# Open Command and Control (OpenC2) Language Specification Version 1.0

## Working Draft 07

## 11 July 2018

Specification URIs

This version:

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

Previous Version:

- http://docs.oasis-open.org/openc2/oc2ls/v1.0/csd04/md/oc2ls-v1.0-wd06.md (Authoritative)
- http://docs.oasis-open.org/openc2/oc2ls/v1.0/csd04/oc2ls-v1.0-csd04.pdf
- http://docs.oasis-open.org/openc2/oc2ls/v1.0/csd04/oc2ls-v1.0-csd04.html

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

- Jason Romano (jdroman@nsa.gov), National Security Agency
- Duncan Sparrell (duncan@sfractal.com), sFractal Consulting

## Abstract

Cyberattacks are increasingly sophisticated, less expensive to execute, dynamic and automated. The provision of cyberdefense via statically configured products operating in isolation is no longer tenable. Standardized interfaces, protocols and data models will facilitate the integration of the functional blocks within a system or enterprise. 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. It should be understood that a language such as OpenC2 is necessary but insufficient to enable

coordinated cyber response. Other aspects of coordinated cyber response such as sensing, analytics, and selecting appropriate courses of action are beyond the scope of OpenC2.

## 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-Lang-v1.0]

Open Command and Control (OpenC2) Language Specification Version 1.0.

Edited by Jason Romano and Duncan Sparrell.

11 July 2018. OASIS Working Draft 07. oasis-to-fill-in-link.html.

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

## Notices

Copyright © OASIS Open 2018. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

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 available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The name "OASIS" is a trademark of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see https://www.oasis-open.org/policies-guidelines/trademark for above guidance.

# Table of Contents

> **Editor's Note** - This document is NOT complete.
>
> The document development process is based on agile software development principles. Iterative, incremental working documents are being developed, reviewed by the Language Subcommittee, and then submitted to the Technical Committee for approval as a Committee Specification Drafts (CSD).
>
> This is iteration 5 and the expectation is there will be at least another CSD iterations before this document is complete and ready to be submitted for approval as a Committee Specification.
>
> Parenthetical "Editor's Notes" will be removed prior to submitting for Committee Specification. Sections that are expected to added in a later iteration (prior to 1.0) will be labeled with "TBSL" for "To Be Supplied Later", optionally with a guestimate as to which iteration it would be supplied in.

1 Introduction

The OpenC2 Language Specification defines a language used to compose messages for command and control of cyber defense systems and components. The OpenC2 command and control interface for a system is defined by the transport used and the subset/extensions of this Language Specification as defined in an Actuator Profile. The transport MAY be specified by an OpenC2 Transport Specifications such as (Ed note - incl refs and links). An Actuator Profile MUST exist, MAY (or MUST?) follow the Actuator Profile Guidelines (Ed note incl refs), and MUST include which of the options in this Language Specification are conformed to, and what extensions (if any) are added.

A message consists of a set of headers and a body. The headers SHOULD be provided as part of transport.

The OpenC2 language defines two message body types:

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 system that requested the Command be invoked, i.e., the OpenC2 Consumer response to the OpenC2 Producer.

The components of the body 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. The inclusion of an actuator and/or command arguments provide additional precision.

Additional detail regarding the TARGET and ACTUATOR may be included to increase the precision of the command. For example, which target (i.e., target specifier), , which actuator(s) (i.e., actuator specifier) .

An OpenC2 Response is issued as a result of an OpenC2 command. OpenC2 responses are used to provide acknowledgement, status, results of command execution, or other information in conjunction with a particular command.

1.1 Goal

> **Editor's Note** - TBSL - This section will be included in a future iteration (probably iteration 5) prior to submitting for Committee Specification.

1.2 Purpose and Scope

The OpenC2 Language Specification defines the set of components to assemble a complete command and control message and provides a framework so that the language can be extended. To achieve this purpose, the scope of this specification includes:

1. the set of actions and options that may be used in OpenC2 commands
2. the set of targets and target specifiers
3. A syntax that defines the structure of commands and responses
4. an organizational scheme that describes an Actuator Profile
5. The MTI serialization of OpenC2 commands, and responses
6. the procedures for extending the language

The OpenC2 language assumes that the event has been detected, a decision to act has been made, the act is warranted, and the initiator and recipient of the commands are authenticated and authorized. The OpenC2 language was designed to be agnostic of the other aspects of cyber defense implementations that realize these assumptions. The following items are beyond the scope of this specification:

1. Language extensions applicable to some actuators
2. Alternate serializations of OpenC2 commands
3. The enumeration of the protocols required for transport, information assurance, sensing, analytics and other external dependencies

1.3 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).

1.4 Terminology

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].

1.5 Document Conventions

> **Editor's Note** - TBSL - This section will be included in a future iteration (probably iteration 5) prior to submitting for Committee Specification.

1.6 Naming Conventions

RFC2119/RFC8174 key words (see section 1.4) are in all uppercase.

All words in type names are capitalized. 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). Words in property names are separated with an underscore (_), while words in string enumerations and type names are separated with a hyphen (-). All type names, property names, object names, and vocabulary terms are between three and 250 characters long.

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

1.7 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.

[Reference] [Full reference citation]

1.8 Non-Normative References

[Reference] [Full reference citation]

2 OpenC2 Language

2.1 Overview

The OpenC2 language has two distinct message types: Command and Response. The OpenC2 Command describes an action performed on a target. The OpenC2 Response is a means to provide information (such as acknowledgement, status, etc.) as a result of an OpenC2 Command.

2.2 OpenC2 Command

The OpenC2 Command communicates an action to be performed on a target and may include information identifying the actuator(s) that is to execute the command. OpenC2 is agnostic of any particular serialization; however, implementations MUST support JSON serialization of the commands.

2.2.1 Command Structure

An OpenC2 Command has four fields: ACTION, TARGET, ACTUATOR and ARGS.

The ACTION and TARGET fields are required and are populated by one of the 'action-types' in Table 2-1 and the 'target-types' in Table 2-2. A particular target-type may be further refined by one or more 'target-specifiers'.

The optional ACTUATOR field identifies the entity or entities that are tasked to execute the OpenC2 Command.

Information with respect to how the action is to be executed is provided with one or more 'actuator-options'.

The optional ARGS field is populated by one or more 'command arguments' that provide information that influences how the command is executed.

The following list summarizes the fields and subfields of an OpenC2 Command. OpenC2 Commands MUST contain an ACTION and TARGET and MAY contain an ACTUATOR and/or ARGS. OpenC2 is agnostic of any particular serialization; however, implementations MUST support JSON serialization of the commands.

- **ACTION** (required): The task or activity to be performed.
- **TARGET** (required): The object of the action. The ACTION is performed on the target.
    - **TARGET-NAME** (required): The name of the object of the action.
    - **TARGET-SPECIFIERS** (optional): The specifier further identifies the target to some level of precision, such as a specific target, a list of targets, or a class of targets.
- **ACTUATOR** (optional): The ACTUATOR may perform the ACTION on the TARGET. The ACTUATOR type will be defined within the context of an Actuator Profile.

    - **ACTUATOR-NAME** (required): The name of the set of functions (e.g., "slpf") performed by the actuator, and the name of the profile defining commands applicable to those functions.
    - **ACTUATOR-SPECIFIERS** (optional): The specifier identifies the actuator to some level of precision, such as a specific actuator, a list of actuators, or a group of actuators.
- **ARGS** (optional): Provide additional information on how the command is to be performed, such as date/time, periodicity, duration etc.

The TARGET of an OpenC2 Command may include a set of targets of the same type, a range of targets, or a particular target. Specifiers provide additional precision for the target.

The OpenC2 ACTUATOR field identifies the entity(ies) that execute the ACTION on the TARGET. Specifiers for actuators refine the command so that a particular function, system, class of devices, or specific device can be identified. Actuator-options indicate how an action is to be done in the context of the actuator.

Actuator is optional. One case where the Actuator is not specified is the case if the transport provides the mutual authentication so the OpenC2 Producer and Consumer both know the Consumer is the Actuator. One example of this would be an https API with mutual authentication. Another example may be a pub/sub such as OpenDXL. Another case where the actuator is not specified is when 'effects-based actions' are being used such as across trust boundaries - i.e., the Producer says the effect desired (e.g., deny ip, mitigate domain, etc.) but leaves it up to decision making in the OpenC2 Consumer to determine what actuator to use to achieve the desired effect.

ARGS influence the command by providing information such as time, periodicity, duration, or other details on what is to be executed. They can also be used to convey the need for acknowledgement or additional status information about the execution of a command.

2.2.2 Action Vocabulary

The normative list of actions is found in section 3.2.1.2.OpenC2 actions can be grouped by their general activity:

- *Actions that Control Information*: These actions are used to gather information needed to determine the current state or enhance cyber situational awareness.
- *Actions that Control Access*: These actions are used to control traffic flow and file permissions (e.g., allow/deny).
- *Actions that Control Activities/Devices*: These actions are used to control the state or the activity of a system, a process, a connection, a host, or a device. The actions are used to execute tasks, adjust configurations, set and update parameters, and modify attributes.
- *Effects-Based Actions*: Effects-based actions are at a higher level of abstraction for purposes of communicating a desired impact rather than a command to execute specific tasks. This level of abstraction enables coordinated actions between enclaves, while permitting a local enclave to optimize its workflow for its specific environment. Effects-based action assumes that the recipient enclave has a decision-making capability

because effects-based actions typically do not have a one-to-one mapping to the other actions.

Each command MUST contain one, and only one, action. Only the actions in Section 3.2.1.2 SHALL be used.

2.2.3 Target Vocabulary

The TARGET is the object of the ACTION (or alternatively, the ACTION is performed on the TARGET). The normative set of TARGETs is in Section 3.2.1.3

The target vocabulary is extensible - see Section 2.2.6.

2.2.4 Actuator

An ACTUATOR is an implementation of a cyber defense function that executes the ACTION on the TARGET. An Actuator Profile is a specification that identifies the subset of ACTIONS, TARGETS and other aspects of this language specification that are mandatory to implement or optional in the context of a particular ACTUATOR. An Actuator Profile also defines ACTUATOR-SPECIFIERS that are meaningful and possibly unique to the actuator.

An Actuator Profile SHALL be composed in accordance with the framework in section 4.

The ACTUATOR field in the command is optional for those cases where the implied actuator(s) are unambiguous, e.g. at 1:1 mutually-authenticated secure transport link.

2.2.5 Command Argument Vocabulary

Command Arguments influence a command. Command Arguments provide additional information to refine how the command is to be performed such as time, periodicity, or duration, or convey the need for status information such as a response is required. The requested status/information will be carried in a RESPONSE.

The valid Command Arguments are in Section 3.2.1.7.

> **Editor's Note** - Additional usage guidance for these command options will be included in a future working draft.

2.2.6 Imported Data

In addition to the targets, actuators, and other language elements defined in this specification, OpenC2 messages may contain data objects imported from other specifications and/or custom data objects defined by the implementers. The details are specified in a data profile which contains:

1. a prefix indicating the origin of the imported data object is outside OpenC2:
    - `x_` (profile)
2. a unique name for the specification being imported, e.g.:

    - For shortname `x_kmipv2.0` the full name would be `/docs.oasis-open.org/openc2/profiles/kmip-v2.0`,
    - For shortname `x_sfslpf` the full name would be `/docs.sfractal.com/slpf/v1.1/x_slpf-profile-v1.1`
3. a namespace identifier (nsid) - a short reference, e.g., `kmipv2.0`, to the unique name of the specification
4. a list of object identifiers imported from that specification, e.g., `Credential`
5. a definition of each imported object, either referenced or contained in the profile
6. conformance requirements for implementations supporting the profile

The data profile itself can be the specification being imported or the data profile can reference an existing specification. In the example above, the data profile created by the OpenC2 TC to represent KMIP could have a unique name of `/docs.oasis-open.org/openc2/profiles/kmip-v2.0`. The data profile would note that it is derived from the original specification `/docs.oasis-open.org/kmip/spec/v2.0/kmip-spec-v2.0`. In the example for shortname `x_sfslpf`, the profile itself could be defined in a manner directly compatible with OpenC2 and would not reference any other specification.

An imported object is identified by namespace identifier and object identifier. While the data profile may offer a suggested nsid, the containing schema defines the nsids that it uses to refer to objects imported from other specifications:

```
import /docs.oasis-open.org/openc2/profiles/kmip-v2.0 as x_kmip_2.0
```

An element using an imported object identifies it using the nsid:

```
{
    "target": {
        "x_kmip_2.0": {
            {"kmip_type": "json"},
            {"operation": "RekeyKeyPair"},
            {"name": "publicWebKey11DEC2017"}
        }
    }
}
```

A data profile can define its own abstract syntax for imported objects, or it can reference content as defined in the specification being imported. Defining an abstract syntax allows imported objects to be represented in the same format as the containing object. Referencing content directly from an imported specification results in it being treated as an opaque blob if the imported and containing formats are not the same (e.g., an XML or TLV object imported into a JSON OpenC2 command, or a STIX JSON object imported into a CBOR OpenC2 command).

The OpenC2 Language MAY be extended using imported data objects for TARGET, TARGET_SPECIFIER, ACTUATOR, ACTUATOR_SPECIFIER, ARGS, and RESULTS. The list of ACTIONS in Section 3.2.1.2 SHALL NOT be extended.

2.3 OpenC2 Response

The OpenC2 Response is a message sent from an entity as the result of a command. Response messages provide acknowledgement, status, results from a query, or other information as requested from the issuer of the command. Response messages are solicited and correspond to a command.

2.3.1 Response Structure

The following list summarizes the fields and subfields of an OpenC2 Response. OpenC2 Responses MUST contain an STATUS and MAY contain an STATUS_TEXT and/or RESULTS. OpenC2 is agnostic of any particular serialization; however, implementations MUST support JSON serialization of the responses.

- **STATUS** (required): An integer containing a numerical status code
- **STATUS_TEXT** (optional): A free-form string containing human-readable description of the response status. The string can contain more detail than is represented by the status code, but does not affect the meaning of the response.
- **RESULTS** (optional): Contains the data or extended status code that was requested from an OpenC2 Command. If not present, the status code is a sufficient response.

3 OpenC2 Property Tables

3.1 Terminology

3.1.1 Data Types

The syntax of valid OpenC2 messages is defined using the following datatypes:

Type	Description
Primitive Types	
Binary	A sequence of octets or bytes. Serialized either as binary data or as a string using an encoding such as hex or base64.
Boolean	A logical entity that can have two values: `true` and `false`. Serialized as either integer or keyword.
Integer	A number that can be written without a fractional component. Serialized either as binary data or a text string.
Number	A real number. Valid values include integers, rational numbers, and irrational numbers. Serialized as either binary data or a text string.
Null	Nothing, used to designate fields with no value. Serialized as a keyword or an empty string.
String	A sequence of characters. Each character must have a valid Unicode codepoint.
Structures	
Array	An ordered list of unnamed fields. Each field has an ordinal position and a type. Serialized as a list.
ArrayOf	An ordered list of unnamed fields of the same type. Each field has an ordinal position and must be the specified type. Serialized as a list.
Choice	One field selected from a set of named fields. The value has a name and a type. Serialized as a one-element map.
Enumerated	A set of id:name pairs. Serialized as either the integer id or the name string.
Map	An unordered set of named fields. Each field has a name and a type. Serialized as a mapping type (referred to in various programming languages as: associative array, dict, dictionary, hash, map, object).
Record	An ordered list of named fields, e.g. a message, record, structure, or row in a table. Each field has an ordinal position, a name, and a type. Serialized as either a list or a map.

3.1.2 Idioms

The following types are defined as value constraints applied to String (text string), Binary (octet string) or Integer values. Idiomatic types have more than one natural representation within an implementation. Interoperability is not affected by how these types are handled internally by an implementation, but the serialized representation must be specified. For JSON format all idiomatic types shown here are converted (if necessary) to String representation before serialization.

Type	Base	Description
Constraints		
Domain-Name	String	RFC 1034 Section 3.5
Email-Addr	String	RFC 5322 Section 3.4.1
Identifier	String	(TBD rules, e.g., initial alpha followed by alphanum or underscore)
URI	String	RFC 3986
JSON	String	JSON value, RFC 7159 Section 3. Note that a JSON value carried in a JSON string requires every quote (") to be escaped.
Idioms		
Date-Time	String	date-time, RFC 3339 Section 5.6
	Integer	32, 64, or 128 bit RFC 5905 NTP time value
Duration	String	duration, RFC 3339 Appendix A (ISO 8601)
	Integer	32 or 64 bit RFC 5905 NTP time value
IP-Addr	String	IPv4 or IPv6 address in CIDR notation, RFC 2673 Section 3.2
	Integer	32 bit IPv4 address or 128 bit IPv6 address
MAC-Addr	String	48 bit Media Access Code, hex encoded without separators
	Integer	48 bit Media Access Code / Extended Unique Identifier
Port	String	Service Name or Transport Protocol Port Number, RFC 6335
	Integer	16 bit RFC 6335 Transport Protocol Port Number
UUID	String	String representation of a UUID, RFC 4122 Section 3
	Integer	128 bit Universal Unique Identifier, RFC 4122 Section 4

3.1.3 Cardinality

Property tables for types based on Array, Map and Record include a cardinality column (#) that specifies the minimum and maximum number of values of a field. The most commonly used cardinalities are:

- 1 Required and not repeatable
- 0..1 Optional and not repeatable

- 1..n Required and repeatable
- 0..n Optional and repeatable

The cardinality column may also specify a range of sizes, e.g.,:

- 3..5 Required and repeatable with a minimum of 3 and maximum of 5 values

> **Editor's Note** - The cardinality column will be applied to all of the Array, Map, and Record property tables in the next iteration of this specification.

3.1.4 Selectors

A Choice field within an Array, Map or Record type may reference the contents of another field within that type to select which element of the choice is present. The selector (key) field can be an enumeration autogenerated from a Choice type by appending ".*" to the type. The Choice type can reference the selector by appending ".&selector-name" to the type. For example:

Type Name: Example

Base Type: Record

ID	Name	Type	#	Description
1	key	Animal.*	1	Selector autogenerated from choice
2	date	String	1	... other fields in this record
3	val	Animal.&key	1	Value of choice selected by "key" field

Type Name: Animal

Base Type: Choice

ID	Name	Type	Description
1	dog	String	Coat color if animal is a dog
2	fish	Number	Length in inches if animal is a fish

3.2 Messages

An OpenC2 Message is a protocol data unit that is exchanged between OpenC2 producers and OpenC2 consumers for purposes of commanding. An OpenC2 message consists of a ‘message body’ and ‘message head’. The message body communicates the intended action on a target and associated response. The message head provides metadata to support the transfer of the message body between the participants of the command/ response.

The scope of this specification is to define the ACTION and TARGET portions of body (or payload) of the OpenC2 message. An OpenC2 body is either a Command or a Responses where the properties of the OpenC2 command are defined in section 3.2.1 and the properties of the response are defined in section 3.2.2.

The message head is beyond the scope of this specification and is defined in transfer specifications such as OpenC2-HTTPS, OpenC2-MQTT, OpenC2-CoAP etc. Transfer specifications SHOULD include the following information:

- Version
- Command id
- Timestamp
- Sender
- Content type

In addition to the ACTION and TARGET, the body of the OpenC2 message has an optional ACTUATOR. Other than identification of namespace identifier, the semantics associated with the ACTUATOR specifiers and ACTUATOR arguments is beyond the scope of this specification. The actuators are specified in 'Actuator Profile Specifications' such as StateLess Packet Filter Profile, Routing Profile etc.

3.2.1 OpenC2 Command

The OpenC2 Command describes an action performed on a target.

3.2.1.1 Type Name: OpenC2-Command

Base Type: Record

ID	Property Name	Type	Description
1	**action** (required)	Action	The task or activity to be performed (i.e., the 'verb')
2	**target** (required)	Target	The object of the action. The action is performed on the target.
3	**actuator** (optional)	Actuator	The subject of the action. The actuator executes the action on the target.
4	**args** (optional)	Args	An object containing additional properties that apply to the command
5	**id** (optional)	Command-ID	Identifier used to link responses to a command

> **Editor's Note** - In a future working draft, we may reformat these tables to include a cardinality column instead of the required/optional tags on the property names.

3.2.1.2 Type Name: Action

Base Type: Enumerated

ID	Property Name	Description
1	**scan**	Systematic examination of some aspect of the entity or its environment in order to obtain information.
2	**locate**	Find an object physically, logically, functionally, or by organization.
3	**query**	Initiate a request for information.
6	**deny**	Prevent a certain event or action from completion, such as preventing a flow from reaching a destination or preventing access.
7	**contain**	Isolate a file, process, or entity so that it cannot modify or access assets or processes.
8	**allow**	Permit access to or execution of a target.
9	**start**	Initiate a process, application, system, or activity.
10	**stop**	Halt a system or end an activity.
11	**restart**	Stop then start a system or an activity.
14	**cancel**	Invalidate a previously issued action.
15	**set**	Change a value, configuration, or state of a managed entity.
16	**update**	Instruct a component to retrieve, install, process, and operate in accordance with a software update, reconfiguration, or other update.
18	**redirect**	Change the flow of traffic to a destination other than its original destination.
19	**create**	Add a new entity of a known type (e.g., data, files, directories).
20	**delete**	Remove an entity (e.g., data, files, flows).
22	**detonate**	Execute and observe the behavior of a target (e.g., file, hyperlink) in an isolated environment.
23	**restore**	Return a system to a previously known state.
28	**copy**	Duplicate a file or data flow.
30	**investigate**	Task the recipient to aggregate and report information as it pertains to a security event or incident.
32	**remediate**	Task the recipient to eliminate a vulnerability or attack point.

The following actions are reserved for future use and are not valid actions In this version of the Language Specification.

- report - Task an entity to provide information to a designated recipient
- pause - Cease operation of a system or activity while maintaining state.
- resume - Start a system or activity from a paused state
- move - Change the location of a file, subnet, network, or process
- snapshot - Record and store the state of a target at an instant in time
- save - Commit data or system state to memory
- throttle - Adjust the rate of a process, function, or activity
- delay - Stop or hold up an activity or data transmittal

- substitute - Replace all or part of the payload
- sync - Synchronize a sensor or actuator with other system components
- mitigate - Task the recipient to circumvent a problem without necessarily eliminating the vulnerability or attack point

3.2.1.3 Type Name: Target

Base Type: Choice

ID	Property Name	Type	Description
1	**artifact**	Artifact	An array of bytes representing a file-like object or a link to that object.
2	**command**	Command-Id	A reference to a previously issued OpenC2 Command.
3	**device**	Device	The properties of a hardware device.
4	**directory**	Directory	The properties common to a file system directory.
7	**domain_name**	Domain-Name	A network domain name.
8	**email_addr**	Email-Addr	A single email address.
9	**email_message**	Email-Message	An instance of an email message, corresponding to the internet message format described in RFC 5322 and related RFCs.
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.
13	**mac_addr**	Mac-Addr	A single Media Access Control (MAC) address.
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	**openc2**	OpenC2	A set of items used with the query action to determine an actuator's capabilities.
17	**process**	Process	Common properties of an instance of a computer program as executed on an operating system.

25	**property**	Property	Data attribute associated with an actuator
18	**software**	Software	High-level properties associated with software, including software products.
19	**uri**	Uri	A uniform resource identifier(URI).
23	**windows_registry_key**	Windows-Registry-Key	The properties of a Windows registry key.
1024	**slpf**	slpf:Target	Target defined in the Stateless Packet Filter profile

The following targets are reserved for future use:

- disk
- disk_partition
- memory
- user_account
- user_session
- volume
- x509_certificate

3.2.1.4 Type Name: Actuator

Base Type: Choice

ID	Property Name	Type	Description
1	generic	Actuator_Specifiers	Generic actuator specifiers
1024	slpf	slpf:Specifiers	Actuator specifiers and options as defined in the Stateless Packet Filter profile, oasis-open.org/openc2/oc2ap-slpf/v1.0/csd01

> **Editor's Note** - The intent is to fill in this table with actuators as they are defined by the AP-SC. The AP-SC profiles will define the actuators and they will only be listed here. Once we have a lot of them (not an issue yet), we may figure out how to just put a reference here to a list maintained by the AP-SC.

> **Editor's Note** - The intent is to for the actuators to be extensible. Ie if a vendor has a function that is not yet in an AP-SC profile, the extensibility would be used to add this new function. The text to go here on how to do that is still under development

3.2.1.5 Type Name: Actuator_Specifiers

Base Type: Map

TBSL

3.2.1.6 Type Name: Args

Base Type: Map

ID	Property Name	Type	Description
1	**start_time** (optional)	Date-Time	The specific date/time to initiate the action
2	**stop_time** (optional)	Date-Time	The specific date/time to terminate the action
3	**duration** (optional)	Duration	The length of time for an action to be in effect
4	**response_requested** (optional)	Response-Type	The type of response required for the action
1024	**slpf (optional)**	slpf:Args	Command arguments defined in the Stateless Packet Filter profile

> **Editor's Note** - version will appear in the OpenC2 message header and in query responses for the OpenC2 version query

3.2.2 OpenC2 Response

3.2.2.1 Type Name: OpenC2-Response

Base Type: Record

ID	Property Name	Type	Description
1	**id** (optional)	Command-ID	ID of the response
5	**id_ref** (required)	Command-ID	ID of the command that induced this response.
2	**status** (required)	Status-Code	An integer status code
3	**status_text** (optional)	String	A free-form human-readable description of the response status
4	**results** (optional)	Results	Data or extended status information that was requested from an OpenC2 Command

Example:

```javascript
{
    "id_ref": "01076931758653239640628182951035",
    "status": 200,
    "status_text": "All endpoints successfully updated",
    "results": {
        "strings": ["wd-394", "sx-2497"]
    }
}
```
```

### 3.2.2.2 Type Name: Status-Code

Base Type: Enumerated

| Value | Description |
|---|---|
| 102 | **Processing** - an interim response used to inform the client that the server has accepted the request but has not yet completed it. |
| 200 | **OK** - the request has succeeded. |
| 301 | **Moved Permanently** - the target resource has been assigned a new permanent URI. |
| 400 | **Bad Request** - the server cannot process the request due to something that is perceived to be a client error (e.g., malformed request syntax). |
| 401 | **Unauthorized** - the request lacks valid authentication credentials for the target resource or authorization has been refused for the submitted credentials. |
| 403 | **Forbidden** - the server understood the request but refuses to authorize it. |
| 500 | **Server Error** - the server encountered an unexpected condition that prevented it from fulfilling the request. |
| 501 | **Not Implemented** - the server does not support the functionality required to fulfill the request. |

## 3.3 Property Details

### 3.3.1 Target Types

#### 3.3.1.1 Target Type: Artifact

Base Type: Record

| ID | Property Name | Type | Description |
|---|---|---|---|
| 1 | **mime_type** (optional) | String | Permitted values specified in the IANA Media Types registry, RFC 6838 |
| 2 | ***** (optional) | Payload | Choice of literal content or URL |
| 3 | **hashes** (optional) | Hashes | Hashes of the payload content |
```

3.3.1.2 Target Type: Command

TBSL

3.3.1.3 Target Type: Device

Base Type: Map

Property Name	Type	Description
hostname (optional)	Hostname	A hostname that can be used to connect to this device over a network
description (optional)	String	A human-readable description of the purpose, relevance, and/or properties of this device
device_id (optional)	String	An identifier that refers to this device within an inventory or management system

3.3.1.4 Target Type: Directory

TBSL

3.3.1.5 Target Type: Domain-Name

Type Name	Type	Description
Domain-Name	String	per RFC 1034

3.3.1.6 Target Type: Email-Addr

Type Name	Type	Description
Email-Addr	String	Email address, RFC 5322, section 3.4.1

3.3.1.7 Target Type: Email-Message

TBSL

3.3.1.8 Target Type: File

Base Type: Map

ID	Property Name	Type	Description
0	**name** (optional)	String	The name of the file as defined in the file system
1	**path** (optional)	String	The absolute path to the location of the file in the file system
2	**hashes** (optional)	Hashes	One or more cryptographic hash codes of the file contents

3.3.1.9 Target Type: IP-Addr

Type Name	Type	Description
IP-Addr	String	IPv4 or IPv6 address or range in CIDR notation. IPv4 address or range in CIDR notation, i.e., a dotted decimal format per RFC TBSL with optional CIDR prefix. IPv6 address or range in CIDR notation, i.e., colon notation per RFC 5952 with optional CIDR prefix

Examples:

- "192.168.10.11" - a single ipv4 address distinguishable because of the dots
- "192.168.10.11/32" - a single ipv4 address in CIDR notation
- "192.168.0.0/16" - a range of 65,536 ipv4 addresses in CIDR notation
- "2001:db8::1" - a single ipv6 address distinguishable because of the colons
- "2001:db8:aaaa:bbbb:cccc:dddd:0:1" - single ipv6 address
- "2001:db8::0/120" - 256 ipv6 addresses

Examples of invalid ipv6 (since violates RFC 5952):

- "2001:DB8::1" - lower case MUST be used
- "2001:db8:0:0:1:0:0:1" - the :: notation MUST be used for zero compression when possible
- "2001:db8::1:1:1:1:1" - the :: notation MUST NOT be used when only one zero is present

3.3.1.10 Target Type: Mac-Addr

TBSL

3.3.1.11 Target Type: IP-Connection

Base Type: Record

ID	Property Name	Type	Description
1	**src_addr**	IP-Addr	ip_addr of source, could be ipv4 or ipv6 - see ip_addr

			section
2	**src_port**	Port	source service per RFC TBSL
3	**dst_addr**	IP-Addr	ip_addr of destination, could be ipv4 or ipv6 - see ip_addr section
4	**dst_port**	Port	destination service per RFC TBSL
5	**protocol**	L4-Protocol	layer 4 protocol (e.g., TCP) - see l4_protocol section

3.3.1.12 Target Type: OpenC2

Base Type: ArrayOf(Query-Item)

3.3.1.13 Target Type: Process

Base Type: Map

Property Name	Type	Description
pid (optional)	Integer	Process ID of the process
name (optional)	String	Name of the process
cwd (optional)	String	Current working directory of the process
executable (optional)	File	Executable that was executed to start the process
parent (optional)	Process	Process that spawned this one
command_line (optional)	String	The full command line invocation used to start this process, including all arguments

3.3.1.14 Target Type: Property

Base Type: Record

Property Name	Type	Description
name (optional)	String	The name that uniquely identifies a property of an actuator.

query_string (optional)	String	A query string that identifies a single property of an actuator. The syntax of the query string is defined in the actuator profile.

3.3.1.15 Target Type: Software

TBSL

3.3.1.16 Target Type: Uri

TBSL

3.3.1.17 Target Type: Windows-Registry-Key

TBSL

3.3.1.18 Target Type: Slpf-Target

TBSL

3.3.2 Data Types

3.3.2.1 Type Name: Command-ID

Type Name	Type	Description
Command-ID	String	Uniquely identifies a particular command

3.3.2.2 Type Name: Hashes

Base Type: Map

Property Name	Type	Description
md5 (optional)	String	Hex-encoded MD5 hash as defined in RFC 1321
sha1 (optional)	String	Hex-encoded SHA1 hash as defined in RFC 6234
sha256 (optional)	String	Hex-encoded SHA256 hash as defined in RFC 6234

3.3.2.3 Type Name: Hostname

Type Name	Type	Description
Hostname	String	A legal Internet host name as specified in RFC 1123

3.3.2.4 Type Name: Identifier

Type Name	Type	Description
Identifier	string = command--UUIDv4	An identifier universally and uniquely identifies an OpenC2 command. Value SHOULD be a UUID generated according to RFC 4122.

3.3.2.5 Type Name: L4-Protocol

Value of the protocol (IPv4) or next header (IPv6) field in an IP packet. Any IANA value, RFC 5237

ID	Property Name	Description
1	**icmp**	Internet Control Message Protocol - RFC 792
6	**tcp**	Transmission Control Protocol - RFC 793
17	**udp**	User Datagram Protocol - RFC 768
132	**sctp**	Stream Control Transmission Protocol - RFC 4960

3.3.2.6 Type Name: Payload

Base Type: Choice

ID	Property Name	Type	Description
1	**payload_bin** (optional)	Binary	Specifies the data contained in the artifact
2	**url** (optional)	uri	MUST be a valid URL that resolves to the un-encoded content

3.3.2.7 Type Name: Port

Type Name	Type	Description
Port	String	Service Name or Transport Protocol Port Number, RFC 6335

3.3.2.8 Type Name: Query-Item

Base Type: Enumerated

Specifies the results to be returned from a query openc2 command.

ID	Property Name	Description
1	**versions**	List of OpenC2 Language versions supported by this actuator
2	**profiles**	List of profiles supported by this actuator
3	**schema**	Definition of the command syntax supported by this actuator
4	**pairs**	List of supported actions and applicable targets

3.3.2.9 Type Name: Response-Type

Base Type: Enumerated

ID	Name	Description
0	**none**	No response
1	**ack**	Respond when command received
2	**status**	Respond with progress toward command completion
3	**complete**	Respond when all aspects of command completed

> **Editor's Note** - Use cases are needed for the different types of responses needed.

3.3.2.10 Type Name: Version

Type Name	Type	Description
Version	String	TBSL

> **Editor's Note** - version will appear in the OpenC2 message header and in query responses for the OpenC2 version query"

3.3.2.11 Type Name: Results

Base Type: Map

ID	Name	Type	#	Description
1	**strings**	String	0..n	Generic set of string values
2	**ints**	Integer	0..n	Generic set of integer values

3	**kvps**	KVP	0..n	Generic set of key:value pairs
4	**versions**	Version	0..n	The list of OpenC2 language versions supported by this actuator
5	**profiles**	Uname	0..n	The list of profiles supported by this actuator
6	**schema**	Schema	0..n	Syntax of the OpenC2 language elements supported by this actuator
7	**actions**	ActionTargets	0..n	List of actions and their supported targets
1024	**slpf**	slpf:Results	0..1	Response data defined in the Stateless Packet Filtering Firewall profile

3.3.2.12 Type Name: KVP

Base Type: Array

ID	Type	#	Description
1	Identifier	1	"key": name of this item
2	String	1	"value": string value of this item

3.3.2.13 Type Name: ActionTargets

Base Type: Array

ID	Type	#	Description
1	Action	1	An action supported by this actuator.
2	Target.*	1..n	List of targets applicable to this action. The targets are enumerated values derived from the set of Target types.

> **Editor's Note**: to be moved elsewhere.
> Example from SLPF Profile Table 2.3-1 - Command Matrix:

Command:

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

Response:

```
{
```

Copyright © OASIS Open 2018. All Rights Reserved.

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

3.3.3 Schema Syntax

3.3.3.1 Type Name: Schema

Base Type: Record

ID	Name	Type	#	Description
1	**meta**	Meta	1	Information about this schema module
2	**types**	Type	1..n	Types defined in this schema module

3.3.3.2 Type Name: Meta

Meta-information about this schema

Base Type: Map

ID	Property Name	Type	Description
1	**module** (required)	Uname	Unique name
2	**title** (optional)	String	Title
3	**version** (optional)	Version	Module version
4	**description** (optional)	String	Description
5	**imports** (optional)	ArrayOf(Import)	Imported modules
6	**exports** (optional)	ArrayOf(Identifier)	Data types exported by this module
7	**bounds** (optional)	Bounds	Schema-wide upper bounds

ID	Property Name	Type	#	Description
1	**module**	Uname	1	Unique name
2	**title**	String	0..1	Title
3	**version**	String	0..1	Patch version (module includes major.minor version)
4	**description**	String	0..1	Description

5	**imports**	Import	0..n	Imported schema modules
6	**exports**	Identifier	0..n	Data types exported by this module
7	**bounds**	Bounds	0..1	Schema-wide upper bounds

3.3.3.3 Type Name: Import

Base Type: Array

ID	Type	#	Description
1	Nsid	1	"nsid": A short local identifier (namespace id) used within this module to refer to the imported module
2	Uname	1	"uname": Unique name of the imported module

3.3.3.4 Type Name: Bounds

Schema-wide default upper bounds. If included in a schema, these values override codec default values but are limited to the codec hard upper bounds. Sizes provided in individual type definitions override these defaults.

Base Type: Array

ID	Type	#	Description
1	Integer	1	"max_msg": Maximum serialized message size in octets or characters
2	Integer	1	"max_str": Maximum text string length in characters
3	Integer	1	"max_bin": Maximum binary string length in octets
4	Integer	1	"max_fields": Maximum number of elements in ArrayOf

3.3.3.5 Type Name: Type

Definition of a data type.

Base Type: Array

ID	Type	#	Description
1	Identifier	1	"tname": Name of this data type
2	JADN-Type.*	1	"btype": Base type. Enumerated value derived from the list of JADN data types.
3	Option	1..n	"opts": Type options
4	String	1	"tdesc": Description of this data type
5	JADN-Type.&2	1..n	"fields": List of fields for compound types. Not present for primitive types.

3.3.3.6 Type Name: JADN-Type

Field definitions applicable to the built-in data types (primitive and compound) used to construct a schema.

Base Type: Choice

ID	Name	Type	Description
1	Binary	Null	Octet (binary) string
2	Boolean	Null	True or False
3	Integer	Null	Whole number
4	Number	Null	Real number
5	Null	Null	Nothing
6	String	Null	Character (text) string
7	Array	FullField	Ordered list of unnamed fields
8	ArrayOf	Null	Ordered list of fields of a specified type
9	Choice	FullField	One of a set of named fields
10	Enumerated	EnumField	One of a set of id:name pairs
11	Map	FullField	Unordered set of named fields
12	Record	FullField	Ordered list of named fields

3.3.3.7 Type Name: Enum-Field

Item definition for Enumerated types

Base Type: Array

ID	Type	#	Description
1	Integer	1	Item ID
2	Identifier	1	Item name
3	String	1	Item description

3.3.3.8 Type Name: Full-Field

Field definition for compound types Array, Choice, Map, Record

Base Type: Array

ID	Type	#	Description
1	Integer	1	Field ID or ordinal position
2	Identifier	1	Field name

3	Identifier	1	Field type
4	Option	0..n	Field options
5	String	1	Field description

3.3.3.9 Type Name: Identifier

Base Type: String

A string beginning with an alpha character followed by zero or more alphanumeric | underscore | dash characters, max length 32 characters

3.3.3.10 Type Name: Nsid

Base Type: String

Namespace ID - a short identifier, max length 8 characters

3.3.3.11 Type Name: Uname

Base Type: String

Unique name (e.g., of a schema) - typically a set of Identifiers separated by forward slashes

3.3.3.12 Type Name: Option

Base Type: String

An option string, minimum length = 1. The first character is the option id. Remaining characters if any are the option value.

4 Core Actuator Profile

> **Editor's Note** - TBSL - This section be included in a future iteration (probably iteration 5) prior to submitting for Committee Specification.
>
> This section defines the core functions applicable to every OpenC2 actuator.
>
> Command and resulting response:
> * One action: query
> * One target: openc2
> * Target specifiers: versions, profiles, schema

5 Conformance

OpenC2 is a command and control language that converges (i.e., common 'point of understanding') on a common syntax, and lexicon. The tables in Section 3 of this document specify the normative rules for determining if an OpenC2 message (command or response) is syntactically valid. All examples in this document are informative; in case of conflict between the tables and an example, the tables are authoritative. Conformant implementations of OpenC2:

A. MUST produce messages that are syntactically valid.
B. SHOULD reject messages that are syntactically invalid.
C. MUST implement the actions designated as mandatory in this document.
D. MUST implement the targets designated as mandatory in this document.
E. MAY implement optional targets defined in this document
F. MAY implement actuator specifiers, target specifiers and/or args as specified in one or more Actuator Profiles.
G. MUST implement JSON serialization of the commands and responses that are consistent with the syntax defined in this document.

> **Editor's Note** - TBSL - More conformance text will be included in a future iteration (probably the next) prior to submitting for Committee Specification.

6 Appendix A. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

> **Editor's Note** - TBSL - This section be included in the final iteration prior to submitting for Committee Specification. The proposal is to include on the list the names of all members of the Language Subcommittee who made contributions to the document (defined very liberally as anyone who either attended a meeting, or sent a contributing email, or contributed text), and all members of the OpenC2 Language Subcommittee that voted on at least one of the drafts

7 Appendix B. Revision History

Revision	Date	Editor	Changes Made
v1.0-wd01	10/31/2017	Romano, Sparrell	Initial working draft
v1.0-csd01	11/14/2017	Romano, Sparrell	approved wd01
v1.0-wd02	01/12/2018	Romano, Sparrell	csd01 ballot comments targets
v1.0-wd03	01/31/2018	Romano, Sparrell	wd02 review comments
v1.0-csd02	02/14/2018	Romano, Sparrell	approved wd03
v1.0-wd04	03/02/2018	Romano, Sparrell	Property tables threads (cmd/resp) from use cases previous comments
v1.0-wd05	03/21/2018	Romano, Sparrell	wd04 review comments
v1.0-csd03	04/03/2018	Romano, Sparrell	approved wd05
v1.0-wd06	05/15/2018	Romano, Sparrell	Finalizing message structure message=header+body Review comments Using word ‘arguments’ instead of ‘options’
v1.0-csd04	5/31/2018	Romano, Sparrell	approved wd06
v1.0-wd07	7/11/2018	Romano, Sparrell	Continued refinement of details Review comments Moved some actions and targets to reserved lists

8 Appendix C. Acronyms

> **Editor's Note** - TBSL - This section be included in the final iteration prior to submitting for Committee Specification.

9 Annex 1. Examples

> **Editor's Note** - TBSL - This section will be populated with examples of json command and responses. The intent is to have each example serve multiple purposes (e.g., one example shows action=allow, command option=start_time, target=....) and then could be referenced with footnotes from several places in spec. This original draft was quite long due to all the inline examples and this is hoped to be a reasonable compromise

9.1 Example 1

> **Editor's Note** - This example shows the structure of an OpenC2 Message containing a `header` and a `body`. This example is for a transport where the header is included in the JSON (eg STIX).

9.1.1 OpenC2 Message

```
{
    "header": {
        "version": "1.0",
        "created": "2018-01-30T18:25:43.511Z"
    },
    "command": {
        "id": "9d43df98-7e34-43d3-bb25-4d1ea7a0a02a",
        "action": "redirect",
        "target": {
            "url": "http://evil.com"
        },
        "args": {
            "destination": "http://newdest.com/home"
        }
    }
}
```

9.2 Example 2

This example is for a transport where the header information is outside the JSON (eg HTTPS API) and only body is in JSON.

```
{
    "id": "3cf4df44-1fbb-4b40-936c-b6139000d9d4",
    "action": "allow",
```

~~~
    "target": {
        "ip_addr": "1.2.3.4"
    },
    "args" {
        "start_time": "now",
        "response_requested": "ack"
    }
}
```
~~~

9.3 Example 3

This example shows the OpenC2 Command and Response for retrieving data from an actuator.

9.3.1 OpenC2 Command

~~~
```
{
    "id": "71be3c32-188f-476d-9b20-35cb4eb60e52",
    "action": "query",
    "target": {
        "property": {
            "name": "battery_percentage"
        }
    },
    "actuator": {
        "endpoint_smart_meter": {
            "actuator_id": "TSLA-00101111",
            "asset_id": "TGEadsasd"
        }
    }
}
```
~~~

9.3.2 OpenC2 Response

~~~
```
{
    "id_ref": "71be3c32-188f-476d-9b20-35cb4eb60e52",
    "status": 200,
    "results": {
        "battery_percentage": 0.577216
    }
}
```
~~~