Web of Things (WoT): WoT Profile - Strawman Proposal

W3C Document 17 March 2020

Latest editor's draft:
https://w3c.github.io/wot-profile/

Editor:
Michael Lagally (Oracle Corp.)

Contributors:
In the GitHub repository

Repository:
We are on GitHub
File a bug
Contribute

Copyright © 2018-2020 W3C® (MIT, ERCIM, Keio, Beihang). W3C liability, trademark and permissive document license rules apply.

Abstract

The W3C WoT Thing Architecture [wot-architecture] and WoT Thing Description [wot-thing-description] define a powerful description mechanism and a format to describe myriads of very different devices, which may be connected over various protocols. The format is very flexible and open and puts very few normative requirements on devices that implement it.

However, this flexibility de-facto prevents interoperability, since, without additional **rules**, it allows implementers to make many choices that do not provide **guarantees** of common behavior between implementations.

These rules have to be prescriptive, to ensure that compliant implementations satisfy the semantic guarantees, that are implied by them. We call this set of rules a Profile.

The **WoT Profile Specification** as defined in the present document serves two purposes:

- It defines a generic **Profiling Mechanism** which provides a mechanism to describe a profile in an unambiguous way. This mechanism can be used to define additional profiles.

- In addition it defines a **Core Profile** of the Thing Description for use with the HTTP(S) and selected notification protocols. The Core Profile formalizes the results of several plug-fests that were conducted by the WoT Interest Group and of tests that were conducted as part of the development. It is expected that additional profiles for thing templates and other protocols will be defined in the near future.

Devices that constrain their use of the Thing Description to the WoT Core Profile can interoperate with each other out-of-the-box.

A later version of this document will define additional profiles:

- **Thing Template Profile**

  The Thing Template Profile enables to apply the vocabulary of the Thing Description Specification for defining a common interface, that can be implemented by different things. It selects an appropriate subset to describe **classes** of things that share the same data model.

- **Digital Twin Profile**

  The Digital Twin Profile allows defining a virtual model of a thing or a group of things.

Profiles are is not exclusive, they can overlap or include others.

Status of This Document

This document is merely a W3C-internal document. It has no official standing of any kind and does not represent consensus of the W3C Membership.

Table of Contents

1. **Introduction**
1.1 Deployment Scenarios
1.2 Why a Core Profile?
1.3 Out-of-the-box interoperability
1.4 Structure of this document

2. **Terminology**

3. **Profiling Mechanism**
3.1 Methodology

4. WoT Core Profile
4.1 WoT Core Data Model
4.1.1 General
4.1.1.1 Mandatory fields
4.1.1.2 Length and Value Limits
4.1.2 Thing
4.1.2.1 Mandatory fields
4.1.2.2 Recommended practice
4.1.3 Data Schemas
4.1.4 Property Affordance
4.1.4.1 Mandatory fields
4.1.4.2 Additional Constraints
4.1.4.3 Recommended Practice
4.1.5 Interaction Affordance
4.1.5.1 Mandatory fields
4.1.5.2 Additional Constraints
4.1.5.3 Recommended Practice
4.1.6 Event Affordances
4.1.6.1 Mandatory fields
4.1.6.2 Additional Constraints
4.1.7 Forms
4.1.7.1 Mandatory fields
4.1.7.2 Additional Constraints
4.1.8 Links
4.1.9 Security
4.1.9.1 Recommended Practice
4.2 Protocol Binding
4.2.1 HTTP Protocol Binding
4.2.1.1 Properties
4.2.1.2 Actions
4.2.1.3 Events
4.3 External TD representations

5. Open Issues

A. References
A.1 Informative references

1. Introduction §

The W3C WoT Architecture [wot-architecture] and the WoT Thing Description [wot-thing-description] (https://w3c.github.io/wot-thing-description) has been developed as a versatile format, that allows describing the interactions between multiple devices and protocols.

This flexibility permits an easy integration of new device types and protocols, however it risks interoperability, since there are *no guarantees* that two devices which are formally spec-compliant, will be able to communicate.

To increase adoption of the WoT specifications, interoperability between on premise devices, edge devices and the cloud is essential. Even if every manufacturer is implementing the current Thing Description specification in full flexibility, there is no interoperability guarantee; many choices are still left to the implementations and there are very few normative requirements that a device has to fulfill.

1.1 Deployment Scenarios §

A Thing Description can be used in two fundamentally different deployment scenarios:

- a "brown-field" scenario, where it is created to describe the interactions with existing systems.
- a "green-field" scenario, where a device model and a thing description are developed together.

For green field deployments, where the implementations are being carried out and corresponding thing descriptions are being created, it is easier to achieve full interoperability by using a small, extensible Core Profile.

In the brown field area, due to the nature of existing deployments and protocols, a broad spectrum of variations and potentially high complexity of thing descriptions inhibits interoperability and will most likely lead to additional profiles of the TD and domain-specific thing consumer implementations.

The WoT Core Profile can be used by green field deployments and gives guidance to new implementers of the WoT specifications. It has already proved in brown-field scenarios in the plug-fests, where existing devices, that already existed as products, prototypes or demonstrators, were described with Thing Descriptions that are constrained to the Core Profile.

1.2 Why a Core Profile? §

During the recent WoT plug-fests there were many de-facto agreements on the use of a small constrained subset of interaction patterns and protocol choices. These de-facto agreements select a common subset of the WoT Thing Description specification, based on proven interoperability among manufacturers.

The aim of this specification is to formalize these agreements by defining a **WoT Core Profile** based on the choices that were made by the implementers of plug fest devices.

The WoT Core Profile contains additional normative requirements that *MUST* be satisfied by devices to be compliant to the profile.

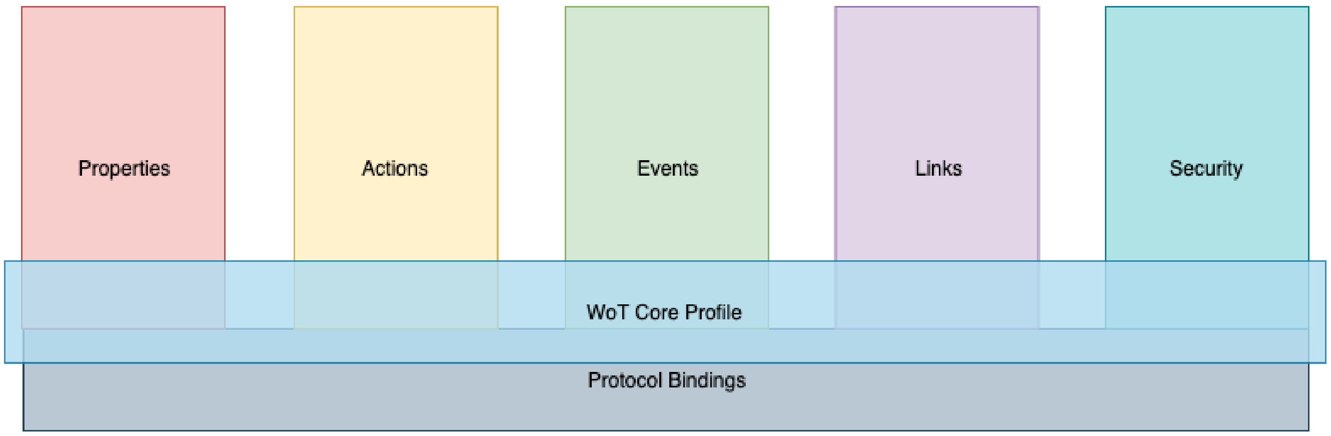

Figure 1 WoT Core Profile

Adoption of the WoT Core Profile will significantly limit the implementation burden of device and cloud implementors.

The WoT Core Profile was defined with the following main goals:

- guarantee interoperability among all implementations of the profile.
- limit the implementation complexity for resource constrained devices.
- ensure that a thing description is also useful for a human reader.

It makes choices on the required metadata fields as well as the supported interactions and protocol endpoints. It introduces some constraints on data schemas for properties and actions which are required for resource constrained devices in real-world deployments. The format does not forbid the use of additional elements of the TD for vendor specific extensions, however this will impact interoperability.

1.3 Out-of-the-box interoperability §

Devices, which implement the Core Profile, are **out-of-the-box interoperable** with other Core Profileem> compliant devices. Furthermore, the Core Profile simplifies device validation and compliance testing since a corresponding conformance test suite can be defined.

NOTE
It is also a goal of the WoT Core Profile to ensure that compliant files can be understood by humans - therefore descriptions, dates, and author fields are either mandatory or highly recommended.

1.4 Structure of this document §

EDITOR'S NOTE
to be added.

2. Terminology §

This specification uses the same terminology as the WoT Architecture and Thing Description specifications.

For convenience of the reader, we use the terms *keyword* and *field* for the linguistic notion *vocabulary term* as defined in the Thing Description Specification.

We use the terms *device* and *thing* in an interchangeable manner.

The keywords "*MUST*", "*MUST NOT*", "*REQUIRED*", "*SHALL*", "*SHALL NOT*", "*SHOULD*", "*SHOULD NOT*", "*RECOMMENDED*", "*MAY*", and "*OPTIONAL*" in this document are to be interpreted as described in RFC 2119. (https://www.ietf.org/rfc/rfc2119.txt)

Additional Definitions: §

Digital Twin Profile
A profile for defining a virtual representation of a thing or a group of things.

Thing Template Profile
A profile for defining a class of things that share the same Data Model.

TD Specification

Synonym for The WoT Thing Description Specification.

Thing Description Specification
: The WoT Thing Description Specification [wot-thing-description].

Thing Description
: A Data Model that conforms to the Thing Description specification [wot-thing-description].

Core Data Model
: A Data Model that conforms to the subset of the Thing Description specification [wot-thing-description] as defined in section .

Core Profile
: Synonym for WoT Core Profile.

Core TD
: Synonym for Core Thing Description.

Core Thing Description
: A Thing Description that conforms to the Core Profile.

Profile
: A set of prescriptive rules, to ensure that compliant implementations satisfy the semantic guarantees that are implied by them.

WoT Core Profile
: The subset of the Thing Description specification defined by the present document.

3. Profiling Mechanism §

This section describes a generic mechanism to define a profile of the TD specification in a unambiguous way.

The W3C WoT Thing Description specification defines a formal language, i.e. a set of vocabulary terms (keywords), a set of classes that are built from these keywords, and a set of additional rules, that define constraints on permitted values and keyword presence (mandatory / optional) dependent on the context where the keyword is used. In addition the TD specification defines relationships and corresponding cardinalities between these classes.

The TD specification already has some constraints, but there is a wide variety of variations that are left to the interpretation or the discretion of an implementer. The rationale for the Core Profile is not to forbid complex things, rather to enable statements like:

- *If you constrain your TD to the Core Profile, all other devices that conform the Core Profile can interoperate with it out of the box.*

- *If you have additional needs your device is free to implement other profiles or add-ons at your own choice, but other devices that only implement the Core Profile will most likely not be able to use these additions."*

3.1 Methodology §

A profile is a set of constraints and rules, which provide additional semantic guarantees that are applied to the TD specification. These constraints define a subset of valid Thing Descriptions by defining additional rules on various aspects of the Thing Description specification.

Constraints on	*Rationale*	*Example*
vocabulary of TD classes	guaranteed set of metadata fields	Make specific vocabulary terms mandatory, remove others
class relationships	unambiguous structure	limited cardinality, e.g. only one form per operation per interaction affordance.
values of vocabulary terms	simplified processing	Limit the length of characters per string, Always use arrays, where the spec permits a string or an array of strings.
data schemas	simplified processing	No arbitrary nested objects or arrays of arrays
security	reduced implementation effort	Only a restricted set of security mechanisms
protocol binding	guaranteed protocol semantics	limited protocol(s) and protocol features, predefined mapping of http verbs (GET/PUT) to operation verbs

These constraints and rules fall into two categories:

- constraints for the data model.
- constraints for the protocol binding.

These two categories are orthogonal to each other, i.e. a data model that conforms to a profile can be mapped to different protocols. The protocol binding for each protocol may contain additional (protocol-specific) constraints.

A profile is not exclusive, i.e. a thing may conform to multiple profiles. Profiles can build on top of each other or overlap, extended profiles can be built on top of the core profile.

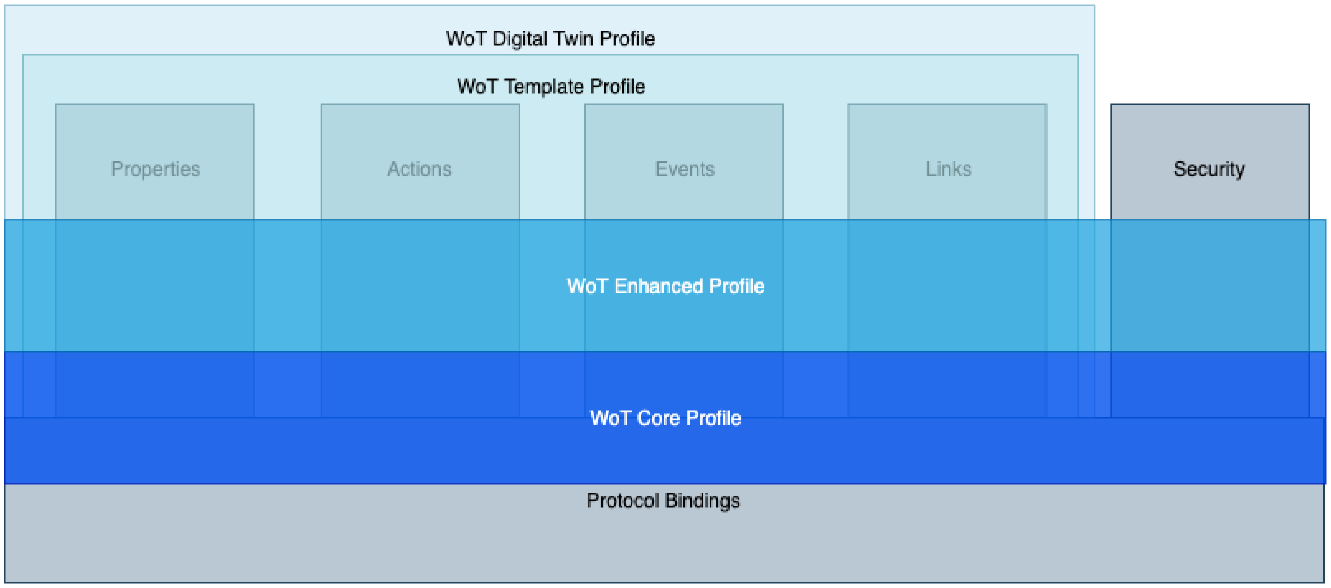

Figure 2 WoT Core Profile

In the present document, we define a Core Profile by defining a Core Data Model and a set of Protocol Binding Rules for selected protocols.

4. WoT Core Profile §

This section defines the Core Profile by defining a Core Data Model and a set of Protocol Binding Rules.

4.1 WoT Core Data Model §

4.1.1 General §

The following rules are applicable to multiple classes of the TD Specification, as they provide clearer semantics, improved readability and simplified processing on resource constrained devices.

4.1.1.1 Mandatory fields §

One of the primary benefits of the WoT Thing Description over a typical IoT format is the additional documentation for a human reader.

Therefore the fields `title` and `description` are MANDATORY for Things, Property Affordances, Action Affordances, Event Affordances and Data Schemas.

It is possible to have empty values for these fields, if, for specific purposes it is not desired to provide documentation, however this is *NOT RECOMMENDED* and the conscious decision is obvious from the TD.

4.1.1.2 Length and Value Limits §

The length of `id` , `description` and `descriptions` values is limited to 512 characters.

The length of `title` and `titles` values is limited to 64 characters.

Where a type permits using an `array of string` or a `string` , an `array of string` *MUST* be used.

Where a type permits using an `array of DataSchema` or a `DataSchema` , an `array of DataSchema` *MUST* be used.

All elements of an `enum` *MUST* be either `string` or `number` . Different types in a single `enum` are NOT PERMITTED.

4.1.2 Thing §

The Core Data Model applies the following constraints and rules to the Thing class of section 5.3.1.1 of the TD Specification.

4.1.2.1 Mandatory fields §

To provide minimum interoperability, the following metadata fields of a Thing *MUST* be contained in an Core TD:

keyword	*type*	*remarks*

keyword	*type*	*remarks*
title	string	human readable documentation
id	urn_type	a globally unique urn of the thing
description	string	human readable documentation
created	date	human readable documentation
modified	date	human readable documentation
support	urn_type	human readable documentation
security	array of string	simplified handling
version	VersionInfo	clear versioning, easy to compare different TDs

4.1.2.2 Recommended practice §

It is *RECOMMENDED* to use the value "Undefined" for strings, where the value cannot be determined.

If a Thing Description is used solely within a company, the email address of the developer *SHOULD* be used in the support field, if the IF-TD is provided externally, a support email address *SHOULD* be used.

It is *RECOMMENDED* to use the following field names for additional fields with specific semantics:

keyword	*type*	*remarks*
serialNumber	string	
hardwareRevision	string	
softwareRevision	string	
loc_latitude	number	Text string representation of latitude as defined in annex H of [ISO-6709]

keyword	*type*	*remarks*
loc_longitude	number	Text string representation of longitude as defined in annex H of [ISO-6709]
loc_altitude	number	Text string representation of altitude as defined in annex H of [ISO-6709]
loc_height	number	Text string representation of height as defined in annex H of [ISO-6709]
loc_depth	number	Text string representation of depth as defined in annex H of [ISO-6709]

If a location is provided, loc_latitude and loc_longitude *MUST* be present. Only one of loc_altitude, loc_height and loc_depth *MAY* be present. The use of loc_altitude is *NOT RECOMMENDED*, since it refers to the *mean sea level*, which is not a globally unique reference point.

NOTE

See https://www.science20.com/news_articles/what_happens_bridge_when_one_side_uses_mediterranean_sea_level_and_another_north_sea-121600 for a justification why loc_altitude is problematic.

4.1.3 Data Schemas §

Data Schemas are used for the values of Properties, Action input and output parameters and Event message payloads. The value of a *Data Schema* can be a simple type (boolean, integer, number, string) or an instance of a structured type (array and object).

The Core Data Model applies the following constraints and rules to the `DataSchema` class of section 5.3.2.1 of the TD Specification.

This section defines a subset of the class `DataSchema` that can be processed on resource-constrained devices.

Data Schema Constraints §

The Core Data Model restricts the use of arrays and objects to the **top level** of Data Schemas, i.e. only a one-level hierarchy is permitted. The members of a top level `object` or `array` *MUST NOT* be array or object types.

NOTE: RATIONALE

This may appear as a severe limitation, however it is motivated by integrating with multiple cloud services. Many enterprise services and applications are based on (relational) databases, where individual property values are stored. Of course databases can also store objects (e.g. encoded as a JSON string), however this will prevent processing by other enterprise applications.

If a property conceptually has a deeper structure, such as grid of lamps with RGB colors, the structure can be represented in the keyword of the property, i.e. lamp1_color_r, lamp1_color_g and lamp1_color_b. A similar mapping can be done for arrays and hierarchical objects. This constraint leads to simpler Thing Descriptions that can be handled by very limited devices.

The following fields *MUST* be contained in a DataSchema:

keyword	*type*	*constraints*
description	human readable description	
type	string	one of boolean, integer, number, string, object or string

The values `object` , `array` , or `null` *MUST NOT* be used in the type field.

4.1.4 Property Affordance §

The Core Data Model applies the following constraints and rules to the `PropertyAffordance` class of section 5.3.1.3 of the TD Specification.

4.1.4.1 Mandatory fields §

The following property fields *MUST* be contained in the `properties` element of a *Core TD*:

keyword	*type*	*constraints*
title	string	unique name among all properties
description	string	human readable description
type	string	one of `boolean` , `string` , `number` , `integer` , `object` or `array` . The type value `null` *MUST NOT* be used.

4.1.4.2 Additional Constraints §

The *Thing Description* permits arbitrary object depths for properties. Parsing of a deeply nested structure is not possible on resource constrained devices. Therefore each property *MAY* contain `array` or `object` elements only at *top level*.

The following additional constraints *MUST* be applied to the Property Affordances of a Thing Description conforming to the Core Profile:

keyword	*type*	*constraint*
const	anyType	*MUST NOT* be used
enum	array of simple type	*Values* of enums *MAY* only be simple types. Handling of *any type* is too complex to implement on resource constrained devices
forms	array of Forms	The `Array of Form` of each property *MUST* contain only a *single endpoint* for each operation `readproperty` , `writeproperty` , `observeproperty` , `unobserveproperty`.
format	string	If the field `format` is used, only formats defined in section 7.3.1-7.3.6 of [JSON-SCHEMA] *MAY* be used.
oneOf	string	The DataSchema field `oneOf` does not make sense for properties and *MUST NOT* be used.

keyword	*type*	*constraint*
uriVariables	Map of DataSchema	`uriVariables` *MUST NOT* be used.

4.1.4.3 Recommended Practice §

It is highly *RECOMMENDED* to always specify a `unit` , if a value has a metric. Authors of *Thing Descriptions* should be aware, that units that are common in their geographic region are not globally applicable and may lead to misinterpretation with drastic consequences.

The field `unit` could be used for non-decimal numeric types as well, e.g. a string value with binary or hex data (`0xCAFEBABE` , `0b01000010`), where the unit is `hex` or `bin` , to indicate how the value should be interpreted. It is strongly *RECOMMENDED* to use the values `hex` , `oct` or `bin` in this case to achieve interoperability.

4.1.5 Interaction Affordance §

The Core Data Model applies the following constraints and rules to the `ActionAffordance` class of section 5.3.1.4 of the TD Specification.

4.1.5.1 Mandatory fields §

The following fields *MUST* be contained in an action element of an Core TD:

keyword	*type*	*constraints*
title	string	unique name among all actions
input	array of DataSchema	all elements of the subclasses objectSchema and dataSchema *MUST* only contain simple types.
output	array of DataSchema	all elements of the subclasses objectSchema and dataSchema *MUST* only contain simple types.

4.1.5.2 Additional Constraints §

The elements of the DataSchema subclasses ArraySchema and ObjectSchema for the fields `input` and `output` are restricted to simple types in a Thing Description conforming to the Core Data Model. Without this limitation a higher implementation burden would be put on resource constrained devices (arbitrary cascaded arrays and multi-level objects) which cannot be satisfied by all consuming devices.

The following additional constraints *MUST* be applied to the Interaction Affordances of a Thing Description conforming to the Core Data Model:

keyword	*type*	*constraint*
forms	array of Forms	The `Array of Form` of each action *MUST* contain only a *single* endpoint.
format	string	If the field `format` is used, only formats defined in section 7.3.1-7.3.6 of [JSON-SCHEMA] *MAY* be used.
oneOf	string	The DataSchema field `oneOf` does not make sense for properties and *MUST NOT* be used.
uriVariables	Map of DataSchema	`uriVariables` *MUST NOT* be used.

EDITOR'S NOTE

TODO:

- no optional parameters

- timeout

4.1.5.3 Recommended Practice §

4.1.6 Event Affordances §

The Core Data Model applies the following constraints and rules to the `EventAffordance` class of section 5.3.1.5 of the TD Specification.

A Thing may provide more than one event mechanism to enable a variety of consumers.

4.1.6.1 Mandatory fields §

The following fields *MUST* be present in an event element of a *Core TD*:

keyword	*type*	*constraints*
title	string	unique name among all events
description	string	human readable description
data	set of DataSchema instances in a JSON object	only the DataSchema subclasses booleanSchema, IntegerSchema, NumberSchema, StringSchema are permitted

4.1.6.2 Additional Constraints §

The following additional constraints *MUST* be applied to the Event Affordances of a Thing Description conforming to the profile:

keyword	*type*	*constraint*
forms	array of Forms	The `Array of Form` of each event *MUST* contain only a *single* endpoint.
uriVariables	Map of DataSchema	`uriVariables` *MUST NOT* be used.

4.1.7 Forms §

A Thing may provide more than one event mechanism to enable a variety of consumers.

4.1.7.1 Mandatory fields §

The following fields *MUST* be present in a form element of a *Core TD*:

keyword	*type*	*constraints*
title	string	unique name among all events
description	string	human readable description
data	set of DataSchema instances in a JSON object	only the DataSchema subclasses booleanSchema, IntegerSchema, NumberSchema, StringSchema are permitted

4.1.7.2 Additional Constraints §

The following additional constraints *MUST* be applied to the Form elements of a Thing Description conforming to the Core profile:

keyword	*type*	*constraint*
security	string or Array of string	`security` at form level *MUST NOT* be used.
scopes	string or Array of string	`scopes` *MUST NOT* be used.

4.1.8 Links §

EDITOR'S NOTE

TODO: Consider selecting a defined set from [RFC6903] chapter 6.

The "type" relationship as defined in chapter 6 of [RFC6903] is reserved for indicating an instance relationship between a thing and a thing template. The Core Data Model does not put additional constraints or requirements on links. The interpretation of a link is out of scope.

4.1.9 Security §

The Core Data Model defines a subset of the security schemes that *MAY* be implemented on resource constrained devices. A security scheme *MUST* be defined at the thing level. The security scheme is applied to the thing as a whole, a thing may adopt multiple security schemes.

The set of security schemes supported in the Core Data Model is based on the plug-fest results. To ensure interoperability, a TD consumer, which compliant with the Core Data Model *MUST* support **all** of the following security schemes:

- no security
- Basic Auth
- Digest
- Bearer Token
- Oauth2

4.1.9.1 Recommended Practice §

When using the "no security" or "Basic Auth" security schemes it is strongly recommended to use transport layer encryption.

4.2 Protocol Binding §

EDITOR'S NOTE
This section is work in progress and not ready for review.

This section describes how the Core Data Model can be bound to different protocols. In addition to a set of mapping rules, it defines additional behavior, e.g. timeouts, error behavior, action semantics, ...

4.2.1 HTTP Protocol Binding §

All communication is using JSON payloads over HTTP(s). The content type header *MUST* be set to "application/json".

4.2.1.1 Properties §

The HTTP verbs GET and PUT are mapped on reading and writing a property - all other protocol verbs return an error "405 Method Not Allowed".

> Note: Since HTTP does not provide a pub/sub mechanism, the observe interaction is not supported directly. The event mechanism can be used instead to send notifications on property changes.

Multiple properties can be set/get by accessing the Properties endpoint.

4.2.1.2 Actions §

Actions can be synchronous and asynchronous. The current TD specification does not distinguish these two cases and does not describe a detailed mechanism.

The HTTP verb POST is mapped to invoking an action on the actions endpoint - all other protocol verbs return an error "405 Method Not Allowed".

4.2.1.3 Events §

The supported protocols for events are WebHooks, WebSockets, SSE and Long polling.

4.3 External TD representations §

The default representation is JSON. Semantic annotations based on JSON-LD *MAY* be present but are not required to perform all interactions with the thing instance.

5. Open Issues §

- define action semantics
- define subscription mechanism
- Examples

A. References §

A.1 Informative references §

[html]
HTML Standard. Anne van Kesteren; Domenic Denicola; Ian Hickson; Philip Jägenstedt; Simon Pieters. WHATWG. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[ISO-6709]
ISO-6709:2008 : Standard representation of geographic point location by coordinates. ISO. 2008-07. Published. URL: https://www.iso.org/standard/39242.html

[JSON-SCHEMA]
JSON Schema Validation: A Vocabulary for Structural Validation of JSON. Austin Wright; Henry Andrews; Geraint Luff. IETF. 19 March 2018. Internet-Draft. URL: https://tools.ietf.org/html/draft-handrews-json-schema-validation-01

[RFC6903]
Additional Link Relation Types. J. Snell. IETF. March 2013. Informational. URL: https://tools.ietf.org/html/rfc6903

[wot-architecture]
Web of Things (WoT) Architecture. Matthias Kovatsch; Ryuichi Matsukura; Michael Lagally; Toru Kawaguchi; Kunihiko Toumura; Kazuo Kajimoto. W3C. 9 April 2020. W3C Recommendation. URL: https://www.w3.org/TR/wot-architecture/

[wot-thing-description]
Web of Things (WoT) Thing Description. Sebastian Käbisch; Takuki Kamiya; Michael McCool; Victor Charpenay; Matthias Kovatsch. W3C. 9 April 2020. W3C Recommendation. URL: https://www.w3.org/TR/wot-thing-description/

↥