface Identifier (IID) `dof-2009-1-pdu-21` 91
Type Identifier `dof-2009-1-pdu-23` 91
Type Context `dof-2009-1-pdu-39` 92
Type Reference `dof-2009-1-pdu-40` 92
Attribute/Value Pair `dof-2008-1-pdu-16` 92
Permission `dof-2008-16-pdu-7` 93

1. About This Document

This document contains information about the Object Access Protocol (OAP). This is critical information for anyone who needs to work with the protocol.

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.

This document is managed by the Technical Steering Committee of the OpenDOF Project, Inc. (ODP-TSC). Contact information is at the beginning of 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.

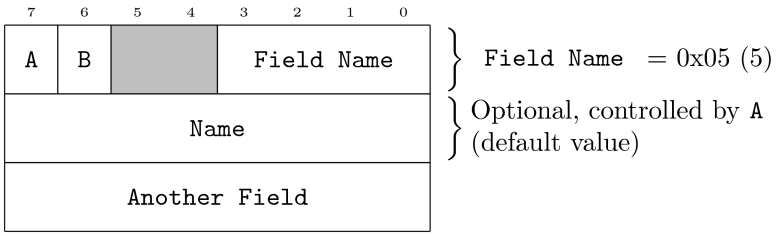
1.3.4. Fields

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. Overview

This document describes the Object Access Protocol, or OAP. The Object Access Protocol defines how a provider or a requestor interacts with other providers and requestors in a network to exchange object information and to allow requestors to manipulate those objects.

OAP is a flexible protocol that allows a wide variety of functionality. At the same time, however, it can be implemented (in restricted forms) on very small processors. This is possible because a node only needs to implement certain aspects of the protocol. Throughout this document, there are hints at how to minimize implementation complexity.

Some nodes fully implement all aspects of the protocol. These nodes can provide functionality for more restricted nodes, which is one of the powerful aspects of OAP.

OAP is implemented as an application protocol in the DOF Protocol Stack, which is fully discussed in the *DOF Protocol Specification*.

The following sections more fully describe aspects of OAP.

These first sections deal specifically with the background of OAP. Everyone that works with OAP should understand the information in these sections. Following the general information are sections that deal more specifically with OAP over lossless and lossy transports. Finally, there is information on each command and response of OAP.

OAP can be negotiated using DSP. Nodes may act as providers, requestors, or both, based on the types of commands that they send and respond to. There is no indication of the role that a node performs, and the role may change over time.

The function of a provider is to react to interest commands and respond by providing bindings. The function of a requestor is to find providers and then issue commands using the provided bindings. Nodes that are both provider and requestor at the same time can act as proxy for other providers and requestors.

Providers may also support subscriptions to both properties and events. Subscriptions allow optimized access by potentially consolidating updates on the network.

OAP can operate securely, and uses several permissions to control access to bindings. OAP can use the partitioning provided by DPS and DOF Security to restrict visibility of bindings to different groups. OAP also supports its own addressing (the binding), along with definitions for OAP broadcast and OAP multicast.

3. Specification

This section contains the specification of the Object Access Protocol.

3.1. Application Identifier Number

The APPID for this version of the Object Access Protocol is 1 (one).

3.2. Command/Response Formats

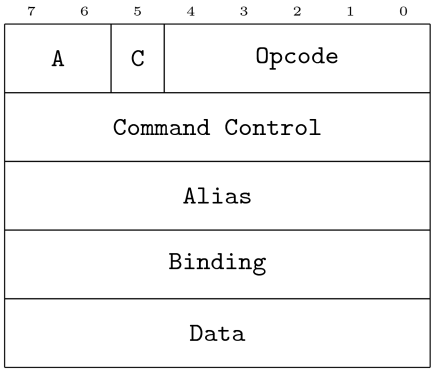
There are two major categories of OAP command: binding-oriented commands and item-oriented commands. All OAP commands fall into one of these categories, and the category determines the format of the associated PDU.



All OAP command and response formats begin with a command byte that has flags in the upper three bits and the opcode in the lower five bits. Further, all OAP command formats use the upper three bits identically (to indicate alias length and command control).



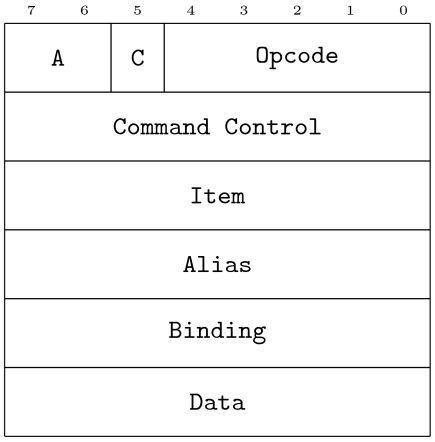
OAP Binding-Oriented Command `dof-2008-2-pdu-1`



`A`	**Two bits.** Controls the format of `Binding`.
`C`	**One bit.** Controls the presence of `Command Control`.
`Opcode`	**Five bits.** Determines the type of the command.
`Command Control`	**Instance of** Command Control$_{\texttt{dof-2008-2-pdu-7}}$.**Optional, controlled by** `C`. If `C` is one then this field is present. This field determines options that apply to the command.
`Alias`	**Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.
`Binding`	**Instance of** Binding$_{\texttt{dof-2008-2-pdu-5}}$.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

`Data`	**Variable length. Optional, controlled by** `Opcode`. Data for the command depending on the `Opcode`.

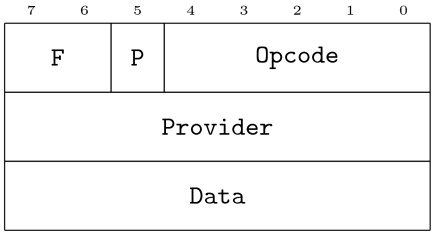
OAP Item-Oriented Command `dof-2008-2-pdu-2`



`A`	**Two bits.** Controls the format of `Binding`.
`C`	**One bit.** Controls the presence of `Command Control`.
`Opcode`	**Five bits.** Determines the type of the command.
`Command Control`	**Instance of** Command Control`dof-2008-2-pdu-7`.**Optional, controlled by** `C`. If `C` is one then this field is present. This field determines options that apply to the command.
`Item`	**Instance of** Item Identifier`dof-2008-2-pdu-4`.Defines the item for the command.
`Alias`	**Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.
`Binding`	**Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.
`Data`	**Variable length. Optional, controlled by** `Opcode`. Data for the command depending on the `Opcode`.

There is a single response format.

⇄ **OAP Response** `dof-2008-2-pdu-3`



`F` **Two bits.** Flags based on the `Opcode`.

`P` **One bit.** Controls the presence of `Provider`.

`Opcode` **Five bits.** Determines the type of the response.

`Provider` **Instance of** Object Identifier (OID)`dof-2009-1-pdu-4`.**Optional, controlled by** `P`. If `P` is one then this field is present. Identifies the provider of the response. This field is required in responses to OAP multicast and broadcast commands. Responses to OAP unicast commands should not use this field.

`Data` **Variable length. Optional, controlled by** `Opcode`. Data for the command depending on the `Opcode`.

3.3. Interface Definition

OAP makes extensive use of interfaces. This section discusses some general principles of interfaces.

Interface definitions may take many forms. The DOF protocols define a binary representation that is suitable for passing over networks. It also defines a command and response that nodes can use to pass the definitions. This is not to imply that the specified binary representation is the best representation, but is easy for programs to work with. Other methods of representing interface definitions could include XML files, other text representations, and even other binary representations.

OAP does not define how to exchange meta-data about interface definitions. This is because this alternate information often relates directly to particular programs, which may place their own requirements on the format and content of the meta-data. For example, there is nothing in the OAP binary representation to describe the formatting requirements of the data. A program that needs to display data for a user may be very concerned with this additional type of information.

This specification does not discuss the means of obtaining this kind of meta-data. There is also other information missing from OAP interface definitions. For example, there is no range information stored for numeric values of choice lists for strings. OAP defines how typed information appears on the wire, and does not include all of the information to define a particular value. This additional information could also be part of the meta-data of the interface.

Each interface (an associated interface identifier and interface definition) is unique. This means that even two interface definitions that include all of the same methods, properties, events, and exceptions are not the same if their interface identifiers are different. The converse is problematic: it is a serious error to have two interface definitions with the same interface identifier but that have different definitions. DOF specifications do not address this type of error, other than to establish permissions on who may define interfaces. Unsecured systems should protect themselves against malicious interface definitions, usually by requiring that all providers and requestors include them so there is no need to ask any other node.

Each different interface identifier must represent a different interface definition. It is anticipated (and encouraged) that the semantics of an interface be represented through the interface definition and identifier. This means, for example, that an interface that returns the current temperature using a single 16-bit fixed-point number and an interface that returns the current light level using the same type (16-bit fixed point) should not use the same interface identifier, even though the specifics of the interface definition may be identical.

In that example, the syntax of the two interfaces is identical. They both appear identical on the wire. However, using the same interface identifier makes it difficult for a thermostat to find the current temperature without confusing it with the current light level.

The specific design considerations for interfaces are beyond the scope of this document. It is a critical aspect of application design, as is the appropriate re-use of existing interface definitions.

The wire format of interface definitions (defined as part of the OAP.13.2: Define (response)) ensures that identical interface definitions will have the same length and a byte comparison will be identical.

3.4. OAP Operation Types

OAP supports three different major types of operations: unicast, broadcast and multicast.



Do *not* confuse OAP unicast, broadcast and multicast with transport unicast broadcast and multicast. They have very similar meanings, but there is not a one-to-one mapping between them. Senders may use transport unicast to send an OAP multicast operation. Similarly, an OAP unicast datagram may use a transport broadcast datagram.

+ This means that the capabilities of OAP are independent of the capabilities of the transport (or transports) that it runs on. This is a key DOF design feature.

OAP operations can be flooded using the DOF Presentation Protocol. This means that nodes can send an operation to many other (or all other) nodes in the network. This is possible when the source of a binding is unknown, or when OAP broadcast or multicast is used. The capabilities of DPP are required for sending a flooded operation to avoid looping/ringing.



Handle all OAP operation types and flooded operations. `dof-2008-2-spec-1`

All nodes must respond correctly to all OAP operation types (unicast, multicast, broadcast). Nodes should optimize routing by using directed operations and only fall back to flooded operations if necessary.

3.4.1. OAP Unicast

OAP unicast operations target a specific binding. This binding must not use the broadcast OID class or the wildcard IID. Note that while typical networking protocols do not have multiple instances of addresses, OAP encourages this practice. This is a primary method of providing redundancy. Because of this, the rules for OAP unicast are more complicated than transport unicast. However, each node determines a single provider for any given binding.

OAP unicast relies on binding route information built up by the node. If there is no routing information available for a given binding and there could be undetected providers then the node should flood the operation if allowed by the operation specification.

3.4.2. OAP Broadcast

OAP broadcast targets all providers of a particular interface. It uses a binding that contains the broadcast OID class, with no attributes, and paired with a specific IID (non-wildcard). Unfortunately, broadcast operations cannot benefit from routing information in the general case. This means that OAP broadcast must flood the network. However, if on a particular session all the bindings are known, then the sender can avoid flooding and send individual operations.

3.4.3. OAP Multicast

OAP multicast targets a specific group of providers of a particular interface. It uses a binding that contains the broadcast OID that includes another attribute or attributes, paired with a specific IID (non-wildcard). In this case, the binding matches all bindings whose object identifiers contain at least the specified attribute values.

OAP multicast has the same issues with routing as broadcast in the lack of ability to optimize based on routing information.

Note that there is not a sense of "membership" in OAP multicast. Systems that desire membership control should use permission assignment. Since OAP multicast uses an OID attribute, it is relatively simple to control access through the binding permission.

In particular, there is no direct relationship between transport multicast groups (even secure groups) and OAP multicast.

Providers specify a group for a binding by adding a specific attribute or attributes. It is possible to add the same object to multiple 'groups', but only by providing the binding multiple times with different attribute values. Access to a binding that is part of a group is not possible without using the correct attributes, unless the provider adds the binding without the attribute.

3.5. Operation Identity and Lifecycle

OAP utilizes the lifecycle defined by the DOF Presentation Protocol. DPS will notify OAP of all appropriate changes to lifecycle, and OAP reacts to these changes. In many cases the behavior will be to forward the operation (again, or for the first time) to a set of other nodes. In some cases, it will cause the update of local state.

Unless specifically stated otherwise, all operations allow retries. These retries have the effect of extending the operation’s lifecycle and generating responses as directed. In addition, operations that cause reactions (such as the interest operations, defined below) cause the reactionary operations to extend their lifecycle – causing a relationship between the lifecycles.

Providers send responses using the facilities of DPS. Each operation has a source, and nodes send responses back to the source. OAP also uses a variety of reactionary operations, and depending on the type of operation, OAP supports fan-out and fan-in.



Manage operation lifecycle. `dof-2008-2-spec-2`

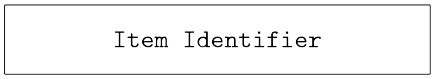
A node must correctly manage operation lifecycle. This includes enforcing appropriate timeouts as well as retry and cancel behaviors.

3.6. Item Identifiers

Properties, events, methods, and exceptions are *items*. Each has an identifier, as discussed earlier. Items have no meaning unless combined with a binding, so items and bindings typically appear together.

Item identifiers use the following format:

⇄ **Item Identifier** `dof-2008-2-pdu-4`



`Item Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The item identifier.

3.7. Bindings

Most commands in OAP include a binding. As discussed above, a binding is a combination of a single object identifier with a single interface identifier. One of the primary requirements of OAP is to exchange information about the presence of bindings.

Bindings are typically large (requiring space to identify both the OID and IID). The next section on aliases discusses how to optimize the information sent on the wire.

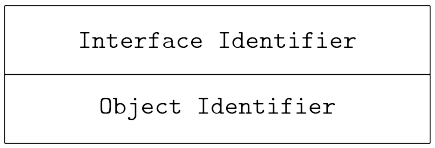
OAP allows multiple routes for the same binding. However, each node must choose which route to use. In many cases, the route can change without causing any problem, but there are some commands (like subscriptions) where route changes cause special behavior. OAP requires that the routing for each binding form a DAG. Later sections discuss the creation of the DAG for a binding.

Operation lifecycle maintains routing information. When the parent of an operation that controls routing changes or a node chooses an alternate route, then that may cause other OAP state to change.

In some cases, a single binding may actually refer to multiple "compatible" bindings (OAP multicast and broadcast, discussed above). This definition of compatibility depends on matching the components of the bindings. Multiple matches may occur when the components contain wildcard interface identifiers or broadcast object identifiers (including multicast).

For example, a binding that includes a broadcast OID would match all bindings with an equivalent IID (because broadcast matches all OIDs). The most general binding includes a broadcast OID and a wildcard IID – it is compatible with *all* bindings.

⇄ **Binding** `dof-2008-2-pdu-5`



`Interface Identifier`	**Instance of** Interface Identifier (IID)`dof-2009-1-pdu-21`.The Wildcard IID is not allowed unless specifically mentioned in the command details.
`Object Identifier`	**Instance of** Object Identifier (OID)`dof-2009-1-pdu-4`.The object identifier may be specific, broadcast, or multicast unless an operation states otherwise.

3.8. Aliases

Commands in OAP that use bindings are usually able to utilize aliases as shortcuts for specifying a full binding (object identifier and interface identifier). Aliases are always an optimization, and are never required. However, they provide a major savings in network bandwidth, and so nodes should use them when possible. An operation may switch between using aliases or bindings.

Aliases are associated with a binding in the Provide command. A provider is not required to support an alias, and if it chooses not to then an alias value of zero is used.



Valid alias values must be different from 0 (zero). `dof-2008-2-spec-3`

The alias value must be different from 0 (zero), which is reserved.

Unlike DPP operation identifiers, aliases are *not* universal. An alias definition exists only between a requestor and a provider. A provider is free to assign a different alias to the same binding in different scopes (session, group, etc.). However, once an alias is associated with a particular binding within a certain scope (either a session, or between two nodes for a given time period), then that alias must not be used for a different binding in that same scope for at least the longest potential lifetime of the provide operation. This prevents ambiguity in the case of bindings being removed, and then added again. Note that even if a node adds, removes, and then adds a binding again it may use the same alias.



The Aliases must remain valid and not reused until the declaring Provide command would time out, not affected by cancellation, and the time out should account for clock skew. `dof-2008-2-spec-4`

Aliases are declared through the Provide operation, and that declaration remains valid until the declaring operation would enter the Timed Out state or become invalid because the session associated with the alias has been closed. In other words, operation cancellation or shortening other than session loss is *not* allowed to shorten this time. Otherwise, a node that misses the other operations may be in an inconsistent state regarding the meaning of the alias - and that node may use the alias up to that time. Note that when a node reboots it will likely lose its memory of assigned aliases. If that node used long durations on Provide commands over lossy sessions then it will not know whether it is causing conflict. Over lossless sessions there is no problem, but if the Provide command was used on a lossy session then the alias must either remain associated with the same binding or must not be used until the command is known to have been completed (by timeout in this case).

Nodes send aliases as variable length numbers.

Aliases are numbers, passed MSB first, and nodes should use the smallest space possible. `dof-2008-2-spec-5`

The space allocated is inherent in the command that uses the alias. Aliases are equal if the numbers are equal, independent of the size used to pass the number.

All of the commands that allow aliases use the top bits of the opcode as an **A** field. This field controls the size and use of the alias as follows:

- 0: No alias, a full binding is present.
- 1: The alias is passed in one byte.
- 2: The alias is passed in two bytes.
- 3: The alias is passed in four bytes.

Providers use aliases in any context, including transport multicast. The provider must be prepared to understand the alias in whatever context the provider used it.

Requestors can only use an alias using transport unicast datagrams (or its equivalent, meaning that a single identified target). Any attempt by a requestor to use an alias where the target is ambiguous is an error. If a command arrives that uses an improper or undefined alias then the command must be silently discarded.

Requestors must not use aliases with ambiguous targets. `dof-2008-2-spec-6`

Because aliases are not global, it is meaningless for a requestor to attempt to use an alias without reference to a specific provider. Receivers must ignore ambiguous aliases so they will ignore the offending command.

3.9. Sessions

Typically, a session refers to a relationship between two nodes. This is true whether the network connection is persistent or not, similar to HTTP sessions. OAP sessions are object-based, and therefore are not node-to-node, but rather requestor-to-provider.

Requestors request OAP sessions on a particular binding. If the provider allows the session, it creates a new object that represents the session. This new object combines the object identifiers of the binding and the session, which is part of the Open command. One benefit of sessions is in the access rules, which are set to restrict access to the session to a single point in the system – the requestor that initiated the session.

Beyond the definition of "ownership" of the session, nodes handle access control for bindings on sessions the same as for all other bindings in the system.

Object identifiers usually allow for distributed bindings, meaning that multiple providers can add bindings using the same object identifier. This is not true for bindings provided on sessions. First, if two requestors attempted to use the same session identifier then the creation would fail because of a duplicate session object. This duplication problem prevents another provider from trying to use the object identifier because it could never receive an Open command that would include the same session identifier.

3.10. Lossless 2-Node Sessions

The following sections deal with requirements and background for using OAP over lossless transport sessions.

Nodes negotiate options that affect the behavior of OAP on a session during session establishment. The DOF Session Protocol (DSP) is used for this negotiation. OAP is a peer-to-peer application protocol, and if either node negotiates its use then both nodes may use it freely.

3.10.1. Protocol Options

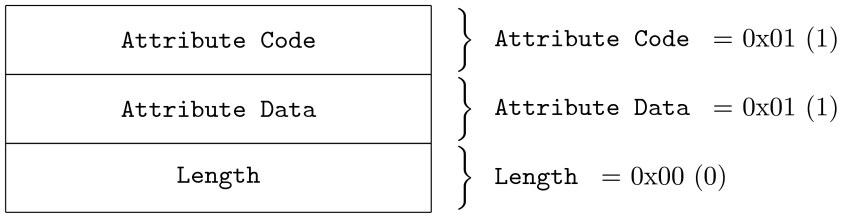
The following sections describe the DSP Attribute-Value Pairs for the OAP protocol. These include the entire AVP format.

This section presents options from the *requesting node's point of view*. A node is responsible to negotiate not only what it wants the other node to do, but also what it wants to do.

OAP does not have any negotiated options, and so the following format is used.

DSP Application Attribute `dof-2008-2-pdu-6`

Instance of Attribute/Value Pair`dof-2008-1-pdu-16`



`Attribute Code` **One byte.** This field identifies the attribute. It must be registered with the ODP-TSC.

`Attribute Data` **Two bytes, MSB.** This field completes the definition of an attribute in combination with the code.

`Length` **One byte.** This field determines the length of the ???. It may be zero, in which case the ??? field is not present.

3.10.2. Effects of Session Closing

Session closure has a tremendous effect on outstanding operations. The definition of these effects is defined the DOF Protocol Stack.

3.10.3. Invalid Opcodes

Invalid opcodes must cause lossless 2-node sessions to close immediately.

Invalid opcodes cause lossless 2-node sessions to close. `dof-2008-2-spec-7`

Invalid opcodes signal either corrupted communication channels or loss of state.

3.11. Lossy Sessions

The following sections discuss using OAP over lossy sessions. A fundamental goal of OAP is to minimize the differences between session types. This is more difficult when using lossy sessions because of the lack of shared state between the sender and receiver. This means that commands and responses must pass additional information.

Lossy transports typically have multiple ways of addressing destination nodes. The terms used for these different addressing modes are identical to the terms used for OAP addressing, and so we denote the difference by using the prefixes "Transport" and "OAP."

3.11.1. Transport Addressing and OAP Addressing

All OAP addressing modes (unicast, multicast, and broadcast) are available on all transport addressing modes (unicast, multicast, and broadcast).

3.11.2. Invalid Opcodes

Invalid opcodes must cause the receiver to ignore the datagram.



Invalid opcodes on lossy sessions cause the receiver to drop the datagram. `dof-2008-2-spec-8`

Invalid opcodes signal either corrupted communication channels or loss of state.

3.12. Responses

Commands usually cause responses. In fact, almost all commands in OAP have some sort of response or cause some kind of behavior resulting in nodes returning datagrams.

Whether or not a response terminates an operation depends on the operation, not the result. For example, an Invoke response terminates an OAP unicast Invoke command, but may not terminate an OAP multicast or broadcast Invoke command. Whether the operation will terminate is determined through the DPP lifecycle controls (either setting DPP INTERMEDIATE or FINAL, in conjunction with the SINGLERESPONSE/MULTIRESPONSE flag).

Note that the OAP Exception response is the only response that can use the DPP INTERMEDIATE indicator.

There are no timeouts defined in OAP other than those of the operation.



Providers send completions, not intermediate nodes. `dof-2008-2-spec-9`

A core aspect of OAP is that providers send completions.

Nodes must not allow commands and responses to cross security boundaries. This means that nodes must take care to ensure (particularly on multicast and broadcast operations) that they evaluate each potential response to ensure that the sender is compatible with the requestor.

This must apply even in cases where the information *should* be the same. For example, the Define command must not receive a Define response from an untrusted source.



Responses must originate from a compatible security partition. `dof-2008-2-spec-10`

Nodes must receive all responses from a compatible security domain to that of the request. Nodes do this evaluation link by link, and they must ensure that there is no crossover.

In order to allow for acknowledgement and discovery, the requestor must know who to acknowledge or who is responding. Acknowledgements are by binding, and so the rule is that if the command does not use OAP unicast then the Provider field must be present to indicate the responding object identifier. This field must be present in all responses where it is appropriate.



Use the OAP provider identifier in all responses where the responder is ambiguous. `dof-2008-2-spec-11`

This usually means that the original operation uses either OAP multicast or broadcast.

3.13. Command Control

OAP is unique in its ability to use OAP multicast and broadcast to retrieve large amounts of data. It is also unique in its use of distributed state to control the behavior of other commands. Command control refers to options that available in OAP to control both of these behaviors. Command control applies to providers and not to proxy nodes.

Many OAP commands cause state changes that may cause reactions in addition to responses. Prime examples are the interest commands, which may generate many provide commands directed back at the sender.

In some cases, the amount of traffic generated by a command is unknown. This can be problematic both for the network and for the requestor.

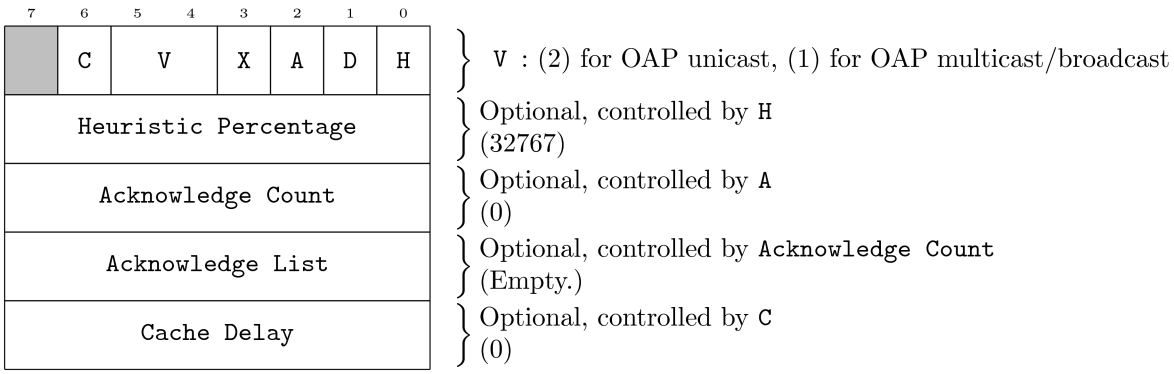
There are typically two solutions to this problem: spread the traffic over time, or minimize the traffic by selecting responders using some heuristic algorithm. OAP supports both methods, including combinations of the two.

First, commands can specify an execution delay. Note that this is not the delay of the *response* but of the *command*. Many commands generate state changes that generate other commands. In this case, the option delays the state change itself (including the reactions to the command).

Second, commands can specify execution on a random subset of the receivers. By starting with a small percentage, the generated responses provide a guess at the total population of potential responders. This guess adjusts the percentage, or the command delay. In this way, the requestor can tailor the rate of response for its benefit and for the benefit of the network.

The following PDU defines the general command control format.

Command Control `dof-2008-2-pdu-7`



`C`	**One bit.** Controls the presence of `Cache Delay`.
`V`	**Two bits.** Controls the command verbosity.
`X`	**One bit.** Controls the command execution.
`A`	**One bit.** Controls the presence of `Acknowledge Count`.
`D`	**One bit.** Controls the command delay.
`H`	**One bit.** Controls the presence of `Heuristic Percentage`.
`Heuristic Percentage`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.**Optional, controlled by** `H`. If `H` is one then this field is present. Controls the random choice of responders.
`Acknowledge Count`	**One byte. Optional, controlled by** `A`. If `A` is one then this field is present. Controls the length of the `Acknowledge List`.
`Acknowledge List`	**Array of** Object Identifier (OID)`dof-2009-1-pdu-4`.**Optional, controlled by** `Acknowledge Count`. If `Acknowledge Count` is non-zero then it is the length of this list. Contains a list of providers to acknowledge.
`Cache Delay`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.**Optional, controlled by** `C`. If `C` is one then this field is present. Determines if providers can use cached data in responses. The default value of zero means cached data is not allowed. Otherwise, it contains the maximum age (in seconds) of data that may be returned.

Commands may use these fields in any combination, with the results controlled as specified by the particular command. Further, the fields may change (which will change the effects) for each operation retry. This allows the behavior to change as the results arrive. Note, however, that not all commands utilize all of the fields. For example, a command that has no responses would not use the **V** field. All commands must understand and ignore any fields that do not apply.

The following sections discuss the use of the specific fields.

3.13.1. Large Population Operations (Heuristic Percentage, Delay, and Acknowledge List)

The following discussion relates to using command control for operations with possibly large, but unknown populations of providers. This section discusses three related options: **Heuristic Percentage**, **Delay**, and **Acknowledge List**.

For the example, assume a system with between zero and 10,000 nodes. The current population is 1,000 nodes. The example will show the effect on two categories of operation: those that create state and responses, and those that do not create state unless requested. The goal is to keep the *response* traffic below 10 responses per second. This means that the shortest time to get responses from all nodes is 100 seconds. A good algorithm will approach this minimum total operation time. *Response* includes commands sent in reaction to state-creating operations like Watch. Finally, for the example, assume that the network has infinite bandwidth and is lossless. This simplifies the example, even though it is never true. The principles described here apply in same way to real-world network behaviors.

First, assume that no command control is used. In this case, all of the nodes will respond within a very short time (approaching instantaneous). The datagram rate will approach infinity (1,000 nodes in instantaneous time). This is clearly not acceptable.

Second, assume that the requestor can control the nodes so that they respond at a random time during the operation. This is the function of the **Delay** field. In this case, the requestor can use duration of 100 seconds and set the **Delay** field. On average, the datagram rate will be 10 responses per second. This setting achieved the goal, *except* that the requestor had to know the population count beforehand. If the requestor had used 10 seconds instead of 100 then the goal would not be met (100 responses per second). On the other hand, if the requestor had chosen 1000 seconds the first goal is met (one response per second, on average) but the second goal is not (it took 1000 seconds, or ten times the optimal total delay to get the responses). This shows that **Delay** works well with a known population.

Also note that if the population is not known and can vary greatly (say between 1 and 10,000 nodes), then **Delay** alone is very difficult to use. The requestor is forced to guess at the population, but if it guesses a duration too low then the network can be easily saturated with a lot of traffic and if it guesses a duration too high then the time to get all the responses (or any response at all) can be much too long. One could argue that the solution is to use a long duration to begin with, and then shorten it over time. This works to some degree, but it can easily be fooled because of the real-world effects of collision on the network. Collisions make it appear to the requestor that the system is working, even though in reality traffic is saturating the network. This amplifies another bad effect (in real networks) that if response collisions start to occur that the ability of the requestor to control the system is greatly diminished because his own attempts to send datagrams collide with the responses. This fact forces a **Delay**-only approach to start with large times and move down, increasing total delay.

A solution is to use the **Heuristic Percentage** in combination with **Delay**. Its effect is to select a percentage of the population and have them respond. The benefit is that the requestor can determine, using datagram rate and probability, the population in the system without committing itself to a particular duration in a way that causes network saturation. To use **Heuristic Percentage** the requestor guesses (or is configured with) the likely maximum population. It then determines the percentage based on this duration and the desired maximum datagram rate. In our example, the node knows the maximum of 10,000 nodes and the desired datagram rate of 10 responses per second. Without randomness, the duration should be 1,000 seconds worst case. Knowing that the actual population may be much less, the requestor decides to use a very short duration and a small percentage of responders. For example, the requestor chooses 1%, with 10-second duration. The number of responses during that period is determined. In the case of the actual population of 1,000 nodes, the number should be 10 (one response per second, on average). At this point the requestor has a guess at the total population (10 responses total is 1% of the population, so the total population is 1,000). At that point, the requestor switches to using **Delay**, with duration of 100 seconds.

Continuing the example of **Heuristic Percentage** shows the effect of different populations. Say that *no* responses arrive in the initial 10 seconds. That means that the operation created no state on any nodes (assuming a lossless transport, as is the case with this example). Using probability the maximum actual population is now less than 100. Anything much larger should have resulted in at least one response. There are two possibilities. The first is that the requestor can switch to **Delay**, assuming the maximum population (yielding duration of 10 seconds for a maximum population of 100 nodes). The second is that the requestor can retry the operation using a higher percentage. In this case, it can choose 10%, with 1-second duration. If no responses arrive then the maximum population is 10. Again, the requestor can switch to **Delay**, this time using 1 second. This method can continue until the duration is at one second (the minimum allowed). Using **Heuristic Percentage** resulted in total responses in 11 seconds in the case of a small population.

However, say that the total population is actually 5,000. In the initial attempt (1%, 10-second duration), 50 responses were received in the 10 seconds. At this point, the operation created state on those 50 nodes, and so the requestor should switch to **Delay**. However, the requestor has a good indication that the total population is 5,000, and so can use the appropriate duration (500 seconds).

This method works well up to the point (discussed earlier) where the number of responses received approaches network saturation. This did not apply in the example, but does apply in the real world. In this case, the requestor has a difficult choice to make. For example, assume a population of 100,000 nodes. This represents 10 times the maximum assumed population. The initial request (1%, 10-second duration) resulted in 100 responses per second (1,000 total responses during 10 seconds). Assuming that this is approaching network saturation, then the actual population may be much higher. These other nodes are "hidden" due to collisions on the network. However, the 1,000 nodes now have state created, and it will expire (the operations will expire) in a short time. The node can switch the operation to **Delay** with the duration set for the guessed population, and then monitor the continued response rate and change the duration if it continues to be too high.

Assume, for the example, that the actual population is 200,000 nodes, but with half of the responses masked because of collision. The operation switches to **Delay** based on an assumed population of 100,000 nodes. In this case, the duration should be 10,000 seconds. DOF specifications do not support this duration currently, but the example will continue assuming it is. The actual datagram rate drops from 100 responses per second to 20 responses per second. At this point network congestion is probably not an issue, resulting in a more accurate estimate of total population. The operation can be retried with a duration of 20,000 seconds (also not supported), and the datagram rate drops to the desired 10 responses per second.

The preceding examples use operations that create state and generate responses. For operations that do not create state by default, the situation changes slightly. In this case the problem of the state timing out (encountered with very large populations) does not exist in the same way. Since requestors must individually acknowledge responders in this case, a different approach to **Heuristic Percentage** can be used called "continuous discovery." In this case, **Heuristic Percentage** is used, without **Delay**, but with **Acknowledge List**. The requestor continues to retry the operation with increasing percentages until reaching the desired response rate. For each response, the requestor acknowledges the responder. This process continues until the percentage is 100%. At that point, no further responses occur unless new providers join the network.

All of these methods rely on relative stability of the population of providers. The requestor monitors the number of responses per second, with large fluctuations used to modify the behavior of the system. However, the continuous discover method is subject to severe swings in traffic if a large population of nodes suddenly joins the network after a period of stability. In this case, the next retry with 100% may cause excessive response traffic. Because of this, when the initial discovery is accomplished a good idea is to switch to **Delay**. In this way, the responses will at least spread out over time, with the operation further modified if the datagram rate goes too high.

The following sections discuss the specific behavior of each of these options and present the specifications for implementations.

3.13.2. Heuristic Percentage

The **Heuristic Percentage** ranges from zero to 32767. The value of zero is equivalent to telling the command not to execute and has the exact same effect as setting the **X** bit. The value 32767 indicates 100% of the receiving nodes should execute, and is the same as not including the **Heuristic Percentage** field at all. If state exists for the operation, meaning the operation has executed and created state or that the requestor acknowledged it, then the receiver ignores this option.

For nodes without state, the behavior depends on the value of **Heuristic Percentage**, and is not deterministic. In this case, the node must pick a random number between 0 and 32767. If the number chosen by the receiver is less than the value specified in the command then the command executes, otherwise the receiver ignores the command. Executing the operation may, or may not, create state automatically. If it does, then the receiver ignores the option in the future. If it is does not, then the same logic (the choosing of a random number, followed by possible execution) will occur again.

This means that using **Heuristic Percentage** without **Delay** (discussed later) on an operation that does not automatically create state will result in repeated execution on some percentage of nodes, and a different set of providers executing each time. This is not a good use of the option. In these cases, the requestor should acknowledge the responders. However, in this case, retries will result in a decreasing number of responders each time. The typical use would then be to acknowledge responders and increase the percentage on each retry.

For operations that create state automatically, requestors should use **Heuristic Percentage** with **Delay**. Otherwise, large numbers of nodes will synchronize their timing of requiring retries and sending responses.

Note that **Heuristic Percentage** does not introduce a phase difference between the command and the response.



Providers must support the heuristic behavior of command control. `dof-2008-2-spec-12`

The reason for this requirement is that the least capable nodes are usually the most numerous in a network. Allowing them to ignore this requested behavior would negate the benefit of having it at all.

3.13.3. Delay

The **D** or **Delay** field controls command execution, enabling delayed behavior. If this field is zero then the command executes with no delay. If the field is one, the node should randomly execute during the duration of the command as received from the DPS. This delay acts as a phase shift between the DPS operation and the OAP operation. However, since DPS maintains the expiration of operations the requestor must retry the operation at the DPS duration or nodes will timeout (for operations with state).

Acknowledged operations ignore the **Delay** field.

If there is no existing state for the operation, then the effect of delay can "reset" for each command. This means choosing a new random offset.

However, for operations that have existing state (in this case because the operations create state – state created by acknowledgement cause the option to be ignored as described above) the case is different and yields some interesting requirements.

Operations that automatically create state must associate with that state the chosen phase shift (random delay). Future retries that include the option for **Delay** must ensure maintenance of the same phase shift

if possible. This means that if the new duration exceeds the existing phase shift then the same phase shift applies to the retried operation.

However, if the new duration is less than the current phase shift then the receiver must choose a new phase shift randomly between the current time and the new duration.



Providers must support the delayed execution behavior of command control.

`dof-2008-2-spec-13`

The reason for this requirement is that the least capable nodes are usually the most numerous in a network. Allowing them to ignore this requested behavior would negate the benefit of having it at all.

DOF libraries typically should not make assumptions about timeliness of responses due to the distributed nature of DOF implementations and the differences in supported transports. For example, a requestor running on 1Gb Ethernet may be bridged to a provider running on 100Kb wireless network.

This tolerance to delay can be used to the advantage of nodes on limited transports by manually introducing a form of delayed execution. This is acceptable to within certain limits. For example, receiver should probably not delay an OAP unicast operation, but an OAP multicast/broadcast operation might be.



Implementations with knowledge of transport characteristics may make use of the delayed execution capability to manage transport bandwidth. However, they should balance the timeliness of the response against the transport problems. For example, providers should not interpret an operation with DPS duration of one hour and no **Delay** as allowing a delay of an hour. The actual permissible delay must not exceed the duration of the operation, and should be as fast as reasonable based on the transport.

3.13.4. Do Not Execute

The **X** field controls execution of the command. If set then receivers do not execute the command, however, other options are still processed.



Providers must support the do not execute behavior of command control.

`dof-2008-2-spec-14`

The reason for this requirement is that the least capable nodes are usually the most numerous in a network. Allowing them to ignore this requested behavior would negate the benefit of having it at all.

3.13.5. Verbosity

The **V** field express control over the types of responses sent. Note that it refers to true responses, not commands sent because of other commands.

There are four values defined for this flag. At the low end, a value of zero indicates *no* responses. On the high end, a value of three indicates *very verbose* responses. This includes information that is may be more useful during product development.

In general, these values have the following meanings:

Table 3.1. General Verbosity Meanings

Value	Meaning
0	No response.
1	Successful responses only (no Exception responses).
2	Exception responses in addition to level 1.
3	Intermediate results in addition to level 2. Intermediate responses mean non-final Exceptions.

Providers must understand and should follow verbosity behavior of command control. `dof-2008-2-spec-28`

Small implementations may elect not to recognize this request. The command control PDU specifies the default verbosity level.

The handling of verbosity level 3 is up to the application. For example, it may send additional, non-final user Exceptions, or may choose to do additional logging. This limits the applicability of this level to cases where the application knows specifics of the implementation, or where some specification of the application behavior exists.

3.13.6. Acknowledgement Lists

OAP allows multicast, broadcast, and even unicast operations that are flooded through the system. Whether submitted over a connected transport or not, these operations may need to be retried. It is desirable to generate responses only once, but datagram loss means that the provider cannot know whether the requestor received the response. To close the loop the requestor will send an acknowledgement.

All operations that generate responses and that are not OAP unicast must deal with acknowledgement. Nodes accept attempts to use acknowledgement with OAP unicast, but should ignore the acknowledgement list. Normally, a provider does not need to keep state for an operation once it has responded. The situation changes when the provider receives an acknowledgement. In this case, the provider should track the operation identifier and not generate further responses for that operation identifier. This logic applies for all transports, so an acknowledgement may arrive over a session and halt responses on a lossy transport.

Providers should recognize acknowledgements and prevent further responses. `dof-2008-2-spec-15`

A provider should maintain information on all acknowledged operations and not generate further responses, independent of the transports of the requests.

It is also possible to acknowledge an operation *on receipt of* the actual command. This means that the logic for handling a new operation must take into account any included acknowledgement list associated with the new operation.

Providers should recognize pre-acknowledgement. `dof-2008-2-spec-16`

A provider should allow for cases of new operations that include acknowledgement lists, and respond appropriately (generate responses only if not pre-acknowledged).

3.14. Types and Values

Many OAP commands and responses pass typed value information. PDUs do not pass type information with the data, and so it is not possible to interpret the data without understanding the context of the command or response. This usually indicates that only someone with knowledge of the binding or alias and the interface definition can create, understand, or respond to these OAP commands.

OAP passes values as the last fields in a PDU. This means that no additional OAP fields can ever follow this kind of typed value.

The *DOF Protocol Specification* and *DOF Common Types* contain information on the formatting of types and values.



Value List `dof-2008-2-pdu-8`



`Value Data` **Variable length.** Value data.

It is not possible for the provider or requestor to validate what type information a PDU contains, and so mismatches can occur. When possible, implementations should provide type checking of application data.



A node must not do type conversion. It should reject the command or response if it knows the required types the data do not match. It should always put data on the wire that matches the type information of the associated interface definition.

3.15. Permissions

OAP defines the following permissions.



Pay particular attention to the handling of security scopes when normalizing permissions. See *DOF Security Specification*, Security.27. Implementations must take special care when dealing with the security scope compatibility as defined in the reference.

3.15.1. Binding Permission with Wildcards

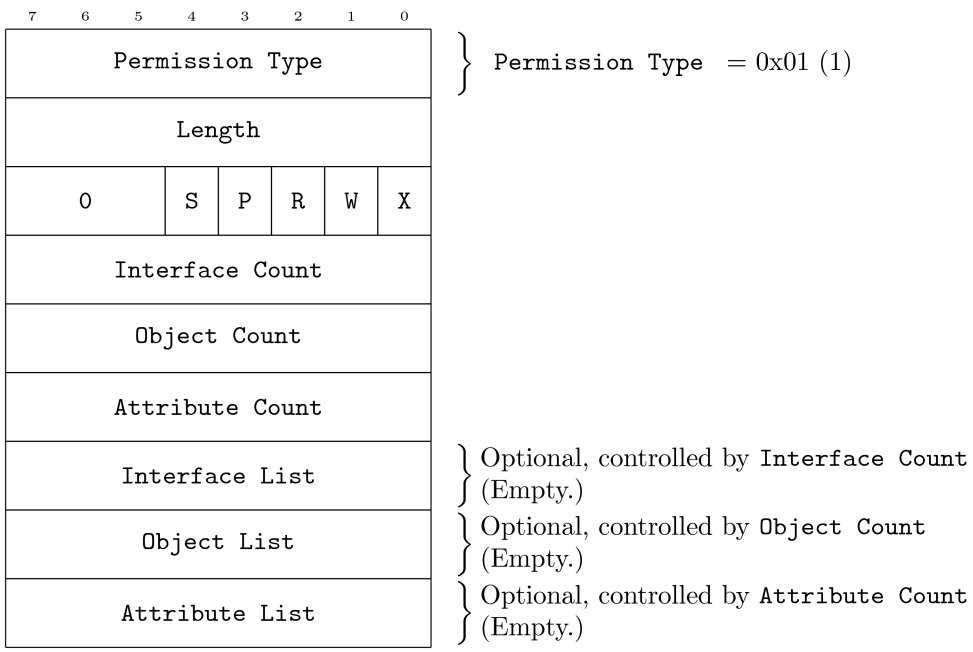
This permission represents the ability to use bindings in different ways and allows the use of wildcards. The presence of wildcards means that this format is more useful for permission requests driven by the user, since dedicated programs would rarely assume a wildcard was possible. Note that there is no explicit permission to "see" a binding. This means that a requestor may have visibility to many bindings that it cannot use.

This permission uses the following format:



Binding Permission with Wildcards `dof-2008-2-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`.

`S` **One bit.** Indicates the Session permission when set.

`P` **One bit.** Indicates the Provide permission when set.

`R` **One bit.** Indicates the Read permission when set.

`W` **One bit.** Indicates the Write permission when set.

`X` **One bit.** Indicates the Execute permission when set.

`Interface Count` **One byte.** Controls the length of the `Interface List`. Indicates a match against all interfaces if zero.

`Object Count` **One byte.** Controls the length of the `Object List`. Indicates a match against all objects if zero, applied to the base identifier only.

`Attribute Count` **One byte.** Controls the length of the `Attribute List`. Indicates a match against all attributes if zero. If 255, indicates that no `Attribute List` is present and that the permission matches no attributes.

`Interface List` **Array of** Interface Identifier (IID)`dof-2009-1-pdu-21`.**Optional, controlled by** `Interface Count`. Indicates interfaces for which the specified actions are allowed when paired with an object and attribute specified in this permission.

`Object List` **Array of** Object Identifier (OID)`dof-2009-1-pdu-4`.**Optional, controlled by** `Object Count`. Indicates objects for which the specified actions are allowed when paired with an interface and attribute

specified in this permission. The permission allows the Broadcast OID to indicate permission for OAP broadcast and multicast. The permission does not allow OID attributes on these OIDs.

`Attribute List` **Array of** Object Identifier Attribute`dof-2009-1-pdu-5`.**Optional, controlled by** `Attribute Count`. Indicates attributes that must be present on the object in order for the permission to apply.

Interfaces must appear in the permission in increasing sorted order. The permission does not allow the wildcard IID. `dof-2008-2-spec-17`

See the definition of interface identifier in *DOF Common Types* for the description on this order.

Objects must appear in the permission in increasing sorted order. The permission allows the Broadcast OID. The permission does not allow attributes on OIDs. `dof-2008-2-spec-18`

See the definition of object identifier for the description on this order. The Broadcast OID indicates OAP multicast/broadcast.

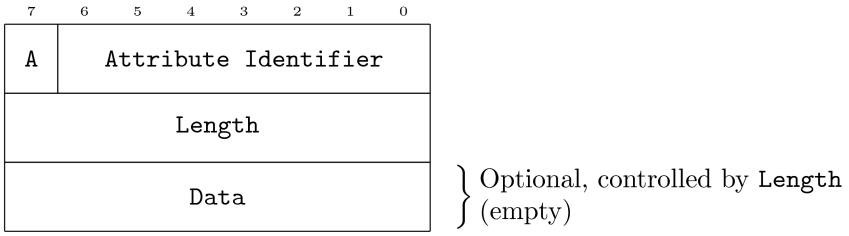
The **Attribute List** contains a list of object identifier AVPs. These follow the same format as in the OID format in the DOF Data Types section, duplicated here in part. The interpretation is slightly different when used in permissions.

Attributes must appear in the permission in increasing sorted order Each Attribute Identifier may appear in the list at most once. `dof-2008-2-spec-19`

See the definition of object identifier for the description on this order. Note that the high-bit, used as described below, does not affect the sort order. This is the same as defined in the object identifier section although the bit has a different meaning there.

Binding Permission Attribute `dof-2008-2-pdu-10`

Instance of Object Identifier Attribute`dof-2009-1-pdu-5`



A **One bit.** This field has a different meaning than in Object Identifier Attribute. Indicates when set that the attribute data is a wildcard, and will match any attribute using the same **Attribute Identifier**. When set the **Length** field must be zero.

	This indicates that the attribute must be present, but can take any value.
`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.

3.15.1.1. Normalization

The formats defined in this section are flexible enough to define many different formats for equivalent permission sets. This flexibility is not a problem unless it allows for different results when permissions are evaluated, which would be a serious error.

In order to prevent this, the definition of a "normalized" permission set exists. Given a set of binding permissions, the following algorithm defines the normalized set of equivalent binding permissions.

- Expand all lists to form an equivalent set of permissions. This expansion will result in (potentially) many more individual binding permissions. Each binding permission will have list lengths equal to either zero or one after this step. This also includes expanding the set of associated security scopes so that each resulting permission is in a single security scope.
- Compare each permission (binary comparison) within a given security scope not including the Actions. If they are equal, then combine the **Actions** and remove the now redundant permission.
- Find each permission within a given security scope that includes a wildcard (for any list) and merge its **Actions** into each permission that it matches. For example, a Read action on all bindings would merge into all other binding permissions. In this case no permission becomes redundant and so none are removed.
- Find matching permissions and consolidate the permission scopes back to a list.

This process does not treat the special case of the **Attribute Count** equal to 255 differently, meaning that the binary comparison will not merge it except with other permissions that use the same special value of 255.

This process normalizes the binding permissions.

3.15.1.2. Comparison

Binding permission A $\leq$ binding (is allowed by) permission B if all of the following are true:

- The **Actions** of A are a subset of the **Actions** of B.
- All objects in A appear in B or the object list in B is empty. An empty object list in A only matches an empty object list in B.
- All attributes in A appear in B, or B contains a list count of zero, or A has no attributes and B has a list count that is either zero or 255. B contains no additional attributes. The values of the Attributes must match (with wildcard values in A only matching wildcard values in B).
- All interfaces in A appear in B or the interface list in B is empty. An empty interface list in A only matches an empty object list in B.

3.15.1.3. Complement

The complement of a binding permission may result in a set of one, four, or five permissions. The complement result set starts out empty, and the implementation follows these steps on the input binding permission A:

- If A contains **P**, then add four permissions, each specifying a single action from **S**, **R**, **W**, and **X**. The binding for the new permission is identical to that of A.
- If A contains **S**, **R**, **W**, or **X**, then add a single permission with the action **P**. The binding for the new permission is identical to that of A.

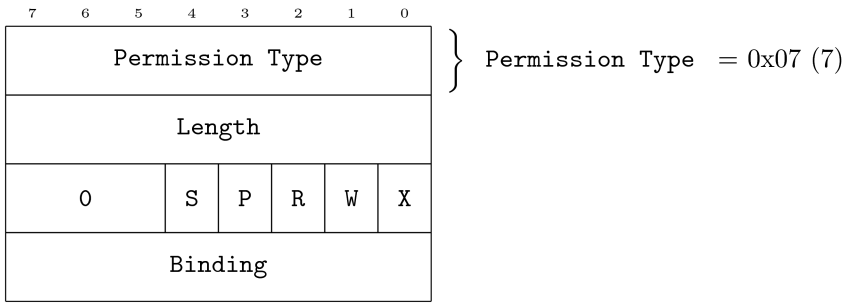
3.15.2. Binding Permission without Wildcards

This permission represents the ability to use bindings in different ways but does not allow wildcards. This is a simpler representation of the more general wildcard permission discussed above, and more directly represents the permissions required by individual operations.

This permission uses the following format:

Binding Permission without Wildcards `dof-2008-2-pdu-11`

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`.

`S` **One bit.** Indicates the Session permission when set.

`P` **One bit.** Indicates the Provide permission when set.

`R` **One bit.** Indicates the Read permission when set.

`W` **One bit.** Indicates the Write permission when set.

`X` **One bit.** Indicates the Execute permission when set.

`Binding` **Instance of** Binding`dof-2008-2-pdu-5`.Indicates the binding of the permission. The permission does not allow the Wildcard IID. The binding may include attributes.

3.15.3. Normalization, Comparison, and Complement

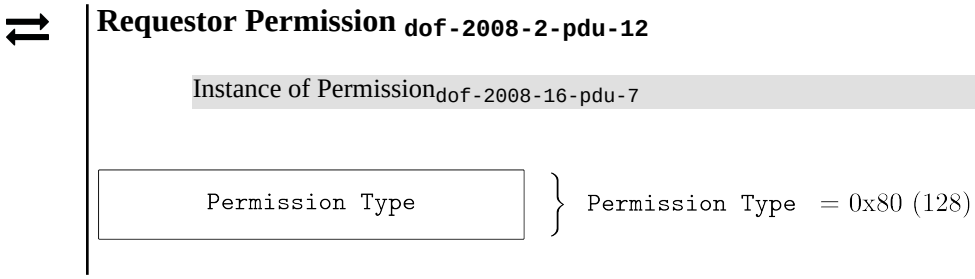
This permission is normalized, compared, and complemented by transforming it into the wildcard permission presented above. The translation is as follows:

- Create a new Binding Permission with Wildcards.
- Separate the OID in the **Binding** into the base part and the attribute part.
- Place the base of the OID into an Object List with a length of one in the new permission.
- Place the IID in the Binding into an Interface List with a length of one in the new permission.
- If there are any attributes (above), then place them into an Attribute List of the appropriate length (greater than zero) in the new permission. If there are no attributes, then use the special value 255 in the attribute list count of the new permission.

Once completing the transformation, the resulting permission is handled as the in the Binding Permission with Wildcard case, above.

3.15.4. Requestor Permission

This permission represents the ability to act as a requestor in a general sense. It does not allow specific access to a binding, but allows other general requestor operations. Negotiating this permission can also act as a signal as to the type of node that is connected.



Nodes that know they are requestors but do not know specific required permissions use this permission.

3.15.4.1. Normalization

The normalization of this permission type is that only a single instance should remain in each security scope – remove all duplicates from the permission set. In addition, if the permission set contains any Binding permissions with actions other than provide in the same security scope then remove this permission entirely.

3.15.4.2. Comparison

Requestor permission A $\leq$ permission B (is allowed by) if either of the following are true:

- Permission B is a Requestor permission.
- Permission B is a Binding permission that includes an action other than provide.

3.15.4.3. Complement

The complement of this permission is the Provider permission.

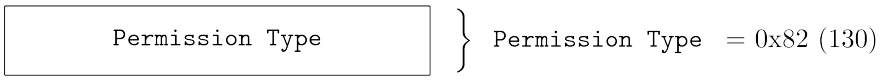
3.15.5. Provider Permission

This permission represents the ability to act as a provider in a general sense. It does not allow providing any specific binding, but allows other general provider operations. Negotiating this permission can also act as a signal as to the type of node that is connected.



Provider Permission `dof-2008-2-pdu-13`

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



Nodes that know they are providers but do not know specific required permissions use this permission.

3.15.5.1. Normalization

The normalization of this permission type is that only a single instance should remain in each security scope – remove all duplicates from the permission set. In addition if the permission set contains any Binding permissions with a provide action or any Define permissions in the same security scope then this permission can be removed entirely.

3.15.5.2. Comparison

Provider permission A $\leq$ permission B (is allowed by) if any of the following is true:

- Permission B is a Provider permission.
- Permission B is a Binding permission that includes a provide action.
- Permission B is a Define permission.

3.15.5.3. Complement

The complement of this permission is the Requestor permission.

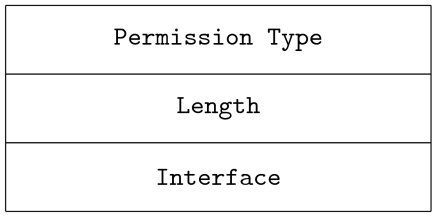
3.15.6. Define Permission

This permission represents the ability to define an interface.



Define Permission `dof-2008-2-pdu-14`

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`.
`Interface`	**Instance of** Interface Identifier (IID)`dof-2009-1-pdu-21`.The interface that can be defined. The Wildcard IID indicates permission to define all interfaces.

Nodes that can define an interface but not provide a binding request this permission.

3.15.6.1. Normalization

Normalization occurs in each security scope and includes removing all duplicates. Further, if any Define permissions include the Wildcard IID then remove all other Define permissions. Finally, if a Binding permission exists that includes the referenced interface then the Define permission can be removed. In this last case, the Wildcard IID in the Define permission may match a Binding permission with an interface count of zero. All comparisons must occur in the same security scope.

3.15.6.2. Comparison

Define permission A $\leq$ permission B (is allowed by) if any of the following is true:

- Permission B is a Define permission with the same interface.
- Permission B is a Define permission with the Wildcard IID.
- Permission B is a Binding permission that includes a provide action and the interface in A is included in B (or B has an interface count of zero).

3.15.6.3. Complement

The complement of this permission is the Requestor permission.

3.16. Access Control

Each command defines the permissions required in order for it to be accepted. Receivers silently drop requests that arrive securely and require permissions not held by the requestor.

Commands that utilize OAP broadcast and multicast must have matching permission (to either broadcast or multicast). Additionally, in the case of a response to an OAP broadcast or multicast command, the provider must have permission to provide the binding being responded to. Receivers silently drop packets that require lacking permissions.

The permissions that affect OAP are as follows:

- Binding(Read): Controls the ability to get properties and subscribe to properties and events.

- Binding(Write): Controls the ability to write properties.
- Binding(Execute): Controls the ability to invoke methods.
- Binding(Provide): Controls the ability to add and remove bindings.
- Binding(Session): Controls the ability to open a session on a binding.
- Define(Interface): Controls the ability to define an interface.
- Requestor: Allows generic requestor operations.
- Provider: Allows generic provider operations.



Nodes must correctly request and enforce permissions on secure sessions. `dof-2008-2-spec-20`

The *DOF Security Specification* contains more detail on the specific permission requirements.

3.17. Domains and Partitions

Systems use security domains and partitions to isolate and control visibility of groups of bindings. The system should function identically if all the sessions belonging to a particular set of compatible partitions were replicated, isolated, put into a topology without partitioning, and connected as they were with partitioning. In the real world, however, things are more complicated. A useful feature is to provide a single service shared by many partitions. This is complicated because such services may in fact add bindings to the system, and those bindings may not be global! In reality, such issues relate more to implementation and not protocol.

The details and specifications of domains, partitioning, and compatibility are defined in the *DOF Security Specification*.

3.18. General Operation State Type

The default rule for operation state type is Mutable unless specified otherwise.

For OAP, the Binding, Item, and Value List in all commands are Invariant.

Aliases are always type Source.

Command control information is always Mutable.

Specific commands will define their own fields.

3.19. General Routing Behavior

Brief mention has been made of "proxy," or the ability of a node to act as (or forward commands to) another node. This feature is a key aspect of DOF protocols, and primarily of OAP. OAP relies on techniques of mesh networking in order to accomplish its proxy responsibilities.

Not all nodes can proxy. Those nodes are free to ignore the specific behaviors required of nodes that can proxy. These sections discuss several common proxy behaviors. In addition, each command contains a section discussing the specific routing rules that apply to it.

OAP routing is based on the general capabilities of the DOF Presentation Protocol. However, OAP defines its own routing based on bindings. It is the responsibility of each OAP command to guarantee a loop-free behavior.

3.19.1. Operation Identification

A proxy will accept an inbound command and possibly forward it to other nodes. The operation identifier used in DPP serves an important purpose in this forwarding: it identifies and prevents ringing and loops.

This means that if the routing of a command could result in loops then it must preserve the operation identifier when forwarding the command. In general, the DPP is not able to determine the potential scope of a command, and so it is up to the implementation of OAP to guarantee this behavior.

If a command will not loop or ring then a node may use a new operation identifier for the forwarded command. These are sometimes smaller on the wire and more desirable for this reason. The command definitions identify these commands as DIRECTED operations.

3.19.2. Forwarding vs. Flooding

Forwarding is the process of identifying potential providers and directing a command only at them. Flooding is the process of sending a command to all possible providers. Forwarding is always preferred to flooding, as it minimizes network utilization and processing. However, the ability to forward requires that the routing provide a DAG – something that requires special management.

In order to determine whether an operation must flood, the node answers the following questions:

- Is the command specific enough to identify the correct provider? For example, for an OAP unicast command the answer is yes. For an OAP broadcast command the answer would be yes *only* if the node knew every provider (which is extremely rare). If the answer to this question is no, then the command should be flooded.
- If the provider (or providers) is specific, then does the node know the route to the provider? If the answer is yes then the node may direct the operation using the route. If the answer is no, then additional questions are required.
- Finally, assuming the node does not know the route, then the node should determine if it has expressed interest in the binding. If the node has expressed interest, and there is no provider for the binding, then the node should drop and *not* flood the operation. If the node has not expressed interest then the operation should be flooded. Nodes may use this determination to limit the number of sessions that are flooded to as well (only send to those sessions with no expressed interest).



Non-routing nodes ignore DPP FLOODED operations that do not match a provided binding. `dof-2008-2-spec-21`

This specification allows terminal nodes to ignore DPP flooded operations, preventing meaningless Exception responses. Nodes do not ignore DIRECTED operations.

3.19.3. Responses

Responses are special in that they always route back to the source of the command. In the case that a response arrives and the command is in a state where the node does not know the source then node must drop the response.

3.20. General Consolidation Behavior

See the section in the *DOF Protocol Specification* for a discussion about consolidation rules.

A general feature of proxy nodes is that may consolidate operations when possible. For example, if a proxy receives an interest operation that it can consolidate with one of its own (non-proxy) operations, then it may consolidate that interest.

Consolidation must not affect the security of the operations involved.

Certain operations, described later, gain a lot of benefit from consolidation and therefore all proxy nodes should consolidation them.

Each operation indicates suggested consolidation, along with the rules that govern the consolidation. However, note that a single requestor (non-proxy) is always able to consolidate its own operations. Individual operations do not explicitly state this consolidation – it is a global property of OAP. Otherwise, operations must explicitly allow consolidation.

3.21. Minimally Implemented Providers

The command descriptions found in the remainder of this section provide hints for minimizing implementation requirements. In general, OAP ensures scalability and compatibility through the full set of requirements covered in this document.

Of these two goals, nodes may sacrifice scalability in some instances. OAP ensures that nodes never sacrifice compatibility. A question arises as to what the *minimal* provider requirements really are. This section addresses that question, and provides some general guidance on minimizing provider implementations.

3.21.1. Transports

A single provider may simultaneously run on multiple transports. This discussion treats a session like a transport.

One of the biggest optimizations a provider can make is to unify its behavior on different transports. Effectively, all of its actions will take place on all transports, even if the action is not required or requested on that transport. For example, if a subscription starts on one transport (or one session) then the provider sends Change commands on *all* transports (and *all* sessions). This is not scalable (for a large system, it would be bad if all providers did this), but it serves to significantly reduce state on a provider.

Of course this optimization does not need to be all or nothing – some things may operate this way and others be tracked by transport or session. This would depend on the implementation.

3.21.2. Operation Lifecycle

A requestor can simultaneously have many outstanding operations against a provider. This compounds the fact that there are many requestors in a system. Operation consolidation significantly reduces the required state.

As discussed earlier, part of the role of a proxy is to consolidate operations. However, a proxy is not *required* to do this and even if it did, there are still large peer-to-peer systems in which there are no proxies, but there are many requestors!

Providers can make many operations that have lifecycle stateless by ignoring acknowledgements. For example, many of the object manipulation commands (Get, Set, and Invoke) can "acknowledge" a result. The provider should track acknowledged operations and not generate further responses when retried. Providers can save state by ignoring these acknowledgements. This is not ideal, and may cause problems for the requestor, who repeatedly has to send acknowledgements, but this section is concerned with minimizing the *provider*, and this is one way to accomplish this.

Another set of operations require state in order to control a desired behavior of the provider. For example, Watch, and Activate control Provide. A minimal provider could choose to ignore the control commands and assume the behavior (Provide all the time). Subscribe and Register control Change and Signal. A minimal provider could choose to ignore Subscribe and Register, and always send Change and Signal commands. Again, this does not scale well, but it minimizes provider implementation.

Finally, there is a middle ground between ignoring control commands and tracking them completely. The idea is to track the desired "result" along with a timeout. For example, the desired result of a Watch is a Provide. If each binding stores a matching timeout then the first Watch or Activate on a binding would set the timeout (to sometime in the future) and the Provide would begin on that binding. If that timeout occurs then the provider cancels the Provide. The arrival of an operation that would cause the same behavior (another Watch or Activate) *and* its timeout is further in the future than the existing timeout then the provider modifies the timeout.

Note that this solution is not entirely correct – it does not handle Cancel and so may not stop at the right time, but it takes significantly less state.

In fact, if the lifecycle optimizations are combined with the transport optimizations then the resulting minimal implementation results. Here are some of the characteristics of these minimal providers:

- They Provide without being requested to.
- They send Signal and Change commands without subscriptions and registrations.
- They do not manage sessions (and probably do not support sessions).
- They respond even after acknowledgement.

The benefit is that they can operate fully (although not optimally by any means) with no operation state. Such minimal implementations should be rare – the mode switching behavior described below is a much more network-friendly way to minimize state.

Note the discussion above regarding the initial operation state that talks about responses based on requested behavior that applies in these situations.

3.21.3. Mode Switching

It is possible for a provider to switch between "correct" or "optimized" behavior and "minimal" behavior. This can be very beneficial. Typically the switch occurs when resources (usually RAM) are not able to store the complete required state. The switch back occurs when enough operations have completed that resources are again sufficient to track all operations.

For example, say that a provider has resource to track five subscriptions. It will work normally as long as there are fewer than six subscriptions. When the sixth (and later) subscriptions arrive, the system falls back to "minimal" behavior. It does this by representing the current state of the five operations in the minimal way, and then begins to send Change and Signal commands on all changes. At some point, the subscriptions all complete, and the system reverts to fully tracking subscriptions.

3.22. Minimally Implemented Requestors

Minimal requestors must solve a different set of problems as compared to providers. Requestors have more control over the state they require, but they may have to keep track of many providers. The primary issue for a minimal requestor is keeping track of bindings, which means keeping track of Provide commands.

The other issue for requestors is keeping outbound operations alive. This requires keeping state on each operation. The primary benefit of a requestor is it usually immune from any scaling problem. In other words, a requestor is in better control to manage its state, even in the presence of many providers.

An important distinction for a requestor is whether it is also a provider. Requestor-only nodes may completely ignore all inbound commands that are associated with providers.

4. Command Reference

The following sections describe in detail each OAP command and response. The commands appear in alphabetical order. Each command includes a summary, which lists the permissions required to use the command, session requirements, and DPS requirements.

Following the summary is an overview that shows the layout on the wire of the command or response and describes each of the fields that they contain.

Following the overview is information relating to minimizing implementation size, which identifies rules or easily overlooked optimizations regarding the command. This is especially important for people writing small implementations of the protocol.

Finally is a details section that provides all of the background and use information for the command, including sequence examples.

4.1. Command Classes

There are three major classes of operations in OAP, although some operations do not fit into a particular class. The following sections define the general requirements and handling of operations in the specified class. Specific operations may override these requirements, but this is rare.

4.1.1. Interest

Interest is the most fundamental of all OAP operation classes. Interest is required in order to determine the routes to bindings. The DOF architecture assumes that there are more potential providers in a system than there are interested requestors. This assumption leads to a design in which providers react to interest, which should minimize the overall operations in the system.

Interest commands view providers as continuously "providing" a set of bindings, and as optionally being able to provide additional bindings. These sets represent "base" bindings and "dynamic" bindings. Further, interest is concerned with the transports used to provide the bindings, either lossless or lossy. The variations of these requests represent the level of the interest, which starts at "watch" and moves to "full connect."

The interest commands do not define the method used to provide matching bindings. The Provide command is required, but the transport and transport address used does not need to match the request. For example, it is possible for a node to determine that it is more efficient for it to use a single transport broadcast Provide to satisfy many incoming unicast interest requests. The method used must be a superset of the method used in the request. Some of the interest commands express some desire about the transport used to provide the bindings as presented above.

Requestors express interest in OAP using flooded DPP operations. All interest follows the same fundamental pattern: a command floods the network, generates no responses, and providers or proxies send Provide commands as reactions.

The binding allows in all interest commands the use of all OAP addressing types (unicast, multicast, and broadcast) and in addition allows the use of the wildcard IID. OID matches follow an *at least* rule, where attributes present in the provided binding may include attributes not present in the interest. Interest commands support the use of aliases, although it would be impossible to use the feature unless the requestor already knows the binding (required to know the alias in the first place).

Interest commands, particularly those that use non-OAP-unicast bindings or wildcard IIDs, can cause a lot of network traffic. This is first due to their flooded nature, and second to the large number of potential

responses. The section on command control, above, can guide requestors in the correct methods to handle the large number of potential reaction packets.

Providers should not provide bindings unless there is interest. Interest commands remain active during their lifecycles, and it the summation of all interest governs provided bindings. This rule is particularly interesting for proxy nodes, which may stop forwarding Provide operations if there is no interest. Note that proxy nodes cannot cancel Provide operations, although they can (and should) be cancelled by their providers if there is no interest.

Interest commands are usually the first commands used in OAP, although new sessions may use the Advertise command in order to speed the reception of matching interest commands.

4.1.1.1. Command Control

All interest commands utilize command control, although only the Heuristic Percentage and Delay options have any effect. Providers ignore other options.

4.1.1.2. Routing

As mentioned above, all interest commands flood to all compatible sessions.

4.1.1.3. Consolidation

There is a great benefit to consolidating related interest commands. The *DOF Protocol Specification* contains general guidelines that nodes must follow. In addition, nodes must propagate the correct interest level. This means, for example, that a node may consolidate a Watch into an Activate, but not the opposite.

However, given the general consolidation rules, the chance for consolidation in a general routing node depends on topology. This is because there may be no local application, and another source would usually not send operations that represented duplicate interest (if owned by that node then it should have only forwarded one operation in the first place).

The exception to this is a topology where a routing node accepts multiple connections and then forwards the requests to another routing node. The first routing node sees operations from a variety of sources, where the rules do not allow it to consolidate them. However, the second routing node sees many interest operations from the same source (the first routing node) and is able to consolidate them effectively.

4.1.1.4. Security

Nodes can abuse any command to cause security issues, but the interest commands that request sessions (Connect and Full Connect) are particularly dangerous. The following guidelines minimize the potential for attacks. The reason is that typical denial-of-service attack prevention looks at the source of the attack, and the Connect and Full Connect commands could "masquerade" the attacker behind another node if it were defined differently or implemented incorrectly. This would make it easy to create a DDoS attack.

The application should ensure that:

- An unsecure Connect/Full Connect only results in an unsecure outbound session.
- A secure Connect/Full Connect only results in a secure outbound session.
- If the session is secure, negotiate permission to provide the binding as part of the initial permission set.
- Use the Advertise command to speed the sending of a matching Connect/Full Connect command.

- Give careful attention to timing issues. This means, for example, that sessions should not remain open indefinitely while waiting for commands to arrive, and they should close within a reasonable time when commands that are keeping them open are terminated or complete.

4.1.2. Actions

Actions take place on bindings whether there is interest expressed or not. Requestors initiate these commands to interact with or discover bindings. Actions do not require interest, and they do not establish a relationship with individual providers outside that of the operation itself.

The typical purpose of an action is to interact with a provider, either sending or receiving information (or both). Actions can use OAP unicast, multicast, or broadcast, and they support aliases. Requestors may use DPP directed operations for OAP unicast commands with a known provider; otherwise, they must use an DPP flooded operation.

A general capability of actions is to interact with multiple providers using the same command. In this case, each provider will respond and the requestor must be prepared to receive the responses.

Another general capability of actions is the use of command control, discussed later.

Whenever the command uses a broadcast OID, the response must indicate the specific provider. As the command always specifies the interface identifier, the combination of the IID and provider uniquely identifies the responding binding.

The handling of actions depends on a number of factors, primarily based on the type of OAP operation (unicast, multicast, or broadcast) and the type of receiver (proxy, provider, or both).

Proxies follow the routing rules, defined below. Providers follow the sequence identified next. Nodes that are both provider and proxy first follow the routing rules, and then follow this sequence. Requestor-only nodes may silently drop the command.

Providers ignore all non-OAP-unicast commands that do not match a binding that they currently provide. Providers take the following steps in the indicated order.

- If the provider doesn’t support the requested action:
  - If transport unicast, send Exception (Unsupported).
  - Otherwise, silently drop the command.
- If the binding matches a provided binding (discussed below):
  - If the command is not valid for the item or if the item is unknown, send Exception (Unsupported).
  - Take the action and send a response.
- If the binding does not match a provided binding:
  - If OAP unicast and transport unicast, send Exception (Binding Not Found).
  - Otherwise, silently drop the command.

The process of generating the response may result in Exception (Provider). If the response data does not match the interface definition then return Exception (Illegal Parameter).

4.1.2.1. Command Control

Command control modifies the action commands in the following ways:

Do Not Execute Causes providers to ignore the command except for the processing of any Acknowledgement List.

Verbosity Follows the general verbosity behavior.

Heuristic Percentage Causes randomized execution as described in the Command Control section.

Delay Causes the command execution to delay. The value returned is as of the execution time, not the reception time.

Cache Allows the return of a cached property, as long as the value is not older than the specified maximum.

Acknowledgement List Providers identified in the list cease sending responses, although they continue to track the operation during its lifecycle. Acknowledgements are to a binding, not a provider. A provider that has multiple matching bindings must keep individual acknowledgment state for each object.

Acknowledgements are to an object (a binding, actually), not a provider. A provider that has multiple matching bindings must keep individual acknowledgment state for each object.

4.1.2.2. Routing

Proxies react to the action commands according to the following two cases:

- OAP unicast: In this case, the proxy forwards the command to the primary provider of the binding. The lack of a primary provider causes the proxy to flood the command.
- OAP broadcast and multicast: The proxy forwards the command to all sessions that could provide a matching binding. This means that an implementation can remove sessions that cannot provide a matching binding. This determination could be because of permissions or any other knowledge based on the specific implementation. If there is nowhere to forward the command, the proxy drops it.

The forwarding of any operation that not directed to a known Provide must be flooded, even if it arrived directed.

4.1.2.3. Consolidation

Nodes must not consolidate the action commands.

4.1.3. Relationships

Finally, operations can establish relationships between requestors and providers. Relationships are fundamentally different from actions in that they require interest in order to maintain the relationship.

Relationships require the establishment of a primary provider. The Provide command describes the logic required to determine the primary provider, but the result is that each node in the network as a determined route for any specific binding. Further, restricting the acceptance of reaction commands to those *only* from the primary provider eliminates looping, even if the commands that cause the reaction do not form a DAG.

This reliance on the primary provider means that the response to the original operation is also restricted. In this were not the case, responses could be accepted from a different provider than becomes part of the relationship.

To summarize, the relationship commands rely on the following:

- Established interest, meaning each provider is established and has a primary provider. Relationships will fail to establish if interest is not present.

- An initial command, which establishes the requestor's sense of the relationship. This may be flooded or directed. The initial command controls the lifecycle of the relationship. Providers should only send reactionary commands while the relationship is active.
- A response, accepted only from the primary provider, that confirms the relationship. Note that in the case of a directed operation only the primary provider should be in a position to respond.
- Command filtering from the provider, where receivers ignore commands from providers that are not primary.

The limit on accepting responses and reaction commands prevents the following two problems:

- Without response filtering, loops in the topology cause multiple responses from the same provider. Responses follow the DAG of the operation at the time the provider sends them, but they do not reflect the topology resulting from the operation once it is stable. This means that basing a DAG on general responses is not safe. The DAG of primary providers is safe.
- Without reactionary filtering, reactionary commands that fan out (based on consolidation) may loop. This occurs when multiple requestors align to form loops, and the fan-out of the reacting commands cannot determine that the loop exists. This is particularly true in the case where the reacting commands do not include operation identifiers.

The initial (setup) commands follow the requirements of the Action class of operations, except that requestors and proxies drop OAP unicast operations that have no route. The reactionary commands may be unique, not following a particular class of operation.

The lifecycle of the relationship relates to the initial command. The requestor controls the operation, but the provider in the relationship can also terminate its participation using an Exception(Terminated), even after sending a response and other reactionary commands.

The relationship between requestor and provider is broken if the primary provider changes. The fact that a relationship may include many providers, each of which can change, and each of which represents a relationship, complicates the matter.

If a provider (non-proxy) terminates the relationship then it sends the exception and modifies the operation state to prevent further responses in the case of flooded operations.

If a proxy terminates the relationship due to the change of primary provider then it sends the exception and modifies its internal state to block further responses from that binding until the next retry.

4.1.3.1. Consolidation

Each relationship-based command specifies its own consolidation requirements.

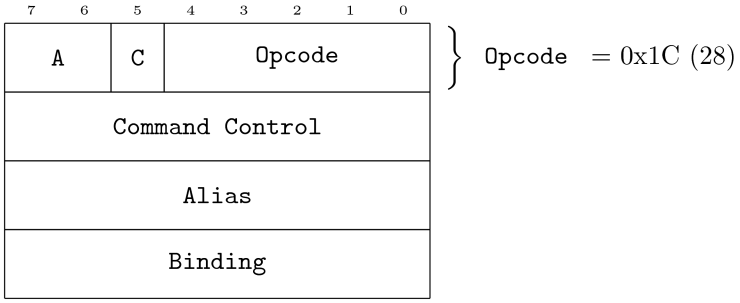
4.2. Activate

Request that matching provide commands be sent to the sender.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Requestor
Permission (response): N/A

See Advertise, Connect, Full Connect, Provide and Watch.

Activate (command) `dof-2008-2-pdu-15`

COMMAND/FLOODED
OPID
Instance of OAP Binding-Oriented Command$_{\texttt{dof-2008-2-pdu-1}}$



`A`	**Two bits.** Controls the format of `Binding`.
`C`	**One bit.** Controls the presence of `Command Control`.
`Opcode`	**Five bits.** Determines the type of the command.
`Command Control`	**Instance of** Command Control$_{\texttt{dof-2008-2-pdu-7}}$.**Optional, controlled by** `C`. If `C` is one then this field is present. This field determines options that apply to the command.
`Alias`	**Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.
`Binding`	**Instance of** Binding$_{\texttt{dof-2008-2-pdu-5}}$.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

4.2.1. Small Implementation Hints

All provider nodes must typically react to this command. The only exception is a provider that always provides and maintains its bindings over all sessions and transports *and* does not support lossless sessions. Such a node would ignore this command. However, such minimal implementations are highly discouraged because of the load that they put on the network.

Requestor-only nodes may silently drop this command.

4.2.2. Flows

Activate() $\rightarrow$ (none) : Success.

4.2.3. Details

The Activate command expresses interest in dynamic bindings that providers can provide even if they are not at the time.

See the section on the Interest class of operations for common requirements.

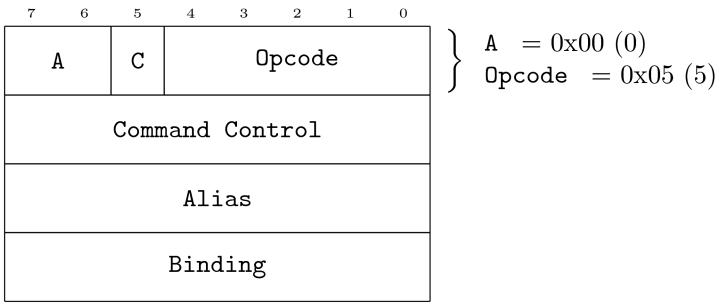
4.3. Advertise

Notify receivers of the ability of the sender to provide a binding.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Provider
Permission (response): N/A

See Activate, Connect, Full Connect, Provide and Watch.

⇄ **Advertise (command)** `dof-2008-2-pdu-16`

COMMAND/FLOODED
OPID
Instance of OAP Binding-Oriented Command`dof-2008-2-pdu-1`



`A`	**Two bits.** Controls the format of `Binding`.
`C`	**One bit.** Controls the presence of `Command Control`.
`Opcode`	**Five bits.** Determines the type of the command.
`Command Control`	**Instance of** Command Control`dof-2008-2-pdu-7`.**Optional, controlled by** `C`. If `C` is one then this field is present. This field determines options that apply to the command.
`Alias`	**Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.
`Binding`	**Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

4.3.1. Small Implementation Hints

Nodes should respond to this command, although it is not required.

4.3.2. Flows

Advertise() $\rightarrow$ (none) : Success.

4.3.3. Details

Providers use the Advertise command to indicate that they are capable of providing a set of bindings if requested. Receiving nodes can use this information to transmit matching interest commands to the sender, which may then cause actual Provide commands. This command gives a way of accelerating the sending of interest retries.

The binding present in the command may represent a single binding, or it can be a superset of those bindings if wildcards are used. This command allows the wildcard IID.

The Advertise command is available on all transports, although it is particularly useful with stateless servers, where the server may not otherwise know to send interest commands to the node. This can happen, for example, because there is no state kept on the stateless server for every node in the network, meaning the server cannot forward interest automatically.

4.3.4. Command Control

Command control affects the Advertise command in the following ways:

Do Not Execute	No effect.
Verbosity	No effect.
Heuristic Percentage	Causes randomized execution as described in the command control section.
Delay	Causes the command execution to delay.
Cache	No effect.
Acknowledgement List	No effect.

4.3.5. Routing Rules

This command must be selectively flooded unless consolidated. There are two cases:

- First, if the receiving node does not maintain and automatically forward existing interest commands over sessions then Advertise must be flooded to all sessions.
- Second, assuming the receiver forwards existing interest automatically, this command must be flooded to all sessions where the receiver does not maintain session state.

The following receivers do not maintain state:

- Stateless servers.
- Multicast groups (secure groups of all kinds).
- Multicast addresses (lossy sessions that use broadcast or multicast addresses).

4.4. Change

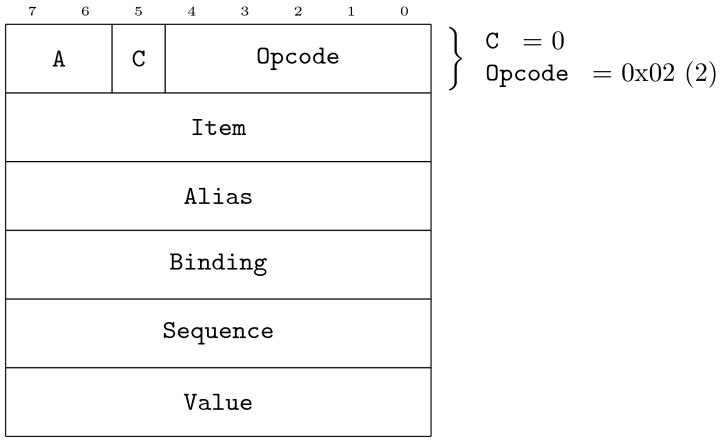
Notify a subscriber of a change to the value of a property.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.

Permission (command): OAP Binding(Provide)
Permission (response): N/A

See Subscribe.

⇄ Change (command) `dof-2008-2-pdu-17`

COMMAND/DIRECTED
NOOPID
Instance of OAP Item-Oriented Command`dof-2008-2-pdu-2`



`A` **Two bits.** Controls the format of `Binding`.

`C` **One bit.** Controls the presence of `???`.

`Opcode` **Five bits.** Determines the type of the command.

`Item` **Instance of** Item Identifier`dof-2008-2-pdu-4`.Defines the item for the command.

`Alias` **Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.

`Binding` **Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

`Sequence` **Two bytes, MSB.** Contains the provider's current change sequence, expressed in a floating window.

`Value` **Instance of** Value List`dof-2008-2-pdu-8`.The value for the changed property.

4.4.1. Small Implementation Hints

If a requestor or provider knows that it does not use or support subscriptions, then it does not need to support this command.

Provider-only nodes may silently drop this command.

Providers and proxies should send this command according to the suggested timing in the Subscribe command. However, small implementations may ignore the suggested timing may be ignored in favor of some other minimum value.

In the worst case, a provider may send Change commands in the absence of subscriptions. The specification allows this, but only in extreme cases and for values that do not change frequently.

4.4.2. Flows

Change() $\rightarrow$ (none) : Success.

4.4.3. Details

Providers and proxies use the Change command to pass information about value changes that occur while a subscription is active on that property. Nodes should only send this command to nodes with active Subscribe commands to the property.

Change is a reactionary command inside of a relationship formed by the Subscribe command. See the section on the Relationship class of operations for common requirements.

Subscriptions exist over a variety of transports, and may exist at the same time from a variety of requestors. However, the definition and behavior of the Subscribe command (the relationship operation type) ensures the formation of a DAG, so the Change command does not need to provide loop prevention. As an optimization, a node that does not currently Provide the binding (even if it could) can eliminate sending Change. This is because nodes drop the Change coming from any node other than the primary provider.

The binding present in the command must use OAP unicast. Providers may use an alias only in response to a subscription that used an alias; otherwise, the provider must use a full binding. Providers may send a single Change command (for a given binding and item) to a node even if the node has multiple outstanding subscriptions.

Note that the Change command contains a **Sequence**. The **Sequence** has operation state type of Mutable. Receiving nodes can use this value to ensure that they accept values in order. For a given property, the provider increases the **Sequence** for each Change sent. The value is a floating window, so wrapping the value is not a problem.

The **Value** is also operation state type Mutable.

4.5. Connect

Request that matching provide commands be sent to the sender.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Requestor
Permission (response): N/A

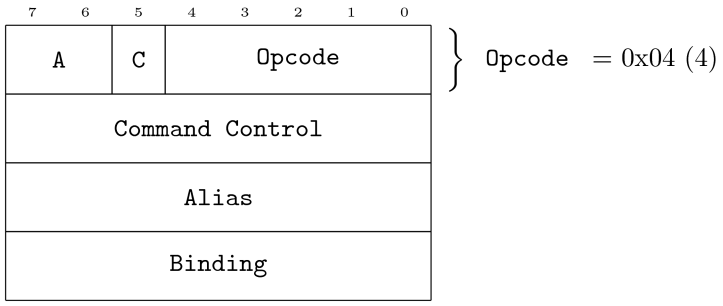
See Activate, Advertise, Full Connect, Provide and Watch.



Connect `dof-2008-2-pdu-18`

COMMAND/FLOODED
OPID

Instance of OAP Binding-Oriented Command$_{\texttt{dof-2008-2-pdu-1}}$



`A`	**Two bits.** Controls the format of `Binding`.
`C`	**One bit.** Controls the presence of `Command Control`.
`Opcode`	**Five bits.** Determines the type of the command.
`Command Control`	**Instance of** Command Control$_{\texttt{dof-2008-2-pdu-7}}$.**Optional, controlled by** `C`. If `C` is one then this field is present. This field determines options that apply to the command.
`Alias`	**Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.
`Binding`	**Instance of** Binding$_{\texttt{dof-2008-2-pdu-5}}$.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

4.5.1. Small Implementation Hints

All provider nodes must typically react to this command. The only exception is a provider that always provides and maintains its bindings over all sessions and transports *and* does not support lossless sessions. Such a node would ignore this command. However, such minimal implementations are highly discouraged because of the load that they put on the network.

Requestor-only nodes may silently drop this command.

4.5.2. Flows

Connect() $\rightarrow$ (none) : Success.

4.5.3. Details

The Connect command is a superset of the Activate command. In addition to the behaviors of the Activate command, the Connect command requests that upon finding a provider for the binding that the receiver start and maintain a lossless session with the provider if possible. However, it does not request that the provider establish such a session. This is appropriate for requestors or proxies that are able to manage outbound sessions.

See the section on the Interest class of operations for common interest requirements. Note that this command only supports OAP unicast, and does not allow the wildcard IID.

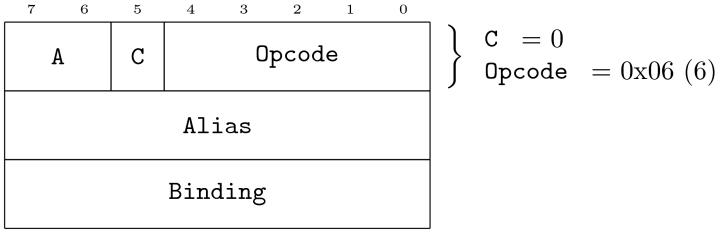
Typically, requestors and proxies use the Full Connect command unless they know that the provider will be unable to open a session back to the requestor. Unless this is the case, nodes forward Connect commands as Full Connect commands.

4.6. Define

Requests an interface definition from a node.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast
Security: Session.
Permission (command): OAP Requestor
Permission (response): OAP Define or OAP Binding

Define `dof-2008-2-pdu-19`

COMMAND/DIRECTED or FLOODED
OPID
COMPLETE ON FINAL
Instance of OAP Binding-Oriented Command`dof-2008-2-pdu-1`



`A` **Two bits.** Controls the format of `Binding`.

`C` **One bit.** Controls the presence of `???`.

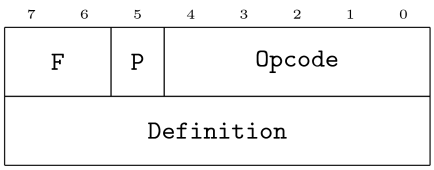
`Opcode` **Five bits.** Determines the type of the command.

`Alias` **Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.

`Binding` **Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

Define `dof-2008-2-pdu-20`

RESPONSE
FINAL
Instance of OAP Response`dof-2008-2-pdu-3`



`F` **Two bits.** Flags based on the `Opcode`.

`P` **One bit.**

`Opcode` **Five bits.** Determines the type of the response.

`Definition` **Instance of** Interface Definition`dof-2008-2-pdu-43`.The interface definition.

⇄ Interface Definition `dof-2008-2-pdu-43`

Type Context
Property Count
Property List
Event Count
Event List
Method Count
Method List
Exception Count
Exception List

`Type Context` **Instance of** Type Context`dof-2009-1-pdu-39`.This structure defines all of the types that are used in the definition.

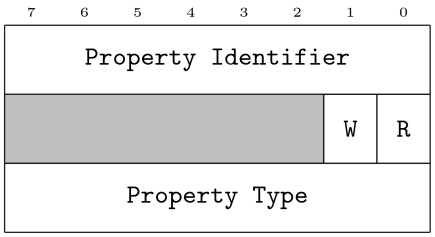
`Property Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This controls the length of the `Property List`.

`Property List` **Array of** Property`dof-2008-2-pdu-21`.**Optional, controlled by** `Property Count`. List of properties in the definition.

`Event Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This controls the length of the `Event List`.

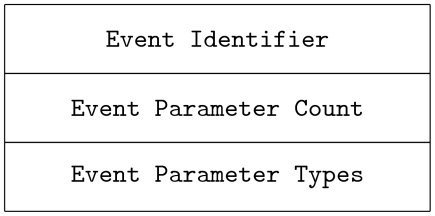
`Event List`	**Array of** Event`dof-2008-2-pdu-22`.**Optional, controlled by** `Event Count`. List of events in the definition.
`Method Count`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This controls the length of the `Method List`.
`Method List`	**Array of** Method`dof-2008-2-pdu-23`.**Optional, controlled by** `Method Count`. List of methods in the definition.
`Exception Count`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.This controls the length of the `Exception List`.
`Exception List`	**Array of** Method`dof-2008-2-pdu-23`.**Optional, controlled by** `Exception Count`. List of exceptions in the definition.

Property `dof-2008-2-pdu-21`



`Property Identifier`	**Instance of** Item Identifier`dof-2008-2-pdu-4`.The item identifier of the described property.
`W`	**One bit.** One if the property is writable, zero otherwise.
`R`	**One bit.** One if the property is readable, zero otherwise.
`Property Type`	**Instance of** Type Reference`dof-2009-1-pdu-40`.The type associated with the property.

Event `dof-2008-2-pdu-22`



`Event Identifier`	**Instance of** Item Identifier`dof-2008-2-pdu-4`.The item identifier of the described event.

`Event Parameter Count` **One byte.** This controls the length of the `Event Parameter Types`.

`Event Parameter Types` **Array of** Type Reference`dof-2009-1-pdu-40`.**Optional, controlled by** `Event Parameter Count`. This list contains one type for each event parameter.



Method `dof-2008-2-pdu-23`

Method Identifier
Method Input Parameter Count
Method Input Parameter Types
Method Output Parameter Count
Method Output Parameter Types

`Method Identifier` **Instance of** Item Identifier`dof-2008-2-pdu-4`.The item identifier of the described method.

`Method Input Parameter Count` **One byte.** This controls the length of the `Method Input Parameter Types`.

`Method Input Parameter Types` **Array of** Type Reference`dof-2009-1-pdu-40`.**Optional, controlled by** `Method Input Parameter Count`. This list contains one type for each input parameter.

`Method Output Parameter Count` **One byte.** This controls the length of the `Method Output Parameter Types`.

`Method Output Parameter Types` **Array of** Type Reference`dof-2009-1-pdu-40`.**Optional, controlled by** `Method Output Parameter Count`. This list contains one type for each event parameter.



Exception `dof-2008-2-pdu-24`

Exception Identifier
Exception Parameter Count
Exception Parameter Types

`Exception Identifier`	**Instance of** Item Identifier`dof-2008-2-pdu-4`.The item identifier of the described exception.
`Exception Parameter Count`	**One byte.** This controls the length of the `Exception Parameter Types`.
`Exception Parameter Types`	**Array of** Type Reference`dof-2009-1-pdu-40`.**Optional, controlled by** `Exception Parameter Count`. This list contains one type for each event parameter.

4.6.1. Small Implementation Hints

Even small implementations should support this command, but it is not required unless interfaces in registry zero (0) (the dynamic registry) are provided.

4.6.2. Flows

Define() → response() : Success.

Define() → (none) : Definition not known.

4.6.3. Details

Requestors use the Define command to request an interface definition. This command requires unicast over lossy transports. If possible, receivers should ignore broadcast and multicast Define commands in order to prevent overrunning the requestor with responses.

The responder should respond immediately if it knows the interface definition.

By definition, registered interface definitions with the same interface identifier are identical. This means that in general requestors and proxies may use any matching response for a request. However, the security rules prohibit communications between incompatible partitions, so the response must come from a compatible partition.

The use of a binding to identify the interface requires explanation. There are two primary classes of interface identifier: registered and dynamic. Dynamic interfaces are always in registry zero (0). All other interfaces are registered (for the purposes of this command).

The following table represents the requirements for accepting a response based on the type of request and the permissions of the responder:

Type of Request\Conditions	**Conditions for Forwarding or Responding**	**Permissions Required for Responder**
Specific Binding, Either Dynamic (registry 0) or Registered	Response sent if currently providing the binding. If the definition is not known then forward the request to the provider.	Binding permission that includes the Provide action on the specified Binding.
Broadcast OID (which implies Registered IID)	Response sent if definition is known. Otherwise, forward as described below.	Define permission on the IID.

There is no defined error response to this request, other than no response. The reason for this is that the requestor is likely not keeping track of the specific nodes that the request was forwarded to, and so it is

unclear when a request can no longer be satisfied (even if many nodes return error responses). This is a trade-off between requestor and responder state and responsiveness.

Responders use the Define response to define the requested interface. Providers of dynamic interface definitions (registry zero) must not respond to requests that use a broadcast OID.

The Define response includes variable-length lists of types, properties, events, methods, and exceptions contained in the interface definition. Any of these lists may have a count of 0 (zero), indicating that there are no definitions for that type of item in the definition.

Each Define response item list must appear in order of increasing Item Identifier. `dof-2008-2-spec-22`

In order to ensure that byte comparison of interface definitions yields correct results, all item definition blocks must appear in order of increasing item identifier within their respective lists.

Duplicate Item Identifiers in a Define response are invalid. `dof-2008-2-spec-23`

It is invalid to have a single interface definition with duplicated Item Identifiers. Receivers must reject any such definition.

4.6.4. Routing Rules

Proxies may forward this command to other nodes if the proxy does not know the requested definition. Proxies may forward broadcast OID requests to any provider of the binding and to configured nodes or nodes that have an appropriate OAP Define permission. Requests that use a specific binding follow the route of the corresponding primary Provide, and if there is no primary provider then the proxy drops the operation.

This means that requests using a specific binding are directed (as the primary provider route is a DAG). Requests using broadcast are flooded.

4.6.5. Consolidation Rules

Define commands should be consolidated if they are of the same type (broadcast or specific). The Define response must match the original request for each requestor. For requests in compatible security partitions, proxies may treat any response as valid for any matching request. This means that proxies can use the first response that arrives for all current commands and that any additional responses may be silently dropped. This also means that responses may be cached and used in responses to later commands, again only within compatible security partitions. Responses are final, causing removal of operation state. This is required to prevent ringing of responses sent to multiple requestors.

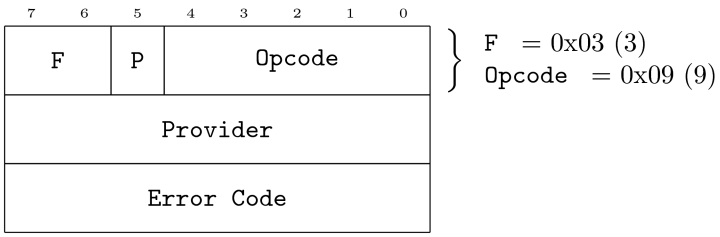
4.7. Exception

Notifies a requestor that an operation has failed or sent some other asynchronous information.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): N/A
Permission (response): OAP Binding(Provide)

See Get, Invoke, Open, Register, Set, Subscribe.

Internal Exception (response) `dof-2008-2-pdu-25`

RESPONSE
FINAL
Instance of OAP Response`dof-2008-2-pdu-3`



`F` **Two bits.** Flags based on the `Opcode`.

`P` **One bit.** Controls the presence of `Provider`.

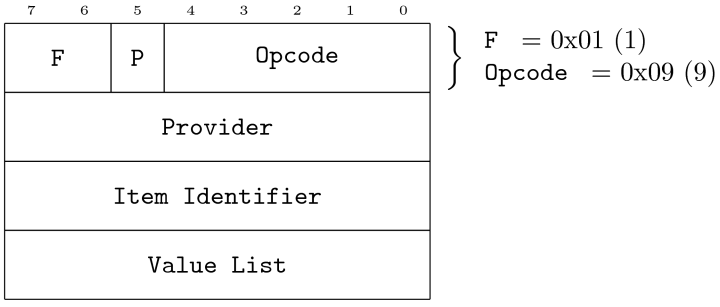
`Opcode` **Five bits.** Determines the type of the response.

`Provider` **Instance of** Object Identifier (OID)`dof-2009-1-pdu-4`.**Optional, controlled by** `P`. If `P` is one then this field is present. Identifies the provider of the response. This field is required in responses to OAP multicast and broadcast commands. Responses to OAP unicast commands should not use this field.

`Error Code` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The error code to return.

Provider Exception (response) `dof-2008-2-pdu-26`

RESPONSE
FINAL or INTERMEDIATE
Instance of OAP Response`dof-2008-2-pdu-3`



`F` **Two bits.** Flags based on the `Opcode`.

`P` **One bit.** Controls the presence of `Provider`.

`Opcode`	**Five bits.** Determines the type of the response.
`Provider`	**Instance of** Object Identifier (OID)$_{\texttt{dof-2009-1-pdu-4}}$.**Optional, controlled by** `P`. If `P` is one then this field is present. Identifies the provider of the response. This field is required in responses to OAP multicast and broadcast commands. Responses to OAP unicast commands should not use this field.
`Item Identifier`	**Instance of** Compressed Unsigned 16-Bit$_{\texttt{dof-2009-1-pdu-1}}$.The item identifier of the reported exception.
`Value List`	**Instance of** Value List$_{\texttt{dof-2008-2-pdu-8}}$.The values associated with the exception.

4.7.1. Small Implementation Hints

This response represents all operation failures in the system, and so it must be supported at least in a limited form.

Provider-only nodes may silently drop this command.

4.7.2. Details

The Exception response indicates failure of an operation, temporarily or permanently, or provides some intermediate information.

The two different variations of the response are used based on whether the exception is an "internal" or a "provider" exception. Internal exceptions use an error code. In this case, there is no **Item Identifier** or **Value List** and only an **Error Code**.

Provider exceptions reference an **Item Identifier** in the same interface as the corresponding request, and may or may not have a **Value List** as specified in the interface definition.

The sending of this response depends on the **V** (Verbosity) field of the corresponding operation. All exceptions require at least a verbosity of 2 (two) in order to be sent.

This response usually takes the place of the normal completion response. However, the DPP FINAL/INTERMEDIATE indicator, which controls DPP lifecycle, governs completion of the operation.

If the operation that generates the exception used a broadcast or multicast OID then the exception must include a **Provider** that contains the object that caused the exception.

The following are defined values of **Error Code**. The ODP-TSC controls all internal error codes. Note that this document uses the short name to reference the particular error, and may or may not correspond to specific error codes in any implementations.

Table 4.1. Exception Error Codes

Reason	Short Name	Description
2	Unsupported	The request does not match the interface (for example, a Get on a write-only property, or an Invoke on an Event). See Get, Invoke, Open, Register, Set, Subscribe.

Reason	Short Name	Description
3	Illegal Parameter	A value or value list is improper. See Get, Invoke, Set, Signal, Subscribe.
4	Insufficient Resources	A node, either a proxy or the provider, does not have resources available to accept or process the command. All commands may return this exception.
5	Terminated	A provider or proxy terminated the operation. See Open, Register, Subscribe.
6	Subscribe Rejected	A provider rejects a subscription – the provider does not support subscriptions on that binding and item. See Subscribe.
7	Binding Not Found	A request was received that should match an existing binding, but no matching binding can be found. See Get, Invoke, Open, Register, Set, Subscribe.
8	Duplicate Session	The requested Session ID is already in use. Retry with a different Session ID. See Open.
9	Too Many Sessions	The provider has reached a maximum supported session count on the requested binding. The request may succeed later. See Open.
10	Application Error	An error occurred in application code, but prevents generation of a valid response. All commands that return responses may return this exception.



The ENC-TC controls Exception error codes. `dof-2008-2-spec-24`

Requestors treat unknown error codes as generic failures.

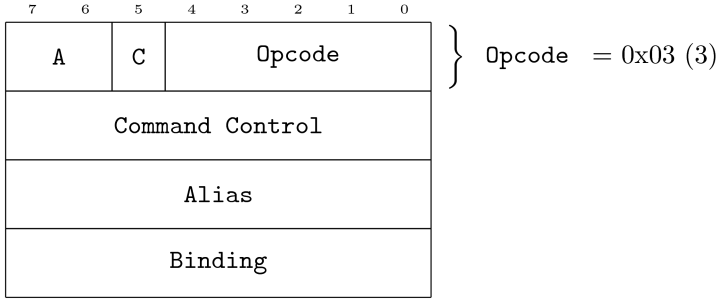
4.8. Full Connect

Request that matching provide commands be sent to the sender.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Requestor
Permission (response): N/A

See Activate, Advertise, Connect, Provide and Watch.

Full Connect (command) `dof-2008-2-pdu-27`

COMMAND/FLOODED
OPID
Instance of OAP Binding-Oriented Command`dof-2008-2-pdu-1`



`A`	**Two bits.** Controls the format of `Binding`.
`C`	**One bit.** Controls the presence of `Command Control`.
`Opcode`	**Five bits.** Determines the type of the command.
`Command Control`	**Instance of** Command Control`dof-2008-2-pdu-7`.**Optional, controlled by** `C`. If `C` is one then this field is present. This field determines options that apply to the command.
`Alias`	**Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.
`Binding`	**Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

4.8.1. Small Implementation Hints

All provider nodes must typically react to this command. The only exception is a provider that always provides and maintains its bindings over all sessions and transports *and* does not support lossless sessions. Such a node would ignore this command. However, such minimal implementations are highly discouraged because of the load that they put on the network.

Requestor-only nodes may silently drop this command.

4.8.2. Flows

Full Connect() $\rightarrow$ (none) : Success.

4.8.3. Details

The Full Connect command is a superset of the Connect command. In addition to the behaviors of the Connect command, this command requests that the receiver establish and maintain a lossless session with

the sender. Note that the difference between this and the Connect command is the direction of the requested session.

See the section on the Interest class of operations for common interest requirements. Note that this command only supports OAP unicast, and does not allow the wildcard IID.

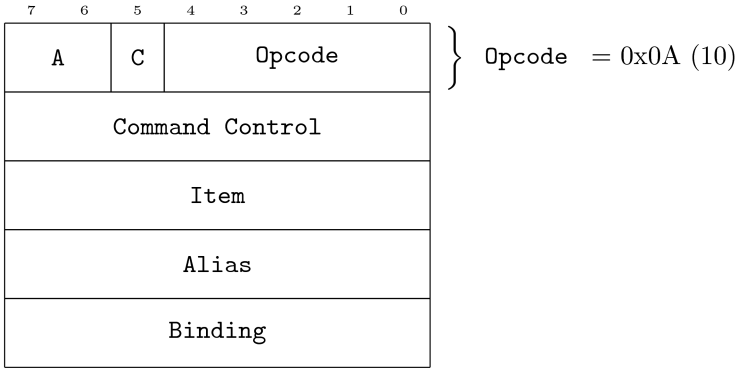
4.9. Get

Gets the current value of a property.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Binding(Read)
Permission (response): OAP Binding(Provide)

See Exception.

⇄ Get (command) `dof-2008-2-pdu-28`

COMMAND/DIRECTED or FLOODED
OPID
Instance of OAP Item-Oriented Command`dof-2008-2-pdu-2`

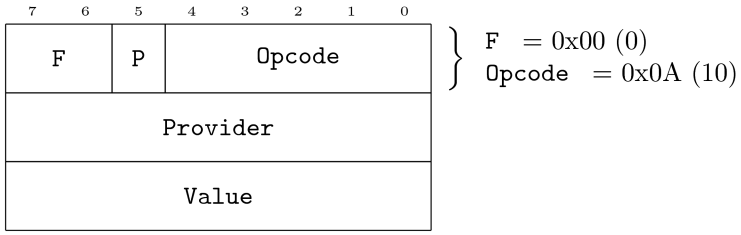


`A`	**Two bits.** Controls the format of `Binding`.
`C`	**One bit.** Controls the presence of `Command Control`.
`Opcode`	**Five bits.** Determines the type of the command.
`Command Control`	**Instance of** Command Control`dof-2008-2-pdu-7`.**Optional, controlled by** `C`. If `C` is one then this field is present. This field determines options that apply to the command.
`Item`	**Instance of** Item Identifier`dof-2008-2-pdu-4`.Defines the item for the command.
`Alias`	**Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.

`Binding`	**Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

⇄ **Get (response)** **`dof-2008-2-pdu-29`**

RESPONSE
Instance of OAP Response`dof-2008-2-pdu-3`



`F`	**Two bits.** Flags based on the `Opcode`.
`P`	**One bit.** Controls the presence of `Provider`.
`Opcode`	**Five bits.** Determines the type of the response.
`Provider`	**Instance of** Object Identifier (OID)`dof-2009-1-pdu-4`.**Optional, controlled by** `P`. If `P` is one then this field is present. Identifies the provider of the response. This field is required in responses to OAP multicast and broadcast commands. Responses to OAP unicast commands should not use this field.
`Value`	**Instance of** Value List`dof-2008-2-pdu-8`.The returned property value.

4.9.1. Small Implementation Hints

Providers only require support for this command if they provide readable properties. A default handler that returns an Exception is permissible.

Requestor-only nodes may silently drop this command.

4.9.2. Flows

Get() $\rightarrow$ response() : Success.

Get() $\rightarrow$ Exception() : Error.

Get() $\rightarrow$ (none) : No match.

4.9.3. Details

Requestors use the Get command to obtain the current value of a property from the specified binding(s). The Get response contains the requested value, and multiple responses indicate multiple providers. Errors may result in an Exception response.

See the section on the Action class of operations for common action requirements.

Related to command control, note in particular the option to allow the return of cached data values. Normally the Get command requires getting the current value from the provider. However, there are situations where intermediate nodes know old values of the property. An example is that a node that is maintaining a subscription to the property may have cached values. In order to increase performance and allow the Get command to access this cached data the command control field can specify a maximum data age that is acceptable. A value of zero prohibits use of cached data.

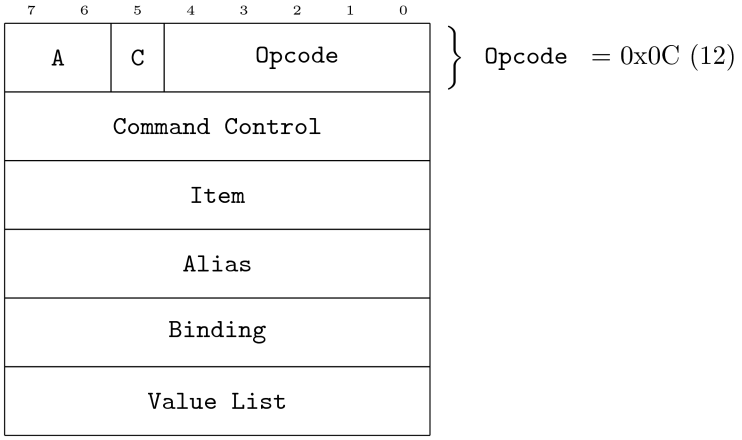
4.10. Invoke

Execute a method.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Binding(Execute)
Permission (response): OAP Binding(Provide)

See Exception.

⇄ Invoke (command) `dof-2008-2-pdu-30`

COMMAND/DIRECTED or FLOODED
OPID
Instance of OAP Item-Oriented Command`dof-2008-2-pdu-2`



`A` **Two bits.** Controls the format of `Binding`.

`C` **One bit.** Controls the presence of `Command Control`.

`Opcode` **Five bits.** Determines the type of the command.

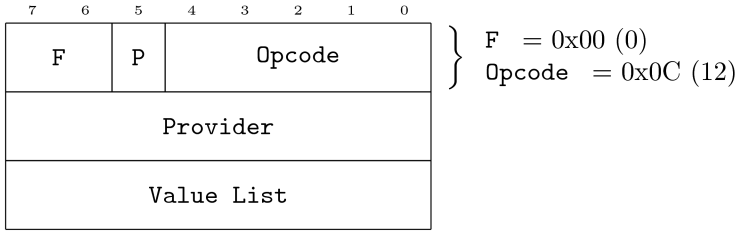
`Command Control` **Instance of** Command Control`dof-2008-2-pdu-7`.**Optional, controlled by** `C`. If `C` is one then this field is present. This field determines options that apply to the command.

`Item` **Instance of** Item Identifier`dof-2008-2-pdu-4`.Defines the item for the command.

`Alias`	**Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length `A`. Defines the alias for the `Binding` when it is not present.
`Binding`	**Instance of** Item Identifier`dof-2008-2-pdu-4`.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.
`Value List`	**Instance of** Value List`dof-2008-2-pdu-8`.Input parameters to the method, which may be empty (void).

⇄ **Invoke (response)** `dof-2008-2-pdu-31`

RESPONSE
Instance of OAP Response`dof-2008-2-pdu-3`



`F`	**Two bits.** Flags based on the `Opcode`.
`P`	**One bit.** Controls the presence of `Provider`.
`Opcode`	**Five bits.** Determines the type of the response.
`Provider`	**Instance of** Object Identifier Attribute`dof-2009-1-pdu-5`.**Optional, controlled by** `P`. If `P` is one then this field is present. Identifies the provider of the response. This field is required in responses to OAP multicast and broadcast commands. Responses to OAP unicast commands should not use this field.
`Value List`	**Instance of** Value List`dof-2008-2-pdu-8`.Return parameters of the method, which may be empty (void).

4.10.1. Small Implementation Hints

Providers only require support for this command if they provide methods. A default handler that returns an Exception is permissible.

Requestor-only nodes may silently drop this command.

4.10.2. Flows

Invoke() $\rightarrow$ response() : Success.

Invoke() $\rightarrow$ Exception() : Error or not supported.

Invoke() $\rightarrow$ (none) : No match.

4.10.3. Details

Requestors use the Invoke command to execute methods on the specified binding(s), providing any required input parameters. The Invoke response contains the return parameters (if any), and multiple responses indicate multiple providers. Errors may result in an Exception response.

See the section on the Action class of operations for common action requirements.

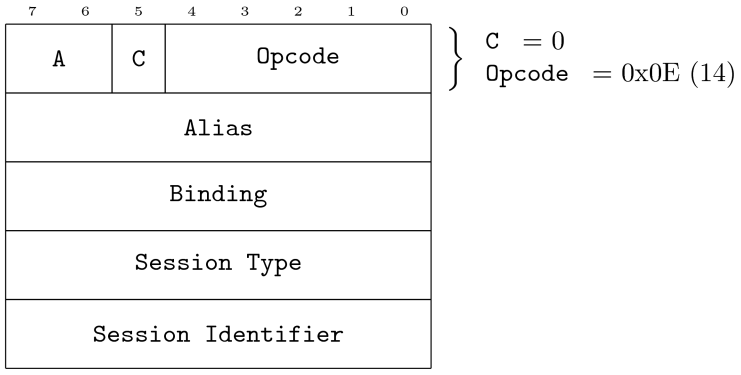
4.11. Open

Create an end-to-end session between a requestor and a provider.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Binding(Open)
Permission (response): OAP Binding(Provide)

See Exception, Provide.

⇄ Open (command) `dof-2008-2-pdu-32`

COMMAND/DIRECTED
OPID
Instance of OAP Binding-Oriented Command`dof-2008-2-pdu-1`

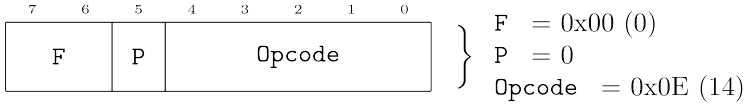


`A`	**Two bits.** Controls the format of `Binding`.
`C`	**One bit.** Controls the presence of `???`.
`Opcode`	**Five bits.** Determines the type of the command.
`Alias`	**Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.
`Binding`	**Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

`Session Type`	**Instance of** Interface Identifier (IID)`dof-2009-1-pdu-21`.An indicator of either a specific requested binding or a more generic session type indicator.
`Session Identifier`	**Instance of** Object Identifier (OID)`dof-2009-1-pdu-4`.A unique identifier for the session, the same attribute that the provider will be add to each provided binding. The A flag of the attribute must be clear. The attribute value must be globally unique, otherwise the session may fail.

⇄ **Open (response)** `dof-2008-2-pdu-33`

RESPONSE
Instance of OAP Response`dof-2008-2-pdu-3`



`F`	**Two bits.** Flags based on the `Opcode`.
`P`	**One bit.** Controls the presence of `???`.
`Opcode`	**Five bits.** Determines the type of the response.

4.11.1. Small Implementation Hints

Providers only require support for this command if they provide sessions. A default handler that returns an Exception is permissible.

Requestor-only nodes may silently drop this command.

4.11.2. Flows

Open() $\rightarrow$ response() : Success.

Open() $\rightarrow$ Exception() : Error or not supported.

4.11.3. Details

The Open command creates a session between the requestor and the provider based on the specified attribute, the session identified by the unique attribute value used with each resulting binding. The Open response indicates success. Errors may result in an Exception response.

See the section on the Relationship and Action classes of operations for common requirements. The Relationship requirements govern the Open response.

This command requires OAP unicast. The primary provider (established previously) is the source of the session. The reactionary commands defined as part of the relationship follow the rule of requiring that the primary provider send them. The fact that these bindings originate at the same provider as the "base"

binding and that they use the same "base" OID modified only by adding another attribute, means that the interest command causing the Provide of the base will also cause the Provide of the reactionary bindings – in other words they will all have the same primary provider.

The **Session Type** defines the session requested. It has an operation state type of Invariant. This represents an interface identifier that the provider should provide on the session, however, it can also be more generic and used only as a type of registered flag for the type of session requested. For example, a session may represent different views of an object. In this case, the **Session Type** indicates the type of view even if not provided.

Providers use a slightly modified sequence of checks before responding to Open. After the other checks and before sending a successful response, providers:

- Check if the session identifier is already in use on the provider, and if so send Exception (Duplicate Session).
- Check if the provider already has too many open sessions, and if so send Exception(Too Many Sessions).

Once accepted and responded to, the behavior on the provider is to introduce new bindings. The bindings use the OID from the original request, augmented with the attribute specified in the request. The provider determines the interface identifiers provided and should include the **Session Type** (although the node may provide others simultaneously).

There are three critical differences between these bindings and others, which place requirements not only on providers but also on proxies:

- The specified **Session Identifier** exists on the OID of each binding. The **Session Identifier** has operation state type of Invariant.
- The provider and any proxies allow access to the binding *only* from the original requestor who sent the Open command. Attempted access from any other requestor must result in the node silently dropping the command.
- Only the provider of the original "base" binding may provide bindings in the session. Attempted provides using the same session identifier from other providers must result in the command being silently dropped. Nodes view the Provide as a reactionary command in a relationship, which limits the reactionary commands to the primary provider.

4.11.4. Command Control

The Open command does not use command control. However, it does use Exception responses. Since this command must use OAP unicast, providers may send Exception (the default value of Verbosity for OAP unicast is two).

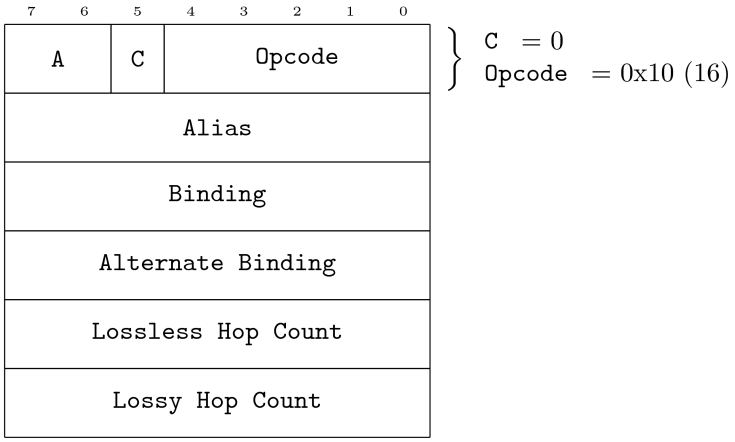
4.12. Provide

Notify a requestor of a binding.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Binding(Provide)
Permission (response): N/A

See Connect, Full Connect, Watch, Activate, Open and Advertise.

Provide (command) `dof-2008-2-pdu-34`

COMMAND/DIRECTED or FLOODED
OPID
Instance of OAP Binding-Oriented Command`dof-2008-2-pdu-1`



`A`	**Two bits.** Controls the format of `Binding`. This field also controls the format of the `Alternate Binding`.
`C`	**One bit.**
`Opcode`	**Five bits.** Determines the type of the command.
`Alias`	**Variable length. Optional, controlled by** A. If A is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.
`Binding`	**Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** A. If A is zero then this field is present. Defines the binding for the command.
`Alternate Binding`	**Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** A. If A is not zero then this field is present. Defines the binding for the command.
`Lossless Hop Count`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The number of lossless hops to the sender from the original provider.
`Lossy Hop Count`	**Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.The number of lossy hops to the sender from the original provider.

4.12.1. Small Implementation Hints

All nodes must handle this command. It is possible to "preemptively" send Provide commands and ignore Watch commands and (to some extent) Activate, Connect and Full Connect commands. This may be easier for some restricted implementations. It may still be easier to respond to interest by sending a Provide, or at least triggering a retry.

For the original provider, the hop counts at the end of the PDU can be omitted entirely (they would both be zero for the original provider).

Provider-only nodes may silently drop this command.

4.12.2. Flows

Provide() $\rightarrow$ (none) : Success.

4.12.3. Details

A provider uses this command to define that it is currently providing the given binding. This information propagates through the network to requestors based on the interest that they express. When received by a requestor it indicates the presence of a binding, which may result in a commands directed to the binding. The Provide commands result in the creation of a DAG for each binding, which other commands may use to direct operations to the binding.

The Provide command is available on all transports.

The binding that it contains must be OAP unicast. Receivers must silently ignore any attempt to use other bindings in this command.

The Provide command is a reaction sent to interest commands, although it does not form a relationship. The combination of all interest commands is not a DAG (discussed in the *DOF Protocol Specification*), which requires that proxy nodes send Provide commands as flooded. The original provider of a binding should use a flooded operation, but may use a directed operation. Any proxy node must convert the directed operation to a flooded operation. This requires each node to track the operation state, and means that each Provide command is unique and identified. A node may forward two separate Provide commands to the same requestor.



Multiple bindings expressed from the same node must use correct operation identifiers. `dof-2008-2-spec-25`

This means that if the same operation identifier is used, then it must represent the same binding. If different operation identifiers are used then the binding must be different *or* it must represent multiple, independent providers of the same binding.

Each binding should define an associated alias. Aliases must maintain their association for the duration of the operation. Providers define the uniqueness of an alias, and they need not change even if provider removes and re-adds the binding. It is illegal, however, to add multiple different bindings with the same alias where confusion could occur. If a provider provides the same binding multiple times it must use the same alias for each Provide.

If the provider is not assigning an alias then the **A** field must be zero. The **Binding** field, if an alias, must not use all zero bytes.



Multiple identical bindings provided from the same node must use the same alias. `dof-2008-2-spec-26`

A node must not provide the same binding with different aliases. An alias must present the binding, independent of how many times the provider provides it. However, if the node stops providing the binding entirely and then begins to provide it again in the future, a different alias may be used. Network reordering can cause race conditions that should be avoided.

Providers must not use the same alias for different bindings at the same time.

`dof-2008-2-spec-27`

This is a serious error, and should cause the requestor to close the session on which it occurs.

Note that these requirements have a large impact on nodes that reboot. In this case, before a node sends a Provide, the node must cancel all of its (potential) previous Provide commands. Alternatively, it can assign aliases to its own bindings in a way that always generates the same resulting alias/binding pairs (which may be useful for small implementations).

Each new interest operation should result in a corresponding Provide operation, even if it is a retry of a previous Provide.

4.12.4. Primary Providers

Each proxy provider must establish a primary provider, and this must result in a DAG for each binding. This requirement applies to the entire system, meaning that from any node the path to a given binding must be direct and not contain loops.

In the simple case of a single provider, the Provide operation source is the root of a DAG. In the more complex case of multiple providers then rules must ensure that all nodes pick the appropriate provider so that loops cannot exist. In order to do this, the primary provider is the provider with the minimum value of **Lossless Hop Count** + (**Lossy Hop Count** << 15). This function results in picking the shortest lossless route, otherwise it picks the most lossless route.

In the case of a tie, the application may provide input, or the node may use either operation. Note that since the Provide operation uses the flooding DPP model the sender ID should be specified in most cases. Nodes connected to the original provider may need to determine the SID to do the comparison. The comparison rules for operation identifiers are defined in *DOF Common Types*.

4.12.5. Routing Rules

Nodes must forward Provide commands to all nodes that have expressed interest in a compatible binding. See the Interest section for information, and the section on bindings for the definition of compatibility. However, nodes forward bindings that include the Session attribute only to the sender of the matching Open command (in addition to requiring interest from that node).

Nodes must update the hop counts for each Provide they send. This can be done by updating the hop counts when the Provide operation is received (and based on the transport it is received on), and then forwarding that same information.

4.12.6. Consolidation Rules

Nodes are only required to forward information for their primary provider (defined earlier). However, they must accurately forward the operation. This means that as the primary provider changes, that the node must forward that specific operation (since it is flooded).

Not forwarding additional Provide operations saves state on the receiver, and so nodes should only forward a single Provide command for any given binding.

In the case where nodes do not forward all Provide operations, the following sections discuss additional requirements.

4.12.6.1. Forwarded Information

The outbound Provide command requires certain source-based information, like the alias. It is up to the node to determine appropriate values for these fields, guided by the meaning that the fields have.

4.12.6.2. Monitoring

A proxy must monitor the available Provide commands. At a minimum, it must cease to be a provider upon removal of all of its available Provide commands. This monitoring may involve forwarding changes to values as the corresponding input values change.

4.12.6.3. Application Input

The application may provide input to the determination of the primary provider only in cases of a tie.

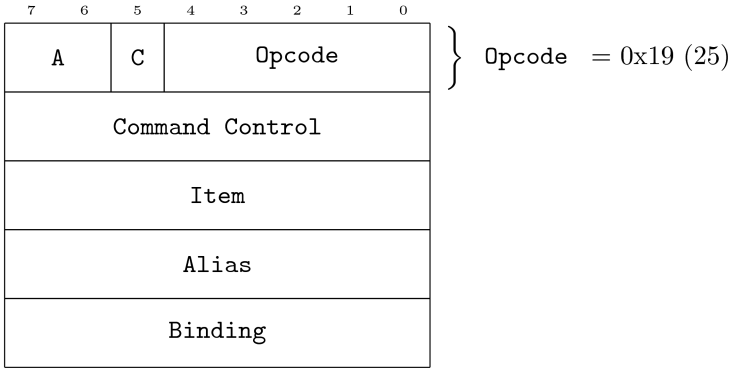
4.13. Register

Registers a node with an event, requesting that it send signals.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Binding(Read)
Permission (response): OAP Binding(Provide)

See Signal and Exception.

Register (command) `dof-2008-2-pdu-35`

COMMAND/DIRECTED or FLOODED
OPID
Instance of OAP Item-Oriented Command`dof-2008-2-pdu-2`



`A` **Two bits.** Controls the format of `Binding`.

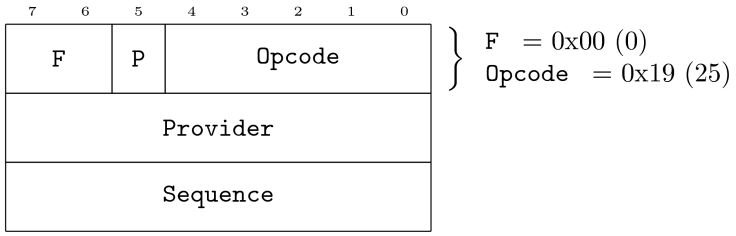
`C` **One bit.** Controls the presence of `Command Control`.

`Opcode` **Five bits.** Determines the type of the command.

`Command Control`	**Instance of** Command Control`dof-2008-2-pdu-7`.**Optional, controlled by** `C`. If `C` is one then this field is present. This field determines options that apply to the command.
`Item`	**Instance of** Item Identifier`dof-2008-2-pdu-4`.Defines the item for the command.
`Alias`	**Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.
`Binding`	**Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

⇄ Register (response) `dof-2008-2-pdu-36`

RESPONSE
Instance of OAP Response`dof-2008-2-pdu-3`



`F`	**Two bits.** Flags based on the `Opcode`.
`P`	**One bit.** Controls the presence of `Provider`.
`Opcode`	**Five bits.** Determines the type of the response.
`Provider`	**Instance of** Object Identifier (OID)`dof-2009-1-pdu-4`.**Optional, controlled by** `P`. If `P` is one then this field is present. Identifies the provider of the response. This field is required in responses to OAP multicast and broadcast commands. Responses to OAP unicast commands should not use this field.
`Sequence`	**Two bytes, MSB.** Contains the provider's current signal sequence for the item, expressed in a floating window.

4.13.1. Small Implementation Hints

Even minimal providers should only send Signal commands if there are active Register commands. However, they may take short cuts regarding how accurately they determine active Register commands. Providers with no events may trigger a common exception.

Requestor-only nodes may silently drop this command.

4.13.2. Flows

Register() → (response), [Signal() …] : Success.

Register() → Exception() : Immediate failure.

Register() → (response), [Signal() …], Exception() : Register cancelled by the provider or proxy.

4.13.3. Details

The Register command creates a relationship between the requestor and providers based on the specified event. The Register response indicates success. Errors may result in an Exception response.

See the section on the Relationship and Action classes of operations for common requirements. The Relationship requirements govern the Register response and routing rules.

The relationship requests that providers send Signal commands to the requestor.

Providers and proxies respond to indicate acceptance of the registration. The response includes the current **Sequence**, which requestors use to preserve the order of Signal commands, prevent duplicates, and detect missed or lost Signal commands. The Signal command discusses these requirements further.

The use of an alias on this command allows, but does not require, the use of an alias on the reactionary Signal commands. The use of a full binding on this command requires the use of a full binding on the corresponding Signal commands. However, any active Request that uses an alias from a node allows a Signal to use an alias to that node.

4.13.4. Consolidation Rules

Nodes should consolidate OAP unicast Register commands. Nodes may consolidate flooded commands according to the general consolidation rules.

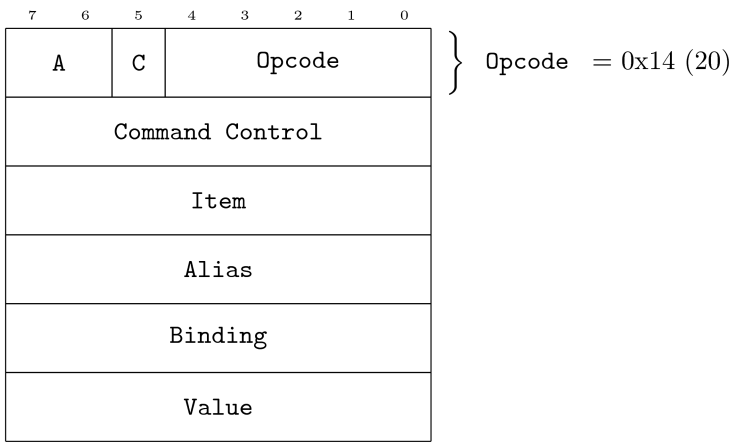
4.14. Set

Sets the current value of a property.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Binding(Write)
Permission (response): OAP Binding(Provide)

See Exception.

⇄ **Set (command)** `dof-2008-2-pdu-37`

COMMAND/DIRECTED or FLOODED
OPID
Instance of OAP Item-Oriented Command`dof-2008-2-pdu-2`



`A` **Two bits.** Controls the format of `Binding`.

`C` **One bit.** Controls the presence of `Command Control`.

`Opcode` **Five bits.** Determines the type of the command.

`Command Control` **Instance of** Command Control`dof-2008-2-pdu-7`.**Optional, controlled by** `C`. If `C` is one then this field is present. This field determines options that apply to the command.

`Item` **Instance of** Item Identifier`dof-2008-2-pdu-4`.Defines the item for the command.

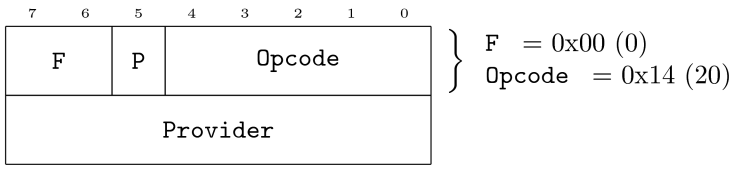
`Alias` **Variable length. Optional, controlled by** `A`. If `A` is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the `Binding` when it is not present.

`Binding` **Instance of** Binding`dof-2008-2-pdu-5`.**Optional, controlled by** `A`. If `A` is zero then this field is present. Defines the binding for the command.

`Value` **Instance of** Value List`dof-2008-2-pdu-8`.The value to set the property to.

⇄ Set (response) `dof-2008-2-pdu-38`

RESPONSE
Instance of OAP Response`dof-2008-2-pdu-3`



`F` **Two bits.** Flags based on the `Opcode`.

`P`	**One bit.** Controls the presence of `Provider`.
`Opcode`	**Five bits.** Determines the type of the response.
`Provider`	**Instance of** Object Identifier (OID)`dof-2009-1-pdu-4`.**Optional, controlled by** `P`. If `P` is one then this field is present. Identifies the provider of the response. This field is required in responses to OAP multicast and broadcast commands. Responses to OAP unicast commands should not use this field.

4.14.1. Small Implementation Hints

Providers only require support for this command if they provide writable properties. A default handler that returns an Exception is permissible.

Requestor-only nodes may silently drop this command.

4.14.2. Flows

Set() $\rightarrow$ response() : Success.

Set() $\rightarrow$ Exception() : Failure.

Set() $\rightarrow$ (none) : No match.

4.14.3. Details

Requestors use the Set command to change the current value of a property one the specified binding(s). The Set response indicates success, and multiple responses indicate multiple providers. Errors may result in an Exception response.

See the section on the Action class of operations for common action requirements.

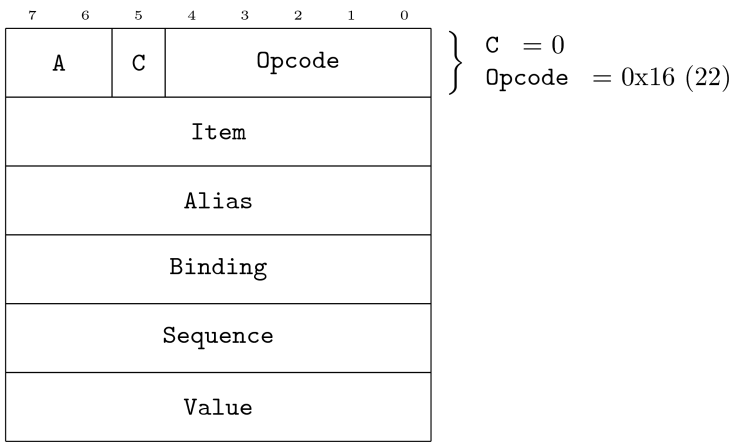
4.15. Signal

Signals an Event.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Binding(Provide)
Permission (response): N/A

See Register.

⇄ **Signal (command)** `dof-2008-2-pdu-39`

COMMAND/DIRECTED
NOOPID
Instance of OAP Item-Oriented Command`dof-2008-2-pdu-2`



A	**Two bits.** Controls the format of Binding.
C	**One bit.** Controls the presence of Command Control.
Opcode	**Five bits.** Determines the type of the command.
Command Control	**Instance of** Command Control$_{\text{dof-2008-2-pdu-7}}$.**Optional, controlled by** C. If C is one then this field is present. This field determines options that apply to the command.
Item	**Instance of** Item Identifier$_{\text{dof-2008-2-pdu-4}}$.Defines the item for the command.
Alias	**Variable length. Optional, controlled by** A. If A is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the Binding when it is not present.
Binding	**Instance of** Binding$_{\text{dof-2008-2-pdu-5}}$.**Optional, controlled by** A. If A is zero then this field is present. Defines the binding for the command.
Sequence	**Two bytes, MSB.** Contains the provider's current change sequence, expressed in a floating window.
Value	**Instance of** Value List$_{\text{dof-2008-2-pdu-8}}$.The values associated with the signal.

4.15.1. Small Implementation Hints

If a requestor or provider knows that it does not support or use registrations, then it does not need to support this command.

Provider-only nodes may silently drop this command.

4.15.2. Flows

Signal() $\rightarrow$ (none) : Success.

4.15.3. Details

Providers and proxies use the Signal command to pass information about signals that occur while a registration is active on that event. Nodes should only send this command to nodes with active Register commands to the event.

Signal is a reactionary command inside of a relationship formed by the Register command. See the section on the Relationship class of operations for common requirements.

Registrations exist over a variety of transports, and may exist at the same time from a variety of requestors. However, the definition and behavior of the Register command (the relationship operation type) ensures the formation of a DAG, so the Signal command does not need to provide loop prevention. As an optimization, a node that does not currently Provide the binding (even if it could) can eliminate sending Signal. This is because nodes drop the Signal coming from any node other than the primary provider.

The binding present in the command must use OAP unicast. Providers may use an alias only in response to a registration that itself used an alias; otherwise, the provider must use a full binding. Providers may send a single Signal command (for a given binding and item) to a node even if the node has multiple outstanding registrations.

Note that the Signal command contains a **Sequence**. This field has operation state type Mutable, as does **Value List**. Receiving nodes can use this value to preserve order, prevent duplicates, or determine if there have been intervening (lost) Signal commands. Since a goal of Signal is to not loose data this would be an error. However, Signal does not make provision to guarantee its delivery. Requestors should use a lossless transport if they desire guaranteed delivery. For a given event, the provider increases the **Sequence** for each Signal sent. The value is a floating window, so wrapping the value is not a problem.

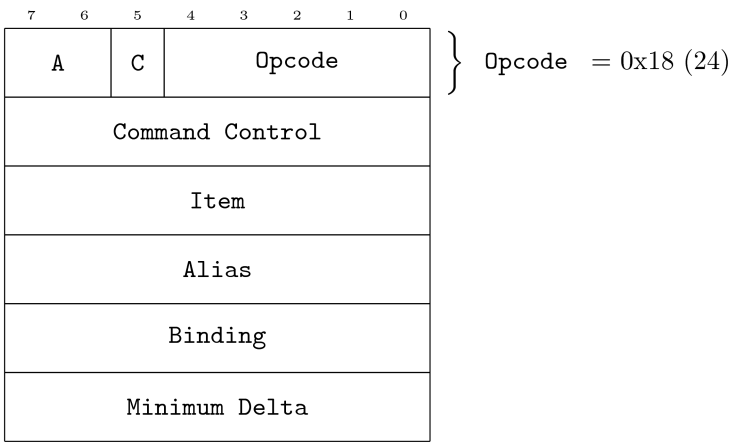
4.16. Subscribe

Subscribes to property, requesting notification of value changes.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Binding(Read)
Permission (response): OAP Binding(Provide)

See Change and Exception.

⇄ Subscribe (command) `dof-2008-2-pdu-40`

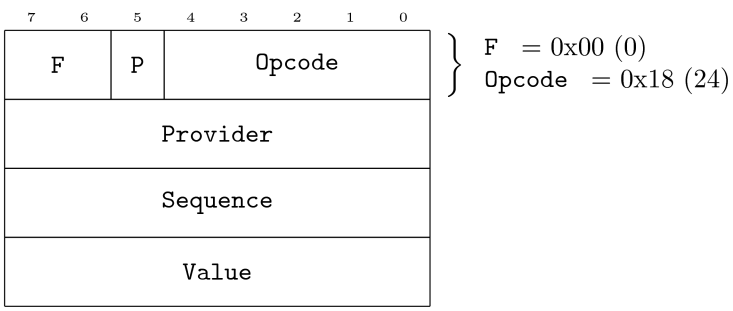
COMMAND/DIRECTED or FLOODED
OPID
Instance of OAP Item-Oriented Command`dof-2008-2-pdu-2`



A	**Two bits.** Controls the format of Binding.
C	**One bit.** Controls the presence of Command Control.
Opcode	**Five bits.** Determines the type of the command.
Command Control	**Instance of** Command Control$_{\texttt{dof-2008-2-pdu-7}}$.**Optional, controlled by** C. If C is one then this field is present. This field determines options that apply to the command.
Item	**Instance of** Item Identifier$_{\texttt{dof-2008-2-pdu-4}}$.Defines the item for the command.
Alias	**Variable length. Optional, controlled by** A. If A is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the Binding when it is not present.
Binding	**Instance of** Binding$_{\texttt{dof-2008-2-pdu-5}}$.**Optional, controlled by** A. If A is zero then this field is present. Defines the binding for the command.
Minimum Delta	**Instance of** Compressed Unsigned 16-Bit$_{\texttt{dof-2009-1-pdu-1}}$.A value used to control update rate.

⇄ Subscribe (response) `dof-2008-2-pdu-41`

RESPONSE
Instance of OAP Response$_{\texttt{dof-2008-2-pdu-3}}$



F	**Two bits.** Flags based on the Opcode.
P	**One bit.** Controls the presence of Provider.
Opcode	**Five bits.** Determines the type of the response.
Provider	**Instance of** Object Identifier (OID)$_{\texttt{dof-2009-1-pdu-4}}$.**Optional, controlled by** P. If P is one then this field is present. Identifies the provider of the response. This field is required in responses to OAP multicast and broadcast commands. Responses to OAP unicast commands should not use this field.
Sequence	**Two bytes, MSB.** Contains the provider's current signal sequence for the item, expressed in a floating window.
Value	**Instance of** Value List$_{\texttt{dof-2008-2-pdu-8}}$.The current property value.

4.16.1. Small Implementation Hints

Even minimal providers should only send Change commands if there are active Subscribe commands. However, they may take short cuts regarding how accurately they determine active Subscribe commands. Providers with no events may trigger a common exception.

Requestor-only nodes may silently drop this command.

4.16.2. Flows

Subscribe() $\rightarrow$ (response), Changed() … : Success.

Subscribe() $\rightarrow$ Exception() : Immediate failure.

Subscribe() $\rightarrow$ (response), Changed () …, Exception() : Subscribe cancelled by the provider or proxy.

4.16.3. Details

The Subscribe command creates a relationship between the requestor and providers based on the specified property. The Subscribe response indicates success and provides the current value. Errors may result in an Exception response.

See the section on the Relationship and Action classes of operations for common requirements. The Relationship requirements govern the Subscribe response and routing rules.

The relationship requests that providers send Change commands to the requestor.

Providers or proxies respond to indicate acceptance of the subscription. The response includes the current **Sequence**, which requestors use to preserve the order of Change commands, and to prevent duplicates. The Change command discusses these requirements further. The response also contains the current **Value**.

The use of an alias on this command allows, but does not require, the use of an alias on the reactionary Change commands. The use of a full binding on this command requires the use of a full binding on the corresponding Change commands. However, a single active Subscribe that uses an alias from a node allows a Change to use an alias to that node.

Readable properties should support subscriptions, but are not required to. A property that does not support subscriptions must immediately return an Exception(Subscribe Rejected) response to each Subscription command. It is not possible to determine whether a provider supports subscriptions before attempting a subscription. Write-only properties must not support subscriptions.

The **Minimum Delta** can help to control the rate at which providers send change notifications. This parameter is a suggestion, but providers and proxies should follow it. The **Minimum Delta** is expressed in milliseconds from 0 to 32767, and defines the minimum time that should elapse between any two Change commands sent because of this subscription. The **Minimum Delta** is Mutable operation state.

A Subscribe response may originate at a proxy that knows the current (or recent) value. In this case, the Subscribe response may reflect the current value on an intermediate node and not the provider. In order to respond, the proxy must have a cached value from a subscription with a **Minimum Delta** that is less than or greater than the requested value.

Providers, in addition to the sequence of the Action class of operation:

- Send Exception(Subscribe Rejected) if they cannot send a Subscribe response.

4.16.4. Consolidation Rules

Proxies should consolidate OAP unicast Subscribe commands. Nodes may consolidate flooded commands according to the general consolidation rules.

However, the **Minimum Delta** of the consolidated (forwarded) operation must represent the minimum of all consolidated values.

4.17. Watch

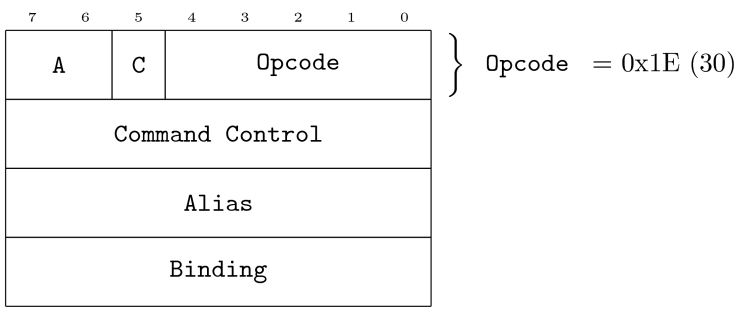
Request that matching provide commands be sent to the sender.
Session: None, Lossy (2-node), Lossy (n-node), Lossless
Addressing: Unicast, Multicast, Unicast
Security: Session.
Permission (command): OAP Requestor
Permission (response): N/A

See Activate, Advertise, Connect, Full Connect, Provide and Watch.



Watch (command) `dof-2008-2-pdu-42`

COMMAND/FLOODED
OPID
Instance of OAP Binding-Oriented Command`dof-2008-2-pdu-1`



A	**Two bits.** Controls the format of Binding.
C	**One bit.** Controls the presence of Command Control.
Opcode	**Five bits.** Determines the type of the command.
Command Control	**Instance of** Command Control$_{\texttt{dof-2008-2-pdu-7}}$.**Optional, controlled by** C. If C is one then this field is present. This field determines options that apply to the command.
Alias	**Variable length. Optional, controlled by** A. If A is non-zero then this field is present and has length $2^{(A-1)}$. Defines the alias for the Binding when it is not present.
Binding	**Instance of** Binding$_{\texttt{dof-2008-2-pdu-5}}$.**Optional, controlled by** A. If A is zero then this field is present. Defines the binding for the command.

4.17.1. Small Implementation Hints

All nodes must typically react to this command. The only exception is a provider that always provides and maintains its bindings over all sessions and transports *and* does not support lossless sessions. Such a node would ignore this command. However, such minimal implementations are highly discouraged because of the load that they put on the network.

Requestor-only nodes may silently drop this command.

4.17.2. Flows

Watch() $\rightarrow$ (none) : Success.

4.17.3. Details

The Watch command expresses interest in base bindings provided at the time. It does not instruct the provider or proxy to take any action beyond forwarding Provide commands of which it is already aware.

See the section on the Interest class of operations for common interest requirements.

5. OAP Sequence Diagrams

There are three major categories of OAP commands: Interest, Object Access, and Subscribe/Register. Within each category, the possible sequences of PDUs are consistent. Because of this, the example sequence diagrams follow the same categorization.

OAP is extremely flexible and it is not possible to show all possible combinations of sequences. Those chosen show major differences in behavior or capability.

5.1. Interest

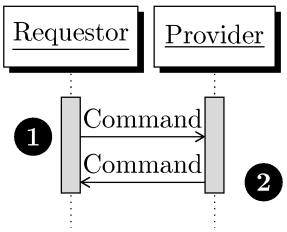
The common feature of interest is that it establishes state throughout the system. This state is used to cause reactions in other nodes, causing related (but different) commands to be created. The creation of state also allows reactions to occur based on topology changes, rather than based on retries.

Interest is flooded. The diagrams do not show this aspect, which show only the reactions between the requestor, provider, and possibly a proxy.

Interest commands do not cause responses. The initial command may be an Advertise, in which case the reaction is an Interest command, or it may be an Interest command, in which case the reaction is a Provide command.

5.1.1. Simple Command and Reaction

Figure 5.1. Sequence for Simple Command and Reaction

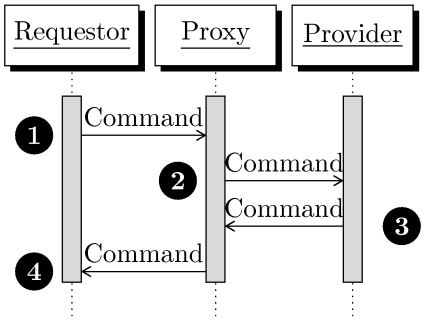


1. Requestor sends the command.

2. Command is received and Provider reacts with a different command.

5.1.2. Proxied Command, Delayed Reaction

Figure 5.2. Sequence for Proxied Command and Reaction



This is an extension to the simple command and reaction sequence showing state on the proxy. . Requestor sends the command. . Proxy forwards command. . Command is received and Provider reacts with a different command. . Second command is forwarded to the original Requestor.

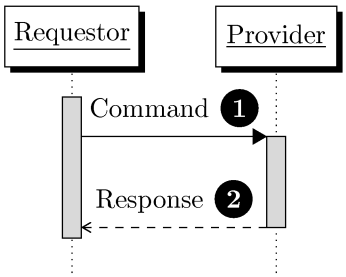
5.2. Object Access

The object access commands cause responses. Requestors may use either directed or flooded operations, based on the state of the system.

Object access commands also use command control, and several examples of its use are included.

5.2.1. Simple Command Response

Figure 5.3. Sequence for Simple Command Response

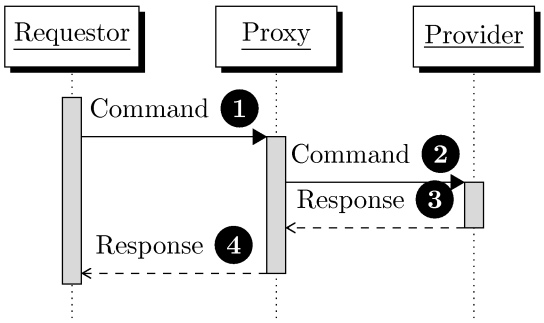


1. Command is sent from Requestor to Provider.

2. Provider immediately responds to the Requestor.

5.2.2. Proxied Command with Immediate Response

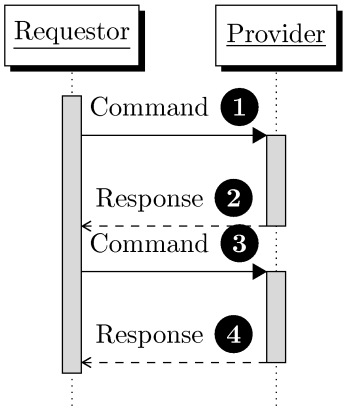
Figure 5.4. Sequence for Proxied Command Response



1. Command is sent from Requestor to Proxy.
2. Proxy forwards to Provider.
3. Provider responds to Proxy.
4. Proxy forwards the response to the Requestor.

5.2.3. Command Response with Retry and Response

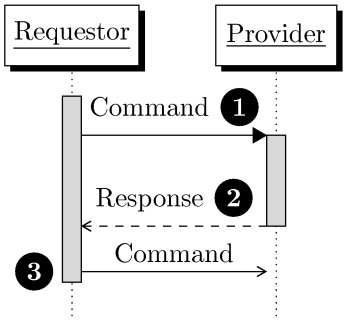
Figure 5.5. Sequence for Command Response with Retry Response



1. Requestor sends the command.
2. Command is received and Provider reacts with a different command.
3. Request retries the command with no command control.
4. Command is received and Provider reacts the same as before.

5.2.4. Command Response with Retry including Acknowledge

Figure 5.6. Sequence for Command Response with Acknowledge



1. Requestor sends the command.

2. Command is received and Provider responds.

3. Request retries the command with acknowledgement of response, no response generated.

5.3. Subscribe/Register

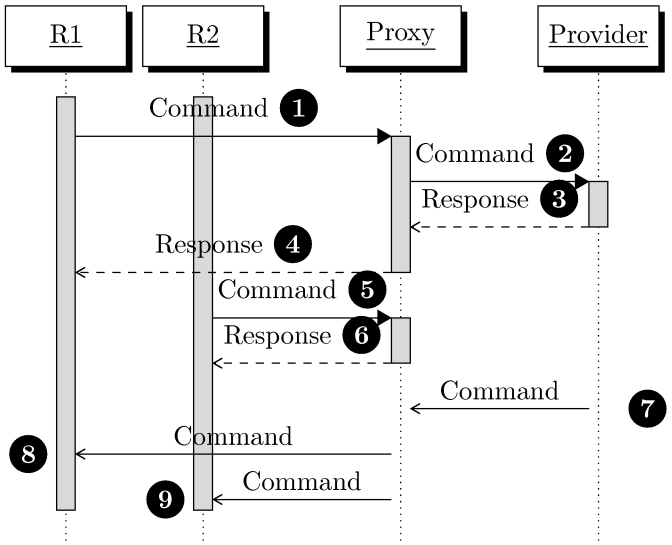
The subscribe/register commands combine a form of interest and object access: they create state throughout the system, they cause providers to send related commands to the requestor, and they generate responses.

A common feature of this category of commands is that proxies can merge, or consolidate them.

This combination means that many of the sequence diagrams already shown apply (with minor modifications) to the subscribe register commands. In general, adding a response to the interest example commands makes them subscribe/register commands.

5.3.1. Merged Proxied Command, Delayed Reaction

Figure 5.7. Sequence for Merged Proxied Command Response with Reaction



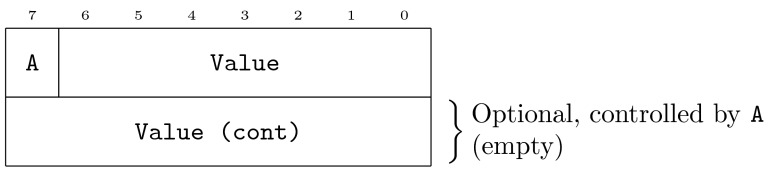
1. Command is sent from R1 to Proxy.
2. Proxy forwards command to Provider.
3. Provider generates response to Proxy.
4. Proxy forwards response to R1.
5. Command is sent from R2 to Proxy, Proxy merges the command.
6. Proxy generates response to R2.
7. Provider sends command to Proxy.
8. Proxy forwards command to R1.
9. Proxy forwards command to R2.

This sequence shows a merged operation generating both responses and additional commands.

6. External PDU Reference

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

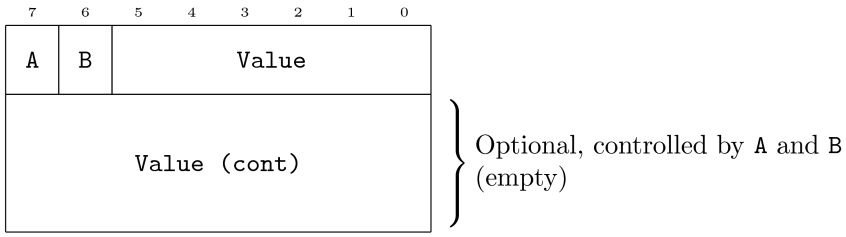
Compressed Unsigned 16-Bit `dof-2009-1-pdu-1`



`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`

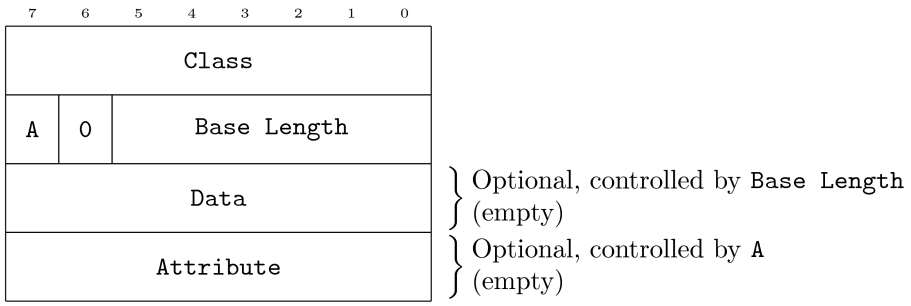


`A` **One bit.** Controls the size of `Value` and the presence of the `B` field.

`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.

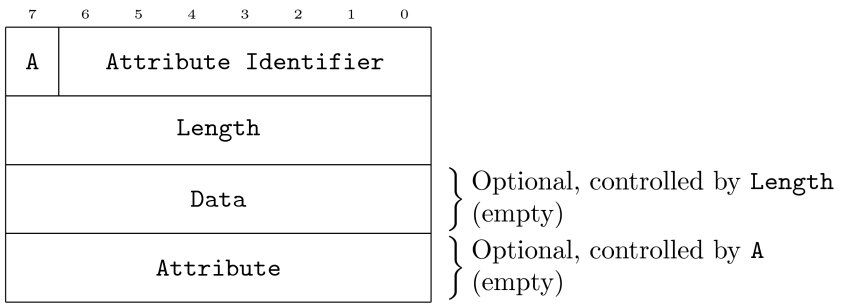
`A` **One bit.** Controls the presence of `Attribute`.

`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`.

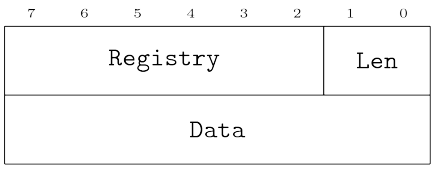
`Attribute Identifier` **Seven bits.** The attribute type, registered with ODP-TSC.

`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.

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

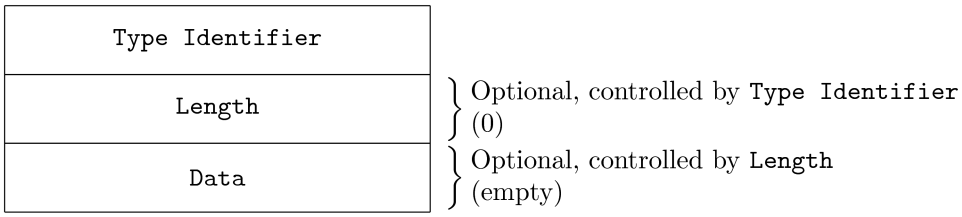


`Registry` **Six bits.** The registry of the IID, registered with the ODP-TSC.

`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`

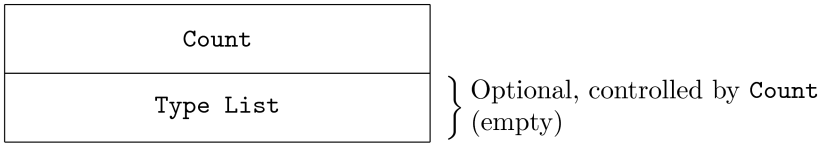


`Type Identifier` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Controls the presence of `Length`. If this value is odd then the `Length` field is present. This identifies each basic type, registered with ODP-TSC.

`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 Context `dof-2009-1-pdu-39`

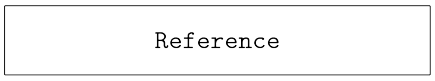


`Count` **Instance of** Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.Controls the length of `Type List`.

`Type List` **Array of** Type Identifier`dof-2009-1-pdu-23`.**Optional, controlled by** `Count`. A list of type definitions. The list starts with index zero, and each type in the list has an index one greater than the preceding type.

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

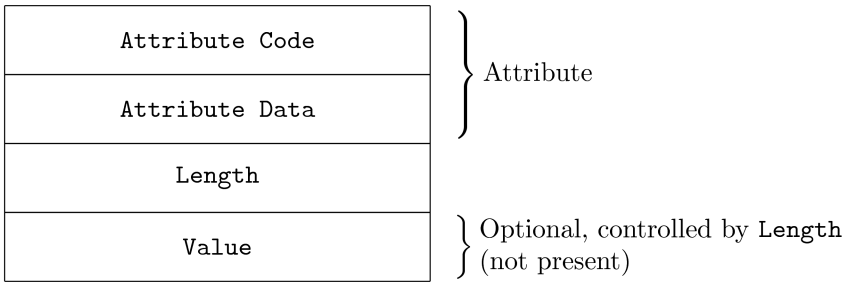
Instance of Compressed Unsigned 16-Bit`dof-2009-1-pdu-1`.



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

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

Attribute/Value Pair `dof-2008-1-pdu-16`



`Attribute Code` **One byte.** This field identifies the attribute. It must be registered with the ODP-TSC.

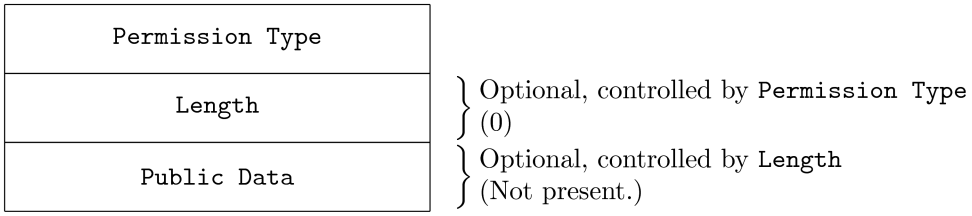
`Attribute Data` **Two bytes, MSB.** This field completes the definition of an attribute in combination with the code.

`Length` **One byte.** This field determines the length of the `Value`. It may be zero, in which case the `Value` field is not present.

`Value` **Variable length. Optional, controlled by** `Length`. Contains the value for the associated attribute.

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

⇄ **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**.