Authoritative

Guide to MBOM

Model formulation processes within Manufacturing Bill of Materials

First Edition

Table of Contents

About the Guide 2
Copyright and License 2

PREFACE 3

THE INNOVATIVE HISTORY OF OWASP CYCLONEDX 4

INTRODUCTION 5

What is CycloneDX Formulation? 5
Design Philosophy and Guiding Principles 6
The Role of MBOM in Supply Chain Transparency 6

CORE CONCEPTS AND ARCHITECTURE 7

The Formulation Framework 7
High-Level MBOM Use Cases 7
Composability and Separation of Concerns 8

CYCLONEDX FORMULATION OBJECT MODEL 9

Overview 9
Object relationships 9
Formula relationships 9
Task relationships 10
Workflow relationships 12
Trigger relationships 13
Step relationships 14
Workspace relationships 15
inputType relationships 15
outputType relationships 16
resourceReferenceChoice explained 17
Use Case: Simple software application 18
MBOM Representation 19
Advanced representation of the build process 24

APPENDIX A: GLOSSARY 26

APPENDIX B: REFERENCES 27

About the Guide

CycloneDX is a modern standard for the software supply chain. It has been ratified as ECMA-424 by Ecma International.

The content in this guide results from continuous community feedback and input from leading experts in the software supply chain security field. This guide would not be possible without valuable feedback from the CycloneDX Industry Working Group (IWG), the CycloneDX Core Working Group (CWG), the many CycloneDX Feature Working Groups (FWG), Ecma International Technical Committee 54, and a global network of contributors and supporters.

Copyright and License

First Edition, 21 October 2025

Version	Changes	Updated On	Updated By
First Edition	Initial Release	2025-10-21	CycloneDX Core Working Group

Preface

Welcome to the Authoritative Guide series by the OWASP Foundation and OWASP CycloneDX. In this series, we aim to provide comprehensive insights and practical guidance, ensuring that security professionals, developers, and organizations alike have access to the latest best practices and methodologies.

At the heart of the OWASP Foundation lies a commitment to inclusivity and openness. We firmly believe that everyone deserves a seat at the table when it comes to shaping the future of cybersecurity standards. Our collaborative model fosters an environment where diverse perspectives converge to drive innovation and excellence.

In line with this ethos, the OWASP Foundation has partnered with Ecma International to create an inclusive, community-driven ecosystem for security standards development. This collaboration empowers individuals to contribute their expertise and insights, ensuring that standards like CycloneDX reflect the collective wisdom of the global cybersecurity community.

One standout example of this model is OWASP CycloneDX, which has been ratified as an Ecma International standard and is now known as ECMA-424. By leveraging the strengths of both organizations, CycloneDX serves as a cornerstone of security best practices, providing organizations with a universal standard for software and system transparency.

As you embark on your journey through this Authoritative Guide, we encourage you to engage actively with the content and join us in shaping the future of cybersecurity standards. Together, we can build a safer and more resilient digital world for all.

Andrew van der Stock
Executive Director, OWASP Foundation

The Innovative History of OWASP CycloneDX

OWASP CycloneDX has carved a legacy steeped in innovation, collaboration, and a commitment to openness. OWASP continues to advance software and system transparency standards, prioritizing capabilities that facilitate risk reduction.

October 2025

OWASP CycloneDX v1.7

First specification supporting citations that improve traceability, attribution, and auditability, and comprehensive support for patents and patent families to address intellectual property transparency.

June 2024

International Standardization

CycloneDX v1.6 ratified as an Ecma International standard and published as ECMA-424.

April 2024

OWASP CycloneDX v1.6

First specification to support cryptographic assets for Post-Quantum Cryptography (PQC) readiness and first general-purpose attestation specification to digitally transform audit and attestation workflows.

December 2023

Ecma TC54 Established

First working group chartered with holistic supply chain goals of standardizing core data formats, APIs, and algorithms that advance software and system transparency.

June 2023

OWASP CycloneDX v1.5

First specification to support AI Transparency, configuration and data components, and formulation describing how components were created, tested, trained, evaluated, and deployed.

January 2022

OWASP CycloneDX v1.4

First specification to introduce vulnerability sharing and transparency, including Vulnerability Disclosure Reports (VDR) and Vulnerability Exploitability eXchange (VEX).

May 2021

OWASP CycloneDX v1.3

First specification to incorporate support for composition completeness surpassing NTIA's framing of "known unknowns".

May 2020

OWASP CycloneDX v1.2

First specification to incorporate SWID (ISO/IEC 19770-2:2015) and services into inventory including data classifications, providers, and relationships between services and components.

March 2019

OWASP CycloneDX v1.1

First specification with complete pedigree support describing component lineage and the commits, patches, and diffs which make a forked version unique.

March 2018

OWASP CycloneDX v1.0

First general-purpose, security-focused Bill of Materials standard supporting software and hardware components. Introduced the world to Package-URL for software security use cases.

Source: https://tc54.org/history

Introduction

CycloneDX is a modern standard for the software supply chain. At its core, CycloneDX is a general-purpose Bill of Materials (BOM) standard capable of representing software, hardware, services, and other types of inventory. CycloneDX is an OWASP flagship project, has a formal standardization process and governance model through Ecma Technical Committee 54, and is supported by the global information security community.

What is CycloneDX Formulation?

CycloneDX formulation provides a universal framework for describing how *anything* came to be - whether software, hardware, services, data, algorithms, processes, or even the BOM document itself. The formulation capability extends the core CycloneDX specification to capture not just *what* components exist in a system, but *how* they were created, manufactured, tested, validated, certified, deployed, configured, or brought into existence through any process.

Formulation can describe the provenance and manufacturing process for virtually any entity:

- **Software Applications**: How code was compiled, tested, and packaged
- **Hardware Components**: Manufacturing processes, quality control, and assembly procedures
- **Services**: Deployment, configuration, testing, and operational procedures
- **Cryptographic Algorithms**: Implementation, validation, and compliance testing (e.g. FIPS 140-2, Common Criteria)
- **Data Sets**: Collection, curation, transformation, and validation processes
- **Infrastructure**: Provisioning, configuration, and deployment of systems
- **Compliance Artifacts**: How certifications, audits, and validations were conducted
- **BOM Documents**: The tools, processes, and procedures used to generate a BOM

The Strategic Importance of Universal Formulation

In today's interconnected world, understanding how things come to be is critical across all domains:

- **Security Assurance**: Verifying that security controls were applied during any creation or validation process
- **Compliance Verification**: Demonstrating adherence to regulatory requirements
- **Reproducible Processes**: Enabling independent verification and reconstruction of any process or artifact
- **Supply Chain Risk Management**: Identifying potential weaknesses or vulnerabilities in any creation, testing, or deployment pipeline
- **Incident Response**: Understanding how compromised or defective components were introduced into any system
- **Quality Assurance**: Documenting quality control processes for any type of deliverable
- **Regulatory Compliance**: Meeting documentation requirements across industries

Design Philosophy and Guiding Principles

The CycloneDX formulation specification is built on several foundational principles that enable universal applicability:

Universal Process Capture: The standard ensures comprehensive capture of any creation, manufacturing, testing, validation, certification, or deployment process, regardless of domain. This includes everything from software compilation to hardware assembly, service deployment, algorithm validation, data curation, and compliance certification.

Process Integration: Formulation captures key elements of any systematic process, leveraging patterns and semantics from workflow management across all domains - whether CI/CD tooling, manufacturing execution systems, laboratory information management systems, or compliance management platforms.

Compositional Integrity: The design assures that compositional elements, their associations, and relationships allow for a complete representation of the formulas and processes necessary to enable repeatable processes with full traceability across any domain.

Standards Compatibility: Formulation information is designed to be compatible with domain-specific standards and frameworks:

- **Software**: Supply-chain Levels for Software Assurance (SLSA), NIST Secure Software Development Framework (SSDF)
- **Hardware**: IPC standards, ISO 9001
- **Cryptography**: FIPS 140-2, Common Criteria (ISO/IEC 15408), NIST Post-Quantum Cryptography
- **Healthcare**: FDA 21 CFR Part 11, ISO 13485, IEC 62304
- **Automotive**: ISO 26262, ISO/SAE 21434
- **Aerospace**: DO-178C, DO-254, AS9100

The Role of MBOM in Supply Chain Transparency

The Manufacturing Bill of Materials serves as a critical bridge between different aspects of supply chain security:

- **SBOM Integration**: Links manufacturing processes to the resulting software components
- **SLSA Compliance**: Provides detailed build provenance information required for SLSA attestations
- **DevSecOps Enablement**: Documents security controls integrated into development workflows
- **Audit Trail Creation**: Maintains comprehensive records for compliance and security audits

Core Concepts and Architecture

The Formulation Framework

The CycloneDX formulation framework describes how something was "formed" or manufactured by capturing the complete manufacturing process. This includes:

Formulas: High-level descriptions of manufacturing processes that can be applied across different contexts and environments.

Workflows: Logical phases of the manufacturing process, organized as directed acyclic graphs of dependent tasks. In CI/CD contexts, workflows correspond to pipelines that execute a series of related operations.

Tasks: Individual units of work within workflows, each with specific inputs, outputs, and execution steps. Tasks represent atomic operations like building code, running tests, or deploying artifacts.

Steps: The granular commands and operations executed within tasks, providing the lowest level of detail about the manufacturing process.

High-Level MBOM Use Cases

The Manufacturing Bill of Materials supports diverse manufacturing scenarios across different domains:

Software Development (SBOM Integration)

- Describes how software components are built and deployed via CI/CD pipelines
- Captures everything from simple Makefile-based builds to complex multi-tier software systems
- Documents the use of build frameworks, dependency management, and deployment automation
- Supports both traditional and cloud-native development practices

Hardware Manufacturing (HBOM Integration)

- Documents hardware component manufacturing processes
- Captures assembly instructions, quality control procedures, and testing protocols
- Links physical manufacturing to digital twins and simulation models

Machine Learning Operations (MLBOM Integration)

- Describes ML model training, quantization, optimization, and deployment processes
- Captures data preprocessing pipelines, model validation procedures, and inference deployment
- Documents MLOps workflows including model versioning and A/B testing

Data Processing Pipelines

- Describes how data is collected, transformed, enhanced, curated, stored, analyzed, and utilized
- Captures data lineage, processing logic, and quality assurance procedures
- Documents data governance and compliance controls

Composability and Separation of Concerns

One of the key architectural decisions in CycloneDX formulation is the separation of creation/manufacturing information from component inventories. This composability approach offers several advantages across all domains:

Security Boundaries: Organizations can share component inventories (SBOMs, HBOMs, etc.) with customers and partners while maintaining strict control over sensitive process details (formulation data). This separation enables appropriate access controls and information sharing policies across different stakeholder groups.

Scalability: Large or complex processes can be decomposed into manageable, reusable components that can be linked and referenced as needed. This applies whether dealing with software builds, hardware assembly lines, or compliance certification processes.

Flexibility: Different stakeholders can access the level of detail appropriate for their use cases, from high-level component lists to detailed procedural documentation.

Formulation Linking Strategy

The recommended approach for managing formulation information involves referencing it from the primary component BOM using CycloneDX's BOM-Link mechanism. This applies universally whether describing software builds, hardware manufacturing, service deployment, or compliance processes:

```
"externalReferences": [
  {
    "type": "formulation",
    "url": "https://example.com/mboms/acme-library-1.0.cdx.json",
    "hashes": [
      {
        "alg": "SHA-256",
        "content": "c7be1ed902fb8dd4d48997c6452f5d7e509fbcdbe2808b16bcf4edce4c07d14e"
      }
    ]
  }
]
```

This approach enables:

- **Independent Access Control**: Different security policies for inventory vs. process data across all domains
- **Modular Management**: Separate lifecycle management for different aspects of the BOM (inventory vs. provenance)
- **Reduced Complexity**: Smaller, focused documents that are easier to process and validate
- **Domain Flexibility**: Different formulation documents for different aspects (manufacturing, testing, deployment, certification)

CycloneDX Formulation Object Model

Overview

The CycloneDX Formulation Object Model establishes a comprehensive hierarchy for representing manufacturing processes. Understanding these relationships is crucial for effective MBOM implementation.

CycloneDX is able to represent a formulation by providing a means to capture manufacturing processes using formulas which describe the workflows, tasks, steps and their relationships along with all referenced resources used during the processes.

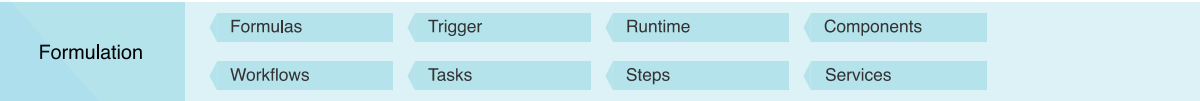
The formulation object model, described in this section, is intended to enable the verification of manufacturing compliance with any requirements or policies for the associated component or product described by its BOM. Additionally, the level-of-detail that can be represented by the model could potentially be leveraged to provide enough information to enable independent reproduction of a component's manufacturing process.

Formulation Structure

Formulation → Formula → Workflows → Tasks → Steps

This hierarchical structure mirrors real-world manufacturing processes, where high-level formulations contain specific formulas, which are implemented through workflows containing tasks that execute specific steps.

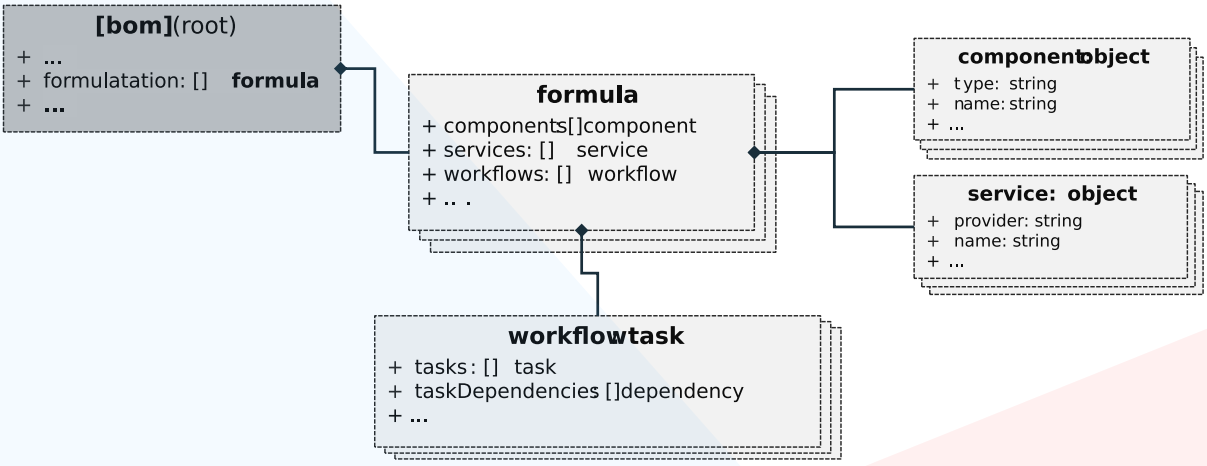
The following diagram shows some of the significant objects that are specifically included in the CycloneDX Formulation Object Model:



Object relationships

Note: *The object diagrams do not show every field that is available for a given object, but only includes those that are relevant to conveying the relational model.*

Formula relationships



formulation

The formulation attribute of the CycloneDX BOM object can be used to describe the *set of processes*, as formula, which detail how the top-level component or service described by the BOM was manufactured.

formula

A formula can describe a set of workflow objects each detailing one or more phases of how the associated component or service was tested, built, delivered, or deployed as a set of dependent tasks.

workflows

The list of workflows which were executed in order to manufacture a BOM's respective component.

Note: *In the context of software Continuous Integration and Delivery (CI/CD), workflows may also we referred to as "pipelines".*

components

The list of software, hardware or other components that are referenced by one or more formula and its workflows or tasks which have not been declared elsewhere within the same BOM document.

For example, build frameworks and tools, scanning tools, test tools and their data, runtime hardware, and software environment information, etc.

Note: *Any component referenced by formula must be declared within the same BOM document for the formula to be considered valid.*

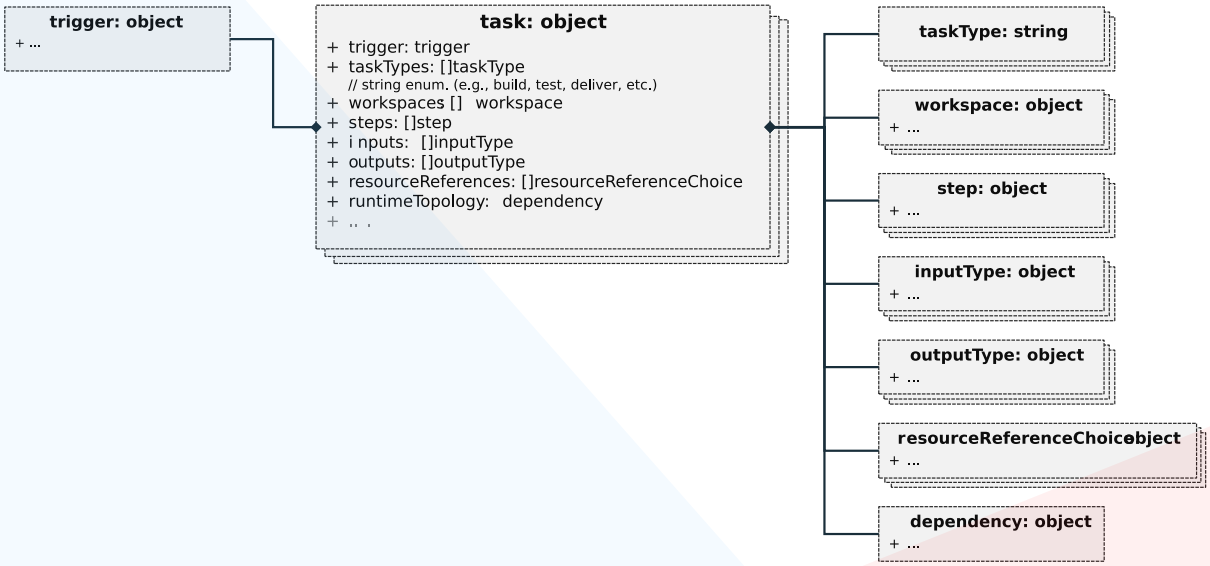
services

The list of services that are referenced by one or more formula and its workflows or tasks which have not been declared elsewhere within the same BOM document.

For example, services used for security scanning, artifact and data storage, logging, testing, deployment, etc.

Note: *Any service referenced by the formula must be declared within the same BOM document for the formula to be considered valid.*

Task relationships



task

Describes the inputs, sequence of steps and resources used to accomplish a task in order to produce its outputs.

Note: *In the context of software Continuous Integration and Delivery (CI/CD), tasks are sometimes referred to as "actions".*

trigger

Describes the **manual** (human) or **automated** action or event that triggered the task execution (i.e., caused its steps to be executed).

taskTypes

Describes the types of tasks, as a list of human-readable, single-word strings, for informational purposes.

The following taskType values are defined:

- **copy**: A task that copies software or data used to accomplish other tasks in the workflow.
- **clone**: A task that clones a software repository into the workflow in order to retrieve its source code or data for use in a build step.
- **lint**: A task that checks source code for programmatic and stylistic errors.
- **scan**: A task that performs a scan against source code, or built or deployed components and services. Scans are typically run to gather or test for security vulnerabilities or policy compliance.
- **merge**: A task that merges changes or fixes into source code prior to a build step in the workflow.
- **build**: A task that builds the source code, dependencies and/or data into an artifact that can be deployed to and executed on target systems.
- **test**: A task that verifies the functionality of a component or service.
- **deliver**: A task that delivers a built artifact to one or more target repositories or storage systems.
- **deploy**: A task that deploys a built artifact for execution on one or more target systems.
- **release**: A task that releases a built, versioned artifact to a target repository or distribution system.
- **clean**: A task that cleans unnecessary tools, build artifacts and/or data from workflow storage.
- **other**: A workflow task that does not match current task type definitions.

Note: *The current set of task types currently favor those that typically appear in modern Continuous Integration and Continuous Delivery (CI/CD) applications and platforms for software. Future versions of this specification may add additional task types for other domains.*

workspaces

The list of workspace objects that are associated with the workflow. A workspace is an accepted abstraction of a filesystem that is shared between tasks and their steps. For example, a workspace can hold the source for the BOM component being built, the binary produced by a build step, output from scanning tools, etc.

steps

Describes the sequence of steps, which may include the actual commands, that were executed by the task.

inputs

Describes references to resources or data made accessible, as input, to the task (and its step's commands) at runtime by the executor. For example, a configuration file used by a tool.

Note: the actual configuration file would be declared as a component or externalReference within the task itself or its parent workflow.

outputs

Describes references to resources or data produced, as output, by the task (and its step's commands). For example, a log file or metrics data.

Note: the actual log or metrics data files would be declared as components or externalReferences within the task itself or its parent workflow.

resourceReferences

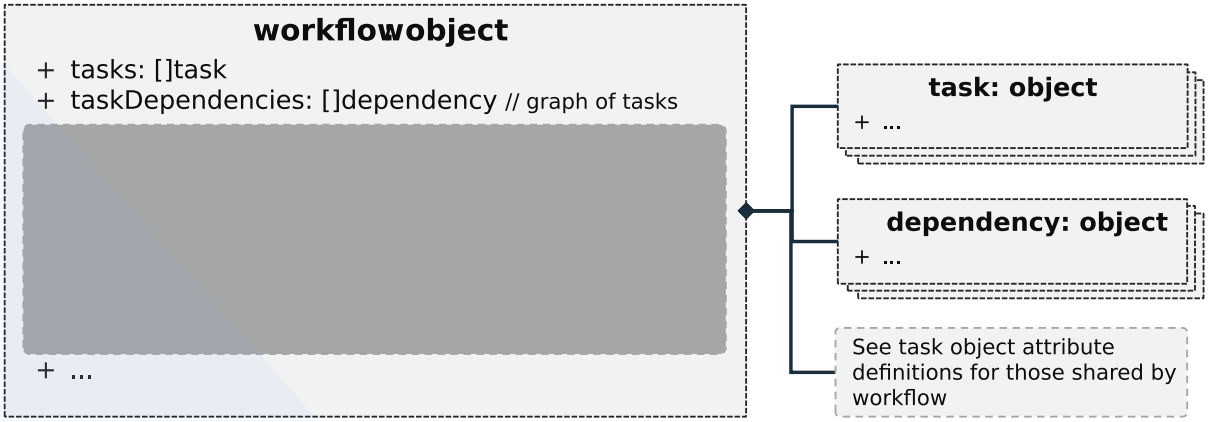
References to component or service resources that are used to realize the resource instance within the execution environment. For example, a logging service or artifact storage service reference.

See section resourceReferenceChoice explained for more details on how to specify resource references.

runtimeTopology

A graph of the component runtime topology for workflow's instance.

Workflow relationships



workflow

A workflow can describe a logical phase of the manufacturing process as a directed acyclic graph of dependent typed task objects.

The workflow object is a viewed (and can be treated) as a specialized "task" which shares most of the same attributes or fields as the task object. This allows a workflow to be referenced as a task in another workflow as part of the taskDependencies graph.

The workflow object uniquely adds the following object attributes described below:

- tasks - *see section below for details.*
- taskDependencies - *see section below for details.*

and duplicates the attributes described for the task object, but are instead relative to the workflow as a whole:

- trigger - *for the workflow as a whole.*

- taskTypes - *inclusive of all tasks listed in the workflow.*
- workspaces - *inclusive of all workspaces available, subject to access control, to all tasks in the workflow.*
- inputs - *to the workflow as a whole which may selectively be provided as inputs to the workflow's tasks.*
- outputs - *from the workflow as a whole.*
- resourceReferences - *made available to the workflow as a whole which may selectively be referenced to the workflow's tasks.*
- runtimeTopology - *Please note that in some execution environments, tasks within a workflow may be configured to run independently in separate runtime environments.*

Note: *The concept of the workflow object as a "near subclass" of a task object was too complex to map easily to JSON schema so it is described here.*

tasks

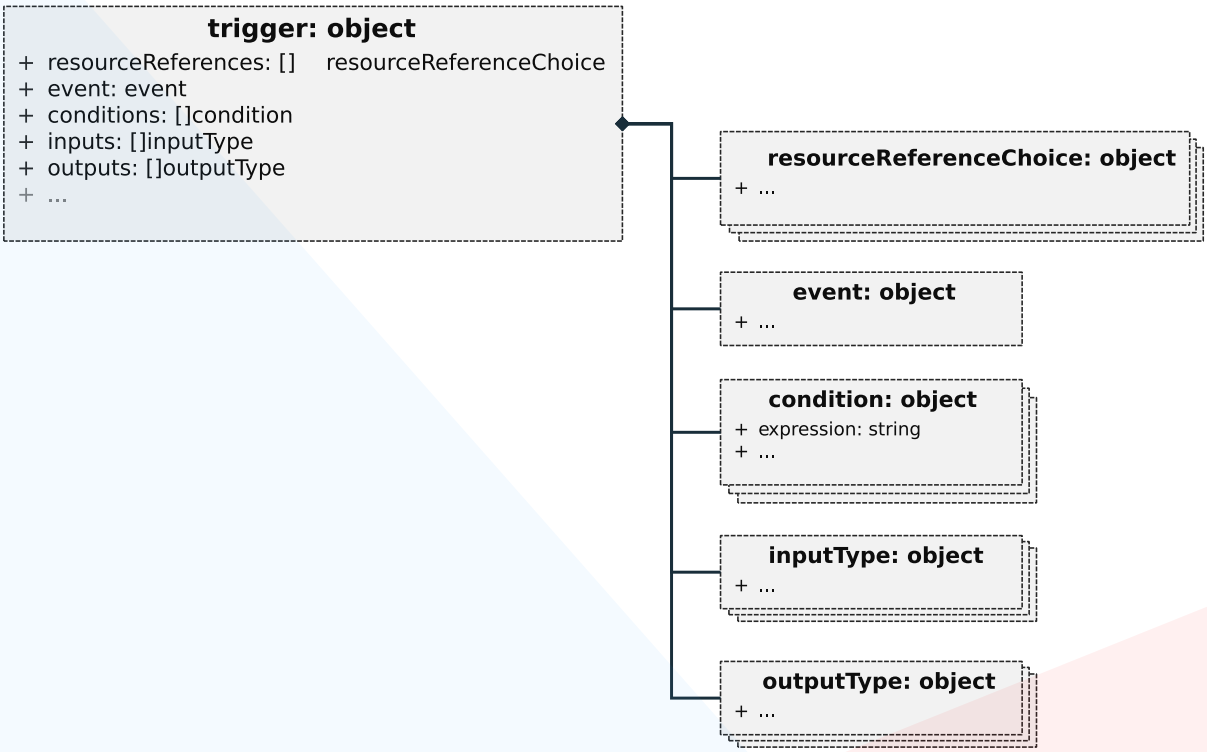
The list of task objects that contain the the low-level steps or commands

taskDependencies

A dependency graph of the tasks for the workflow indicating (observed) execution order.

Note: *The task dependency graph should be acyclic and map to the production of one or more output artifacts*

Trigger relationships



trigger

Describes a resource that can conditionally activate (or "fire") tasks based upon associated events and their data. Triggers are a common event-driven concept that can be defined and managed within the context of typical CI/CD platforms or systems. They enable the conditional execution of associated workflows or tasks in response to manual or automated events.

Triggers are an important part in understanding the context of why a workflow was run and affirm that any security and compliance policies were adhered to.

resourceReferences

References to component or service resources that are used to instantiate the trigger. These can include references to component or service resources, apart from the event data, that were used by the trigger to evaluate conditions (along with inputs) or produce outputs that would be consumed by the associated task or workflow.

event

Describes the event data that caused the associated trigger to be executed.

conditions

A list of conditions used to determine if a trigger should be activated and cause its associated task or workflow to be executed. Each condition captures the logical expression, and optionally any interpolated values, that the execution environment used to evaluate the condition.

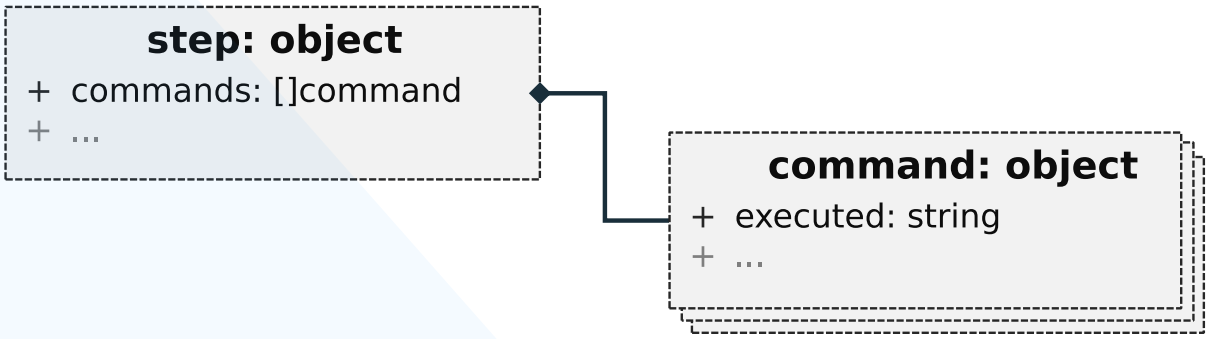
inputs

Represents resources and data provided to the trigger at runtime by the underlying execution environment that provide additional information used to evaluate conditions.

outputs

Represents resources and data provided by the trigger at runtime to the associate task or workflow.

Step relationships



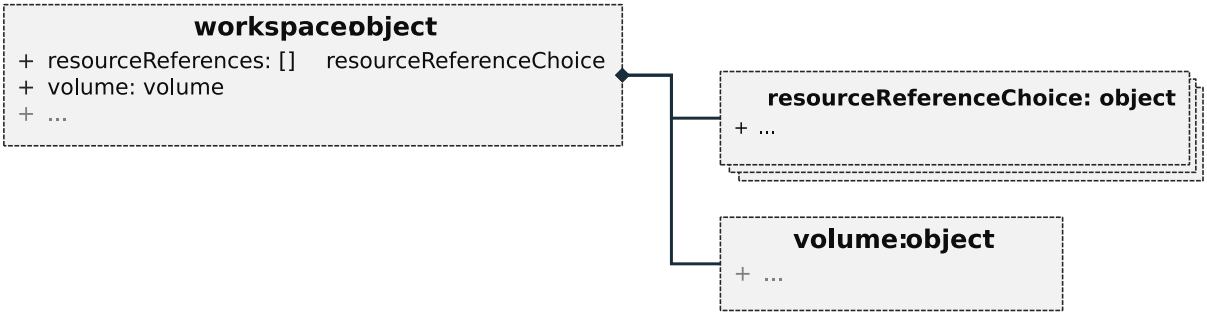
step

Describes the specific the ordered set of commands executed in order to accomplish its owning task.

commands

A text representation of the executed command. For example, this might be an interpolated shell command that copied files or ran a tool.

Workspace relationships



workspace

A named, logical resource typically backed by a filesystem or data resource shareable by workflow tasks. In some cases, these workspaces are implemented with access control to limit access to specific workflows or tasks.

resourceReferences

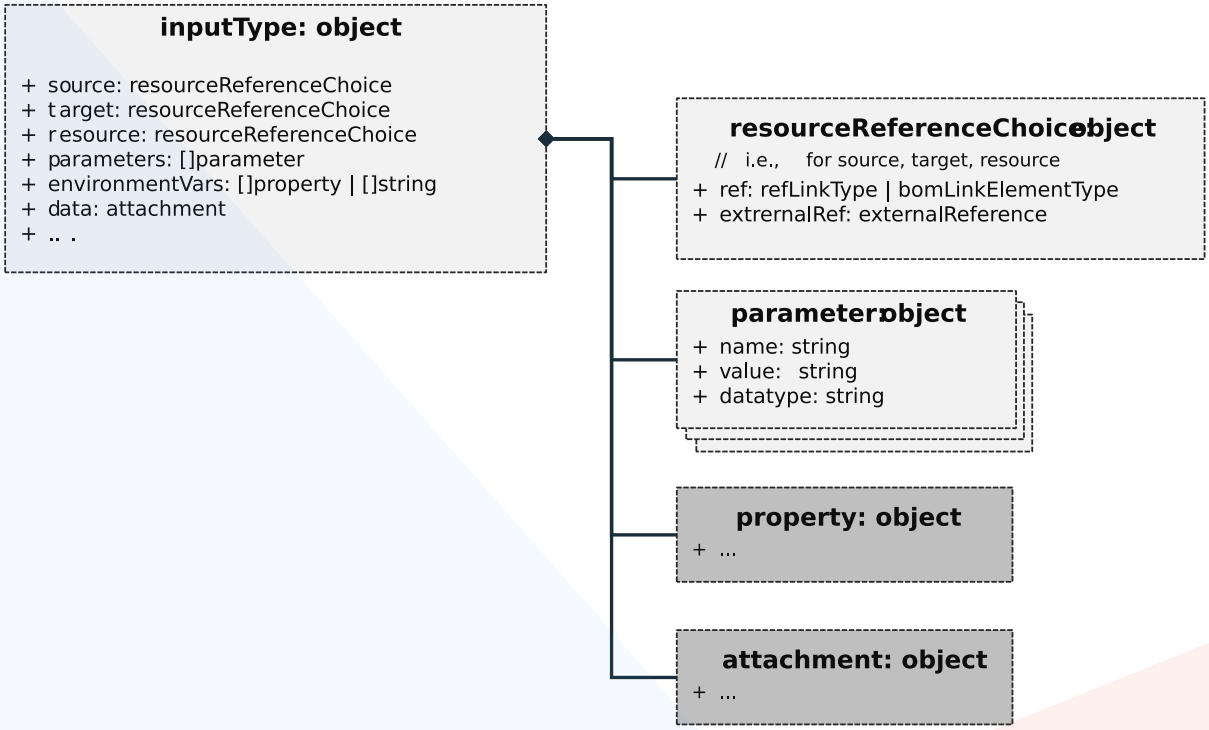
References to component or service resources that are used to realize the workspace. These could include references to resources such as storage services.

See section resourceReferenceChoice explained for more details on how to specify resource references.

volume

Information about the actual volume instance, if applicable, allocated to workspace.

inputType relationships



inputType

Describes different types of possible inputs to workflows, tasks, triggers and other objects in the model.

resourceReferenceChoice

This type is used to reference one of the CycloneDX types that points to a resource.

Specifically for inputs, they can describe the source of the input data, the target for the input data and/or a resource that adds additional data to the input.

For example, the source of input data may be the output from a previous task, while conversely the output of a task can declare its intended target.

parameter

A representation of a functional parameter.

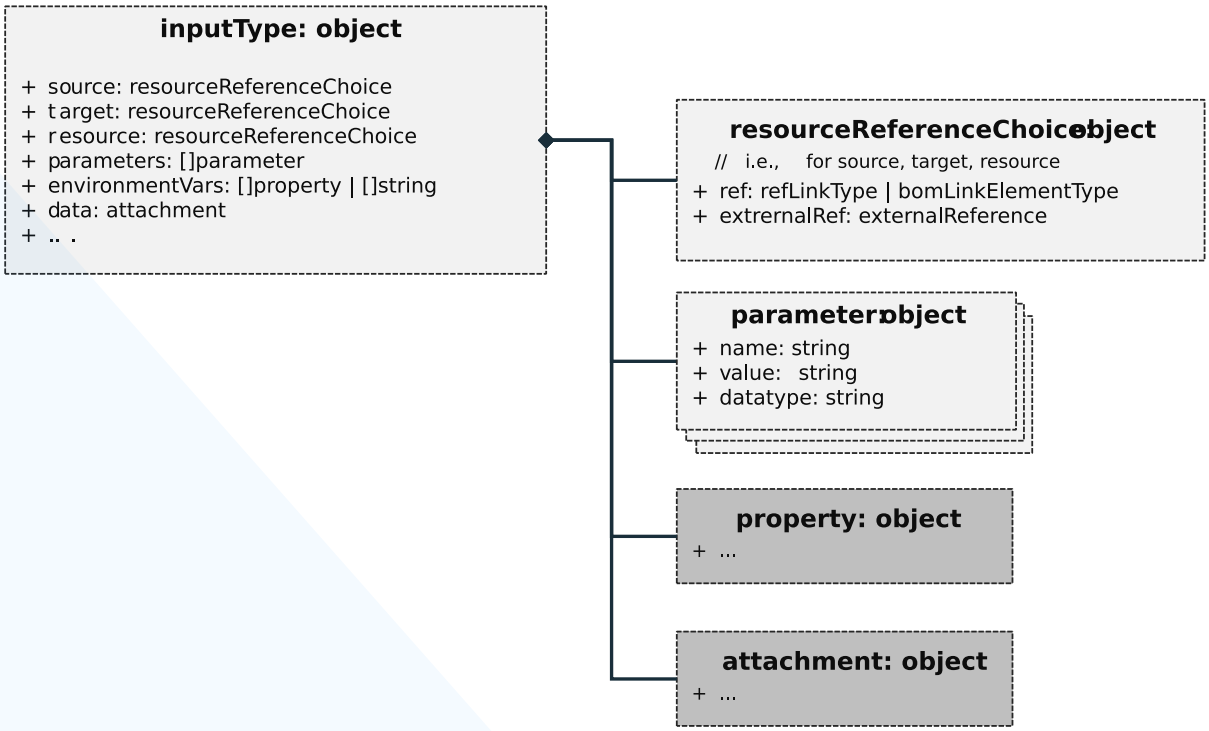
property

A property is a lightweight, name-value pair and defined as part of the core CycloneDX specification.

attachment

An attachment Specifies the metadata (e.g., content type, encoding, etc.) and content for an content data and defined as part of the core CycloneDX specification.

outputType relationships

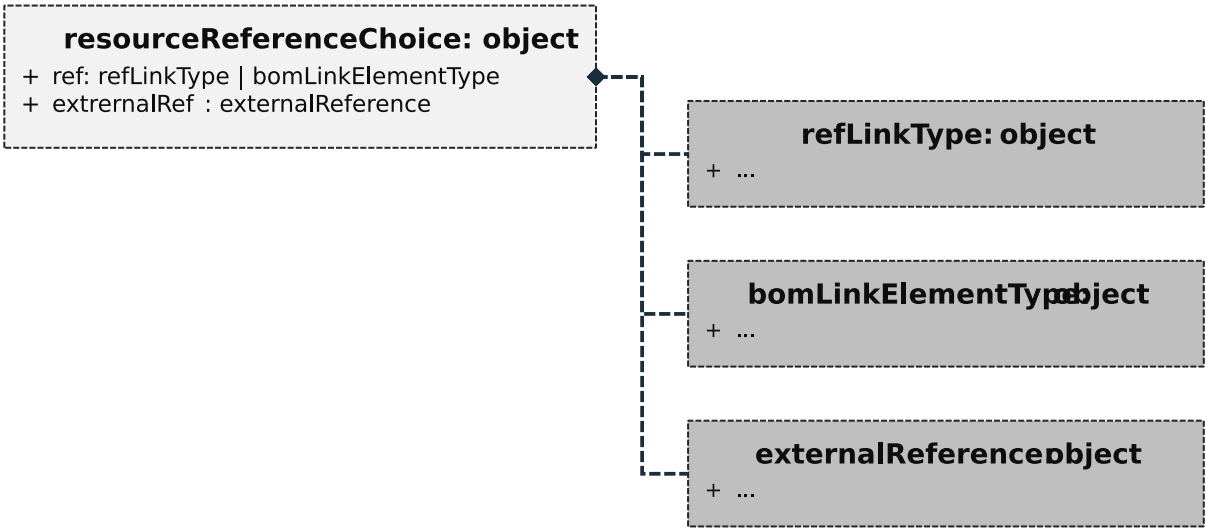


outputType

Describes different types of possible outputs from workflows, tasks, triggers and other objects in the model.

Most of the object attributes are identical to those described in the inputType relationships section.

resourceReferenceChoice explained



resourceReferenceChoice

This type is used composite existing CycloneDX reference types allowing different parts of a model to "point to" resources either defined within other parts with the same BOM document, another BOM or external to the BOM using a URL.

These include:

- bomLinkElementType - Descriptor for an element in a BOM document. See https://cyclonedx.org/capabilities/bomlink/.
- refLinkType - Descriptor for an element identified by the attribute bom-ref in the same BOM document. In contrast to bomLinkElementType.
- externalReference - This type is used to reference a resource external to the current BOM document using a url.

Use Case: Simple software application

This example shows how a simple helloworld application's build process can be captured by an MBOM.

Workflow overview

Application source code

The application itself is composed a single "C" source file, helloworld.c, which contains the following code:

```
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Application Makefile

The application is built using the GCC compiler using the following Makefile:

```
CC = gcc
CFLAGS = -Wall

build: clean hello

hello: helloworld.c
    $(CC) $(CFLAGS) -o hello helloworld.c

clean:
    rm -f hello
```

Build process

The application can be built by manually running the following command in a terminal/shell of a suitable operating system:

```
$ make build
```

which would cause the Makefile's build target (task) to be executed which would, in turn, case the dependent clean and hello targets to be executed in order and result in the creation of an executable file called hello.

Assumptions

When representing the manufacturing process in CycloneDX format, this example assumes:

- The formulation represents a local, manual build process that is executed on a single machine of source code already cloned from a GitHub repository.
- All referenced "tools" are already installed on the local system.
- For readability, component name values will use "short" names. For example, helloworld.c will be used instead of a best practice name CycloneDX/MBOM-examples/simple-application-makefile/helloworld.c; however, the corresponding bom-ref values will be based on the GitHub repository URL and commit hash to preserve uniqueness of identity.

- We will not attempt to encode the non-essential components for the Software Bill-of-Materials (SBOM) which is better show in other guides. For example use case, the "include" (header) file stdio.h is not represented.

MBOM Representation

For effectively conveying the essential representation of the build process using the CycloneDX Formulation objects, this example will initially focus on capturing only the key build artifacts, tools, and information. Then, we will show how additional information can be added to encode a more complete picture of the entire manufacturing process.

In order to simplify the readability of relationships in an MBOM, CycloneDX bom-ref values shown in the example will take the URI form: "cdx:mbom:<CycloneDX entity name>:uuid:<uuid>" although this is not a requirement of the CycloneDX Formulation.

Components

This section defines the essential component objects referenced in building the simple application. For files in this example, we will use the file:// URI scheme with an empty host to reference the local file system.

The component objects are defined as follows:

Source components

- helloworld.c:

```
{
  "bom-ref": "file:///CycloneDX/MBOM-examples/simple-application-makefile/helloworld.c",
  "type": "file",
  "name": "helloworld.c",
  "version": "1.0",
  "hashes": [
    {
      "alg": "SHA-256",
      "content": "..."
    }
  ]
}
```

Build components

- Makefile

```
{
  "bom-ref": "file:///CycloneDX/MBOM-examples/simple-application-makefile/Makefile",
  "type": "file",
  "name": "Makefile",
  "version": "1.0",
  "hashes": [
    {
      "alg": "SHA-256",
      "content": "..."
    }
  ]
}
```

- gcc - GCC compiler

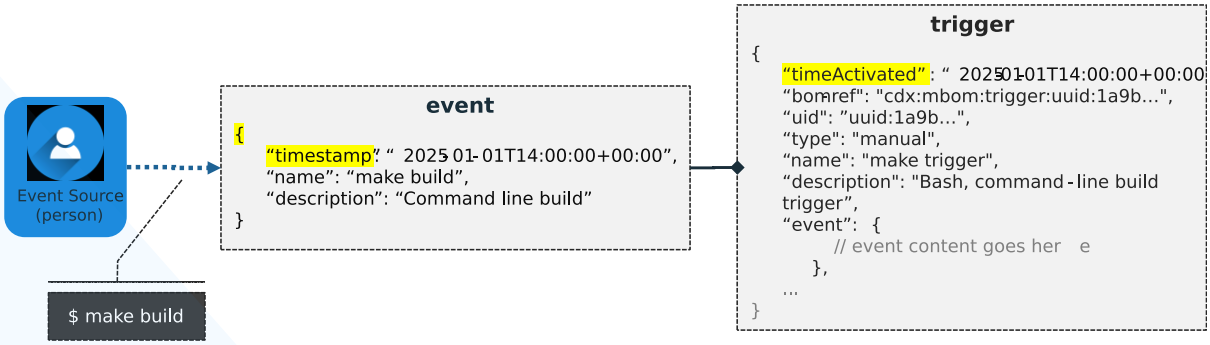
```
{
  "bom-ref":
"file:///Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/gcc",
  "type": "application",
  "name": "gcc",
  "version": "16.0.0 (clang-1600.0.26.4)"
}
```


- make utility


```
{
  "bom-ref":
"file:///Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/make
",
  "type": "application",
  "name": "GNU Make",
  "version": "3.81"
}
```


Event-Trigger relationship

This section describes how the human action make build can be represented in event and trigger data structures as shown here:



Event

In all cases, workflows are triggered by some sort of explicit, human or automated event. In this example, a person manually executed the following command in a Bash command prompt:

```
make build
```

this event could be represented as follows:

```
timestamp: "2025-01-01T14:00:00+00:00",
name: "make build",
description: "Command line build"
```

Note: *Workflows may be triggered by events dynamically received from other systems or services. In these cases, the event could include the raw event data itself as well as information.about the source system or service the event was sent by.*

Trigger

The trigger provides context about an event, as well as describing any additional information or resources used to augment an event before "triggering" an associated workflow. For this use case, the event and

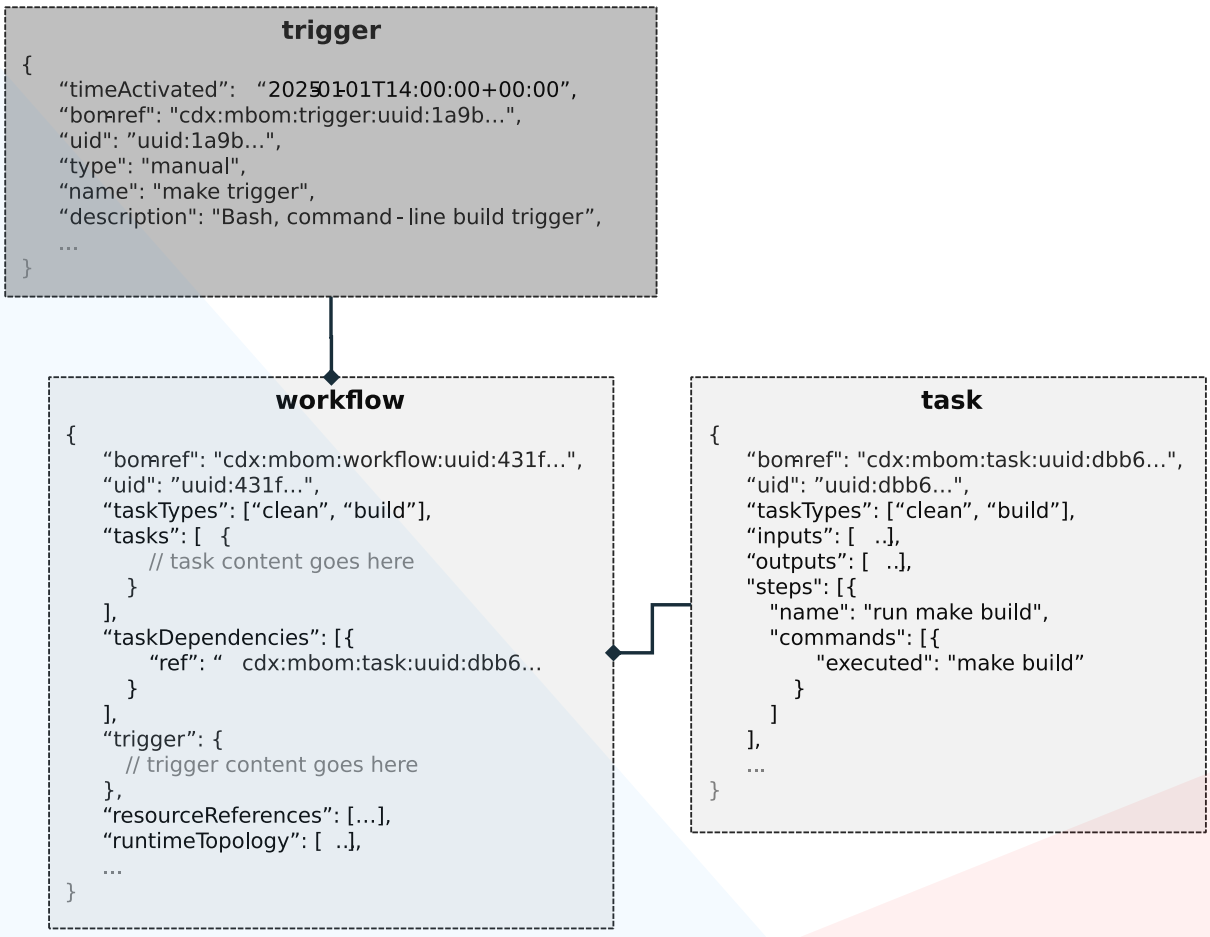
trigger represents a "manual" event type with a clear name and identifier (i.e., a uid) along with a more detailed description.

This could be represented as follows:

```
{
  “timeActivated”: “2025-01-01T14:00:00+00:00”,
  “bom-ref”: "cdx:mbom:trigger:uuid:1a9b...",
  “uid”: ”uuid:1a9b...",
  “type”: "manual",
  “name”: "make trigger",
  “description”: "Bash, command-line build trigger”
  “event”: {
    // event content goes here
  },
  ...
}
```

Note: *In this simple example, the trigger directly represents the event itself so the event's timestamp value is the same as the trigger's timeActivated value. However, in more complex event-driven build systems, the trigger represents a separable action subject to external rules such that the event timestamp value would reflect an earlier date-time than the trigger's timestamp value.*

Task-workflow relationship



Tasks

In this example, there is only one logical "task"; that is, the build process initiated by the make build command step. This task itself can be represented as:

```
{
  "bom-ref": "cdx:mbom:task:uuid:dbb6c5c0-6958-4a18-ac67-d897dbee76b6",
  "uid": "uuid:dbb6c5c0-6958-4a18-ac67-d897dbee76b6",
  "taskTypes": ["clean", "build"],
  "name": "make build task",
  "description": "A task that captures 'make build' step.",
  ...
}
```

As you can see we provide the two logical taskType values of clean and build to represent the logical steps the make command would perform as a result of resolving the target dependencies within the Makefile.

Adding steps to the task

The single command-line, build step can be added to the task:

```
{
  "bom-ref": "cdx:mbom:task:uuid:dbb6....",
  "uid": "uuid:dbb6...",
  "name": "make build task",
  ...
  "steps": [
    {
      "name": "run make build",
      "commands": [
        {
          "executed": "make build"
        }
      ]
    }
  ]
}
```

The trigger defined previously can be added to the task as follows:

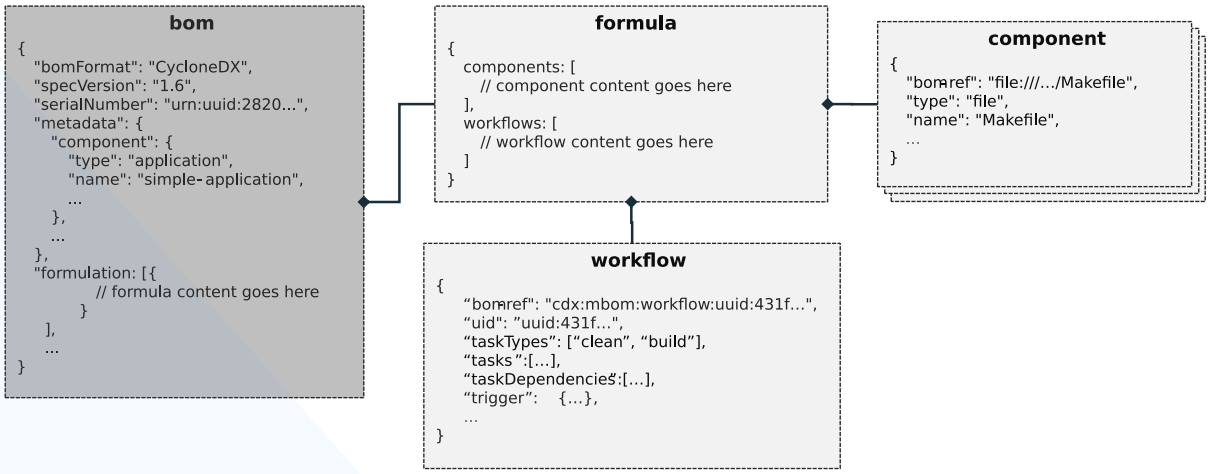
```
{
  "bom-ref": "cdx:mbom:task:uuid:dbb6....",
  "uid": "uuid:dbb6...",
  "name": "make build task",
  ...
  "trigger": {
    "bom-ref": "cdx:mbom:trigger:uuid:1a9b...",
    "uid": "uuid:1a9b....",
    "type": "manual",
    "name": "make trigger",
    "description": "Bash, command-line build trigger",
  },
  ...
}
```

Workflow

In this example, the workflow represents the single task execution as follows:

```
{
  “bom-ref”: "cdx:mbom:workflow:uuid:431ff656-8f90-410b-a614-c3916b842036",
  “uid”: ”uuid:431ff656-8f90-410b-a614-c3916b842036",
  “taskTypes”: [“clean”, “build”],
  “tasks”: [
    // task goes here
  ],
  “taskDependencies”: [
    {
      “ref”: “cdx:mbom:task:uuid:dbb6…"
    }
  ],
  “trigger”: {
    // trigger goes here
  },
  “resourceReferences”: [...],
  “runtimeTopology”: [...],
  ...
}
```

Formulation-Formula-Components relationship



Formula

The formulafor building this example application, in addition to describing the single workflow for this example, also includes the full listing (or manifest) of resources referenced by the workflow and its task. These elements can be represented as follows:

```
{
  components: [
      // component content goes here
  ],
  workflows: [
     // workflow content goes here
  ]
}
```

and finally the formula is placed under the CycloneDX BOM's formulation keyname of the Software Bill of Materials (SBOM):

```
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.7",
  "serialNumber": "urn:uuid:2820...",
  "metadata": {
    "component": {
      "type": "application",
      "name": "simple-application",
      ...
    },
    ...
  },
  "formulation: [{
      // formula content goes here
    }
  ],
  ...
}
```

Why list components under the formula?

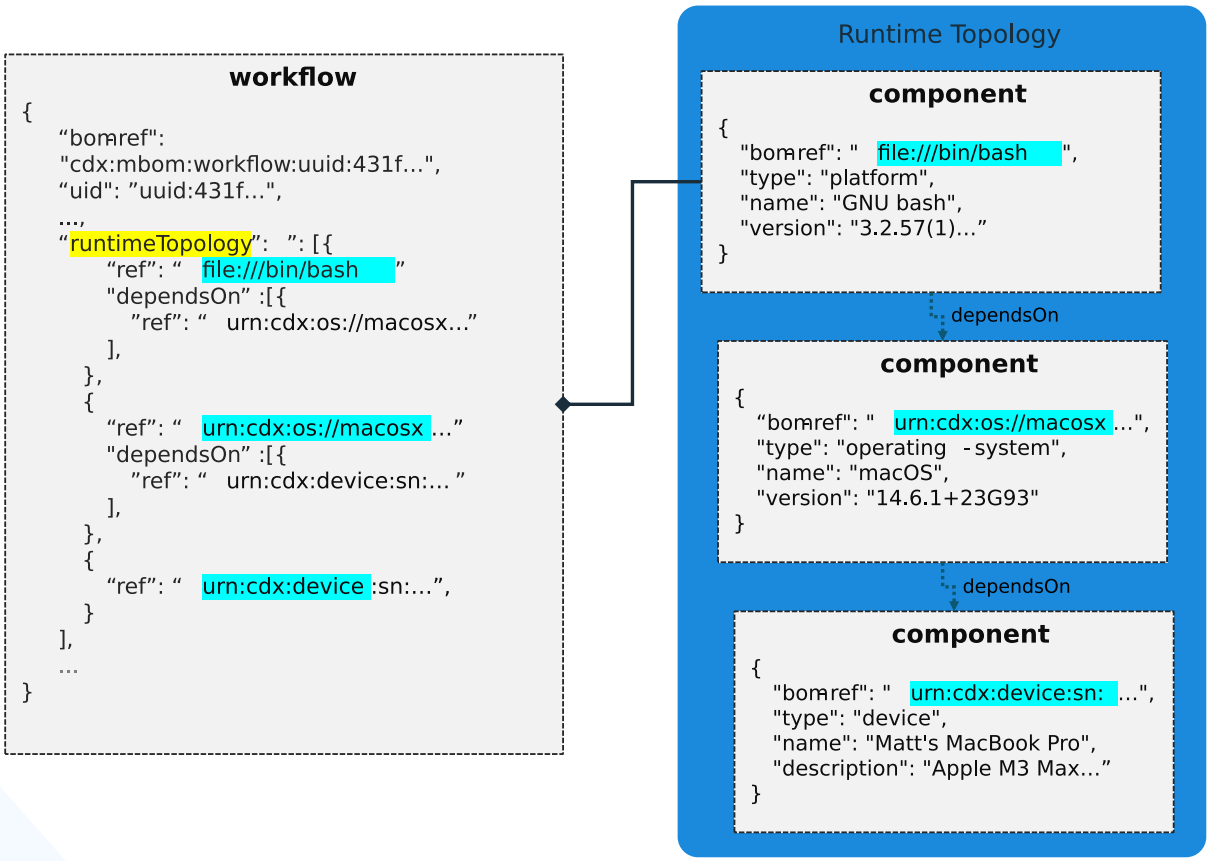
In our example, we chose to list components used to build the application under the formula keyname. However, it is possible to instead list them under the top-level components array's keyname.

This choice was made since this allows the MBOM information to be separated into a separate document from the associated SBOM and linked via the CycloneDX BOM-Link capability. This would be accomplished in the same manner as described for separating vulnerability information using the CycloneDX Vulnerability Disclosure Report (VDR) capability.*

Advanced representation of the build process

For many security and compliance use cases, it is necessary to represent the runtime topology (i.e., the build or manufacturing platform) of a software application and allowing independent verification the process is repeatable. This section shows how to add some of this information to the MBOM for this example.

Runtime topology



The runtime topology represents any software frameworks, platforms, tools, hardware and other resources used to create the software application used to run the workflow and its tasks.

This section will show how to represent the runtime topology for the simple application's build process which includes a Bash shell running on a Mac OS X machine.

Platform

For this example, we can choose to represent the key platform elements used to run the make command. This could include the shell and the operating system used to run the build process as CycloneDX components. For example:

- The Bash shell used to run the make command:

```
{
  "bom-ref": "file:///bin/bash",
  "type": "platform",
  "name": "GNU bash",
  "version": "3.2.57(1)-release (arm64-apple-darwin23)"
}
```

- The OS X operating system the Bash terminal was running on:

```
{
  "bom-ref": "urn:cdx:os://macosx@14.6.1+23G93",
  "type": "operating-system",
  "name": "macOS",
```

```
  "version": "14.6.1+23G93"
}
```

Hardware

Additionally, we could describe the actual device used for the build process to an appropriate level of detail:

- Mac OS X machine

```
{
  "bom-ref": "urn:cdx:device:sn:CBFX71DM3",
  "type": "device",
  "name": "Matt's MacBook Pro",
  "description": "Apple M3 Max, 16 inch"
}
```

Runtime topology relationships

Including the BOM creation in the Makefile

It is envisioned that Software-Bill-of-Materials (SBOM) will be created as part of the build process including Manufacturing (MBOM) information. This could be reflected as a post-build target in the Makefile which would bring additional SBOM generation tooling into the manufacturing process itself and be reflected in the SBOM/MBOM document as well.

Appendix A: Glossary

- Formulation - Describes the set of processes, for how a component or service was manufactured, tested, delivered and/or deployed. These processes are captured as formula which describe the workflows, tasks and steps along with components and services used (observed) in those processes.
- Workflow - Workflows are used to manage repetitive processes and tasks that occur in a particular order. In the context of Continuous Integration and Continuous Delivery (CI/CD) pipelines, workflows are used to define the sequence of steps that need to be executed in order to build, test, and deploy an application. Workflows can be defined using various tools such as Jenkins, Tekton, or CircleCI.

Appendix B: References

The following resources may be useful to users and adopters of this standard:

- Package-URL specification: https://github.com/package-url/purl-spec/
    - Specifically, the pURL types reference in examples:
        - github
            - *Example*: pkg:github/package-url/purl-spec@244fd47e07d1004#everybody/loves/dogs
        - golang
            - *Example*:pkg:golang/github.com/gorilla/context@234fd47e07d1004f0aed9c#api
        - huggingface
            - *Example*: pkg:huggingface/microsoft/deberta-v3-base@559062ad13d311b87b2c455e67dcd5f1c8f65111?repository_url=https://hub-ci.huggingface.co

References in examples

- gcc https://gcc.gnu.org/
- Tekton https://tekton.dev/

- SPDX License IDs
- SPDX License List
- OpenChain
- OWASP CycloneDX
- OWASP CycloneDX Tool Center
- OWASP CycloneDX BOM Repository Server
- OWASP Dependency-Track
- OWASP Software Component Verification Standard (SCVS)
- OWASP Software Component Verification Standard (SCVS) BOM Maturity Model