The W3C Web of Things (WoT) is intended to enable interoperability across IoT Platforms and application domains. Primarily, it provides mechanisms to formally describe IoT interfaces to allow IoT devices and services to communicate with each other, independent of their underlying implementation, and across multiple networking protocols. Secondarily, it provides a standardized way to define and program IoT behavior.

This document describes the abstract architecture for the W3C Web of Things. It is derived from a set of use cases and can be mapped onto a variety of concrete deployment scenarios, several examples of which are given. This document is focused on the standardization scope of W3C WoT, which consists of three initial building blocks that are briefly introduced and their interplay explained.

The WoT Thing Description (TD) provides a formal mechanism to describe the network interface provided by IoT devices and services, independent of their implementation. Provision of a TD is the primary requirement for a device to participate in the Web of Things. In fact, defining a Thing Description for an existing device allows that device to participate in the Web of Things without having to make any modifications to the device itself. WoT Binding Templates define how a WoT device communicates using a concrete protocol. The WoT Scripting API—whose use is not mandatory—provides a convenient mechanism to discover, consume, and expose Things based on the WoT Thing Description.

Other non-normative architectural blocks and conditions underlying the Web of Things are also described in the context of deployment scenarios. In particular, recommendations for security and privacy are included, while the goal is to preserve and support existing device mechanisms and properties. In general, W3C WoT is designed to describe what exists rather than to prescribe what to implement.

Please contribute to this draft using the [GitHub Issue](https://github.com/w3c/wot-architecture/issues) feature of the [WoT Architecture](https://github.com/w3c/wot-architecture/) repository. For feedback on security and privacy considerations, please use the [WoT Security and Privacy](https://github.com/w3c/wot-security/issues) Issues, as they are cross-cutting over all our documents.

# Introduction

The "Web of Things" (WoT) started as an academic initiative in the form of publications and, starting in 2010, a yearly [International Workshop on the Web of Things](https://webofthings.org/events/). Its goal is to improve interoperability as well as usability in the Internet of Things (IoT). With the increasing role of IoT services using other web standards in commercial and industrial applications, the W3C chartered an [Interest Group](https://www.w3.org/WoT/IG/) in 2015 to identify technological building blocks for Recommendation Track standardization. With the WoT [Working Group](https://www.w3.org/WoT/WG/) chartered end of 2016, the first set of WoT building blocks is now being standardized:

* the [WoT Thing Description](https://w3c.github.io/wot-thing-description/),
* the [WoT Binding Templates](https://w3c.github.io/wot-binding-templates/), and
* the [WoT Scripting API](https://w3c.github.io/wot-scripting-api/).

This document serves as an umbrella for the W3C WoT draft specifications and defines the basics such as terminology and the underlying abstract architecture of the W3C Web of Things. In particular, the purpose of this document is to provide

* a set of use cases that lead to the W3C WoT Architecture,
* a set of requirements for WoT implementations,
* an overview of the WoT building blocks being standardized and their interplay,
* a guideline to map the abstract architecture to software stacks and hardware components,
* and security considerations to be aware of when implementing WoT building blocks.

## Terminology

This document uses the following terms as defined here. The WoT prefix is used to avoid ambiguity for terms that are defined specifically for Web of Things concepts.

Please refer to [this markdown file](file:///Users/mike-work/git/mlagally/wot-architecture/proposals/terminology.md). Once the terminology definitions are stable, they will be included here.

Term

Definition

## Terminology imported by Toru Kawaguchi

This document uses the following terms as defined here. The WoT prefix is used to avoid ambiguity for terms that are defined specifically for Web of Things concepts.

Action

An Interaction Pattern to invoke procedure on a Thing, potentially a long running physical process (e.g., lowering of window blinds).

Application

A running or periodically running process that consumes one or more Things (read and processes there data, control their state) and/or exposes one or more Things (provide access to locally gathered data or computed from consumed Things, provide remote control and management interfaces). The application can be implemented using the Scripting API or native platform APIs. The application may also be distributed over multiple Things and other hosts.

Binding Templates

A re-usable collection of blueprints for the communication with different IoT platforms. The blueprints includes the required vocabulary for the Thing Description to map Interactions to platform-specific messages as well as implementation notes for the required protocol stacks or dedicated communication drivers.

Client API

Programming interface that allows scripts to access remote Things over the network, local Things in a different execution environment, or directly attached hardware (which is abstracted as Things).

to consume a Thing

To read in a Thing Description and create a Consumed Thing software object for the application in the local runtime environment.

Consumed Thing

A software object instance created through the WoT object of the Scripting API that represents a remote Thing used by the local application. It is defined by the ConsumedThing interface of the WoT Scripting API.

Discovery API

Programming interface that allows scripts to discover other Things (local, nearby, or remote).

Domain-specific vocabulary

Linked Data vocabulary that can be used in the WoT Thing Description, but is not defined by W3C WoT.

Event

An Interaction Pattern to subscribe to data pushed by a WoT Server. These can be stand-alone events such as an alarm, change-of-value notifications, or time series of data.

Execution Environment

A sandbox within the Runtime that isolates scripts running on the same Servient.

to expose a Thing

To create an Exposed Thing software object in the local runtime environment that enables the application to provide local state and calls as Interactions over the network.

Exposed Thing

A software object instance created through the Server API that is implemented locally and can be accessed over the network by remote WoT Clients. It is defined by the ExposedThing interface of the WoT Scripting API.

Interaction

A functionality exposed by Thing that is defined by the TD Interaction Model.

Interaction Model

A formal model that syntactically and semantically maps different network-facing interfaces (i.e., WoT Interfaces) to application-facing interfaces (e.g., the Scripting API -- note that these can also be proprietary APIs when not implementing the WoT Runtime). The Thing Description is built around this model and connects it to Link Data vocabularies.

Interaction Pattern

A default solution to describe common Interactions: Property, Action, and Event.

IoT platform

A specific IoT ecosystem such as OCF, oneM2M, or Mozilla Project Things with its own specifications for application-facing APIs, data model, and protocols or protocol configurations.

Local Discovery

A discovery method that can discover Things directly connected to a Servient (e.g., sensor or actuator that is abstracted as Thing).

Manual Discovery

A discovery method where the URI of the used consumed Thing Descriptions is provided manually (e.g., through user configuration or hard-coding in a script).

Nearby Discovery

A discovery method where the physical location is considered (e.g., BLE, Audio Watermarking, ...).

Network Discovery

A discovery method that can discover Things in local networks (e.g. SSDP, mDNS/DNS-SD, ...).

Property

An Interaction Pattern to read a specific value of a Thing and optionally to write it.

Protocol Binding

An implementation of the mapping between Interaction in the TD Interaction Model (e.g., Property read) and specific operations of a protocol (e.g., HTTP GET).

Remote Discovery

A discovery method which supports lookup of remote Things also beyond network boundaries, for instance by using a directory service. The endpoint of the directory must be supported.

Scripting API

The application-facing programming interface provided by a Servient; comparable to the Web browser APIs.

Server API

Programming interface that allows scripts to expose local functionality as Things to WoT Clients.

Servient

A software stack that implements the WoT building blocks. A Servient can host and expose Things (server role) and/or consume Things (client role). Servients usually have multiple Protocol Bindings to enable interaction with different platforms.

TD

Short for Thing Description.

TD Vocabulary

A controlled Linked Data vocabulary by W3C WoT to tag the metadata of Things in the WoT Thing Description including communication metadata of WoT Binding Templates.

Thing

An abstraction of a physical or virtual entity whose metadata and interfaces are described by a WoT Thing Description. This entity can be an existing device augmented with a Thing Description, a logical component of a device that has a software stack that natively provides Thing Descriptions (see Servient), a local hardware component of a Servient, or a logical entity (e.g., location) that does not directly provide any software stack such as a room or building.

Thing Description (TD)

Structured data describing a Thing. A TD includes metadata, domain-specific metadata, a list of offered interactions, the supported protocol bindings for each interaction, and links to related Things. The Thing Description is built around a formal Interaction Model.

Thing Directory

A directory service for TDs that provides a Web interface to register TDs (see [draft-ietf-core-resource-directory](https://tools.ietf.org/html/draft-ietf-core-resource-directory)) and look them up (e.g., using SPARQL queries or CoRE Link Format).

WoT Client

An entity that can connect with a network interface described by a WoT Thing Description (i.e., consume a Thing). WoT Clients usually implement multiple Protocol Bindings. WoT Client is also used to refer to a Servient in client role only.

WoT Interface

The network-facing interface of a Thing as defined by its Thing Description.

WoT Object

The WoT object is the Scripting API entry point within a WoT Runtime. It provides methods to discover, consume, and expose Things.

WoT Runtime

A runtime system for application scripts with the WoT Scripting API. Implementing a WoT Runtime is optional for Servients.

WoT Server

An entity that exposes a network interface consistent with a WoT Thing Description. WoT Server is also used to refer to a Servient in server role only.

CoAP

Acronym for Constrained Application Protocol [RFC7252]

CWT

CBOR Web Token

JSON-LD

A JSON document that is augmented with support for Linked Data by providing an @context property with a defining URI [JSON-LD](https://www.w3.org/TR/json-ld/).

JWT

JSON Web Token [RFC7519]

RDF

The Resource Description Framework (RDF) of the Semantic Web [rdf11-concepts](https://www.w3.org/TR/rdf11-concepts/)

# Use Cases

This section presents the use cases targeted by the W3C WoT and which are used to derive the abstract architecture discussed in . While Smart Home use cases might appear predominant in this section, they should be seen as simply a vehicle to identify fundamental requirements inherent to most application domains. The Smart Home domain is suitable for identifying such general requirements, as most stakeholders can relate to it.

## Smart Home

The Smart Home is one of the application domains targeted by W3C WoT. As stated above its use cases are eligible to convey the fundamental requirements on discovery, connectivity, and provided functionality. Also security has become a central issue in consumer systems. Furthermore, Smart Home use cases have stronger privacy requirements than commercial and industrial ones. In general, however, the fundamental aspects of the Smart Home use cases given also apply to other use cases.

### Device Controllers

The first use case is a local device controlled by user-operated remote controller as depicted in . For example, an electronic appliance such as an air conditioner with Web server functionality might be connected directly to a local home network. A remote controller can access the air conditioner through the local home network directly. In this case, the remote controller can be realized by a browser or native application.

Device Control

### Thing-to-Thing

shows an example of direct Thing-to-Thing interaction. The scenario is as follows: when a sensor detects the room temperature is surpassing a set threshold (e.g., 25°C), a control agent issues a "power-on" command to an air conditioner.

Control Agent

### Multiple Network Interfaces

The third use case is a mobile remote controller (e.g., on a smartphone) as shown in . When at home, the smartphone can use Wi-Fi or Bluetooth/BLE to control an electronic appliance locally, while outside, it can use the cellular network.

Multiple Network Interfaces

### Gateways

shows a use case based on a Smart Home gateway. It is placed between a home network and the Internet. The gateway manages electronic appliances inside the house and can receive commands from a remote controller over the Internet, e.g., from a smartphone as in the previous use case.

Smart Home Gateway

### Cloud Proxies

Cloud proxies can be realized in different ways:

The complexity of the use cases here increases too quickly. They will be split up to progressively add and explain each new feature.

#### Cloud-ready Devices

shows an example where electronic appliances are connected directly to the cloud. The cloud mirrors the appliances and, acting as a proxy, can receive commands from remote controllers (e.g., a smartphone). Authorized controllers can be located anywhere, as the proxy is globally reachable.

Proxies with Cloud-ready Devices

### Legacy Devices

shows an example where legacy electronic appliances cannot directly connect to the cloud. Here, a gateway is needed to relay the connection. The gateway works as:

* integrator of a variety of legacy communication protocols both in the physical and logical view
* firewall toward the Internet
* privacy filter which substitutes real image and/or speech, and logs data locally
* local agent in case the Internet connection is interrupted
* emergency services running locally when fire alarms and similar events occur

The cloud mirrors the gateway with all connected appliances and acts as an agent that manages them in the cloud in conjunction with the gateway. Furthermore, the cloud can receive commands from remote controllers (e.g., a smartphone), which can be located anywhere.

Proxies with Legacy Devices

## Smart Factory

shows an example of a Smart Factory. In this case, cell controllers automate factory equipment with specialized communication such as RS-485 or EtherCAT. Line controllers orchestrate multiple cell controllers over IP-based networks. A factory controller integrates different line controllers. A cloud service collects data from the factory controller and analyzes them for predictive maintenance. Users can monitor the factory through a dashboard. Note that factories usually do not reveal the internal topology of their lines and cells.

Smart Factory

### Connected Car

shows an example of a Connected Car. In this case, a gateway connects to car components through CAN and to the car navigation system through a proprietary interface. Services running in the cloud collect data pushed from car components and analyze the data from multiple cars to determine traffic patterns. The gateway can also consume cloud services, in this case, to get traffic data and show it to the driver through the car navigation system.

Connected Car

# Industrial IoT Use Cases (Michael Lagally)

The following chapter describes industrial use cases, that are applicable to different industry verticals. Due to the nature of overlaps in the application scenarios, different verticals have similar use cases.

## Manufacturing

### Predictive Maintenance

Monitoring and predicting machine failures and anomalies.

### Maintenance automation

Automating the process of creating a maintenance/service ticket based on reactive or predictive detection of problems.

### Quality Improvements

Improving quality through monitoring the condition of manufacturing equipment.

### Industrial safety

Reducing industrial safety incidents/accidents.

### Environment monitoring

Monitoring environment in the manufacturing plant for presence of poisonous gases, excessive noise/heat, etc.

### Real-time monitoring

Real-time monitoring and KPI calculations of production equipment.

## Transportation & Logistics

### Shipment tracking

Tracking of shipment en-route - condition, quality and location monitoring.

### Fleet tracking

Monitoring fleet vehicles, fuel costs, maintenance needs, assignments.

### Cold chain monitoring

Ensuring integrity of the cold chain - from warehouses to refrigerated trucks to delivery.

### Warehouse monitoring

Monitoring equipment and stock in warehouses, yards.

## Utilities

### Advanced Metering Infrastructure

Automated reading of residential and C&I (Commercial and Industrial) meters, and billing.

### Distributed Energy Resource management

Monitoring the condition and output of distributed renewable energy generation equipment.

### Distribution Automation

Monitoring and remote-controlling Distribution equipment.

### Crew safety

Improving safety of utilities crew in the field.

## Oil and Gas

### Pipeline monitoring

Offshore platform monitoring Industrial safety on offshore platforms.

### Leakage detection

Detecting/Predicting leakage through the pipelines.

### Tank/Reservoir level monitoring

Monitoring and Controlling the levels in tanks and reservoirs.

### Automated Stock Taking

Automated calculation of a distributed stock through various storage tanks and delivery pipes/trucks.

## Insurance

### Proactive Asset Monitoring

Monitoring of high value assets such as connected structures, fleet vehicles, etc.

### Usage based insurance

Usage tracking and customized insurance policies.

### Safety monitoring

Employee/Occupant safety monitoring.

### Loss reduction - leakage detection

Detecting leakages to limit the losses.

### Loss prevention - garaging fleet vehicles

Predictive weather monitoring and re-routing fleet vehicles to covered garages to limit loss due to hail damage, tree damage.

## Engineering and Construction

### Worker Safety

Worker monitoring for industrial safety.

### Construction Asset Monitoring

Monitoring assets at construction site.

## Connected Car

### Operation monitoring

Monitoring of operation status, prediction of service needs, optimisation of maintenance.

### Driver Safety

Enhance driver safety with notifications of an early warning system for critical road and traffic conditions.

## Agriculture

### Soil condition monitoring

Monitoring soil conditions and creating optimal plans for watering, fertilizing.

### Produce monitoring

Monitoring produce conditions.

--

# Healthcare

### Clinical Trials

Data collection and analytics of clinical trial data.

### Pharmaceutical monitoring

Monitoring of medicines and totes throughout the supply chain.

### Remote Patient Monitoring

Monitoring of patients after hospitalization.

--

# Smart Cities

### Infrastructure monitoring

Monitoring of Bridges, Dams, Levees, Canals - material condition, deterioration, vibrations monitoring.

### Smart Parking

Tracking usage, availability of parking spaces and automating billing/reservations.

### Smart Lighting

Smart control of street lights based on presence detection, weather predictions, etc.

### Waste Management

Monitoring of garbage containers to optimize the trash collection route.

### Highway Monitoring

Monitoring of highways and providing appropriate signage.

--

# Environment Monitoring

### Air and water pollution

Monitoring of air pollution, water pollution and other environmental risk factors such as fine dust, ozone, volatile organic compound, radioactivity, temperature, humidity to detect critical environment conditions.

### Disaster prediction and prevention

Analysis of sensor data from distributed sensors to detect anomalies and predict critical environment conditions.

--

# Smart Buildings

### Energy management

Monitoring energy usage throughout the building.

### Equipment monitoring, Occupant satisfaction

Monitoring equipment in the buildings such as HVAC, Elevators, etc. Monitoring satisfaction of occupants.

### Cross-domain Collaboration (added by Kunihiko Toumura)

show an example of a Cross-domain collaboration. In this case, each system involves other systems in other domains, such as Smart Factrory with Smart City, Smart City with Smart Home. This type of system is called ["Symbiotic" ecosystem](https://www.iec.ch/whitepaper/pdf/iecWP-futurefactory-LR-en.pdf). There are two collaboration models: direct collaboration and indirect collaboration. In the direct collaboration model, systems exchange their own information directly with each other in peer-to-peer manner. In the indirect collaboration, systems exchange their own information via some collaboration platform. In order to maintain and continue this collaboration, each system provide the metadata of their ability and interfaces and adopt itself to others.

Cross-domain collaboration

# Functional Requirements

This section defines the properties required in an abstract Web of Things (WoT) architecture.

## Flexibility

There are a wide variety of physical device configurations for WoT implementations. The WoT abstract architecture should be able to be mapped to and cover all of the variations.

## Compatibility

We already have many existing IoT solutions and ongoing IoT standardization activities in many business fields. The WoT should provide a bridge between these existing and developing IoT solutions and Web technology based on WoT concepts. The WoT should be upwards compatible with existing IoT solutions and current standards.

## Security and Privacy

This section will likely undergo significant revision and reorganization. Please see the [WoT Security and Privacy](https://github.com/w3c/wot-security/) repository for work in progress. In particular, the [WoT Threat Model](https://github.com/w3c/wot-security/blob/master/wot-threat-model.md) defines the main WoT security stakeholders, assets, attack surfaces, and threats.

The functional WoT architecture should enable the use of best practices in security and privacy. Generally, the WoT security architecture must support the goals and mechanisms of the IoT protocols and systems it connects to. These systems vary in their security requirements and risk tolerance, so security mechanisms will also vary based on these factors. However, the WoT architecture needs to do no harm: it should support security and privacy at least as well as the systems it connects to.

Security means the system should preserve its integrity and functionality even when subject to attack. Privacy means that the system should maintain the confidentiality of personally identifiable information. In general, security and privacy cannot be guaranteed but the WoT architecture should support best practices.

Security and privacy are especially important in the IoT domain since IoT devices need to operate autonomously and in many cases have access to both personal data and/or can be in control of safety-critical systems. Compared to personal systems, IoT devices are subject to different and in some cases higher risks than IT systems. It is also important to protect IoT systems so that they can not be used to launch attacks on other computer systems.

# Functional Requirements added by Toru Kawaguchi

This section defines the properties required in Web of Things (WoT) architecture.

## Principles

* WoT architecture should enable mutual interworking of different eco-systems using web technology.
* WoT architecture should be based on the web architecture using RESTful APIs.
* WoT architecture should allow to use multiple payload formats which are commonly used in the web.

## Thing functionalities

* WoT architecture should allow thing&apos;s to have functionalities such as
  + reading thing&apos;s status information
  + updating thing&apos;s status information which might cause actuation
  + subscribing to, receiving and unsubscribing to the notification of the change of thing&apos;s status information.
  + invoking functions with input and output parameters which would cause certain actuation or calculation.
  + subscribing to, receiving and unsubscribing to the notification which does not directly relate to thing&apos;s status information.

## Search and discovery

* WoT architecture should allow clients to know thing&apos;s attributes, functionalities and their access points, prior to access to the thing itself.
* WoT architecture should allow clients to search things by its attributes and functionalities.
* WoT architecture should allow semantic search of its functionalities based on unified vocabulary, regardless of naming of its functionalities.

## Description mechanism

* WoT architecture should support common description mechanism which enables describing things and their functions.
* Such description should be not only human-readable, but also machine-readable.
* Such description should allow semantic annotation of its structure and described contents.
* Such description should be able to be exchanged using multiple formats which are commonly used in the web.

## Description of attributes

* WoT architecture should allow describing thing&apos;s attributes such as
  + name
  + explanation
  + version of spec, format and description itself
  + links to other related things and information
* Such description should support internationalization.

## Description of functionalities

* WoT architecture should allow describing thing&apos;s functionalities which is shown in ## Thing functionalities

## Network

* WoT architecture should support multiple web protocols which are commonly used.
* Such protocols include 1)protocols commonly used in the internet and 2)protocols commonly used in the local area network
* WoT architecture should allow using multiple web protocols to access to the same functionality.
* WoT architecture should allow using mixture of multiple protocols to the functionalities of the same thing (e.g. HTTP and WebSocket).

## Deployment

* WoT architecture should support wide variety of thing capability such as edge devices with resource restrictions and virtual things on the cloud, based on the same model.
* WoT architecture should support multiple levels of thing hierarchy with intermediate entities such as gateways and proxies.
* WoT architecture should support accessing things in the local network from the outside of the local network (the internet or other local network), considering network address translation.

## Application

* WoT architecture should allow describing applications on the wide variety of things such as edge device, gateway, cloud and UI/UX device, using web standard technology based on the same model.

## Legacy adaption

* WoT architecture should allow mapping of legacy IP and non-IP protocols to web protocols, supporting wide variety of hierarchy, where such legacy protocols are terminated and translated.
* WoT architecture should allow transparent use of existing IP protocols without translation, which follow RESTful architecture.

## Security

* WoT architecture should support multiple security mechanism commonly used in the web, such as Basic, Digest, Bearer and OAuth2.0.

# WoT Building Blocks

This section presents the initial WoT building blocks that build up the abstract architecture for the Web of Things. This architecture is derived from the use cases in Section and the requirements in Section . summarizes the high-level goals and requirements and shows the three levels where the WoT building blocks can be applied:

* the device level,
* the gateway level (or "edge"), and
* the cloud level.

Abstract Architecture of W3C WoT

shows a conceptional view of how a component in the WoT Architecture makes use of the WoT building blocks. Each block is described in more detail in the following sections.

Conceptional Architecture of the WoT Building Blocks

## Thing

A [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) is the abstraction of a physical or virtual entity that needs to be represented in IoT applications. This entity can be a device, a logical component of a device, a local hardware component, or even a logical entity such as a location (e.g., room or building).

[Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) provide a network-facing API for interaction ([WoT Interface](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-interface)) based on a formal model. These [WoT Interfaces](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-interface) are a superset of Web APIs, as [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) can also be available over non-Web protocols such as MQTT or ZigBee. The outward-facing [WoT Interface](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-interface) is not to be confused with the [Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api), which is optional and interfaces with application scripts inside the software stack of a [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing).

There can be [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing), however, that do not provide a [WoT Interface](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-interface) and only consist of metadata that is relevant to the application (e.g., the room in which devices are located). In W3C WoT however, a [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) must have a [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td); therefore, everything that has a [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) is a [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing).

## WoT Thing Description

The [WoT Thing Description](https://w3c.github.io/wot-thing-description/) (TD) is structured data that adheres to a formal model and closes the gap between Linked Data vocabularies and functional APIs of IoT systems. It can be seen as the "HTML for Things". A [TD](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) provides general metadata of a [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) as well as metadata about the [Interactions](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-interaction), data model, communication, and security mechanisms of a [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing). Usually, [TDs](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) make use of domain-specific metadata for which WoT provides explicit extension points. However, any [domain-specific vocabulary](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-domain-specific-vocabulary) is out-of-scope of the W3C standardization activity.

The [WoT Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) is built around a formal [Interaction Model](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-interaction-model) that can support multiple messaging paradigms (i.e, request-response, publish-subscribe, and message passing). The default [Interaction Patterns](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-interaction-pattern) are [Property](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-property), [Action](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-action), and [Event](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-event). These were found to be able to cover the network-facing APIs provided by most IoT Platforms. [Properties](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-property) abstract data points that can be read and often written. [Actions](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-action) abstract invokable processes that may run for a certain time; yet they can also abstract RPC-like interactions in general. [Events](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-event) abstract interactions where the remote endpoint pushes data asynchronously.

[Thing Descriptions](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) are serialized to JSON-LD [[!JSON-LD]] by default. More serialization formats are planned in the future, in particular more concise formats for resource-constrained [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing). For now, JSON-LD offers a good trade-off between machine-understandable semantics and usability for developers.

[Thing Descriptions](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) can be managed in [Thing Directories](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-directory), which are aligned with the CoRE Resource Directory [[CoRE-RD]]. They provide a Web interface for registration, registration updates, and removal, and automatic removal after a given lifetime expired without registration update. [Thing Directories](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-directory) also provide a Web interface for lookups, usually including a SPARQL endpoint for semantic queries in addition to simple CoRE Resource Directory [[CoRE-RD]] lookups.

The [WoT Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) fosters interoperability in two ways: First, and foremost, [TDs](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) enable machine-to-machine communication in the Web of Things. Second, [TDs](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) can serve as a common, uniform format for developers to document and retrieve all details necessary to access IoT devices and make use of their data.

## WoT Binding Templates

A great challenge for the WoT is to enable interactions with a myriad of different IoT Platforms (e.g., OCF, oneM2M, RESTful devices not following any particular standard but providing an HTTP or CoAP interface, etc.). The IoT uses a variety of protocols for accessing devices, since no one protocol is appropriate in all contexts. W3C WoT is tackling this variety by including communication metadata in the [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td). This metadata can be used to configure the communication stack to produce compliant messages for a wide variety of target [IoT Platforms](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-iot-platform) and protocols.

The [WoT Binding Templates](https://w3c.github.io/wot-binding-templates/) are an informal collection of communication metadata blueprints that explain how to interact with different [IoT Platforms](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-iot-platform). When creating a [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) for a particular device, the [Binding Template](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-binding-templates) for the corresponding [IoT Platform](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-iot-platform) can be used and instantiated in the [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) for that device.

From Binding Templates to Protocol Bindings

shows how [Binding Templates](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-binding-templates) are applied. A [WoT Binding Template](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-binding-templates) is created only once for each [IoT Platform](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-iot-platform) and then instantiated, and hence reused in all [TDs](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) for its devices. The [WoT Client](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-client) consuming that [TD](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) must implement the corresponding [Protocol Binding](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) to be able to interact with the [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) described. The communication metadata of a Binding spans four dimensions:

* **IoT Platform:** [IoT Platforms](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-iot-platform) often introduce proprietary tweaks to protocols such as platform-specific HTTP header fields or CoAP options. This spans another dimension within the transfer protocol used. Usually, these tweaks can be described with standard [TD Vocabulary](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-td-vocabulary). If not, platform-specific vocabulary along with a corresponding [Protocol Binding](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-protocol-binding) implementation is required.
* **Transfer Protocol:** The [WoT Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-thing-description-td) comes with vocabulary to identify standard protocols (e.g., HTTP, CoAP, WebSockets, or MQTT). Their configuration (e.g., port or variations such as HTTP long-polling [[!RFC6202]] to support event notification) is also done through [TD Vocabulary](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-td-vocabulary). If a transfer protocol is not covered by the [TD Vocabulary](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-td-vocabulary), [domain-specific vocabulary](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-domain-specific-vocabulary) can be used. This, of course, needs to be backed by a corresponding [Protocol Binding](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-protocol-binding) implementation in the [WoT Client](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-client).
* **Media Type:** [IoT Platforms](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-iot-platform) often differ in the representation formats (or serializations) used for exchanging data. The Internet Meda Type [[!RFC6838]] identifies these formats.
* **Security:** Security can be applied at different places of the communication, and hence opens another dimension of variability. Examples are (D)TLS [[!RFC5246]]/[[!RFC6347]], IPSec [[!RFC6071]], OAuth [[!RFC6749]], and ACE [[!RFC7744]].

## WoT Scripting API

The [WoT Scripting API](https://w3c.github.io/wot-scripting-api/) is an optional building block that eases IoT application development. Traditionally, device logic is implemented in firmware, which underlies the productivity constraints of embedded development. The [WoT Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api) enables having a runtime system for IoT applications similar to a Web browser, and aims to improve productivity and reduce integration costs. Furthermore, standardized APIs enable portability for application modules, for instance, to move compute-intense logic from a device up to a local gateway, or to move time-critical logic from the cloud down to a gateway or edge node.

The [Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api) is built on top of the [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) abstraction and the TD [Interaction Model](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-interaction-model). There are three sub APIs:

* **WoT object** as API entry point to discover, consume, and expose [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-thing)
* **ConsumedThing interface** as Client API to consume [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-thing) over the network or locally (e.g., physically attached hardware)
* **ExposedThing interface** as Server API to configure and expose [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-thing) over the network

# WoT Servient Architecture

A [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) is a software stack that implements the WoT building blocks presented in the previous section. [Servients](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) can host and expose [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) and/or consume [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing). Thus, [Servients](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) can perform in both the server and client roles; the name (a portmanteau of server and client) is based on this dual role.

The previous shows how the WoT building blocks conceptionally relate to each other. When implementing these concepts, a more detailed view is necessary that takes certain technical aspects into account. The detailed architecture of a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) is shown in .

Implementation View of a Servient

The role and functionality of each module shown in is as follows:

## Application

[Applications](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-application) running on a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) are usually implemented through scripts (i.e., JavaScript). The application scripts must be provided along with security metadata that defines their [Execution Environment](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-execution-environment) and consequently how scripts must be isolated. The security metadata also needs to include keying material or certificates to authenticate the [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) the script exposes.

Note that the [WoT Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api) building block is optional. There can be minimal [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) implementations where applications are implemented natively for the software stack. These do not have the [Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api) and [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-runtime) modules.

## WoT Scripting API

The standardized [WoT Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api) is the contract between applications and the runtime system of a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient), the so-called [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-runtime). The [WoT Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api) is equivalent to any platform API, and hence there must be mechanisms to prevent malicious access to the system. As mentioned above, this building block, including the underlying WoT Runtime implementation, is optional.

See [WoT Scripting API](https://w3c.github.io/wot-scripting-api/) for details.

## WoT Runtime

The [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) abstraction and [Interaction Model](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-interaction-model) is implemented in a runtime system that offers the application-facing [WoT Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api). This [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-runtime) interfaces with the [Protocol Bindings](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) to access remote [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) and with the system API to access local hardware and proprietary means for communication. Note that both local hardware and devices behind proprietary communications protocols are also represented as [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) in the runtime environment, that is, they are also accessed through the [Client API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-client-api). The [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-runtime) is also tasked with generating the [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) based on the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) metadata (e.g., location), application metadata (e.g., provided [Interactions](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-interaction)), and available [Protocol Bindings](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) (e.g., implemented [WoT Binding Templates](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-binding-templates)).

## Protocol Bindings

[Protocol Bindings](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) are implementations of the [Binding Templates](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-binding-templates). They produce the messages to interact with [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) over the network based on the information given in the [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) of the [Consumed Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-consumed-thing). [Servients](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) usually have multiple [Protocol Bindings](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) to enable interaction with different [IoT Platforms](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-iot-platform).

In many cases, where standard protocols are used, generic protocol stacks can be used to produce the platform-specific messages (e.g., one for HTTP(S) dialects, one for CoAP(S) dialects, and one for MQTT solutions, etc.). In this case, the communication metadata from the [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) is used to select and configure the right stack (e.g., HTTP with the right header fields or CoAP with the right options). Parsers and serializers for the expected representation format identified by the Internet Media Type can also be shared across these generic protocol stacks.

In some cases, where no aspects can be shared, the [Protocol Binding](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) is more comparable to a platform-specific driver that is selected and configured through the communication metadata in similar way as above.

See [WoT Binding Templates](https://w3c.github.io/wot-binding-templates/) for details.

## System API

The implementation of a [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) may access local hardware or system services (e.g. storage) through proprietary APIs or other means. This block is out of scope of WoT standardization.

A [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-runtime) may provide local hardware or system services to application scripts through the [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) abstraction, as if they were accessible over a network protocol. In this case the implementation should be optimized to avoid the overhead that would be involved in a literal implementation of a network protocol while maintaining a consistent [WoT Interface](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-interface). The details of such "System Things" are out of the scope of standardization at present, although W3C WoT will document several informational examples.

A device may be physically external to a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient), but connected via proprietary protocols. In this case, the implemented runtime environment may access legacy devices with such protocols (e.g., Echonet Lite, X10, I2C, SPI) through proprietary APIs, but again exposes them to applications as [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) through the [Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api). A script can then act as gateway to the legacy devices. This should only be done if the legacy device cannot be described using the [WoT Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td).

## WoT in the Web Browser

This section is an early outline to realize WoT with the existing browser APIs. More details and a native WoT integration into the Web browser will be added as the WG drafts progress.

shows how a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) implementation for Web browsers would look like.

Implementing WoT in the Web Browser

The Web browser implicitly isolates applications in tabs using the [same-origin policy](https://www.w3.org/Security/wiki/Same_Origin_Policy). Thus, the security metadata is not mandatory. The application scripts would be part of a Web page that can provide visualization and user interaction.

The [WoT Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api) needs to be added by a WoT library loaded together with the application scripts by the Web page. This library would also implement [TD](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) handling (i.e., parsing for consuming [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) and generating for exposing [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing)) and provide glue code to use the browser APIs. The other aspects of the [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-runtime) are provided by the browser JavaScript runtime system.

The [Protocol Bindings](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) are limited to the protocols implemented by Web browsers. These are:

* HTTP(S) (1.0/1.1/2.0) [[!RFC7230]]/[[!RFC7540]] (including Server-Sent Events [[!HTML5]]),
* WebSockets [[!RFC6455]] (which enables the WoT library to implement MQTT-over-WebSockets and CoAP-over-WebSockets)
* (S)RTP of WebRTC [[!WEBRTC]], and
* FTP [[!RFC765]] (which might be of limited use similar to the file protocol)

The other browser APIs (e.g., Geolocation, Vibration, and Web Storage) are comparable to the System API of normal [Servients](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) and can enable access to local hardware.

# WoT Deployment Scenarios and Guidelines

There are many possibilities for deploying the abstract WoT Architecture and mapping the functions to physical devices and network structures. This section, which is informative but not normative, lists a number of design patterns that may be used when implementing the Web of Things.

## WoT Client

Servient Consuming Thing

shows the basic scenario, where a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) in client role, a [WoT Client](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-client), is consuming an existing device that is augmented with a [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) (TD). The latter could be generated from digital descriptions already available for the device within the corresponding the [IoT Platform](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-iot-platform). It could also be written by a developer using tools or even manually.

The Application Script is loaded and executed by the [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-runtime). Scripts can be manually configured with the URI of the device [TD](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) located on a reachable Web server. Using the [WoT object](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-object), the Application Script retrieves the [TD](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) and instantiates a corresponding [Consumed Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-consumed-thing). The Application Script can use the metadata of the [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) and inspect what [Interactions](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-interaction) it provides. The communication metadata within the [TD](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) is used by the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) to select the matching [Protocol Binding](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) and to configure its protocol stack. These communication details are hidden from the Application Script. However, the script can only interact with the [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing), if the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) implements a [Protocol Binding](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) that matches the communication metadata given in the [TD](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td).

Using the [Client API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-client-api) (i.e., ConsumedThing interface), the Application Script can read or write a [Property](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-property), invoke an [Action](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-action), or subscribe for [Events](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-event) offered by the [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing). The selected [Protocol Binding](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) maps these [Interactions](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-interaction) to the low-level protocol operations and serializations understood by the remote [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing). When a message is returned by the [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing), the [Protocol Binding](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-protocol-binding) parses the response and maps it back to the [Interaction](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-interaction) abstraction. The [Interaction](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-interaction) output is delivered back to the Application Script by resolving a (JavaScript) Promise.

Each sub-section shall describe the technical details to realize the scenario. They shall describe how to discover the involved WoT components, how to realize connectivity, and what security mechanisms can be used. For now, the draft only contains strawman proposals and placeholders that will be replaced as the draft progresses.

Discovery

* Manual (Application Script is configured with TD URI)

Connectivity

* Local over (W)LAN or LPWANs (ZigBee, Z-Wave, etc.)
* Remote over globally reachable IP address of the Thing

Security

* Security metadata to sandbox the Application Script
* Access control implemented on the Thing
* Integrity protection when retrieving the TD
* Potentially confidentiality and integrity protection when interacting with the Thing

## Servient on Device

Servient on Device Itself

In this case, a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) is running on the device itself. The right most [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) in shows an LED Light, whose controller has a powerful CPU and a large memory and is able to provide web server functionality connected directly to the Internet. Then the leftmost browser and/or another application on the internet can access the LED light through the Internet directly.

Resource-constrained Device as Thing

Devices that are not powerful enough to host a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) can still act as [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing). In this case, a classic firmware is providing a [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) that describes the functionality and protocols implemented. The right most [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) in shows a legacy device that has a constrained CPU and a small memory and is able to provide web server functionality connected directly to the internet. Then the leftmost browser and/or another application on the internet can access the device through the Internet directly like .

Discovery

* WoT Client discovers [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-servient) on the same network [network discovery].

Connectivity

* (W)LAN
* LPWANs (ZigBee, Z-Wave, etc.)

Security

* t.b.d.

## Servient on Smartphone

This example uses a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) running on a Smartphone. Smartphones are not only popular but have enough performance to provide gateway functionality. This functionality can be used to bridge between the internet and a legacy device without any intermediate hardware.

Servient on Smartphone

shows an example of a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on a smartphone, which can act as a gateway to existing devices (e.g., via Bluetooth or local Wi-Fi). The Web browser with the user interface can either run on the smartphone directly or remotely on a separate device (such as a laptop).

Discovery

* WoT Client discovers an electronic appliance when the remote controller is nearby [nearby discovery].
* WoT Client discovers [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-servient) remotely when the remote controller is outside [remote discovery].

Connectivity

* t.b.d.

Security

* t.b.d.

## Servient on Gateway

Servient on Gateway

shows an example of a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on a gateway. Gateway are often introduced as a home automation and/or home energy management solution. In the case of consumer electronics, there are very wide variety of physical communication formats such as WiFi, 802.15.4g, Bluetooth Low Energy, HDPLC and so on. In order to normalize those variations, almost all home automation systems introduce a gateway. In , a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) wraps various mechanisms for communicating with legacy devices and provides to other clients a universal device accessing method. Inside the home, HTTP/TCP/IP/WiFi can then be used as the sole unfied communication method between the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on the gateway and a user interface device such as a Web broswer.

Discovery

* [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-servient) discovers electronic appliances nearby [nearby discovery].
* WoT Client discovers [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-servient) remotely [remote discovery].

Connectivity

* t.b.d.

Security

* t.b.d.

## Servient on Cloud and Gateway

Client Apps can control devices at home through a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on a gateway. But in this case the location of client apps is restricted to the home, because physical communication path "WiFi" and/or wired Ethernet between gateway and client apps such as a Web browser is limited to the physical domain provided by the WiFi signal. To provide for controlling devices at home from outside the house, a HTTP/TCP/IP interface to a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) running in the cloud with a globally reachable address could be used. However, in this case the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) in the cloud cannot generally access devices running in the home through only local interfaces such as Bluetooth.

Servient on Cloud Server and Gateway

shows an example of [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) running on a cloud server paired with another [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) running on a gateway. In the case of , a browser accesses the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on the cloud Server named "Cloud". This [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) provides its interface through the Internet globally. So, wherever a browser user is, they can access this [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient). The [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on "Cloud" can accept the request of the browser and/or other application through HTTP, CoAP, and so on. Then the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on the cloud server finds out the route to access another [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on the gateway. After finding out the route, the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on the cloud server transfers the request from the browser to the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on the gateway. After that, the gateway processes the request according the use case. The [Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) of the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on the cloud server can be just a mirror of that on the gateway, since it will generally just pass interactions directly through. More generally, though, one or both [Servients](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) can provide services such as privacy filtering or sub-setting. When the user is home, they can also access the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) in the gateway directly.

Discovery

* [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-servient) discovers WoT Server remotely [remote discovery].
* WoT Client discovers [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-servient) remotely [remote discovery].

Connectivity

* t.b.d.

Security

* t.b.d.

## Servient on Cloud Server

Servient on Cloud Server Only

shows a second example of [Servients](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) in the cloud. In this case, a browser accesses a [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) on a cloud server, similar to . This [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) provides access through the global Internet. So, wherever the browser user is, they can access this [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient). The cloud [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) accepts the requests of the browser and/or other applications through HTTP, CoAP, etc. Then it finds out the route to access a proprietry discovery service running on a gateway. In , the [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) running in the cloud could talk to another [Servient](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) running on the gateway. However, many service providers have already provided IoT services using proprietary IoT interfaces or some other IoT standard. In this case, the gateway can still support the same functionalities, as in the previous case, but using methods outside of the WoT definitions.

Discovery

* Servient1 discovers Servient2 remotely [remote discovery].
* Servient2 discovers electronic appliances nearby [nearby discovery].
* WoT Client discovers Servient1 remotely [remote discovery].

Connectivity

* t.b.d.

Security

* t.b.d.

# Security and Privacy Considerations

Security and privacy considerations are still under discussion and development; the content below should be considered preliminary. Due to the complexity of the subject we are considering producing a separate document containing a detailed security and privacy considerations discussion including a risk analysis, threat model, recommended mitigations, and appropriate references to best practices. A summary will be included here. Work in progress is located in the [WoT Security and Privacy](https://github.com/w3c/wot-security/) repository. Please file any security or privacy considerations and/or concerns using the [GitHub Issue](https://github.com/w3c/wot-security/issues) feature.

Security is a cross-cutting issue that needs to be taken into account in all [WoT building blocks](#sec-building-blocks). The W3C WoT does not define any new security mechanisms, but provides guidelines to apply the best practices from Web security, IoT security, and information security for general software and hardware considerations.

The [WoT Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) must be used together with integrity protection mechanisms and access control policies. Users must ensure that no sensitive information is included in the [TDs](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-td) themselves.

The [WoT Binding Templates](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-binding-templates) must correctly cover the security mechanisms employed by the underlying [IoT Platform](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-iot-platform). Due to the automation of network interactions necessary in the IoT, operators need to ensure that [Things](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) are exposed and consumed in a way that is compliant with their security policies.

## WoT Runtime Security and Privacy Risks

The [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-wot-runtime) implementation for the [WoT Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api) must at least have mechanisms to prevent malicious access to the system and isolate scripts in multi-tenant [Servients](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient). More specifically the [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) implementation for the [WoT Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-scripting-api) should take into account the below security and privacy risks and implement the recommended mitigations.

##### Cross-Script Security and Privacy Risk

In basic WoT setups, all scripts running inside the [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) are considered trusted, and therefore there is no strong need to perform strict isolation between each running script instance. However, depending on device capabilities and deployment use case scenario risk level it might be desirable to do so. For example, if one script handles sensitive privacy-related data and well-audited, it might be desirable to separate it from the rest of the script instances to minimize the risk of data exposure in case some other script inside WoT gets compromised during runtime. Another example is mutual co-existence of different tenants on a single WoT device. In this case each WoT runtime instance will be hosting a different tenant, and isolation between them is required.

Mitigation:

The WoT runtime should perform isolation of script instances and their data in cases when scripts handle privacy-related or other critical security data. Similarly, the [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) implementation should perform isolation of [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) instances and their data if a WoT device has more than one tenant. Such isolation can be performed within the [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) using platform security mechanisms available on the device. For more information see Sections "WoT Servient Single-Tenant" and "WoT Servient Multi-Tenant" of [[!WOT-SECURITY-CONSIDERATIONS#]].

##### Physical Device Direct Access Security and Privacy Risk

In case a script is compromised or misbehaving, the underlying physical device (and potentially surrounded environment) can be damaged if a script can use directly exposed native device interfaces. If such interfaces lack safety checks on their inputs, they might bring the underlying physical device (or environment) to an unsafe state (i.e. device overheats and explodes).

Mitigation:

The [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) should avoid directly exposing the native device interfaces to the script developers. Instead a [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) implementation should provide a hardware abstraction layer for accessing the native device interfaces. Such hardware abstraction layer should refuse to execute commands that might put the device (or environment) to an unsafe state. Additionally, in order to reduce the damage to a physical WoT device in cases a script gets compromised, it is important to minimize the number of interfaces that are exposed or accessible to a particular script based on its functionality.

##### Provisioning and Update Security Risk

If the [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) implementation supports post-manufacturing provisioning or updates of itself, WoT scripts, or any related data (including security credentials), it can be a major attack vector. An attacker can try to modify any above described element during the update or provisioning process or simply provision attacker's code and data directly.

Mitigation:

Post-manufacturing provisioning or update of scripts, the [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) itself or any related data should be done in a secure fashion. A set of recommendations for secure update and post-manufacturing provisioning can be found in [[!WOT-SECURITY-CONSIDERATIONS]].

##### Security Credentials Storage Security and Privacy Risk

Typically the [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) needs to store the security credentials that are provisioned to a WoT device to operate in WoT network. If an attacker can compromise the confidentiality or integrity of these credentials, then it can obtain access to the WoT assets, impersonate WoT things or devices or create Denial-Of-Service (DoS) attacks.

Mitigation:

The [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) should securely store the provisioned security credentials, guaranteeing their integrity and confidentiality. In case there are more than one tenant on a single WoT-enabled device, a [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) implementation should guarantee isolation of each tenant provisioned security credentials. Additionally, in order to minimize a risk that provisioned security credentials get compromised, the [WoT Runtime](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-runtime) implementation should not expose any API for scripts to query the provisioned security credentials.

# Summary

An abstract architecture for the Web of Things and a functional architecture for [Servients](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) has been introduced. It is based on WoT building blocks, which are to be covered by additional WoT specifications:

* the [WoT Thing Description](https://w3c.github.io/wot-thing-description/),
* the [WoT Binding Templates](https://w3c.github.io/wot-binding-templates/), and
* the [WoT Scripting API](https://w3c.github.io/wot-scripting-api/).

Of these, the [WoT Thing Description](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing-description-td) is the primary building block, as it describes the network-facing interface of a Thing ([WoT Interface](https://github.com/w3c/wot-architecture/blob/master/terminology.md" \l "user-content-wot-interface)), whether or not it uses [WoT Binding Templates](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-binding-templates) or the [WoT Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api) internally. The implementation of [WoT Binding Templates](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-binding-templates) results in multiple possible Protocol Bindings, so that a [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) can communicate with different [IoT Platforms](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-iot-platform) (i.e., IoT ecosystems or standards). When a [Thing](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-thing) uses the [WoT Scripting API](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-scripting-api) internally, its application logic can be programmed against a standardized contract using JavaScript. This way, it simplifies IoT application development and enables portability across vendors and WoT network components.

The architecture described here applies well to a wide variety of different use cases. Based on these, we have described serveral scenarios where one or more [Servients](https://github.com/w3c/wot-architecture/blob/master/terminology.md#user-content-servient) were used together to overcome, for instance, limitations on the reach of specific communication protocols. These examples are not exhaustive and are only meant to illustrate the applicability and flexibility of the WoT approach.

# Example Sequences of interaction patterns

This section is non-normative.

This section illustrates example sequences of interaction patterns such as read propery, invoke action and event, among WoT client and WoT server.

For the sake of simplicity, remote and local proxies between client and server are omitted from following sequences.

## Property

### Read property (binding = HTTP)

Following sequence illustrates example sequence of read property operation with HTTP binding.

### Write property (binding = HTTP)

Following sequence illustrates example sequence of write property operation with HTTP binding.

### Observe property (binding = HTTP Long Polling)

Following sequence illustrates example sequence of observe property operation with HTTP Long Polling binding.

### Observe property (binding = HTTP Server Sent Event)

Following sequence illustrates example sequence of observe property operation with HTTP Server Sent Event binding.

### Observe property (binding = Simple WebSocket)

Following sequence illustrates example sequence of observe property operation with simple WebSocket binding.

## Action

### Invoke action (binding = HTTP)

Following sequence illustrates example sequence of invoke action operation with HTTP binding, where operation is synchronous and the request to server is blocked until action completes.

Note: Action with asynchronous operation is subject to further study.

## Event

### Subscribe, notify and unsubscribe event (binding = HTTP Long Polling)

Following sequence illustrates example sequence of subscribe, notify and unsubscribe event operation with HTTP Long Polling binding.

### Subscribe, notify and unsubscribe event (binding = HTTP Server Sent Event)

Following sequence illustrates example sequence of subscribe, notify and unsubscribe event operation with HTTP Server Sent Event.

### Subscribe, notify and unsubscribe event (binding = Simple WebSocket)

Following sequence illustrates example sequence of subscribe, notify and unsubscribe event operation with simple WebSocket binding.

## Further things to consider

* CoAP, MQTT bindings
* Webhook binding
* Multiplexing of event
* Action and event with descriptors

## Acknowledgements

Special thanks to all active Participants of the W3C Web of Things Interest Group and Working Group for their technical input and suggestions that led to improvements to this document.