



Specification for Transfer of OpenC2 Messages via MQTT Version 1.0

Committee Specification Draft 01

07 July 2020

This version:

<https://docs.oasis-open.org/openc2/transf-mqtt/v1.0/csd01/transf-mqtt-v1.0-csd01.md>

(Authoritative)

<https://docs.oasis-open.org/openc2/transf-mqtt/v1.0/csd01/transf-mqtt-v1.0-csd01.html>

<https://docs.oasis-open.org/openc2/transf-mqtt/v1.0/csd01/transf-mqtt-v1.0-csd01.pdf>

Previous version:

N/A

Latest version:

<https://docs.oasis-open.org/openc2/transf-mqtt/v1.0/transf-mqtt-v1.0.md> (Authoritative)

<https://docs.oasis-open.org/openc2/transf-mqtt/v1.0/transf-mqtt-v1.0.html>

<https://docs.oasis-open.org/openc2/transf-mqtt/v1.0/transf-mqtt-v1.0.pdf>

Technical Committee:

[OASIS Open Command and Control \(OpenC2\) TC](#)

Chairs:

Joe Brule (jmbrule@radium.ncsc.mil), [National Security Agency](#)

Duncan Sparrell (duncan@sfractal.com), [sFractal Consulting](#)

Editors:

Joe Brule (jmbrule@radium.ncsc.mil), [National Security Agency](#)

Danny Martinez (danny.martinez@hii-tsd.com), [G2, Inc.](#)

David Lemire (david.lemire@hii-tsd.com), [National Security Agency](#)

Related work:

This specification is related to:

- *Open Command and Control (OpenC2) Specification for Transfer of OpenC2 Messages via HTTPS Version 1.0*. Edited by David Lemire. Latest version: <https://docs.oasis-open.org/openc2/open-impl-https/v1.0/open-impl-https-v1.0.html>.

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. Message Queuing Telemetry Transport (MQTT) is a widely used publish / subscribe (pub/sub) transfer protocol. This specification describes the use of MQTT as a transfer mechanism for OpenC2 messages.

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 specification 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-MQTT-v1.0]

Specification for Transfer of OpenC2 Messages via MQTT Version 1.0. Edited by Joe Brule, Danny Martinez, and David Lemire. 07 July 2020. OASIS Committee Specification Draft 01. <https://docs.oasis-open.org/openc2/transf-mqtt/v1.0/csd01/transf-mqtt-v1.0-csd01.html>. Latest version: <https://docs.oasis-open.org/openc2/transf-mqtt/v1.0/transf-mqtt-v1.0.html>.

Notices

Copyright © OASIS Open 2020. 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.

As stated in the OASIS IPR Policy, the following three paragraphs in brackets apply to OASIS Standards Final Deliverable documents (Committee Specification, Candidate OASIS Standard, OASIS Standard, or Approved Errata).

[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 Standards Final Deliverable, 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 deliverable.]

[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 OASIS Standards Final Deliverable 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 OASIS Standards Final Deliverable. 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 OASIS Standards Final Deliverable 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 Standards Final Deliverable, 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

[1 Introduction](#)

[1.1 IPR Policy](#)

[1.2 Normative References](#)

[1.3 Non-Normative References](#)

[1.4 Terminology](#)

[1.5 Document Conventions](#)

[1.5.1 Naming Conventions](#)

[1.5.2 Font Colors and Style](#)

[1.6 Overview](#)

[1.7 Goal](#)

[2 Operating Model](#)

[2.1 Publishers, Subscribers, and Brokers](#)

[2.2 Default Topic Structure](#)

[2.3 Message Format](#)

[2.4 Quality of Service](#)

[2.5 MQTT Client Identifier](#)

[2.6 Keep-Alive Interval](#)

[3 Protocol Mappings](#)

[4 Security Considerations](#)

[5 Conformance](#)

[Appendix A. Message Examples](#)

[Appendix B. Acknowledgments](#)

[Appendix C. Revision History](#)

1 Introduction

This section is non-normative.

NOTE: The content of Section 1 is currently a direct copy-and-paste from previous OpenC2 specifications. It is anticipated that this section will be greatly abbreviated once the relevant material is captured in the *OpenC2 Architecture Specification*. Relevant content for reviewer is currently in [Section 2](#) and [Appendix A](#).

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.

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 specification 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>).

1.2 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., ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <http://www.rfc-editor.org/info/rfc8259>

[OpenC2-Lang-v1.0]

Open Command and Control (OpenC2) Language Specification Version 1.0. Edited by Jason Romano and Duncan Sparrell. Latest version: <http://docs.oasis-open.org/openc2/oc2ls/v1.0/oc2ls-v1.0.html>.

[mqtt-v3.1.1]

MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS Standard. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>. Latest version: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>.

1.3 Non-Normative References

[RFC3552]

Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[IACD]

M. J. Herring, K. D. Willett, "Active Cyber Defense: A Vision for Real-Time Cyber Defense," *Journal of Information Warfare*, vol. 13, Issue 2, p. 80, April 2014.

Willett, Keith D., "Integrated Adaptive Cyberspace Defense: Secure Orchestration", *International Command and Control Research and Technology Symposium*, June 2015.

[Sparkplug-B]

Eclipse Foundation, "Sparkplug (TM) MQTT Topic & Payload Definition", Version 2.2, October 2019, <https://www.eclipse.org/tahu/spec/Sparkplug%20Topic%20Namespace%20and%20State%20Management%20appendix%20B%20format%20-%20Eclipse.pdf>

1.4 Terminology

- **Action:** The task or activity to be performed (e.g., 'deny').
- **Actuator:** The entity that performs the action (e.g., 'Stateless Packet Filtering').
- **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 (e.g., IP Address).

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 [\[RFC2119\]](#) [\[RFC8174\]](#) when, and only when, they appear in all capitals, as shown here.

A list of acronyms is provided in [Annex A](#).

1.5 Document Conventions

1.5.1 Naming Conventions

- [\[RFC2119\]](#)/[\[RFC8174\]](#) key words (see section 1.2) 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:

```
PUT AN EXAMPLE HERE
```

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 specifications focus on the language content and meaning at the producer and consumer of the command and response while the transfer specifications focus on the protocols for their exchange.

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 OpenC2 Language Specification [[OpenC2-Lang-v1.0](#)] 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. Actuator Profiles may define command arguments and targets 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).

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 sent back to the OpenC2 Producer as a result of the command.

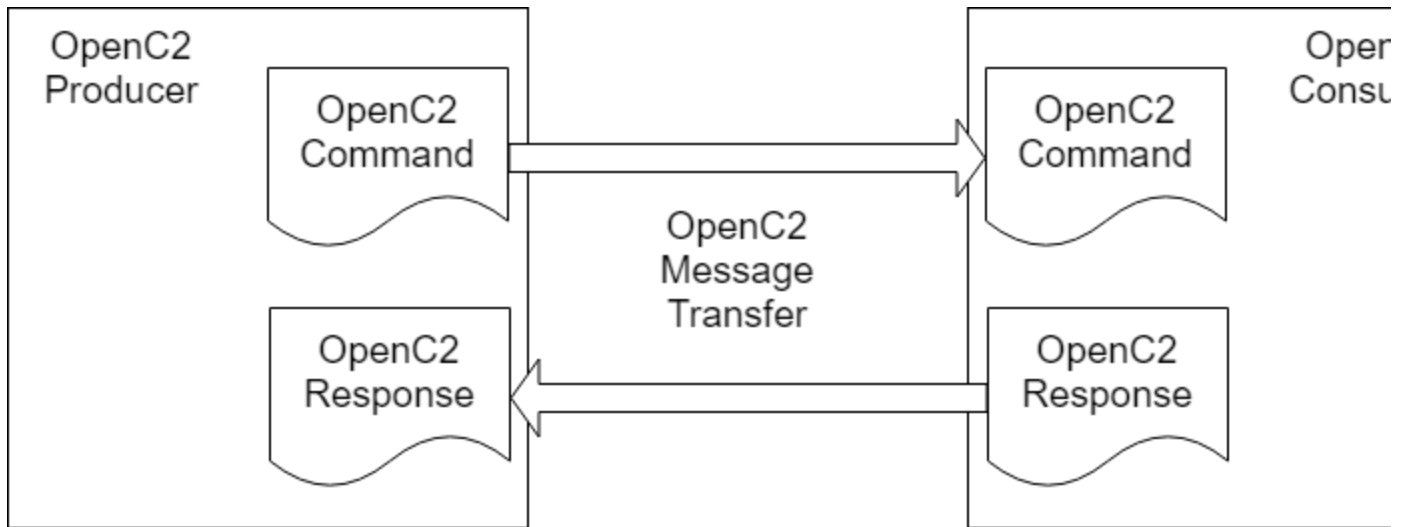


Figure 1-1. OpenC2 Message Exchange

OpenC2 implementations integrate the related OpenC2 specifications described above with related industry specifications, protocols, and standards. Figure 1-2 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 any particular approach to implementing the needed functionality (for example, the use of an application-layer message signature function to provide message source authentication and integrity).

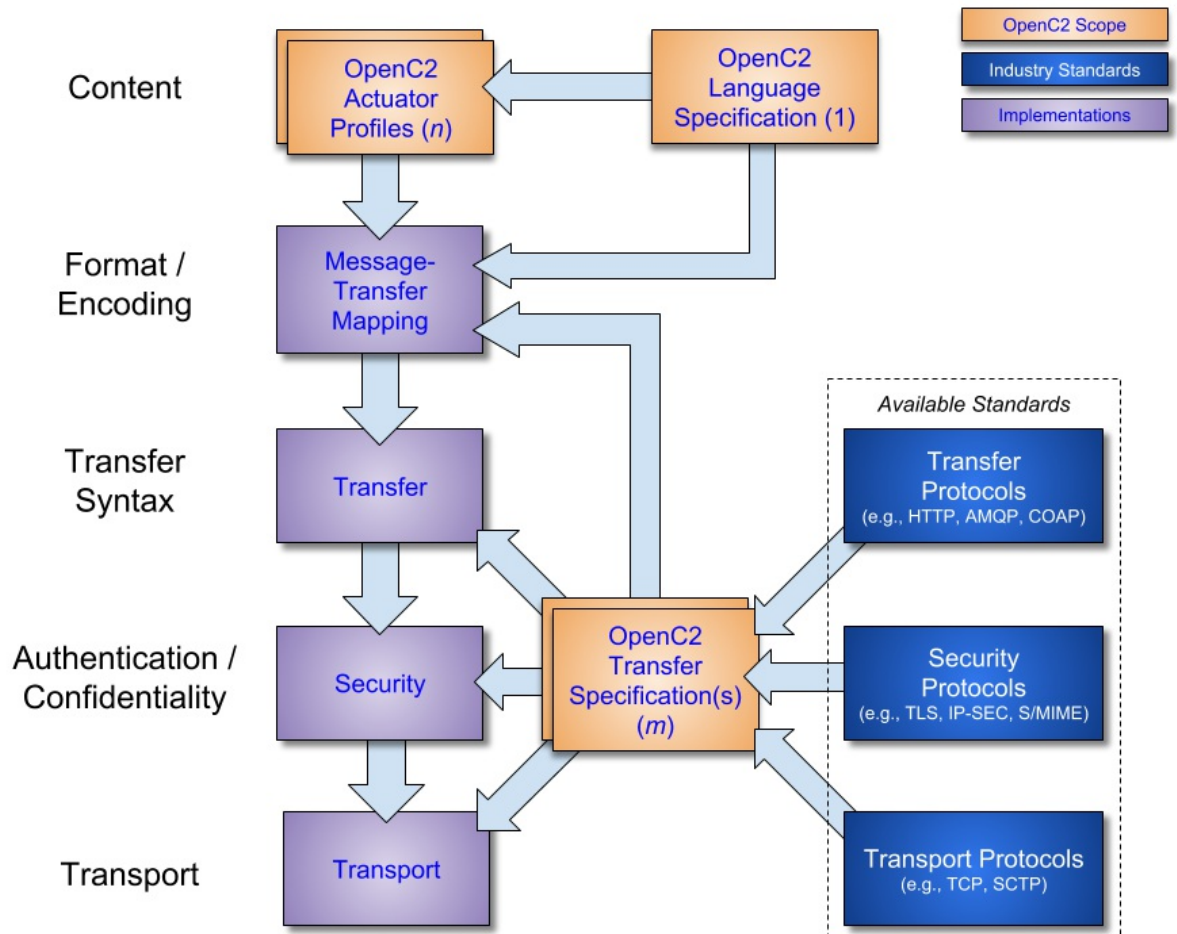


Figure 1-2. OpenC2 Documentation and Layering Model

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	OpenC2 Language Specification
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 implementation 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.

2 Operating Model

This section is non-normative.

This section provides an overview of the approach to employing MQTT as a message transfer protocol for OpenC2 messages.

NOTE: Tentative list of Qs the MQTT Transfer Spec should answer; feedback on additional questions or questions that might be out-of-scope / SEP (someone else's problem) is welcome. As consensus is developed on each aspect of the operating model, the corresponding question(s) should be deleted.

- What is the required interoperable topic structure?
 - A proposal is contained in [2.2 Default Topic Structure](#).
- Is OpenC2 going to use the MQTT Will feature? If so, what should be used for the will topic(s)?
 - Should be addressed in Section 2.2 once resolved.
- What is the OpenC2 message format over MQTT?
 - See [Section 2.3](#)
- Are there any special requirements for the MQTT ClientId?
 - See [Section 2.5](#); a proposal for ClientId assignment is TBD.
- How does a Producer discover the active consumers in a pub/subs space?
- How does a Producer discover the capabilities of active consumers in a pub/sub

space?

- The above two questions have an element of *registration* (making Consumers known to the Producer) vs. *discovery* (enabling the Producer to know what Consumers are currently active in the Producer's sphere of control).
 - *Proposed*: Discovery as defined above is an appropriate topic for a transfer specification, registration is outside the scope of a transfer specification
 - *Proposed*: Determination of actuator capabilities is outside the scope of a transfer specification, but a transfer specification might facilitate use of the OpenC2 Language's features to make such determination (details TBD)
-
- What is the appropriate QoS for MQTT messaging for OpenC2?
 - See [Section 2.4](#).

 - Should Consumers publish any kind of birth and/or death messages?
 - MQTT includes a "last will" mechanism to provide information when a device is disconnected
 - The [Sparkplug B specification](#) defines a birth certificate mechanism to provide information when devices become connected.
 - The operating model should address whether and how OpenC2 should leverage either of those capabilities.

 - Should we recommend a maximum keep-alive interval?
 - [Section 2.6](#) proposes an approach

 - Do we need to describe the nature / structure of the Consumer Device / Actuator(s)?
 - The in-development [Architecture Specification](#) is the appropriate location for this information; transfer specifications should reference the architecture, once it's published.

 - Is there a need to describe a state model for the Producer or Consumer?

2.1 Publishers, Subscribers, and Brokers

When transferring OpenC2 Command and Response messages via MQTT, both Producers and Consumers act as both publishers and subscribers:

- Producers publish Commands and subscribe to receive Responses
- Consumers subscribe to receive Commands and publish Responses

The MQTT broker and MQTT client software used by Producers and Consumers are beyond the scope of this specification, but are assumed to be conformant with the MQTT v3.1.1 specification [MQTT-V3.1.1]. In the content of OpenC2, and in accordance with the Terminology section (1.2) of [MQTT-V3.1.1]:

- MQTT Brokers are Servers
- OpenC2 Producers and Consumer are Clients

2.2 Default Topic Structure

NOTE: a brief Slack discussion on this proposed topic structure can be found [here](#).

The MQTT topic structure below is used to exchange OpenC2 messages. The "oc2" prefix on the topic names segregates OpenC2-related topics from other topics that might exist on the same broker. Topic name components in brackets (e.g., [actuator_profile]) are placeholders for specific values that would be used in implementation. For example, a device that includes a Stateless Packet Filter AP would subscribe to `oc2/cmd/ap/slpf`.

NOTE: a point for consideration is whether to use abbreviations (e.g., `dt` for `device_type`) to shorten the topic names. If we adopt v5.0 instead of v3.1.1, the option to use integer "topic aliases" is also available.

Topic	Purpose	Producer	Consumer
<code>oc2/cmd/ap/[actuator_profile]</code>	Used to send OpenC2 commands to all instances of specified Actuator Profile.	Pub	Sub

Standards Track Work Product

Topic	Purpose	Producer	Consumer
<code>oc2/cmd/device_type/[device_type]</code>	<p>Used to send OpenC2 commands to all instances of a particular device type. It is assumed that a device of a given type may support multiple APs, and that all devices of the same type support the same set of APs.</p>	<p>Pub</p>	<p>Sub</p>
<code>oc2/cmd/device_id/[device_id]</code>	<p>Used to send OpenC2 commands to all APs within a specific device.</p>	<p>Pub</p>	<p>Sub</p>

Topic	Purpose	Producer	Consumer
<code>oc2/cmd/action_target/[action_target]</code>	Used to send commands to all devices and/or actuators that implement the specified command (i.e., action-target pair)	Pub	Sub
<code>oc2/cmd/action/[action]</code>	Used to send OpenC2 commands to all devices and/or actuators that implement the specified action.	Pub	Sub
<code>oc2/rsp</code>	Used to return OpenC2 response messages.	Sub	Pub

In order to receive commands intended for its security functions, a Consumer device registering with the broker would subscribe to:

- `oc2/cmd/ap/[acutator_profile]` for all actuator profiles the device implements
- `oc2/cmd/device_type/[device_type]` for that device's TYPE
- `oc2/cmd/device_id/[device_id]` for that device's ID

- `oc2/cmd/action_target/[action_target]` for all action-target pairs in the union set of actuator profiles the device implements
- `oc2/cmd/action/[action]` for all actions in the union set of actuator profiles the device implements

In order to receive responses to the commands it sends, a Producer registering with the broker would subscribe to:

- `oc2/rsp`

NOTE (from Duncan Sparrell on Slack): I think a lot of this depends on our model of APs within a 'device' (which may be in a 'device') and what operates at which level (AP/ inner device/outer device) which we haven't discussed much. And I think that discussion depends on the 'lots of little atomic APs' or 'fewer compound APs with optional pieceparts' (which BTW I'll argue is just the lots of little atomic with an added layer). I think the pub/sub discussion "informs" the atomic/compound AP discussion but I also think reality of today's tech informs the discussion and we should look at how real world products work today

2.3 Message Format

NOTE: The format proposed by Dave Kemp in [Language Spec issue #353](#), or similar, seems appropriate for use with pub/sub protocols. It encapsulates all of the needed information. This draft MQTT Transfer Specification anticipates the adoption of this message format and utilizes its structure.

OpenC2 messages transferred using MQTT utilize the `OpenC2-Message` structure containing the message elements listed in Section 3.2 of [OpenC2-Lang-v1.0](#).

```
OpenC2-Message = Record {
  1 content          Content,           // Message body
  as specified by msg_type (the ID/Name of Content)
  2 request_id      String optional,    // A unique
  identifier created by Producer and copied by Consumer into
  responses
  3 created         Date-Time optional,  // Creation
  date/time of the content
  4 from            String optional,     // Authenticated
  identifier of the creator of / authority for a request
  5 to              ArrayOf(String) optional // Authenticated
  identifier(s) of the authorized recipient(s) of a message
}
```

```
Content = Choice {
```

```

1 request      OpenC2-Command,          // The initiator
of a two-way message exchange.
2 response    OpenC2-Response,         // A response
linked to a request in a two-way message exchange.
3 notification OpenC2-Notification     // A (one-way)
message that is not a request or response. (Placeholder)
}

```

A Producer sending an OpenC2 Command includes its identifier in the message `from` field, allowing Consumers receiving the command to know its origin. A Consumer sending a response to an OpenC2 command includes its identifier in the message `from` field, allowing responses from different actuators to be identified by the Producer receiving the response.

The `to` field is not utilized, as the MQTT Topic Structure and Client subscriptions regulate which recipients receive each individual message.

The `request_id` field can contain any string; UUIDv4 format is recommended for request IDs.

Note: The selection of the IMF-fixdate format is a carryover from the HTTPS Transfer Spec. There may be date formats more suitable for use with MQTT.

The `created` field is populated with the date/time when the message was created, in the preferred IMF-fixdate format as defined by Section 7.1.1.1 of RFC 7231; the conditions for populating the Date: header specified in Section 7.1.1.2 of RFC 7231 SHALL be followed.

2.4 Quality of Service

[mqtt-v3.1.1](#) Section 4.3, *Quality of Service Levels and Protocol Flows* defines three quality of service (QoS) levels:

- **QoS 0: "At most once"**, where messages are delivered according to the best efforts of the operating environment. Message loss can occur. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
- **QoS 1: "At least once"**, where messages are assured to arrive but duplicates can occur.
- **QoS 2: "Exactly once"**, where message are assured to arrive exactly once. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.

QoS 1 is appropriate for most OpenC2 applications and should be specified as the default. Implementers have the option of electing to use QoS 2 where the additional overhead is justified by application requirements. QoS 0 is not recommended for use in OpenC2 messaging.

2.5 MQTT Client Identifier

As described in [mqtt-v3.1.1](#) Section 3.1, *CONNECT* – *Client requests a connection to a Server*, the Client Identifier (ClientId) is a required field in the CONNECT control packet. Further requirements are contained in Section 3.1.3.1, *Client Identifier*, which defines the ClientId as a UTF-8 string containing only letters and numbers of between 1 and 23 bytes (MQTT servers may accept longer ClientIds). [mqtt-v3.1.1](#) provides no further definition regarding the format or assignment of ClientIds.

NOTE: the approach for creating ClientIds for OpenC2 MQTT clients is TBD.

2.6 Keep-Alive Interval

[mqtt-v3.1.1](#) section 3.1.2.10 provides a keep alive feature where a Client connected to a Broker must send either a Control Packet or a PINGREQ to the broker before a specified time interval has elapsed to prevent the Broker from disconnecting from the Client. The specification notes that "The actual value of the Keep Alive is application specific; typically this is a few minutes. The maximum value is 18 hours 12 minutes and 15 seconds."

This transfer specification leaves the selection of a keep alive interval to the implementer but defines a value of 5 minutes (300 seconds) as the maximum value for conformant implementations. For reliability, an OpenC2 client should send an MQTT PINGREQ when 95% of the Keep Alive interval has expired without any other control packets being exchanged.

3 Protocol Mappings

TBSL The protocol mappings will be specified once consensus has been achieved on the operating model.

4 Security Considerations

- Bare minimum requirement for operational instance should be use of TLS 1.2 or higher for operational client-broker connections. Basically, extract and use the TLS guidance from the v1.0 HTTPS Transfer CS.
- Unsecured MQTT should only be used for testing purposes.

(Note: OASIS strongly recommends that Technical Committees consider issues that could affect security when implementing their specification and document them for implementers and adopters. For some purposes, you may find it required, e.g. if you apply for IANA registration.

While it may not be immediately obvious how your specification might make systems

vulnerable to attack, most specifications, because they involve communications between systems, message formats, or system settings, open potential channels for exploit. For example, IETF [[RFC3552](#)] lists “eavesdropping, replay, message insertion, deletion, modification, and man-in-the-middle” as well as potential denial of service attacks as threats that must be considered and, if appropriate, addressed in IETF RFCs.

In addition to considering and describing foreseeable risks, this section should include guidance on how implementers and adopters can protect against these risks.

We encourage editors and TC members concerned with this subject to read *Guidelines for Writing RFC Text on Security Considerations*, IETF [[RFC3552](#)], for more information.

Remove this note before submitting for publication.)

5 Conformance

TBSL Conformance requirements will be developed once the protocol mappings have been developed.

(Note: The [OASIS TC Process](#) requires that a specification approved by the TC at the Committee Specification Public Review Draft, Committee Specification or OASIS Standard level must include a separate section, listing a set of numbered conformance clauses, to which any implementation of the specification must adhere in order to claim conformance to the specification (or any optional portion thereof). This is done by listing the conformance clauses here. For the definition of "conformance clause," see [OASIS Defined Terms](#).

See "Guidelines to Writing Conformance Clauses":
<http://docs.oasis-open.org/templates/TCHandbook/ConformanceGuidelines.html>.

Remove this note before submitting for publication.)

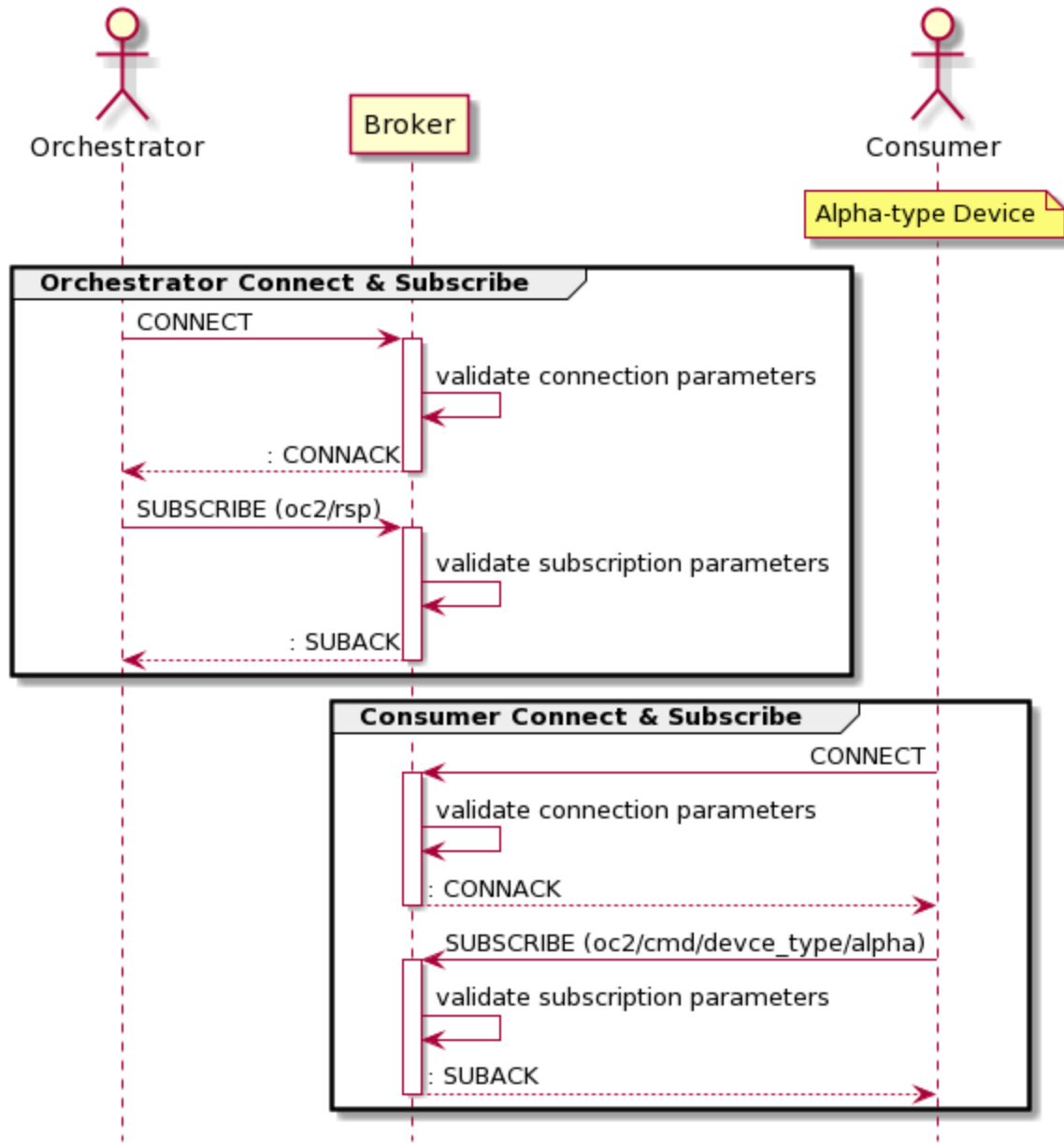
Appendix A: Message Examples

NOTE: Example message creation and presentation format are work-in-progress and two alternative representations as currently provided. The editors would welcome suggestions for the most useful presentation format.

A.1 Example 1: Connect and Subscribe

The following diagram illustrates the process of the Orchestrator and a Consumer each connecting to the MQTT broker and subscribing to a relevant channel.

Connect and Subscribe



Example CONNECT packed fields and values.

Region	Field	Value
FH	Type	CONNECT
FH	Remaining Length	
VH	Protocol Name - Length	4

Region	Field	Value
VH	Protocol Name - Value	MQTT
VH	Protocol Level	4
VH	Connect Flags (bitmap)	
	Clean Session	TBD
	Will Flag	TBD
	Will QoS	TBD
	Will Retain	TBD
	User Name Flag	TBD
	Password Flag	TBD
VH	Keep Alive	Number < 300 (seconds)
PL	Client Identifier	
PL	Will Topic	TBD string
PL	Will Message	TBD string
PL	Username	TBD
PL	Password	TBDS

NOTE: Further example messages to-be-supplied

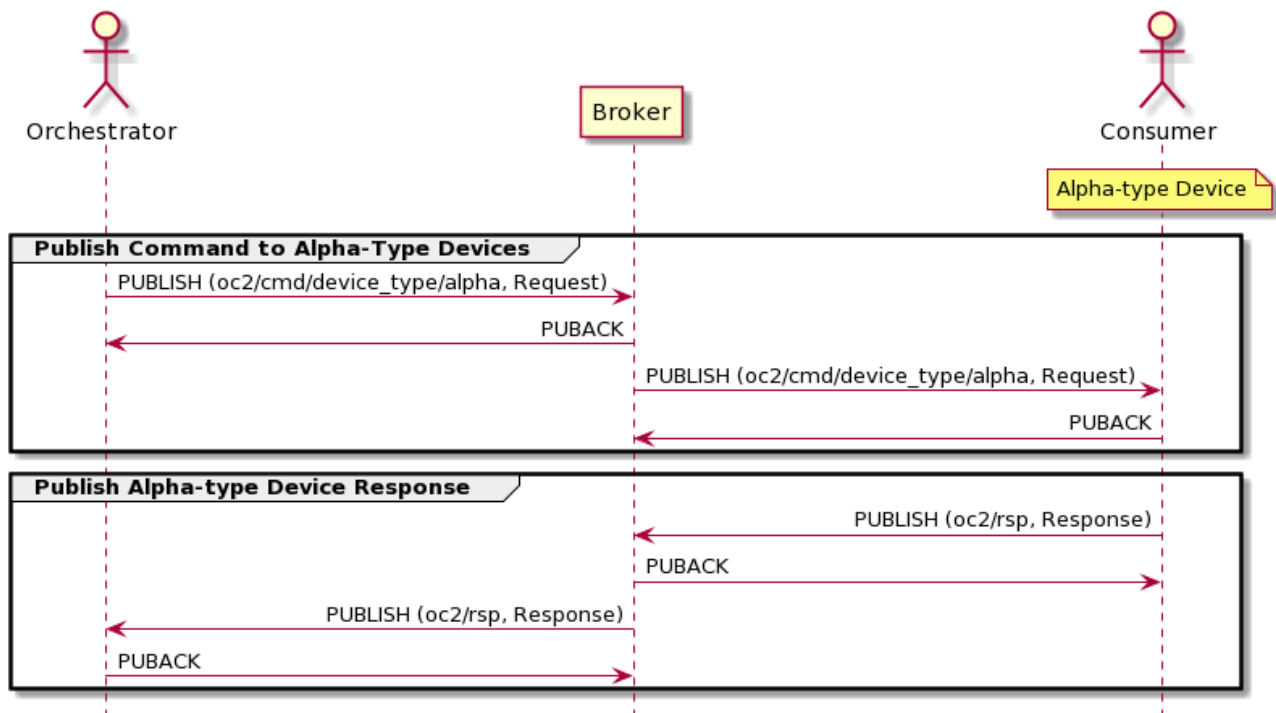
A.2 Example 2: Command / Response Exchange

The example messages in A.2.1 and A.2.2 illustrate the process of an OpenC2 Producer publishing a command to a channel for a specific device type, `oc2/cmd/device_type/alpha`, with Quality of Service level 1. A similar exchange would then occur between the broker and every device subscribed to `oc2/cmd/device_type/alpha` to distribute the command to the intended recipients. The examples assume a notional device type named "Alpha" exists and that one or more devices of that types are subscribed to the appropriate `device_type` channel.

The response message in the sequence diagram below is published with a QoS of 1, which requires the broker to respond to the PUBLISH packet with a PUBACK packet. If response

messages are sent with QoS of 0 no reply from the broker would be required.

Request, and Response



A.2.1: Orchestrator PUBLISHes a Command to All Devices of Type "alpha"

NOTE: This example shows the required information for the MQTT PUBLISH message, but the presentation needs fine tuning / verification. Two different approaches are shown for the first example MQTT Control Packet (PUBLISH).

Bullet-list representation of control packet

Fixed Header

- Type: PUBLISH
- Dup: 0
- QoS: 1
- Retain: 0
- Remaining Length: [computed]

Variable Header

- Topic Name: `oc2/cmd/device_type/alpha`
- Packet Identifier: 1234

Payload

- Content: request (JSON-encoded OpenC2 command)

```
{
  "action": "contain",
  "target": {
    "device": {
      "device_id": "9BCE8431AC106FAA3861C7E771D20E53"
    }
  }
}
```

- request_id: d1ac0489-ed51-4345-9175-f3078f30afe5
- created: Wed, 19 Dec 2018 22:15:00 GMT
- from: producer_one

Tabular representation of control packet

The values FH, VH, and PL represent the Fixed Header, Variable Header, and Payload portions, respectively, of the MQTT Control Packet.

Region	Field	Value
FH	Type	PUBLISH
FH	Dup	0
FH	QoS	1
FH	Retain	0
FH	Remaining Length	<computed>
VH	Topic Name	oc2/cmd/device_type/alpha
VH	Packet Identifier	1234
PL	Content	request (JSON-encoded OpenC2 command)
PL	request_id	d1ac0489-ed51-4345-9175-f3078f30afe5
PL	created	Wed, 19 Dec 2018 22:15:00 GMT
PL	from	producer_one

The JSON-encoded command in the PL:Content field is:

```
{
  "action": "contain",
  "target": {
    "device": {
      "device_id": "9BCE8431AC106FAA3861C7E771D20E53"
    }
  }
}
```

A.2.2: Broker Acknowledges the PUBLISH Control Packet

Bullet-list representation of control packet

Fixed Header

- Type: PUBACK
- Remaining Length: 2

Variable Header

- Packet Identifier: 1234

Tabular representation of control packet

Region	Field	Value
FH	Type	PUBACK
FH	Remaining Length	2
VH	Packet Identifier	1234

Appendix B. Acknowledgments

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

OpenC2 TC Members:

First Name	Last Name	Company
------------	-----------	---------

Standards Track Work Product

First Name	Last Name	Company
TBD	TBD	TBD

Appendix C. Revision History

Revision	Date	Editor	Changes Made
transf-mqtt-v1.0-wd01	2020-xx-xx	David Lemire	Initial working draft