Open Command and Control (OpenC2) Profile for Stateless Packet Filtering Version 1.0

Working Draft 04

16 October 2018

Specification URIs

This version:

- oasis-to-fill-in-link.pdf (Authoritative)
- oasis-to-fill-in-link.md
- oasis-to-fill-in-link.html

Previous Version:

- Not Applicable

Latest Version:

- oasis-to-fill-in-link.md (Authoritative)
- oasis-to-fill-in-link.html
- oasis-to-fill-in-link.pdf

Technical Committee:

- OASIS Open Command and Control (OpenC2) TC

Chairs

- Joe Brule (jmbrule@nsa.gov), National Security Agency
- Sounil Yu (sounil.yu@bankofamerica.com), Bank of America

Editors

- Joe Brule (jmbrule@nsa.gov), National Security Agency
- Duncan Sparrell (duncan@sfractal.com), sFractal Consulting
- Alex Everett (alex.everett@unc.edu), University of North Carolina, Chapel Hill

Abstract

Open Command and Control (OpenC2) is a concise and extensible language to enable the command and control of cyber defense components, subsystems and/or systems in a manner that is agnostic of the underlying products, technologies, transport mechanisms or other aspects of the implementation. Stateless packet filtering is a cyber defense mechanism that denies or allows traffic based on static properties of the traffic (such as address, port, protocol,

etc.). This profile defines the actions, targets, specifiers, and options that are consistent with version 1.0 of the OpenC2 Language Specification in the context of stateless packet filtering.

Status

This document was last revised or approved by the OASIS Open Command and Control (OpenC2) TC on the above date. The level of approval is also listed above. Check the "Latest version" location noted above for possible later revisions of this document. Any other numbered Versions and other technical work produced by the Technical Committee (TC) are listed at https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=openc2#technical.

TC members should send comments on this specification to the TC's email list. Others should send comments to the TC's public comment list, after subscribing to it by following the instructions at the "Send A Comment" button on the TC's web page at https://www.oasis-open.org/committees/openc2/.

This Draft is provided under the Non-Assertion Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/openc2/ipr.php).

Note that any machine-readable content (Computer Language Definitions) declared Normative for this Work Product is provided in separate plain text files. In the event of a discrepancy between any such plain text file and display content in the Work Product's prose narrative document(s), the content in the separate plain text file prevails.

Citation format:

When referencing this specification the following citation format should be used:

[OpenC2-SLPF-v1.0]

Open Command and Control (OpenC2) Profile for Stateless Packet Filtering Version 1.0. Edited by Joe Brule, Duncan Sparrell and Alex Everett. 16 October 2018. OASIS Working Draft 04. oasis-to-fill-in-link.html.

Latest version: N/A.

Notices

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made

Table of Contents

1 Introduction

OpenC2 is a suite of specifications that enables command and control of cyber defense systems and components. OpenC2 typically uses a request-response paradigm where a command is encoded by an OpenC2 producer (managing application) and transferred to an OpenC2 consumer (managed device or virtualized function) using a secure transport protocol, and the consumer can respond with status and any requested information. The contents of both the command and the response are fully described in schemas, allowing both parties to recognize the syntax constraints imposed on the exchange.

OpenC2 allows the application producing the commands to discover the set of capabilities supported by the managed devices. These capabilities permit the managing application to adjust its behavior to take advantage of the features exposed by the managed device. The capability definitions can be easily extended in a noncentralized manner, allowing standard and non-standard capabilities to be defined with semantic and syntactic rigor.

1.1 IPR Policy

This Working Draft is being developed under the Non-Assertion Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC's web page (https://www.oasis-open.org/committees/openc2/ipr.php).

1.2 Terminology

- **Action**: The task or activity to be performed.
- **Actuator**: The entity that performs the action.
- **Command**: A message defined by an action-target pair that is sent from a producer and received by a consumer.
- **Consumer**: A managed device / application that receives Commands. Note that a single device / application can have both consumer and producer capabilities.
- **Producer**: A manager application that sends Commands.
- **Response**: A message from a consumer to a producer acknowledging a command or returning the requested resources or status to a previously received request.
- **Target**: The object of the action, i.e., the action is performed on the target.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] and [RFC8174].

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.3 Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, http://www.rfc-editor.org/info/rfc2119.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, http://www.rfc-editor.org/info/rfc8174.

[RFC8259] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", December 2017, https://tools.ietf.org/html/rfc8259.

[RFC1123] Author, T., "Requirements for Internet Hosts", October 1989. https://tools.ietf.org/html/rfc1123.

[RFC4291] Hinden, R., Deering S. , T., "IP Version 6 Addressing Architecture ", February 2006, https://tools.ietf.org/html/rfc4291.

[RFC2673] Crawford, M., "Binary Labels in Domain Name System", August 1999, https://tools.ietf.org/html/rfc2673.

[RFC3339] Kline, G., "Date and Time on the Internet: Timestamps", July 2002, https://tools.ietf.org/html/rfc3339.

[RFC5237] Arkko, J., Erricsson, S. , "IANA Allocation Guidelines for the Protocol Field", February 2008, https://tools.ietf.org/html/rfc5237.

[OpenC2-Lang-v1.0] *Open Command and Control (OpenC2) Language Specification Version 1.0*.

Edited by Jason Romano and Duncan Sparrell.

xx August 2018. OASIS Working Draft 08. oasis-to-fill-in-link.html.

Latest version: http://docs.oasis-open.org/openc2/oc2ls/v1.0/oc2ls-v1.0.html.

1.4 Non normative References

[OpenC2-HTTPS-v1.0] *Specification for Transfer of OpenC2 Messages via HTTPS Version 1.0*.

Edited by David Lemire.

16 October 2018. OASIS Committee Specification Draft 03.

http://docs.oasis-open.org/openc2/open-impl-https/v1.0/csd03/open-impl-https-v1.0-csd03.html.

Latest version: http://docs.oasis-open.org/openc2/open-impl-https/v1.0/open-impl-https-v1.0.html.

1.5 Document Conventions

1.5.1 Naming Conventions

- RFC2119/RFC8174 key words (see section 1.4) are in all uppercase.
- All property names and literals are in lowercase, except when referencing canonical names defined in another standard (e.g., literal values from an IANA registry).
- All words in structure component names are capitalized and are separated with a hyphen, e.g., ACTION, TARGET, TARGET-SPECIFIER.
- Words in property names are separated with an underscore (_), while words in string enumerations and type names are separated with a hyphen (-).
- The term "hyphen" used here refers to the ASCII hyphen or minus character, which in Unicode is "hyphen-minus", U+002D.
- All type names, property names, object names, and vocabulary terms are between three and 40 characters long.

1.5.2 Font Colors and Style

The following color, font and font style conventions are used in this document:

- A fixed width font is used for all type names, property names, and literals.
- Property names are in bold style – **`created_at`**
- All examples in this document are expressed in JSON. They are in fixed width font, with straight quotes, black text and a light shaded background, and 4-space indentation. JSON examples in this document are representations of JSON Objects. They should not be interpreted as string literals. The ordering of object keys is insignificant. Whitespace before or after JSON structural characters in the examples are insignificant [RFC8259].
- Parts of the example may be omitted for conciseness and clarity. These omitted parts are denoted with the ellipses (...).

Example:

~~~
```javascript
{
    "action": "contain",
    "target": {
        "user_account": {
            "user_id": "fjbloggs",
            "account_type": "windows-local"
        }
    }
}
```
~~~

1.6 Overview

OpenC2 is a suite of specifications to command actuators that execute cyber defense functions. These specifications include the OpenC2 Language Specification, Actuator Profiles, and Transfer Specifications. The OpenC2 Language Specification and Actuator Profile(s) specifications focus on the standard at the producer and consumer of the command and response while the transfer specifications focus on the protocols for their exchange.

- The OpenC2 Language Specification provides the semantics for the essential elements of the language, the structure for commands and responses, and the schema that defines the proper syntax for the language elements that represents the command or response.

- OpenC2 Actuator Profiles specify the subset of the OpenC2 language relevant in the context of specific actuator functions. Cyber defense components, devices, systems and/or instances may (in fact are likely) to implement multiple actuator profiles. Actuator profiles extend the language by defining specifiers that identify the actuator to the required level of precision and may define command arguments that are relevant and/or unique to those actuator functions.

- OpenC2 Transfer Specifications utilize existing protocols and standards to implement OpenC2 in specific environments. These standards are used for communications and security functions beyond the scope of the language, such as message transfer encoding, authentication, and end-to-end transport of OpenC2 messages.

The OpenC2 Language Specification defines a language used to compose messages for command and control of cyber defense systems and components. A message consists of a header and a payload (*defined* as a message body in the OpenC2 Language Specification Version 1.0 and *specified* in one or more actuator profiles).

In general, there are two types of participants involved in the exchange of OpenC2 messages, as depicted in Figure 1-1:

1. **OpenC2 Producers**: An OpenC2 Producer is an entity that creates commands to provide instruction to one or more systems to act in accordance with the content of the command. An OpenC2 Producer may receive and process responses in conjunction with a command.
2. **OpenC2 Consumers**: An OpenC2 Consumer is an entity that receives and may act upon an OpenC2 command. An OpenC2 Consumer may create responses that provide any information captured or necessary to send back to the OpenC2 Producer.

The language defines two payload structures:

1. **Command**: An instruction from one system known as the OpenC2 "Producer", to one or more systems, the OpenC2 "Consumer(s)", to act on the content of the command.
2. **Response**: Any information captured or necessary to send back to the OpenC2 Producer that issued the Command, i.e., the OpenC2 Consumer's response to the OpenC2 Producer.

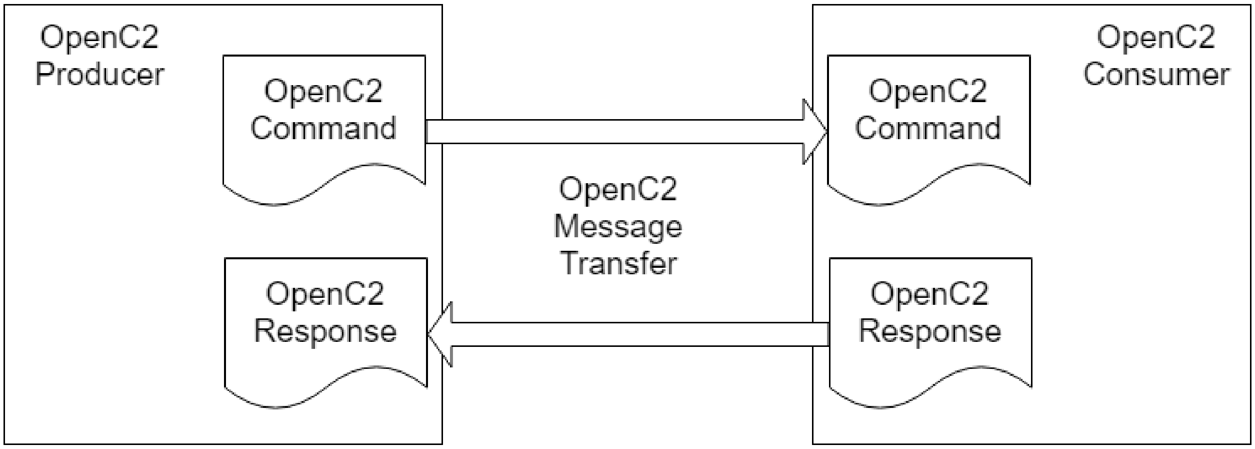

Figure 1-1. OpenC2 Message Exchange

OpenC2 implementations integrate the related OpenC2 specifications described above with related industry specifications, protocols, and standards. Figure 1 depicts the relationships among OpenC2 specifications, and their relationships to other industry standards and environment-specific implementations of OpenC2. Note that the layering of implementation aspects in the diagram is notional, and not intended to preclude, e.g., the use of an application-layer message signature function to provide message source authentication and integrity.

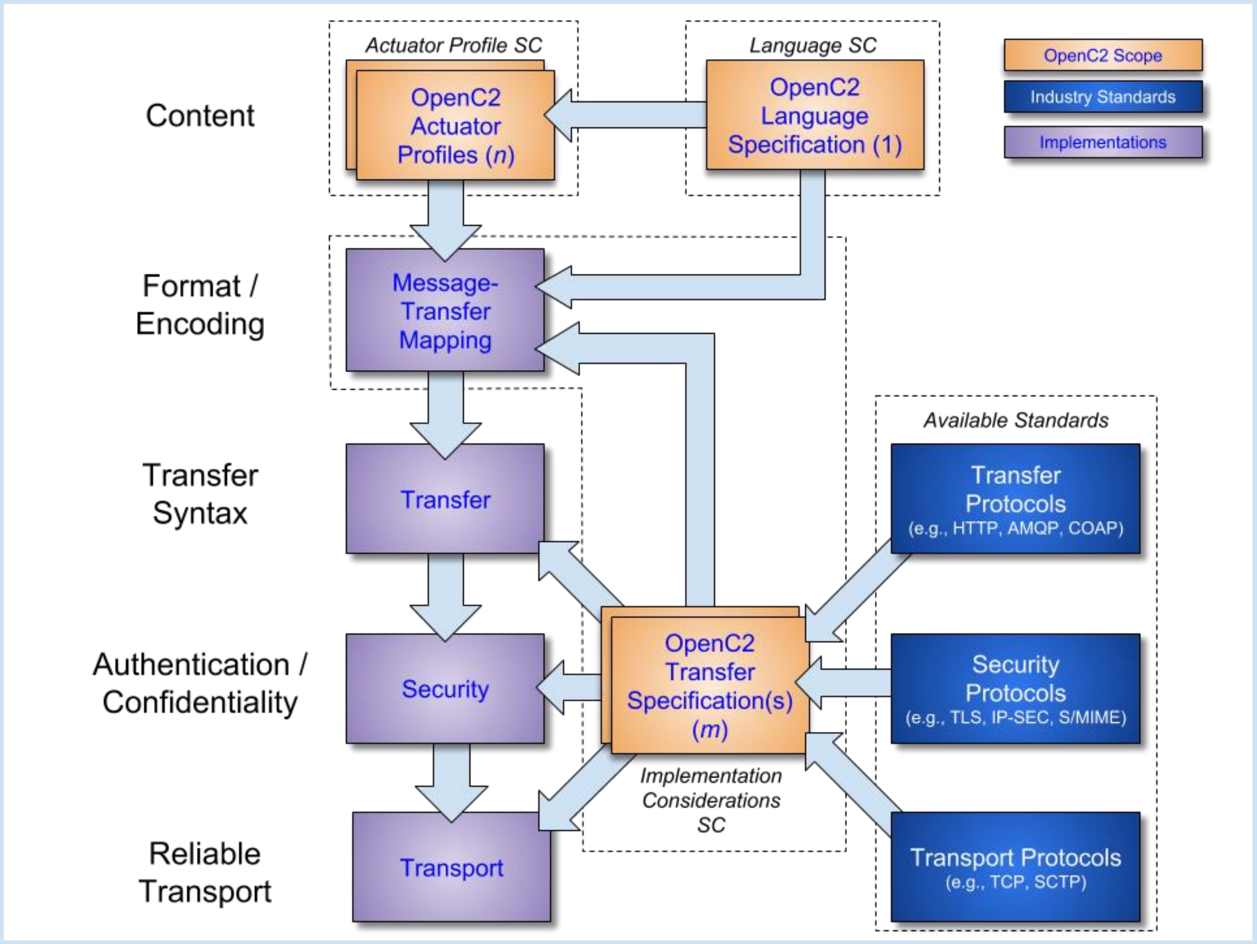
Content
Format / Encoding
Transfer Syntax
Authentication / Confidentiality
Reliable Transport
Actuator Profile SC
OpenC2 Actuator Profiles (n)
Language SC
OpenC2 Language Specification (1)
OpenC2 Scope
Industry Standards
Implementations
Message-Transfer Mapping
Transfer
Security
Transport
OpenC2 Transfer Specification(s) (m)
Implementation Considerations SC
Available Standards
Transfer Protocols (e.g., HTTP, AMQP, COAP)
Security Protocols (e.g., TLS, IP-SEC, S/MIME)
Transport Protocols (e.g., TCP, SCTP)

Figure 1-2. OpenC2 Documentation and Layering Model

OpenC2 is conceptually partitioned into four layers as shown in Table 1-1.

Table 1-1. OpenC2 Protocol Layers

Layer	Examples
Function-Specific Content	Actuator Profiles (standard and extensions)
Common Content	Language Specification (this document)
Message	Transfer Specifications (OpenC2-over-HTTPS, OpenC2-over-CoAP, …)
Secure Transport	HTTPS, CoAP, MQTT, OpenDXL, …

- The **Secure Transport** layer provides a communication path between the producer and the consumer. OpenC2 can be layered over any standard transport protocol.
- The **Message** layer provides a transport- and content-independent mechanism for conveying requests, responses, and notifications. A transfer specification maps transport-specific protocol elements to a transport-independent set of message elements consisting of content and associated metadata.
- The **Common Content** layer defines the structure of OpenC2 commands and responses and a set of common language elements used to construct them.
- The **Function-specific Content** layer defines the language elements used to support a particular cyber defense function. An actuator profile defines the implementation conformance requirements for that function. OpenC2 Producers and Consumers will support one or more profiles.

The components of an OpenC2 Command are an action (what is to be done), a target (what is being acted upon), an optional actuator (what is performing the command), and command arguments, which influence how the command is to be performed. An action coupled with a target is sufficient to describe a complete OpenC2 Command. Though optional, the inclusion of an actuator and/or command arguments provides additional precision to a command, when needed.

The components of an OpenC2 Response are a numerical status code, an optional status text string, and optional results. The format of the results, if included, depend on the type or response being transferred.

1.7 Goal

The goal of the OpenC2 Language Specification is to provide a language for interoperating between functional elements of cyber defense systems. This language used in conjunction with OpenC2 Actuator Profiles and OpenC2 Transfer Specifications allows for vendor-agnostic cybertime response to attacks.

The Integrated Adaptive Cyber Defense (IACD) framework defines a collection of activities, based on the traditional OODA (Observe–Orient–Decide–Act) Loop [IACD]:

- Sensing: gathering of data regarding system activities
- Sense Making: evaluating data using analytics to understand what's happening
- Decision Making: determining a course-of-action to respond to system events
- Acting: Executing the course-of-action

The goal of OpenC2 is to enable coordinated defense in cyber-relevant time between decoupled blocks that perform cyber defense functions. OpenC2 focuses on the Acting portion of the IACD framework; the assumption that underlies the design of OpenC2 is that the sensing/

analytics have been provisioned and the decision to act has been made. This goal and these assumptions guides the design of OpenC2:

- **Technology Agnostic:** The OpenC2 language defines a set of abstract atomic cyber defense actions in a platform and product agnostic manner
- **Concise:** An OpenC2 command is intended to convey only the essential information required to describe the action required and can be represented in a very compact form for communications-constrained environments
- **Abstract:** OpenC2 commands and responses are defined abstractly and can be encoded and transferred via multiple schemes as dictated by the needs of different implementation environments
- **Extensible:** While OpenC2 defines a core set of actions and targets for cyber defense, the language is expected to evolve with cyber defense technologies, and permits extensions to accommodate new cyber defense technologies.

1.8 Purpose and Scope

A ‘Stateless Packet Filter’ (SLPF) is a policy enforcement mechanism that restricts or permits traffic based on static values such as source address, destination address, and/or port numbers. A Stateless-Packet-Filter does not consider traffic patterns, connection state, data flows, applications, or payload information. The scope of this profile is limited to Stateless-Packet-Filtering herein referred to as SLPF.

This actuator profile specifies the set of actions, targets, specifiers, and command arguments that integrates SLPF functionality with the Open Command and Control (OpenC2) command set. Through this command set, cyber security orchestrators may gain visibility into and provide control over the SLPF functionality in a manner that is independent of the instance of the SLPF function.

All components, devices and systems that provide SLPF functionality will implement the OpenC2 ACTIONS, TARGETS, SPECIFIERS and ARGS identified as required in this document. Actions that are applicable, but not necessarily required, for SLPF will be identified as optional.

The purpose of this document is to:

- Identify the required and optional OpenC2 ACTIONS for actuators with SLPF functionality.
- Identify the required and optional TARGET types and associated specifiers for each action in the SLPF class of actuators.
- Identify ACTUATOR-SPECIFIERS, ACTUATOR-ARGS and COMMAND-ARGS for each action-target pair that are applicable and/or unique to the SLPF class of actuators
- Annotate each Action/ Target pair with a justification and example, and provide sample OpenC2 commands to a SLPF with corresponding responses
- Provide an abstract schema that captures the specifiers and options for a SLPF

This SLPF profile:

- Does not define or implement ACTIONS beyond those defined in Version 1.0 of the Language Specification.
- Is consistent with version 1.0 of the OpenC2 Language Specification

Cyber defense systems that are utilizing OpenC2 may require the following components to implement the SLPF profile:

- OpenC2 Producers: Devices that send commands, receive responses, and manage the execution of commands involving one or more SLPF or other actuators with SLPF capability. The OpenC2 producer needs *a priori* knowledge of which commands the actuator can process and execute, therefore must understand the profiles for any device that it intends to command.
- OpenC2 Consumers: Devices or instances that provide stateless packet filtering functions. Typically these are actuators that execute the cyber defense function, but could be orchestrators (i.e., a device or instance that forwards commands to the actuator).

Though cyber defense components, devices, systems and/or instances may may implement multiple actuator profiles, a particular OpenC2 message may reference at most a single actuator profile. The scope of this document is limited to SLPF.

This specification is organized into three major sections.

Section One (this section) provides a nonnormative overview of the suite of specifications that realize OpenC2. This section provides references as well as defines the scope and purpose of this specification.

Section Two (normative) binds this particular profile to the OpenC2 Language Specification. Section Two enumerates the components of the language specification that are meaningful in the context of SLPF and defines components that are applicable to this distinct profile. Section Two also defines the commands (i.e., the action target pairs) that are permitted in the context of SLPF.

Section Three (normative) presents definitive criteria for conformance so that cyber security stakeholders can be assured that their products, instances and/or integrations are compatible with OpenC2.

This specification provides three non-normative Annexes. OpenC2 is intended for machine to machine interactions, therefore a schema for SLPF and the applicable portions of the OpenC2 Language schema are provided to facilitate development. There is also an Annex that provides multiple examples of SLPF commands (JSON serialization).

2 OpenC2 Language Binding

This section defines the set of ACTIONS, TARGETS, SPECIFIERS, and ARGS that are meaningful in the context of an SLPF. This section also describes the format of the response frame's status and results field. This section organized into three major subsections; Command Components, Response Components and Commands.

2.1 OpenC2 Command Components

The components of an OpenC2 command include ACTIONS, TARGETS, ACTUATORS and associated ARGS and SPECIFIERS. Appropriate aggregation of the components will define a command-body that is meaningful in the context of an SLPF.

This specification identifies the applicable components of an OpenC2 command. The components of an OpenC2 command include:

- ACTION: A subset of the ACTIONs defined in the OpenC2 Language specification that are meaningful in the context of a SLPF.
    - This profile does not define ACTIONs that are external to Version 1.0 of the OpenC2 Language Specification.
    - This profile MAY augment the definition of the actions in the context of a SLPF.
    - This profile SHALL NOT define ACTIONs in a manner that is inconsistent with version 1.0 of the OpenC2 language specification.
- TARGET: A subset of the TARGETs and target-specifiers defined in the Language specification that are meaningful in the context of SLPF and one TARGET (and its associated specifier) that is defined in this specification.
- ARGS: A subset of the COMMAND-ARGS defined in the Language Specification and a set of ACTUATOR-ARGS defined in this specification.
- ACTUATOR: A set of specifiers defined in this specification that are meaningful in the context of SLPF.

2.1.1 Actions

Table 2.1.1-1 presents the OpenC2 actions defined in version 1.0 of the Language Specification which are meaningful in the context of an SLPF. The particular action/target pairs that are required or optional are presented in section 2.3.

Table 2.1.1-1. Actions Applicable to SLPF

Type: Action (Enumerated)

ID	Name	Description
3	**query**	Initiate a request for information. Used to communicate the supported options and determine the state or settings.
6	**deny**	Prevent traffic or access.
8	**allow**	Permit traffic or access.
16	**update**	Instructs the actuator to update its configuration by retrieving and processing a configuration file and update.

20	**delete**	Remove an access rule.

2.1.2 Targets

2.1.2.1 Common Targets

Table 2.1.2-1 lists the TARGETs defined in the OpenC2 Language specification that are applicable to SLPF. The particular action/target pairs that are required or optional are presented in section 2.3.

Table 2.1.2-1. Targets Applicable to SLPF

Type: Target (Choice)

ID	Name	Type	Description
10	**file**	File	Properties of a file.
11	**ip_addr**	IP-Addr	The representation of one or more IP addresses (either version 4 or version 6) expressed using CIDR notation.
15	**ip_connection**	IP-Connection	A network connection that originates from a source and is addressed to a destination. Source and destination addresses may be either IPv4 or IPv6; both should be the same version
16	**features**	Features	A set of items such as action target pairs, profiles versions, options that are supported by the actuator. The target is used with the query action to determine an actuator's capabilities.
1024	**slpf**	slpf:Target	Targets defined in the Stateless Packet Filter profile.

2.1.2.2 SLPF Targets

The slpf:Target type is defined in this specification and is referenced under the slpf namespace. Implementations that choose to include this type MUST import it in accordance with the procedures defined in section 2.2.6 of Version 1.0 of the OpenC2 Language Specification:

1. The unique name of the SLPF schema is `oasis-open.org/openc2/v1.0/ap-slpf`
2. The namespace identifier (nsid) referring to the SLPF schema is: `slpf`
3. The list of types imported from the SLPF schema is: `Target`, `Actuator`, `Args`, and `Results`.

4. The definitions of and conformance requirements for these types are contained in this document.

Type: Target (Choice)

ID	Name	Type	Description
1	**rule_number**	Rule-ID	Immutable identifier assigned when a rule is created, Identifies a rule to be deleted.

Implementations that choose to support slpf:Target MUST support the **rule_number** target.

2.1.3 Command Arguments

Arguments provide additional precision to a command by including information such as how, when, or where a command is to be executed. Table 2.1.3-1 summarizes the command arguments defined in Version 1.0 of the OpenC2 Language Specification as they relate to SLPF functionality. Table 2.1.3-2 summarizes the command arguments that are defined in this specification.

2.1.3.1 Common Args

Table 2.1.3.1-1 lists the command arguments defined in the OpenC2 Language specification that are applicable to SLPF.

Table 2.1.3-1. Command Arguments applicable to SLPF

Type: Args (Map)

ID	Name	Type	#	Description
1	**start_time**	Date-Time	0..1	The specific date/time to initiate the action
2	**stop_time**	Date-Time	0..1	The specific date/time to terminate the action
3	**duration**	Duration	0..1	The length of time for an action to be in effect
4	**response_requested**	Response-Type	0..1	The type of response required for the action: `none`, `ack`, `status`, `complete`.
1024	**slpf**	slpf:Args	0..1	Command arguments defined in the Stateless Packet Filter profile

The semantics/requirements as they relate to common arguments:

- start-time/end-time/duration
    - If none are specified then the start time is now, the end time is never, and the duration is infinity

  - Only two of the three are allowed on any given command and the third is derived from the equation end-time = start-time + duration
  - If only start time is specified then end-time is never and duration is infinity
  - If only end time is specified then start-time is now and duration is derived
  - If only duration is specified then start-time is now and end-time is derived
- response_requested
  - If absent or not explicitly set in an OpenC2 Command, then a Consumer MUST respond the same as response_type `complete`.

2.1.3.2 SLPF Args

The command arguments defined in this document are referenced under the slpf namespace.

Table 2.1.3-2. Command Arguments Unique to SLPF

Type: Args (Map)

ID	Name	Type	#	Description
1	**drop_process**	Drop-Process	0..1	Specifies how to handle denied packets
2	**running**	Boolean	0..1	Normal operations assumes any change to a device are to be implemented as persistent changes. Setting the running modifier to TRUE results in a change that is not persistent in the event of a reboot or restart.
3	**direction**	Direction	0..1	Specifies whether to apply rules to incoming or outgoing traffic. If omitted, rules are applied to both.
4	**insert_rule**	Rule-ID	0..1	Specifies the identifier of the rule within a list, typically used in a top-down rule list.

Type: Drop-Process (Enumerated)

ID	Name	Description
1	**none**	Drop the packet and do not send a notification to the source of the packet.
2	**reject**	Drop the packet and send an ICMP host unreachable (or equivalent) to the source of the packet.
3	**false_ack**	Drop the traffic and send a false acknowledgement.

Type: Direction (Enumerated)

ID	Name	Description
1	**ingress**	Apply rules to incoming traffic only
2	**egress**	Apply rules to outgoing traffic only

Type: Rule-ID

Type Name	Type	Description
Rule-ID	Integer	Access rule identifier

The semantics/ requirements as they relate to SLPF arguments:

- insert_rule:
    - The value MUST be immutable - i.e. the identifier assigned to an access rule at creation must not change over the lifetime of that rule.
    - The value MUST be unique within the scope of a command sent to an openc2 consumer - i.e. a rule_number maps to exactly one deny <target> or allow <target>
- directionality:
    - Entities that do not support directionality MUST NOT reply with 200 OK and SHOULD return a 501 error code.
    - If absent, then the command MUST apply to both.
- drop_process: If absent or not explicitly set, then the actuator MUST NOT send any notification to the source of the packet
- running: If absent or not explicitly set, then the value is FALSE and any changes are persistent.

2.1.4 Actuator Specifiers

An ACTUATOR is the entity that provides the functionality and performs the action. The ACTUATOR executes the ACTION on the TARGET. In the context of this profile, the actuator is the SLPF and the presence of one or more specifiers further refine which actuator(s) shall execute the action.

Table 2.1.4-1 lists the specifiers that are applicable to the SPLF actuator. Annex C provides sample commands with the use of specifiers.

The actuator specifiers defined in this document are referenced under the slpf namespace.

Table 2.1.4-1. SLPF Specifiers

Type: Specifiers (Map)

ID	Name	Type	#	Description

1	**hostname**	String	0..1	RFC 1123 hostname (can be a domain name or IP address) for a particular device with SLPF functionality
2	**named_group**	String	0..1	User defined collection of devices with SLPF functionality
3	**asset_id**	String	0..1	Unique identifier for a particular SLPF
4	**asset_tuple**	String	0..10	Unique tuple identifier for a particular SLPF consisting of a list of up to 10 strings

2.2 OpenC2 Response Components

Response messages originate from the ACTUATOR as a result of a command.

Responses associated with required actions MUST be implemented. Implementations that include optional ACTIONS MUST implement the RESPONSE associated with the implemented ACTION. Additional details regarding the command and associated response are captured in section 2.3. Examples will be provided in Annex C.

2.2.1 Common Results

Table 2.2.1-1 lists the results defined in the OpenC2 Language specification that are applicable to SLPF.

Table 2.2.1-1. Results Applicable to SLPF

Type: OpenC2-Response (Map)

ID	Name	Type	#	Description
1	**status**	Status-Code	0..1	An integer status code
2	**status_text**	String	0..1	A free-form human-readable description of the response status
6	**versions**	Version	0..n	List of OpenC2 language versions supported by this actuator
7	**profiles**	jadn:Uname	0..n	List of profiles supported by this actuator
8	**schema**	jadn:Schema	0..1	Syntax of the OpenC2 language elements supported by this actuator
9	**pairs**	Action-Targets	0..n	List of targets applicable to each supported action
10	**rate_limit**	Number	0..1	Maximum number of requests per minute supported by design or policy
1024	**slpf**	slpf:Results	0..1	Response data defined in the Stateless Packet Filtering profile

Table 2.2.1-2 lists the Status Codes defined in the OpenC2 Language specification that are applicable to SLPF.

Table 2.2.1-2. Status Codes

Type: Status-Code (Enumerated.ID)

Value	Description
102	Processing. Command received but action not necessarily complete
200	OK.
400	Bad Request. Unable to process command, parsing error
500	Internal Error. For response type complete, one of the following MAY apply: ● Cannot access file or path ● Rule number currently in use ● Rule not updated
501	Not implemented. For response type complete, one of the following MAY apply: ● Target not supported ● Option not supported ● Command not supported

2.2.2 SLPF Results

The results defined in this document are presented in Table 2.2-2. The results are referenced under the slpf namespace within the OpenC2-Response type defined in the OpenC2 language specification.

Table 2.2-2. SLPF Results

Type: Results (Map)

Type Name	Type	Description
rule_number	Rule-ID	Rule identifier returned from allow or deny command.

2.3 OpenC2 Commands

An OpenC2 command consists of an ACTION/TARGET pair and associated SPECIFIERS and ARGUMENTs. This section enumerates the allowed commands, identify which are required or optional to implement, and present the associated responses.

Table 2.3-1 defines the commands allowed by the SLPF profile and indicates if implementation of the command is required or optional for Openc2 Producers and/or Openc2 Consumers. An ACTION (the top row in Table 2.3-1) paired with a TARGET (the first column in Table 2.3-1) defines an allowable command. The subsequent subsections provide the property tables applicable to each OpenC2 command.

Table 2.3-1. Command Matrix

	Allow	Deny	Query	Delete	Update
ip_connection	required	required			
ip_addr	required	required			
features			required		
slpf:rule_number				optional	
file					optional

Table 2.3-2 defines the command arguments that are allowed for a particular command by the SLPF profile. A command (the top row in Table 2.3-2) paired with an argument (the first column in Table 2.3-2) defines an allowable combination. The subsection identified at the intersection of the command/ argument provides details applicable to each command as influenced by the argument.

Table 2.3-2. Command Arguments Matrix

	Allow <target>	Deny <target>	Query features	Delete slpf:rule_number	Update file
response	2.3.1	2.3.2	2.3.3.1	2.3.4.1	2.3.5.1
start-time	2.3.1	2.3.2		2.3.4.1	2.3.5.1
end-time	2.3.1	2.3.2			
duration	2.3.1	2.3.2			
running	2.3.1	2.3.2			
direction	2.3.1	2.3.2			

insert_rule	2.3.1	2.3.2			
drop_process		2.3.2			

2.3.1 ‘Allow’

Table 2.3.1-1 summarizes the command options that apply to all of the commands consisting of the ‘allow’ action and a valid target type.

Upon receipt of an unsupported command argument, SLPF consumers

- MUST NOT respond with a OK/200.
- SHOULD respond with the 501 status code.
- SHOULD respond with “Option not supported” in the status text.
- MAY respond with the 500 status code.

Products that send ‘allow target’ commands and support the ‘delete slpf:rule_number’ command:

- MUST support the slpf:rule_number target type as defined in section 2.1.2.2
- SHOULD populate the command options field with “response_requested" : ”complete”
- MAY populate the command arguments field with the “insert_rule” : <integer> option.
- MUST populate the command options field with “response_requested" : “complete” if the insert_rule argument is populated.

Products that receive and successfully parse ‘allow <target>’ commands but cannot implement the ‘allow <target>’ :

- MUST NOT respond with a OK/200.
- SHOULD respond with the 501 status code.
- SHOULD respond with ‘Rule not updated’ in the status text.
- MAY respond with the 500 status code.

Products that receive ‘allow <target>’ commands and support the ‘delete slpf:rule_number’ command:

- MUST support the slpf:rule_number target type as defined in section 2.1.2.2
- Upon successful implementation of the ‘allow <target>’, MUST return the rule_number associated with the rule if the “response_requested" : “complete” option is populated.

Products that receive ‘allow target’ commands and support the ‘insert_rule’ command argument:

- MUST assign the rule number provided if the “insert_rule” : <integer> option is populated.
- If the rule number is currently in use, then

    - MUST NOT respond with a OK/200.
    - SHOULD respond with the 501 status code.
    - SHOULD respond with ‘Rule number currently in use’ in the status text.
    - MAY respond with the 500 status code.

The valid target types, associated specifiers, and options are summarized in sections 2.3.1.1 and 2.3.1.2. Sample commands are presented in Annex C.

2.3.1.1 ‘Allow ip_connection’

The ‘allow ip_connection’ command is required for openc2 producers implementing the SLPF.

If the ‘allow ip_addr’ target is not implemented, then SLPF consumers MUST implement the ‘allow ip-connection’ command. Otherwise it is OPTIONAL.

The command permits traffic that is consistent with the specified ip_connection. A valid ‘allow ip_connection’ command has at least one property of the ip_connection populated and may have any combination of the five properties populated. An unpopulated property within the the ip_connection target MUST be treated as an ‘any’.

Products that receive but do not implement the ‘allow ip_connection’ command:

- MUST NOT respond with a OK/200.
- SHOULD respond with the 501 response code.
- SHOULD respond with ‘Target type not supported’ in the status text.
- MAY respond with the 500 status code.

2.3.1.2 ‘Allow ip_addr’

The ‘allow ip_addr’ command is required for openc2 producers implementing the SLPF.

If the ‘allow ip_connection’ target is not implemented, then SLPF consumers MUST implement the ‘allow ip_addr’ command. Otherwise the ‘allow ip-addr’ command is OPTIONAL.

The command permits traffic as specified by the ip_addr property and may be an IPV4 or IPV6 address. The ip-addr supports CIDR notation. The address specified in the ip_addr MUST be treated as a source OR destination address.

Products that receive but do not implement the ‘allow ip_addr’ command:

- MUST NOT respond with a OK/200.
- SHOULD respond with the 501 response code
- SHOULD respond with ‘Target type not supported’ in the status text.
- MAY respond with the 500 status code.

2.3.2 'Deny'

'Deny' can be treated as mathematical complement to 'allow'. With the exception of the additional 'drop_process' actuator-argument, the targets, specifiers, options and corresponding responses are identical to the two 'allow' commands. Table 2.3-2 summarizes the command arguments that apply to all of the commands consisting of the 'deny' action and valid target type.

Upon receipt of a command with an ARGUMENT that is not supported by the actuator, actuators:

- SHOULD respond with the 501 status code
- SHOULD respond with 'Option not supported' in the status text.
- MAY respond with the 500 status code.

Products that send 'deny target' commands and support the 'delete slpf:rule_number' command:

- MUST support the slpf:rule_number target type as defined in section 2.1.2.1.
- SHOULD populate the command options field with '"response_requested" : ”complete”
- MAY populate the command arguments field with the “insert_rule” : <integer> option.
- MUST populate the command options field with "response_requested" : “complete” if the insert_rule argument is populated.

Products that receive 'deny <target>' commands and support the 'delete slpf:rule_number' command:

- MUST support the slpf:rule_number target type as defined in section 2.1.2.1.
- MUST return the rule number assigned in the slpf object if the “response_requested” : “complete” argument is populated.

Products that receive 'deny target' commands and support the 'insert_rule' command argument:

- MUST assign the rule number provided if the “insert_rule” : <integer> argument is populated.
- If the rule number is currently in use, then
    - MUST NOT respond with a OK/200.
    - SHOULD respond with the 501 status code
    - SHOULD respond with 'Rule number currently in use' in the status text.
    - MAY respond with the 500 status code.

2.3.3 'Query'

The valid target type, associated specifiers, and options are summarized in section 2.3.3.1. Sample commands are presented in Annex C.

2.3.3.1 ‘Query features’

The ‘query openc2’ command MUST be implemented in accordance with Version 1.0 of the OpenC2 language specification.

2.3.4 ‘Delete’

The slpf:rule_number is the only valid target type for the delete action. The associated specifiers, and options are summarized in section 2.3.4.1. Sample commands are presented in Annex C.

2.3.4.1 ‘delete slpf:rule_number’

The ‘delete slpf:rule_number’ command is used to remove a firewall rule rather than issue an allow or deny to counteract the effect of an existing rule. Implementation of the ‘delete slpf:rule_number’ command is OPTIONAL. Products that choose to implement the ‘delete slpf:rule_number’ command MUST implement the slpf:rule_number target type described in section 2.1.2.1.

Products that send the ‘delete slpf:rule_number’ command:

- MAY populate the command arguments field with ‘response_requested" : ”complete”.
- MUST NOT include other command arguments.
- MUST include exactly one rule_number.

Products that receive the ‘delete slpf:rule_number’ command:

- but cannot parse or process the ‘delete slpf:rule_number’ command:
    - MUST NOT respond with a OK/200.
    - SHOULD respond with status code 400.
    - MAY respond with the 500 status code
- but do not support the slpf:rule_number target type
    - MUST NOT respond with a OK/200.
    - SHOULD respond with the 501 status code
    - SHOULD respond with ‘target not supported’ in the status text.
    - MAY respond with the 500 status code
- MUST respond with response code 200 upon successful parsing of the ‘delete slpf:rule_number’ command and subsequent removal of the corresponding rule.
- upon successful parsing but failure to remove the corresponding rule
    - MUST NOT respond with OK/200
    - MUST respond with response code 500
    - SHOULD respond with ‘firewall rule not removed or updated’ in the status text.

Refer to Annex C for sample commands.

2.3.5 Update

The ‘file’ target as defined in Version 1.0 of the Language Specification is the only valid target type for the update action. The associated specifiers, and options are summarized in section 2.3.5.1. Sample commands are presented in Annex C.

2.3.5.1 Update file

The ‘update file’ command is used to replace or update files such as configuration files, rule sets, etc. Implementation of the update file command is OPTIONAL. OpenC2 consumers that choose to implement the ‘update file’ command MUST must include all steps that are required for the update file procedure such as retrieving the file(s), install the file(s), restart/ reboot the device etc. The end state shall be that the firewall operates with the new file at the conclusion of the ‘update file’ command. The atomic steps that take place are implementation specific.

Table 2.3-2 presents the valid options for the ‘update file’ command. Products that choose to implement the ‘update file’ command MUST NOT include options other than the options identified in table 2.3-2

Products that send the ‘update file’ command:

- MAY populate the arguments field with the “response_requested” argument. “Complete”, “Ack” and “None” are valid Response-type for ‘update file’
- MUST NOT include other command arguments.
- MUST populate the name specifier in the target.
- SHOULD populate the path specifier in the target.

Products that receive the ‘update file’ command:

- but cannot parse or process the command
  - MUST NOT respond with a OK/200.
  - SHOULD respond with status code 400.
  - MAY respond with the 500 status code
- but do not support the ‘update file’ command type
  - MUST NOT respond with a OK/200.
  - SHOULD respond with status code 501
  - SHOULD respond with ‘command not supported’ in the status text.
  - MAY respond with status code 500
- but cannot access the file specified in the file target
  - MUST respond with status code 500
  - SHOULD respond with ‘cannot access file’ in the status text.
- upon successful parsing and initiating the processing of the ‘update file’ command, products MAY respond with response code 102.
- upon completion of all the steps necessary to complete the update and the actuator commences operations functioning with the new file, actuators products SHOULD respond with response code 200.

Refer to Annex C for sample commands.

3 Conformance statements

Definitions: The following terms apply to this section:

- **OpenC2 SLPF Producers:** Entities that send commands to and receive responses from OpenC2 SLPF consumers.
- **Basic SLPF Producers:** OpenC2 SLPF producers that are conformant to all of the normative requirements identified in this specification as REQUIRED to implement.
- **Complete SLPF Producers:** OpenC2 SLPF producers that are conformant to all of the normative requirements identified in this specification
- **OpenC2 SLPF Consumers:** Entities that receive commands from and send responses to OpenC2 SLPF Producers.
- **Basic SLPF Consumers:** OpenC2 SLPF consumers that are conformant to all of the normative requirements identified in this specification as REQUIRED to implement.
- **Complete SLPF Consumers:** OpenC2 SLPF consumers that are conformant to all of the normative requirements identified in this specification

A conformant OpenC2 implementation SHALL meet all the normative requirements specified in the SLPF Profile as well as applicable normative requirements specified in the Language Specification. Table 3-1 provides a overview of the applicable normative requirements. The traceability for conformance criteria involving commands (action target pairs) are ‘derived’, where derived is defined as a combination of more than a single normative statements from the language specification into a single criteria within the SLPF specification. Sections 3.1 through 3.X provide a concise summary of the corresponding conformance criteria.

Table 3-1: SLPF Traceability Matrix

Conformance Criteria	SLPF Section Reference	Language Specification (V 1.0) Reference	Conformance Criteria Reference
JSON Serialization		2.2	3.1-1.1 and 3.2-1.1
OpenC2 Transfer Specification	1.1	5	3.1-1.3, 3.2-1.3, 3.3-1.2 and 3.4-1.2
Actions	2.1.1	3.3.1.2	
Targets	2.1.1.2	3.4.1.3, 3.4.1.8, 3.4.1.9, 3.4.1.11, 3.4.1.12,	
Slpf:rule_number Target	2.1.1.2.1	SLPF-specific	
‘Query features’ command	2.3.3.1	4	3.1-2.1.5 and 3.2-2.1.3
‘Allow	2.3.1	Derived	3.1-2.1.1, 3.1-2.1.2,

ip_connection| ip_addr'			3.2-2.1.1 and 3.4-2.1.3
Deny ip_connection| ip_addr'	2.3.2	Derived	3.1-2.1.3, 3.1-2.1.4, 3.2-2.1.2 and 3.4-2.1.4
'Delete slpf:rule_number'	2.3.4.1	SLPF-specific	3.3-2.1.1 and 3.4-2.1.1
'Update file'	2.3.5.1	Derived	3.3-2.1.2 and 3.4-2.2
Command Argument: Response_requested	2.1.3	3.3.1.5	3.1-3.1, 3.2-3.1, 3.2-3.2.1 and 3.2-3.2.2
Command Argument: start_time, end_time and/or duration.	2.1.3	3.3.1.5	3.3-3.1, 3.3-3.2.1, 3.3-3.2.2 3.4-3.1, 3.4-3.2.1, 3.4-3.2.2
Command Argument: running, direction and/or drop_process	2.1.3	SLPF-specific	3.3-3.3.1, 3.3-3.3.2, 3.3-3.4 3.4-3.3.1, 3.4-3.3.2, 3.4-3.4
Response Codes	2.2.1	3.3.2.2	

3.1 Conformance Clause 1: Basic SLPF Producers

The Actuator Profile for the basic Stateless Packet Filtering Producers specifies the minimum functionality required in order for an OpenC2 SLPF Producer implementation to be conformant.

1. General Conformance:
   1.1. **MUST** support JSON serialization of OpenC2 commands that are syntactically valid in accordance with the property tables presented in Section 2.1.
   1.2. All serializations **MUST** be implemented in a manner such that the serialization validates against and provides a one-to-one mapping to the property tables in section 2.1 of this specification.
   1.3. **MUST** support the use of a Transfer Specification that is capable of delivering authenticated, ordered, lossless and uniquely identified OpenC2 messages.
   1.4. **MUST** be conformant with Version 1.0 (or higher) of the Language Specification

2. Base Commands (ACTION and TARGET pairs):
    - 2.1. **MUST** implement the following action target pairs where the actions and targets are defined in version 1.0 of the Language Specification.
        - 2.1.1. ‘allow ip_connection’ in accordance with the normative text provided in section 2.3.1 of this specification
        - 2.1.2. ‘allow ip_addr’ in accordance with the normative text provided in section 2.3.1 of this specification
        - 2.1.3. ‘deny ip_connection’ in accordance with the normative text provided in section 2.3.2 of this specification
        - 2.1.4. ‘deny ip_addr’ in accordance with the normative text provided in section 2.3.2 of this specification
        - 2.1.5. ‘query openc2’ in accordance with the normative text provided in version 1.0 of the OpenC2 Language Specification.

3. Command Arguments:
    - 3.1. **MUST** implement the ‘response_requested’ command argument as a valid option for any command:

3.2 Conformance Clause 2: Basic SLPF Consumers

The Actuator Profile for Stateless Packet Filtering Consumers specifies the minimum functionality required in order for a basic SPLF Consumer implementation to be conformant.

1. General Conformance:
    - 1.1. **MUST** support JSON serialization of OpenC2 commands that are syntactically valid in accordance with the property tables presented in Section 2.1.
    - 1.2. All serializations **MUST** be implemented in a manner such that the serialization validates against and provides a one-to-one mapping to the property tables in section 2.1 of this specification.
    - 1.3. **MUST** support the use of a transfer specification that is capable of delivering authenticated, ordered, lossless and uniquely identified OpenC2 messages.
    - 1.4. **MUST** be conformant with Version 1.0 (or higher) of the Language Specification

2. Base Commands (ACTION and TARGET pairs):
    - 2.1. **MUST** implement the following action target pairs where the actions and targets are defined in version 1.0 of the Language specification.
        - 2.1.1. ‘allow ip_connection’ or ‘allow ip_addr’ in accordance with the normative text provided in section 2.3.1 of this specification
        - 2.1.2. ‘deny ip_connection’ or ‘deny ip_addr’ in accordance with the normative text provided in section 2.3.2 of this specification

2.1.3. ‘query openc2’ in accordance with the normative text provided in version 1.0 of the OpenC2 Language Specification.

3. Command Arguments:

3.1. **MUST** implement the ‘response_requested’ command argument as a valid option for any command:

3.2. Processing response_requested command arguments

3.2.1. All commands received with the response argument set to ‘none’ **MUST** process the command and **MUST NOT** send a response. This conformance clause supersedes all other normative text as it pertains to responses.

3.2.2. All commands received without the response argument (or response argument not set) **MUST** process the command and respond in a manner that is consistent with "response_requested" : “complete”.

3.3 Conformance Clause 3: Complete SLPF Producers

OpenC2 SLPF producers that are conformant to all of the normative requirements identified in this specification.

1. General Conformance:

1.1. **MUST** meet all of conformance criteria identified in Conformance Clause 1 of this specification

1.2. **MUST** support the use of one or more published OpenC2 Transfer Specifications which identify underlying transport protocols such that an authenticated, ordered, lossless, delivery of uniquely identified OpenC2 messages is provided as referenced in section 1 of this specification

2. Commands (ACTION and TARGET pairs):

2.1. **MUST** implement the following action target pairs where: Version 1.0 of the Language Specification defines the actions, Version 1.0 of the Language Specification defines the ‘file’ target; and the ‘slpf:rule_number’ target type is defined in this specification

2.1.1. ‘delete slpf:rule_number’ in accordance with the normative text provided in section 2.3.4.1 of this specification

2.1.2. ‘update file’ in accordance with the normative text provided in section 2.3.5.1 of this specification

3. Command Arguments:

3.1. **MUST** implement the start_time command argument as a valid option for any command other than ‘query <target>’

3.2. **MUST** implement the following command arguments as a valid option for any command other than ‘query <target>’ and ‘update file’

3.2.1. end_time

3.2.2. duration

3.3. **MUST** implement the following command arguments as a valid option for 'allow <target>' and/or 'deny <target>' commands

3.3.1. running

3.3.2. direction

3.4. **MUST** implement the drop_process command argument as a valid option for the 'deny <target>' command

3.4 Conformance Clause 4: Complete SLPF Consumers

OpenC2 SLPF producers that are conformant to all of the normative requirements identified in this specification.

1. General Conformance:

1.1. **MUST** meet all of conformance criteria identified in Conformance Clause 2 of this specification

1.2. **MUST** support the use of one or more published OpenC2 Transfer Specifications which identify underlying transport protocols such that an authenticated, ordered, lossless, delivery of uniquely identified OpenC2 messages is provided as referenced in section 1 of this specification

2. Commands (ACTION and TARGET pairs):

2.1. **MUST** implement the following action target pairs where version 1.0 of the Language specification defines the 'file' target and actions; and the 'slpf:rule_number' target type is defined in this specification

2.1.1. 'delete slpf:rule_number' in accordance with the normative text provided in section 2.3.4.1 of this specification

2.1.2. 'update file' in accordance with the normative text provided in section 2.3.5.1 of this specification

2.1.3. 'allow ip_connection' and 'allow ip_addr' in accordance with the normative text provided in section 2.3.1 of this specification

2.1.4. 'deny ip_connection' and 'deny ip_addr' in accordance with the normative text provided in section 2.3.2 of this specification

3. Command Arguments:

3.1. **MUST** implement the start_time command argument as a valid option for any command other than 'query <target>'

3.2. **MUST** implement the following command arguments as a valid option for any command other than 'query <target>' and 'update file'

3.2.1. end_time

3.2.2. duration

3.3. **MUST** implement the following command arguments as a valid option for 'allow <target>' and/or 'deny <target>' commands

3.3.1. running

3.3.2. direction

3.4. **MUST** implement the drop_process command argument as a valid option for the 'deny <target>' command

Annex A SLPF Schema

This annex defines the data objects used by conforming SLPF implementations, as shown in Section 2. This annex is normative, however in the event of a conflict between this schema and the property tables presented in section 2, the property tables are authoritative.

Schema Files:

- https://github.com/oasis-tcs/openc2-oc2ls/tree/master/xxx.jadn (authoritative)
- https://github.com/oasis-tcs/openc2-oc2ls/tree/master/xxx.pdf (pretty-printed)

~~~
```
{
 "meta": {
  "module": "oasis-open.org/openc2/v1.0/ap-slpf",
  "patch": "wd03",
  "title": "Stateless Packet Filter",
  "description": "Data definitions for Stateless Packet Filtering (SLPF)
functions",
  "exports": ["Target", "Specifiers", "Args", "Results"]
 },
 "types": [
  ["Target", "Choice", [], "", [
    [1, "rule_number", "Rule-ID", [], ""]]
  ],
  ["Args", "Map", [], "", [
    [1, "drop_process", "Drop-Process", ["[0"], ""],
    [2, "running", "Boolean", ["[0"], ""],
    [3, "direction", "Direction", ["[0"], ""],
    [4, "insert_rule", "Rule-ID", ["[0"], ""]]
  ],
  ["Drop-Process", "Enumerated", [], "", [
    [1, "none", ""],
    [2, "reject", ""],
    [3, "false_ack", ""]]
  ],
  ["Direction", "Enumerated", [], "", [
    [1, "ingress", ""],
    [2, "egress", ""]]
  ],
  ["Rule-ID", "Integer", [], ""],
  ["Specifiers", "Map", [], "", [
    [1, "hostname", "String", ["[0"], ""],
    [2, "named_group", "String", ["[0"], ""],
    [3, "asset_id", "String", ["[0"], ""],
    [4, "asset_tuple", "String", ["[0", "]10"], ""]]
  ],
  ["Results", "Map", [], "", [
~~~

~~~
    [1, "rule_number", "Rule-ID", ["[0"], ""]]
  ]]
}
```
~~~

Annex B Tailored OpenC2 Schema

This annex is a copy of the schema from the OpenC2 Language Specification tailored to include only elements needed to support the SLPF functions defined in this document. This subset defines the elements of the Language Specification that are meaningful in the context of SLPF, however an implementation may have capabilities beyond the scope of an SLPF therefore may support additional elements of the OpenC2 language beyond those included here.

This annex is normative, however in the event of a conflict with the schema in the OpenC2 Language Specification, the Language Specification is authoritative.

Schema Files:

- https://github.com/oasis-tcs/openc2-oc2ls/tree/master/xxx.jadn (authoritative)
- https://github.com/oasis-tcs/openc2-oc2ls/tree/master/xxx.pdf (pretty-printed)

~~~~
```
{
 "meta": {
  "module": "oasis-open.org/openc2/v1.0/openc2-lang",
  "patch": "wd09-slpf",
  "title": "OpenC2 Language Objects",
  "description": "OpenC2 Language content used by Stateless Packet Filters.",
  "imports": [
   ["slpf", "oasis-open.org/openc2/v1.0/ap-slpf"],
   ["jadn", "oasis-open.org/openc2/v1.0/jadn"]
  ],
  "exports": ["OpenC2-Command", "OpenC2-Response"]
 },
 "types": [
  ["OpenC2-Command", "Record", [], "", [
    [1, "action", "Action", [], ""],
    [2, "target", "Target", [], ""],
    [3, "args", "Args", ["[0"], ""],
    [4, "actuator", "Actuator", ["[0"], ""]
  ]],
  ["Action", "Enumerated", [], "", [
    [3, "query", ""],
    [6, "deny", ""],
    [8, "allow", ""],
    [16, "update", ""],
    [20, "delete", ""]
  ]],
  ["Target", "Choice", [], "", [
    [16, "features", "Features", [], ""],
    [10, "file", "File", [], ""],
    [11, "ip_addr", "IP-Addr", [], ""],
~~~~

```
    [15, "ip_connection", "IP-Connection", [], ""],
    [1024, "slpf", "slpf:Target", [], ""]
  ]],
  ["Actuator", "Choice", [], "", [
    [1024, "slpf", "slpf:Specifiers", [], ""]
  ]],
  ["Args", "Map", [], "", [
    [1, "start_time", "Date-Time", ["[0"], ""],
    [2, "stop_time", "Date-Time", ["[0"], ""],
    [3, "duration", "Duration", ["[0"], ""],
    [4, "response_requested", "Response-Type", ["[0"], ""],
    [1024, "slpf", "slpf:Args", ["[0"], ""]
  ]],
  ["OpenC2-Response", "Map", [], "", [
    [1, "status", "Status-Code", ["[0"], ""],
    [2, "status_text", "String", ["[0"], ""],
    [6, "versions", "Version", ["[0", "]0"], ""],
    [7, "profiles", "jadn:Uname", ["[0", "]0"], ""],
    [8, "schema", "jadn:Schema", ["[0"], ""],
    [9, "pairs", "Action-Targets", ["[0", "]0"], ""],
    [10, "rate_limit", "Number", ["[0"], ""],
    [1024, "slpf", "slpf:Results", ["[0"], ""]
  ]],
  ["Status-Code", "Enumerated", ["="], "", [
    [102, "Processing", ""],
    [200, "OK", ""],
    [400, "Bad Request", ""],
    [500, "Internal Error", ""],
    [501, "Not Implemented", ""]
  ]],
  ["Features", "ArrayOf", ["*Feature", "[0"], ""],
  ["File", "Map", [], "", [
    [1, "name", "String", ["[0"], ""],
    [2, "path", "String", ["[0"], ""],
    [3, "hashes", "Hashes", ["[0"], ""]
  ]],
  ["IP-Addr", "Binary", ["@ip-addr"], ""],
  ["IP-Connection", "Record", [], "", [
    [1, "src_addr", "IP-Addr", ["[0"], ""],
    [2, "src_port", "Port", ["[0"], ""],
    [3, "dst_addr", "IP-Addr", ["[0"], ""],
    [4, "dst_port", "Port", ["[0"], ""],
    [5, "protocol", "L4-Protocol", ["[0"], ""]
  ]],
  ["Request-Id", "Binary", [], ""],
  ["Date-Time", "Integer", [], ""],
  ["Duration", "Integer", [], ""],
  ["Hashes", "Map", [], "", [
    [1, "md5", "Binary", ["[0"], ""],
    [4, "sha1", "Binary", ["[0"], ""],
```

~~~
    [6, "sha256", "Binary", ["[0"], ""]
  ]],
  ["L4-Protocol", "Enumerated", [], "", [
    [1, "icmp", ""],
    [6, "tcp", ""],
    [17, "udp", ""],
    [132, "sctp", ""]
  ]],
  ["Port", "Integer", ["[0", "]65535"], ""],
  ["Feature", "Enumerated", [], "", [
    [1, "versions", ""],
    [2, "profiles", ""],
    [3, "schema", ""],
    [4, "pairs", ""],
    [5, "rate_limit", ""]
  ]],
  ["Response-Type", "Enumerated", [], "", [
    [0, "none", ""],
    [1, "ack", ""],
    [2, "status", ""],
    [3, "complete", ""]
  ]],
  ["Version", "String", [], ""],
  ["Action-Targets", "Array", [], "", [
    [1, "action", "Action", [], ""],
    [2, "targets", "Target", ["]0", "*"], ""]
  ]]
 ]
}
```
~~~

Annex C Sample commands (Informative)

This section will summarize and provide examples of OpenC2 commands as they pertain to SLPF firewalls. The sample commands will be encoded in verbose JSON, however other encodings are possible provided the command is validated against the schema presented in Annex A. Examples of corresponding responses will be provided where appropriate.
The samples provided in this section are for illustrative purposes only and are not to be interpreted as operational examples for actual systems.

The following examples include Binary fields which are serialized in Base64url format. The examples show JSON-serialized commands; the conversion of Base64url serialized values to Binary data and String display text is:

Base64url	Binary	Display String
AQIDBA	01020304	1.2.3.4
xgIDBA	c6020304	198.2.3.4
xjNkEQ	c6336411	198.51.100.17

The examples include Integer Date-Time fields; the conversion of Integer values to String display text is:

Integer	Display String
1534775460000	Monday, August 20, 2018 2:31:00 PM GMT, 2018-08-20T10:31:00-04:00

C.1 Deny and Allow

Deny and allow are mandatory to implement and can be treated as mathematical complements of each other. Unless otherwise stated, the example targets, specifiers, modifiers and corresponding responses are applicable to both actions.

C.1.1 Deny a particular connection

Block a particular connection within the domain and do not send a host unreachable

Command:

~~~
```
{
  "action": "deny",
~~~

~~~
  "target": {
    "ip_connection": {
      "protocol": "tcp",
      "src_addr": "AQIDBA",
      "src_port": 10996,
      "dst_addr": "xgIDBA",
      "dst_port": 80
    }
  },
  "args": {
    "start_time": 1534775460000,
    "duration": 500,
    "response_requested": "ack",
    "slpf": {
      "drop_process": "none"
    }
  },
  "actuator": {
    "slpf": {
      "asset_id": "30"
    }
  }
}
```
~~~

Response:

~~~
```
{
    "status": 200
}
```
~~~

C.1.2 Block all outbound ftp transfers

Block all outbound ftp data transfers, send false acknowledgement and request ack. Note that the five-tuple is incomplete. Note that the response_type field was not populated therefore will be ‘complete’. Also note that the actuator called out was SLPF with no additional specifiers, therefore all endpoints that can execute the command should.

Command:

~~~
```
{
  "action": "deny",
  "target": {
    "ip_connection": {
      "protocol": "tcp",
      "src_port": 21
    }
~~~

```
  },
  "args": {
    "slpf": {
      "drop_process": "false_ack"
    }
  },
  "actuator": {
    "slpf": {}
  }
}
```

Responses:

Case One: the actuator successfully issued the deny.

```
{"status": 200}
```

Case Two: the command failed due to a syntax error in the command. Optional status text can provide error details for debugging or logging.

```
{
  "status": 400,
  "status_text": "Validation Error: Target: ip_conection"
}
```

Case Three: the command failed because an argument was not supported.

```
{
  "status": 501
}
```

C.1.3 Block all inbound traffic from a particular source.

Block all inbound traffic from 1.2.3.4 and do not respond. In this case the ip_addr target and the direction argument was used. In this case only the perimeter filters should update the rule.

Command:

```
{
  "action": "deny",
  "target": {
    "ip_addr": "AQIDBA"
  },
  "args": {
    "response_requested": "none",
    "slpf": {
      "direction": "ingress"
    }
  },
  "actuator": {
    "slpf": {
      "named_group": "perimeter"
    }
  }
}
```

C.1.4 Permit ftp transfers to a particular destination.

Permit ftp data transfers to ip address 198.51.100.17 from any source. (Note that an actual application would also need to allow ftp-data (port 20) in order for transfers to be permitted.)

Command:

```
{
  "action": "allow",
  "target": {
    "ip_connection": {
      "protocol": "tcp",
      "dst_addr": "xjNkEQ",
      "src_port": 21
    }
  },
  "actuator": {
    "slpf": {}
  }
}
```

In this case the actuator returned a rule number associated with the allow.

Response:

```
{
  "status": 200,
  "slpf": {
    "rule_number": 1234
  }
}
```

~~~~
```
~~~~

C.2 Delete Rule

Used to remove a firewall rule rather than issue an allow or deny to counteract the effect of an existing rule. Implementation of the ‘delete slpf:rule_number’ command is OPTIONAL.

In this case the rule number assigned in a previous allow will be removed (refer to the final example in section C.1)

Command:

~~~~
```
{
  "action": "delete",
  "target": {
    "slpf": {
      "rule_number": 1234
    }
  },
  "args": {
    "response_requested": "complete"
  },
  "actuator": {
    "slpf": {}
  }
}
```
~~~~

C.3 Update file

Implementation of the Update action is optional. Update is intended for the device to process new configuration files. The update action is a compound action in that all of the steps required for a successful update (such as download the new file, install the file, reboot etc.) are implied. File is the only valid target type for Update.

Instructs the firewalls to acquire a new configuration file. Note that all network based firewalls will install the new update because no particular firewall was identified. Host based firewalls will not act on this because network firewalls were identified as the actuator.

Command:

~~~~
```
{
  "action": "update",
  "target": {
    "file": {
      "path": "\\\\someshared-drive\\somedirectory\\configurations",
      "name": "firewallconfiguration.txt"
~~~~

```
      }
  },
  "actuator": {
    "slpf": {
      "named_group": "network"
    }
  }
}
```

Responses:

Successful update of the configuration

```
{"status": 200}
```

This actuator does not support the update file command

```
{
  "status": 501,
  "status_text": "Update-File Not Implemented"
}
```

This actuator could not access the file

```
{
  "status": 500,
  "status_text": "Server error, Cannot access file"
}
```

C.4 Query openc2

Implementation of query openc2 is required. The query openc2 command is intended to enable the openc2 producer to determine the capabilities of the actuator. The query openc2 command can also be used to check the status of the actuator.

C.4.1 No query items set

This command uses query openc2 with no query items to verify that the actuator is functioning.

Command:

~~~
```
{
  "action": "query",
  "target": {
    "openc2": []
  }
}
```
~~~

Response:

The actuator is alive.

~~~
```
{"status": 200}
```
~~~

C.4.2 Version of Language specification supported

This command queries the actuator to determine which version(s) of the language specification are supported. The language specifications use semantic versioning ("major.minor"); for each supported major version the actuator need only report the highest supported minor version.

Command:

~~~
```
{
  "action": "query",
  "target": {
    "openc2": ["versions"]
  }
}
```
~~~

Response:

The Actuator supports language specification versions 1.0 - 1.3.

~~~
```
{
  "status": 200,
  "versions": ["1.3"]
}
```
~~~

C.4.3 Actuator profiles supported

This command queries the actuator to determine both the language versions and the actuator profiles supported.

Command:

~~~
```
{
  "action": "query",
  "target": {
    "openc2": ["versions", "profiles"]
  }
}
```
~~~

Response:

The actuator device is apparently a smart toaster for which an extension actuator profile has been written. The device supports both the standard slpf functions and whatever commands are defined in the extension profile.

~~~
```
{
  "status": 200,
  "versions": ["1.3"],
  "profiles": [
    "oasis-open.org/openc2/v1.0/ap-slpf",
    "example.com/openc2/products/iot-toaster"
    ]
}
```
~~~

C.4.4 Specific Commands Supported

This command queries the actuator to determine which action-target pairs are supported. Not all targets are meaningful in the context of a specific action, and although a command such as "update ip_connection" may be syntactically valid, the combination does not specify an operation supported by the actuator.

Command:

For each supported action list the targets supported by this actuator.

~~~
```
{
  "action": "query",
  "target": {
    "openc2": ["pairs"]
  }
}
```
~~~

Response:

The actuator supports all action-target pairs shown in Table 2.3-1 - Command Matrix.

```
{
  "status": 200,
  "pairs": [
    ["allow", ["ip_addr", "ip_connection"]],
    ["deny", ["ip_addr", "ip_connection"]],
    ["query", ["openc2"]],
    ["delete", ["slpf:rule_number"]],
    ["update", ["file"]]
  ]
}
```

C.4.5 Actuator Schema

This command queries the actuator for the syntax definition for all supported commands.

Command:

```
{
  "action": "query",
  "target": {
    "openc2": ["schema"]
  }
}
```

Response:

The result is a single schema defining the syntax of all commands supported by this actuator. It is constructed from:

1. the tailored OpenC2 schema module (Annex B), merged with
2. each imported module (e.g., the SLPF schema module of Annex A, schemas from other profiles supported by this actuator), and,
3. further tailored for the specific actuator product by removing any unsupported optional elements.

Schema File:

The non-normative merged schema example shown in this response is available from:

- https://github.com/oasis-tcs/openc2-oc2ls/tree/master/xxx.jadn

```
{
 "status": 200,
 "schema": {
```

```
 "meta": {
  "module": "oasis-open.org/openc2/v1.0/openc2-lang",
  "patch": "wd09-slpf_merged",
  "title": "OpenC2 Language Objects",
  "description": "OpenC2 Language content used by Stateless Packet Filters.",
  "exports": ["OpenC2-Command", "OpenC2-Response"]
 },
 "types": [
  ["OpenC2-Command", "Record", [], "", [
    [1, "action", "Action", [], ""],
    [2, "target", "Target", [], ""],
    [3, "args", "Args", ["[0"], ""],
    [4, "actuator", "Actuator", ["[0"], ""]
  ]],
  ["Action", "Enumerated", [], "", [
    [3, "query", ""],
    [6, "deny", ""],
    [8, "allow", ""],
    [16, "update", ""],
    [20, "delete", ""]
  ]],
  ["Target", "Choice", [], "", [
    [16, "features", "Features", [], ""],
    [10, "file", "File", [], ""],
    [11, "ip_addr", "IP-Addr", [], ""],
    [15, "ip_connection", "IP-Connection", [], ""],
    [1024, "slpf", "slpf:Target", [], ""]
  ]],
  ["Actuator", "Choice", [], "", [
    [1024, "slpf", "slpf:Specifiers", [], ""]
  ]],
  ["Args", "Map", [], "", [
    [1, "start_time", "Date-Time", ["[0"], ""],
    [2, "stop_time", "Date-Time", ["[0"], ""],
    [3, "duration", "Duration", ["[0"], ""],
    [4, "response_requested", "Response-Type", ["[0"], ""],
    [1024, "slpf", "slpf:Args", ["[0"], ""]
  ]],
  ["OpenC2-Response", "Map", [], "", [
    [1, "status", "Status-Code", ["[0"], ""],
    [2, "status_text", "String", ["[0"], ""],
    [6, "versions", "Version", ["[0", "]0"], ""],
    [7, "profiles", "jadn:Uname", ["[0", "]0"], ""],
    [8, "schema", "jadn:Schema", ["[0"], ""],
    [9, "pairs", "Action-Targets", ["[0", "]0"], ""],
    [10, "rate_limit", "Number", ["[0"], ""],
    [1024, "slpf", "slpf:Results", ["[0"], ""]
  ]],
  ["Status-Code", "Enumerated", ["="], "", [
    [102, "Processing", ""],
    [200, "OK", ""],
    [301, "Moved Permanently", ""],
    [400, "Bad Request", ""],
    [401, "Unauthorized", ""],
    [403, "Forbidden", ""],
    [404, "Not Found", ""],
    [500, "Internal Error", ""],
    [501, "Not Implemented", ""],
    [503, "Service Unavailable", ""]
  ]],
  ["Features", "ArrayOf", ["*Feature", "[0"], ""],
  ["File", "Map", [], "", [
    [1, "name", "String", ["[0"], ""],
    [2, "path", "String", ["[0"], ""],
    [3, "hashes", "Hashes", ["[0"], ""]
  ]],
  ["IP-Addr", "Binary", ["@ip-addr"], ""],
  ["IP-Connection", "Record", [], "", [
    [1, "src_addr", "IP-Addr", ["[0"], ""],
    [2, "src_port", "Port", ["[0"], ""],
    [3, "dst_addr", "IP-Addr", ["[0"], ""],
    [4, "dst_port", "Port", ["[0"], ""],
    [5, "protocol", "L4-Protocol", ["[0"], ""]
  ]],
  ["Request-Id", "Binary", [], ""],
  ["Date-Time", "Integer", [], ""],
  ["Duration", "Integer", [], ""],
  ["Hashes", "Map", [], "", [
    [1, "md5", "Binary", ["[0"], ""],
    [4, "sha1", "Binary", ["[0"], ""],
    [6, "sha256", "Binary", ["[0"], ""]
  ]],
  ["L4-Protocol", "Enumerated", [], "", [
    [1, "icmp", ""],
    [6, "tcp", ""],
    [17, "udp", ""],
    [132, "sctp", ""]
  ]],
  ["Port", "Integer", ["[0", "]65535"], ""],
  ["Feature", "Enumerated", [], "", [
    [1, "versions", ""],
    [2, "profiles", ""],
    [3, "schema", ""],
    [4, "pairs", ""],
    [5, "rate_limit", ""]
  ]],
  ["Response-Type", "Enumerated", [], "", [
    [0, "none", ""],
    [1, "ack", ""],
    [2, "status", ""],
    [3, "complete", ""]
  ]],
  ["Version", "String", [], ""],
  ["Action-Targets", "Array", [], "", [
    [1, "action", "Action", [], ""],
    [2, "targets", "Target", ["]0", "*"], ""]
  ]],
  ["slpf:Target", "Choice", [], "", [
    [1, "rule_number", "slpf:Rule-ID", [], ""]
  ]],
  ["slpf:Args", "Map", [], "", [
    [1, "drop_process", "slpf:Drop-Process", ["[0"], ""],
    [2, "running", "Boolean", ["[0"], ""],
    [3, "direction", "slpf:Direction", ["[0"], ""],
    [4, "insert_rule", "slpf:Rule-ID", ["[0"], ""]
  ]],
  ["slpf:Drop-Process", "Enumerated", [], "", [
    [1, "none", ""],
    [2, "reject", ""],
    [3, "false_ack", ""]
  ]],
  ["slpf:Direction", "Enumerated", [], "", [
    [1, "ingress", ""],
    [2, "egress", ""]
  ]],
  ["slpf:Rule-ID", "Integer", [], ""],
  ["slpf:Specifiers", "Map", [], "", [
    [1, "hostname", "String", ["[0"], ""],
    [2, "named_group", "String", ["[0"], ""],
    [3, "asset_id", "String", ["[0"], ""],
    [4, "asset_tuple", "String", ["[0", "]10"], ""]
  ]],
  ["slpf:Results", "Map", [], "", [
    [1, "rule_number", "slpf:Rule-ID", ["[0"], ""]
  ]],
  ["jadn:Schema", "Record", [], "", [
    [1, "meta", "jadn:Meta", [], ""],
    [2, "types", "jadn:Type", ["]0"], ""]
  ]],
  ["jadn:Meta", "Map", [], "", [
    [1, "module", "jadn:Uname", [], ""],
    [2, "patch", "String", ["[0"], ""],
    [3, "title", "String", ["[0"], ""],
    [4, "description", "String", ["[0"], ""],
    [5, "imports", "jadn:Import", ["[0", "]0"], ""],
    [6, "exports", "jadn:Identifier", ["[0", "]0"], ""],
    [7, "bounds", "jadn:Bounds", ["[0"], ""]
  ]],
  ["jadn:Import", "Array", [], "", [
    [1, "nsid", "jadn:Nsid", [], ""],
```

~~~~
    [2, "uname", "jadn:Uname", [], ""]
  ]],
  ["jadn:Bounds", "Array", [], "", [
    [1, "max_msg", "Integer", [], ""],
    [2, "max_str", "Integer", [], ""],
    [3, "max_bin", "Integer", [], ""],
    [4, "max_fields", "Integer", [], ""]
  ]],
  ["jadn:Type", "Array", [], "", [
    [1, "tname", "jadn:Identifier", [], ""],
    [2, "btype", "jadn:JADN-Type", ["*"], ""],
    [3, "opts", "jadn:Option", ["]0"], ""],
    [4, "desc", "String", [], ""],
    [5, "fields", "jadn:JADN-Type", ["&btype", "]0"], ""]
  ]],
  ["jadn:JADN-Type", "Choice", [], "", [
    [1, "Binary", "Null", [], ""],
    [2, "Boolean", "Null", [], ""],
    [3, "Integer", "Null", [], ""],
    [4, "Number", "Null", [], ""],
    [5, "Null", "Null", [], ""],
    [6, "String", "Null", [], ""],
    [7, "Array", "jadn:FullField", ["]0"], ""],
    [8, "ArrayOf", "Null", [], ""],
    [9, "Choice", "jadn:FullField", ["]0"], ""],
    [10, "Enumerated", "jadn:EnumField", ["]0"], ""],
    [11, "Map", "jadn:FullField", ["]0"], ""],
    [12, "Record", "jadn:FullField", ["]0"], ""]
  ]],
  ["jadn:EnumField", "Array", [], "", [
    [1, "", "Integer", [], ""],
    [2, "", "String", [], ""],
    [3, "", "String", [], ""]
  ]],
  ["jadn:FullField", "Array", [], "", [
    [1, "", "Integer", [], ""],
    [2, "", "jadn:Identifier", [], ""],
    [3, "", "jadn:Identifier", [], ""],
    [4, "", "jadn:Options", [], ""],
    [5, "", "String", [], ""]
  ]],
  ["jadn:Identifier", "String", ["$^[a-zA-Z][\\w-]*$", "[1", "]32"], ""],
  ["jadn:Nsid", "String", ["$^[a-zA-Z][\\w-]*$", "[1", "]8"], ""],
  ["jadn:Uname", "String", ["[1", "]100"], ""],
  ["jadn:Options", "ArrayOf", ["*jadn:Option", "[0", "]10"], ""],
  ["jadn:Option", "String", ["[1", "]100"], ""]
 ]
 }
}
```
~~~~

Annex D Acknowledgements

The Actuator Profile Subcommittee was tasked by the OASIS Open Command and Control Technical Committee (OpenC2 TC) which at the time of this submission, had 132 members. The editors wish to express their gratitude to the members of the OpenC2 TC.

The following individuals are acknowledged for providing comments, suggested text and/or participation in the SLPF CSD ballots:

- Barry, Michelle- AT&T
- Brule, Joseph - National Security Agency
- Darley, Trey- New Context
- Darnell, David- North American Energy Standards Board
- Everett, Alex- University of North Carolina at Chapel Hill
- Farrel, Travis- Anomali
- Gray, Anderson- ForeScout
- Gurney, John-Mark- New Context
- Hamilton, David- AT&T
- Hunt, Christian- New Context
- Jackson, April- G2, Inc.
- Kakumaru, Takahiro- NEC Corporation
- Kasavchenko, Kirill- Arbor Networks
- Kemp, Dave- National Security Agency
- Lemire, Dave- G2, Inc.
- MacGregor, Scott- McAfee
- Martinez, Danny- G2, Inc.
- Mathews, Lisa- Department of Defense
- Ortiz, Efrain- Symantec
- Rajarathnam, Nirmal- ForeScout
- Riedel, Daniel- New Context
- Romano, Jason- National Security Agency
- Royer, Phillip- Splunk
- Skeen, Duane- Northrup Grumman
- Sparrell, Duncan- sFractal
- Stair, Michael- AT&T
- Storms, Andrew- New Context
- Stueve, Gerald- Fornetix
- Trost, Bill- AT&T
- Voit, Eric- Cisco
- Webb, Jason- LookingGlass
- White, Chuck- Fornetix
- Yu, Sounil- Bank of America

Annex E Revision History

Revision	Date	Editor	Changes Made
Committee Specification Draft 1	31 AUG 2018	Brule, Joe	Initial draft
Committee Specification Draft 2	04 OCT 2018	Brule, Joe	Added Document overview, complete rewrite of introduction, modified components section to be consistent with Language Specification and address ballot comments, added schema, added conformance section, added examples, added acknowledgements section.
Committee Specification Draft 3	16 OCT 2018	Brule, Joe	Aligned section 1 with other OpenC2 specifications; other changes to track dependencies on the language specification: 1) replace openc2 target with features target, 2) flatten response examples so that there is not a separate "results" layer.