Web of Things (WoT) Thing Description: Implementation Report

Version: 17 December 2018

Contributors:

Sebastian Kaebisch (Siemens AG) (Editor in Chief)
Takuki Kamiya (Fujitsu Laboratories of America, Inc.)
Michael McCool (Intel K.K. (Japan))
Others (TBD)

Table of Contents

1. Introduction

The Web of Things (WoT) Thing Description Specification entered the Candidate Recommendation period on XX February 2019.

The planned date for entering Proposed Recommendation is XX March 2019. This document summarizes the results from the Web of Things (WoT) TD implementation reports received and describes the process that the Web of Things (WoT) Working Group followed in preparing the report.

1.1 Implementation report objectives

1. Must verify that the specification is implementable.

1.2 Implementation report non-objectives

1. The IR does not attempt conformance testing of implementations.

2. Work during the Candidate Recommendation period

During the CR period, the Working Group carried out the following activities:

1. Clarification and improvement of the exposition of the specification
2. Disposing of comments that were communicated to the WG during the CR period. These comments and their resolution are detailed in the Disposition of Comments document for the CR period.
3. Preparation of this Implementation Report.

3. Participating in the implementation report

Implementors were invited to contribute to the assessment of the W3C Web of Things (WoT) Thing Description Specification by submitting implementation reports describing the coverage of their implementations with respect to the test assertions outlined in the table below.

Implementation reports, comments on this document, or requests made for further information were posted to the Working Group's public mailing list wot-wg@w3.org (archive) and as issues on the github repository i https://github.com/w3c/wot-thing-description.

4. Entrance criteria for the Proposed Recommendation phase

The Web of Things (WoT) Working Group established the following entrance criteria for the Proposed Recommendation phase in the Request for CR:

1. Sufficient reports of implementation experience have been gathered to demonstrate that Things can be described by Thing Descriptions in sufficient detail to allow interoperabilty.
2. Specific Implementation Report Requirements (outlined below) have been met.
3. The Working Group has formally addressed and responded to all public comments received by the Working Group.

All three of these criteria have been met. The testimonials below indicate the broad base of support for the specification. All of the required features had at least two implementations. All of the optional features received at least two implementations. However, these optional features do not have conformance requirements that have an impact on interoperability.

5. Implementation report requirements

5.1 Detailed requirements for implementation report

1. Testimonials from implementers are included in the IR when provided to document the utility and implementability of the specification.
2. IR must cover all specified features in the specification. For each feature the IR should indicate:
    - Feature status: required, optional, other.

  - Feature utility/usefulness based on feedback from implementers.
  - Implementability of the feature specification.
3. Feature status is a factor in test coverage in the report:
   - Required specification features must have at least two implementations. Implementations that do not implement a required specification feature must document the reason for not implementing the feature.
   - Optional specification features must have at least one implementation. Implementations that do not implement an optional specification feature should document the reason for not implementing the feature.

5.2 Notes on testing

1. A implementation report must indicate the outcome of evaluating the implementation with respect to each of the test assertions. Possible outcomes are "`pass`", "`fail`" or "`not-impl`". "

5.3 Out of scope

This implementation Report will not cover:

- Combinations of protocols and security schemes not listed in Web of Things (WoT) Security Best Practices.

6. Systems

This section contains descriptions of each of the implementations of the WoT Thing Description specification from all organizations that submitted implementation reports. Each implementation represents a working system that either exposes or consumes at least one WoT Thing Description. Implementations that expose a Thing Description act as a network server with an interface as described in the Thing Description it exposes. Implementations that consume a Thing Description act as a network client and issue network requests consistent with the target Thing Description. In some cases a given implementation may be used for multiple Things and a single Thing may also act as both client and server on multiple interfaces.

We only count systems with mostly independent code bases as distinct implementations. There are however some cases (documented in the following) where implementations shared components but were still considered substantially independent implementations. In cases where a substantial component was shared across implementations the features supported by that component were only counted once.

W3C/ERCIM

Arena Web Hub is an open source node-based application server for the Web of Things that has been implemented with support from the European Commission for the F-Interop project as a means to demonstrate high level interoperability testing for the Web of Things.

One of the example applications is a browser-based test suite for both client and server in respect to the contract implied by a Thing Description. This uses a specially designed Thing Description, together with associated client and server applications, and has been developed to cover the constraints implied by the data schemas defined by the Thing Description specification.

Arena Client: TD Consumer

This is a browser-based JavaScript library designed to support multiple server platforms, including the Arena Web Hub, Eclipse ThingWeb (node-wot), and Mozilla's Things Gateway.

Arena WebHub: TD Producer

Arena Web Hub is a secure by design application server with support for HTTPS and WebSockets. HTTPS is used together with a bearer token for access control, along with support for Server-Sent Events and Long Polling. You can get or set the values of all properties in a single transaction. The WebSockets sub-protocol is initiated via a protocol upgrade request from HTTPS, and includes suppport for properties, actions and events. This makes use of a RESTful request/response pattern with the same status codes as for HTTPS. Arena has minimal dependencies on other modules and can be installed via npm or a git clone of the open source repository. Applications can combine the producer and consumer modules as needed.

Hitachi, Ltd.

Hitachi understands the importance of standardization to make the world seamlessly connected, and strongly supports W3C's activities.

Hitachi expects that standardization of Web of Things enables easy integration accross various Things and IoT platforms. We create a tool for rapid application development using Node-RED, called Node generator.

Node generator for Node-RED: TD consumer

Node generator is command line tool to generate Node-RED node modules from several various sources including Thing Description of Web of Things. Using this tool, node developers can dramatically reduce their time to implement Node-RED node modules.

Intel Corporation

Intel supports this specification as it enables interoperation between devices from different manufacturers, supporting the growth of an ecosystem of interoperating IoT services and easing barriers to adoption.

Three separate implementations were developed, all based on Node.js. The core implementations focused on basic local network access without built-in security. A separate project, used by all three implementations, provided security (encryption and authentication) by means of a reverse proxy. For the purposes of validation this reverse proxy was considered a shared component of all three implementations, and the features it implemented were only counted once in the results.

Security Proxy: Shared Component

This component, shared across all implementations provided by Intel, was a reverse proxy service that provided authentication (via various mechanisms indicated by the schemes indicated in the Thing Description) and encrypted transport termination. The reverse proxy service was made accessible over a secure tunnel, and depending on the circumstances, was run in the cloud, on a gateway computer, or locally on a device. As this component implemented the security features in all other implementations from Intel, for the purposes of validation these features are only recorded as being supported in a single implementation.

OCF Metadata Translator: TD Producer

This implementation translated metadata made available by devices implemented using the OCF standard and re-expressed that metadata in the form of WoT Thing Descriptions. The actual network interfaces were however provided either by the OCF devices themselves directly via CoAP, or via a CoAP/HTTP bridge developed by OCF. The metadata translator also had the ability to add semantic markup to the generated WoT Thing Descriptions, using a database that related OCF Resource Types to iotschema.org capabilities. It also had the capability to register generated TDs with directory services. Secure interfaces (e.g. HTTPS combined with an authentication scheme) were provided via a reverse proxy attached to the HTTP interface provided by the OCF CoAP/HTTP bridge.

The OCF devices were based on the OCF Smart Home Demo and included both real and simulated devices. Real devices used for testing included

1. ON\OFF Lights (various: MOSFET, Relay, LEDs),
2. RGB Light,
3. Potentiometer (analog input),
4. Toggle Switch,
5. Pushbutton,
6. IR Motion Sensor,
7. Illuminance Sensor, and
8. Buzzer.

There were also multiple instances of many of these devices, each given unique identifiers.

WebSpeak: TD Producer

This implementation provided a service to convert text to speech. English text set to its network interface was spoken audibily. This service was used to test several accessibility scenarios, including automatic conversion of visible status and event information to spoken announcements in order to support blind users. This implementation only supported an HTTP network interface, without security. Secure interfaces (eg HTTPS combined with an authentication scheme) were provided via a reverse proxy attached to the HTTP interface. The TDs for this service were generated using a template mechanism, and the service also had the capability to automatically register these TDs with a directory service.

Simple Web Camera: TD Producer

This implementation provided a service to capture still images from a camera and provide them over its network interface. This was used to test various aspects of actions, including support for inputs and outputs with different content types. The service was also capaple of introspecting the parameters made available via any particular attached video4linux camera and made those parameters available as WoT Thing Description properties. This implementation only supported an HTTP network interface, without security. Secure interfaces (eg HTTPS combined with an authentication scheme) were provided via a reverse proxy attached to the HTTP interface. The TDs for this service were generated using a template mechanism, and the service also had the capability to automatically register these TDs with a directory service.

Oracle Corporation

Oracle supports this specification as it enables manufacturers of IoT cloud platforms and applications to integrate devices across multiple vendors, who describe their features and functionality in a uniform way. This enables application scenarios that allow monitoring and control of devices from different manufacturers. Flexible rules can be used to define alert conditions that trigger actions on devices based on sensor data from other devices. Sensor data combined with big data analytics can be used to predict maintenance needs of devices to prevent downtime.

A common way of describing devices grows the ecosystem of devices that can be easily integrated into cloud platforms out of the box and thus enables creating digital twins of physical devices for asset monitoring, production monitoring, fleet management, and the management of buildings and smart cities.

Oracle offers an IoT cloud platform that enables management of devices, messages and alerts. This platform is complemented with applications for asset monitoring, production monitoring, fleet management and connected workers. This platform interoperates with WoT servients as described in the sections below.

Thing Description to Device Model Converters: TD Consumer and Producer

Oracle's IoT cloud platform defines a device model abstraction that is used to describe the common behaviour of a class of devices via properties (attributes), actions, messages and alerts. This format is similar to the WoT thing description and thing descriptions can be converted to device models and vice versa.
Devices that are managed by the IoT cloud platform can be expose using the WoT thing description format and devices that are described via thing descriptions can be consumed from the IoT cloud service. Oracle provides open source implementations of converters to generate a device model from a thing description (td2dm) and vice versa (dm2td).

Oracle Digital Twin Simulator: TD Producer

Oracle's IoT Cloud Service includes a simulator for devices which allows to model and test asset monitoring scenarios without having the physical device already available. This allows experimenting with different device models and finding the right abstraction before a device is actually manufacturered and thus reduce the implementation risk. Various device simulations were defined:

- HVAC
- BluePump
- Truck
- Connected Car

These simulations were exposed via TDs that were generated from device models using the converters above. The simulator uses HTTP/REST with JSON to interact with the devices. TD's for devices from other manufacterer were imported using the td2dm converter and simulators were defined for the following devices:

- Fujitsu's rotary beacon light
- Intel's RGB light
- Panasonic's air conditioner
- Panasonic's hue light
- Siemens Festo Plant
- KETI environment sensor

Node-WoT with Oracle Binding: TD Consumer

Node-Wot contains a binding to Oracle's IoT Cloud Service. The binding uses Oracle's JavaScript client library that encapsulates Oracle's proprietary device to cloud communication protocol. The integration allows node-wot to consume TDs for devices from other manufacturers and to read properties and trigger actions on these devices.

Oracle Asset Monitoring: Interoperability across Manufacturers

Oracle's Asset Monitoring application is used to define scenarios that combine devices from different manufacturers. Flexible rules trigger actions and issue alerts based on sensor data from other devices.
In an industrial scenario that integrates devices from KETI, Fujitsu, Siemens and Intel was defined and several rules control RGB lights, speech output, a valve and a pump based on sensor data from different devices.

Panasonic Corporation

Panasonic supports this specification as it enables device manufacturers to describe features and functionality of multiple devices in a uniform way. This will make it possible to build mash-up applications easily with multiple devices not only from single vendor but also from different vendors, which will expand the ecosystem.

Panasonic already has various IoT devices in the market and this specification is the first candidate to be used for the directory system to discover those devices. At the moment, Panasonic implements two types of clients (TD consumer) as well as two types of servers (TD producer), together with independent authentication server as a shared component, for experimental purposes.

Authentication Server: Shared Component

This component provides authentication functionality for users who access Panasonic things (TD producers) such as the real things or simulated things provided by other implementations. This component supports HTTPS, receives a predetermined userid and password via an HTTP POST method and sends back a bearer token based on JWT. The token should be placed in the Authentication request header upon access to a Thing supporting the `"bearer"` security scheme. The URL and other information of this component is provided in the security elements of the Thing Descriptions produced by Panasonic things.

Browser Based Client: TD Consumer

This implementation is a TD Consumer which works on the Chrome browser. This implementation reads a Thing Description from a designated URL via HTTP(S), expands its properties, actions and events into UI form, then allows user to read, write or observe a property value, invoke an action, or subscribe/unsubscribe to an event through the UI. When the TD indicates that the thing needs a bearer token, this implementation also supports the settting of user credentials such as userid and password and retrieves the access token from the authentication service at the URL designated in the TD, such as the Panasonic Authentication Server explained above. This implementation also allows the manual addition of any HTTP headers to the request message sent to the things.

This implementation basically supports only HTTP(S) and does not support CoAP and MQTT. For observe property and events, this implementation supports both HTTP(S) long poll and a simple WebSocket protocol which contains only a data value in its transaction. This implementation only supports application/json as message body and does not support text/plain. Since this implementation is browser based and uses Ajax, its default mode requires support of CORS from the server exposing the Things, unless the chrome browser is launched with the `--disable-web-security` option.

Node-RED Based Client: TD Consumer

This implementation is a TD Consumer which works on Node-RED. This implementation reads a Thing Description from a designated URL via HTTP(S) and from a local folder, finds a specific thing according to a semantic annotation, and turns the thing on/off. The implementation supports only HTTP(S) with `"nosec"` and `"basic"` authorization. For MQTT and WebSockets, this implementation needs a hard-coded URL due to the restriction of normal Node-RED.

WoT Server for Real Devices: TD Producer

This implementation is a TD producer which is hosted on a cloud server and connected to real things in the lab through a proprietary tunneling mechanism alike to remote and local WoT proxy. This implementation accepts HTTP bindings with bearer token authorization issued by the Panasonic Authentication Server explained above. This implementation currently provides the following things:

1. Two Home Air Conditioners,
2. One robotics cleaner,
3. One set of Philips Hue Lighting and
4. Two LED bulletin boards.

This implementation is based on Apache HTTP server with plugins written in PHP.

WoT Simulator: TD Producer

This implementation is a TD producer which is hosted on a cloud server and hosts virtual things running on the server. This implementation accepts HTTP bindings with bearer token authorization issued by the Panasonic Authentication Server explained above. This implementation currently provides simulation of things such as

1. Home Air Conditioner,
2. Robotics cleaner,
3. Philips Hue Lighting and
4. Simple LED lighting.

This implementation is based on Node.js. This implementation is portable and also can be run on a local machine such as a Raspberry Pi.

Siemens AG

Siemens supports this specification as it allows manufacturers to attach metadata to heterogeneous IoT devices and services in a uniform way. The extensible model with a standardized serialization format in particular remedies interoperability challenges with an installed base of long-lived industrial devices. Furthermore, this specification is a central building block for several digitalization topics such as device onboarding, device management, digital twins, and data analytics.

Siemens commits to several open-source implementations of W3C Web of Things, which are managed in the public Eclipse Thingweb project. Siemens also implemented concepts of this specification in products and aims at aligning the implementations and opening the features to customers once this specification reaches REC status.

node-wot: TD Consumer and Producer

The node-wot implementation is a Node.js-based framework for WoT Servients. It features an implementation of the WoT Scripting API to both consume TDs to instantiate Consumed Things and produce Exposed Things that provide a TD. The node-wot implementation supports several Protocol Bindings including HTTP, CoAP, WebSockets, and MQTT, with corresponding security mechanisms such as DTLS (CoAP); TLS (HTTP, MQTT), Basic, Digest, and Bearer Token authentication (HTTP), PSK (CoAP), and username/password authentication (MQTT).

Thingweb Directory: TD Consumer and Producer

The Thingweb Directory provides a directory service written in Java to register TDs and look them up through queries (primarily SPARQL). The implementation consumes TDs registered with it through a web API, represent their metadata in a KnowledgeGraph, and (re-)produces TDs to return the results of queries. Currently it supports HTTP and CoAP bindings.

WoT-fxUI: TD Consumer

WoT-fxUI is a generic user interface (UI) for Things based on Java with JavaFX. It consumes TDs to render UI elements that allow human users to interact with Things. The implementation currently supports HTTP and CoAP bindings.

7. Security

The Web of Things (WoT) Thing Description Specification includes features to support security. Functional aspects of assertions associated with security features are validated in the same fashion as other functional features. In addition, however, the Web of Things (WoT) WG Charter requires the development of a test plan that includes adversarial testing. An appropriate security test plan, including a description of how existing web service penetration testing tools can be used, is given in Web of Things (WoT) Security Testing Plan.

Security validation is in addition to a description of more general security and privacy considerations given in Web of Things (WoT) Security and Privacy Considerations and the security and privacy considerations given in the specification itself.

Note the WoT TD specification is designed to support the description of existing devices which may not be secure by current standards, such adversarial testing will normally be limited to security configurations following the best-practices security configurations given in Web of Things (WoT) Security Best Practices. for which security is an expectation.

8. Test results

8.1 Test assertion results

The aim of this section is to describe the range of test assertions developed for the Web of Things (WoT) Thing Description Specification and summarize the results from the implementation reports received in the CR period. The table lists all assertions derived from the Web of Things (WoT) Thing Description Specification.

- The **ID** column uniquely identifies the assertion. and links to the assertion in the context of the specification.
- The **Assertion** column specifies the constraint which must be met.
- The **Req** column is a Y/N value indicating whether the test assertion is for a feature which is required. Note however that the feature may only be required if another feature is also used, as indicated in the **Context** column.
- The **Parent** column indicates another more general assertion that this one is a specialization of. A specialization indicates a more restrictive context or provides additional detail to facilitate testing. The more general assertion is only considered to have a "pass" status if all its required or implmented specializations in a given implementation have a "pass" status.
- The **Context** column indicates that the "required" status of this feature depends on the use of other features. If the context is optional, then the value of **Req** actually indicates whether the feature is required if the indicated context exists.
- The **Result** column is annotated with the number of 'pass' (**P**), 'fail' (**F**), and 'not implemented' (**N**) status results in the individual implementation reports.
- The **T** column indicates the total number of implementations (or implementation components in the case of shared components) reporting a status (of any kind) on each assertion.

ID	Category	Assertion	Req	Parent	Context	Results			
						P	F	N	T
td-vocabulary	Syntax	All vocabulary restrictions noted in these tables MUST be followed, including mandatory items and default values.	Y			6		2	8
td-unique-identifiers	Syntax	Each instance of a `Property`, `Action`, and `Event` class MUST have an identifier that is unique within the context of a Thing Description document.	Y			6		2	8

ID	Category	Assertion	Req	Parent	Context	Results			
						P	F	N	T
td-jsonld-keywords	Syntax	Thing Description instances MAY contain JSON-LD keywords such as `@context` and `@type`.	N			6		2	8
td-string-type	Syntax	Vocabulary terms that use simple types string and anyURI MUST be serialized as JSON string.	Y			6		2	8
td-integer-type	Syntax	Vocabulary terms that use simple type unsignedInt MUST be serialized as JSON integer.	Y			4		4	8
td-number-type	Syntax	Vocabulary terms that use simple type double MUST be serialized as JSON number.	Y			3		5	8
td-context	Syntax	The root object of a Thing Description instance MAY include the `@context` name from JSON-LD 1.1 with the value URI of the Thing description context file `http://www.w3.org/ns/td`.	N			6		2	8
td-context-jsonld	Syntax	When a Thing Description instance is processed and interpreted by a JSON-LD 1.1 processor the `@context` member MUST be present	Y			6		2	8
td-additional-contexts	Syntax	When a single Thing Description instance involves several contexts, additional namespaces with prefixes MUST be appended to the `@context` array structure.	Y			7		4	11
td-context-toplevel	Syntax	Each mandatory and optional member name as defined in the class `Thing` MUST be serialized as a JSON name in the root object of the Thing Description instance.	Y			6		2	8
td-objects	Syntax	The type of the members `properties`, `actions`, `events`, `version`, and `securityDefinitions` MUST be a JSON object.	Y			6		2	8
td-arrays	Syntax	The type of the members `links`, `scopes`, and `security` MUST be a JSON array.	Y			6		2	8
td-properties	Syntax	Properties (and sub-properties) offered by a Thing MUST be collected in the JSON-object based `properties` member with (unique) Property names as JSON names.	Y			5		3	8
td-property-names	Syntax	Each mandatory and optional vocabulary term as defined in the class `Property`, as well as its two superclasses `InteractionPattern` and `DataSchema`, MUST be serialized as a JSON name within a Property object.	Y	td-properties		5		3	8

ID	Category	Assertion	Req	Parent	Context	Results			
						P	F	N	T
td-property-objects	Syntax	The type of the members `properties` and `items` MUST be serialized as a JSON object.	Y	td-properties		8		3	11
td-property-arrays	Syntax	The type of the members `forms`, `required`, and `enum`, `scopes`, and `security`, MUST be serialized as a JSON array.	Y	td-properties		6		5	11
td-property-semantic	Syntax	Similar to the case at the `Thing` level, properties MAY have additional semantic annotations based on JSON-LD 1.1 keywords.	N	td-properties		5		3	8
td-property-defaults	Syntax	When a Thing Description instance is processed and interpreted by a JSON-LD 1.1 processor, each `property` MUST contain the vocabulary terms `observable` and `readOnly` due to the open-world assumption of Linked Data.	Y	td-properties		2	2	3	7
td-actions	Syntax	Actions offered by a Thing MUST be collected in the JSON-object based `actions` member with (unique) Action names as JSON names.	Y			5		3	8
td-action-names	Syntax	Each optional vocabulary term as defined in the class `Action` and its superclass `InteractionPattern` MUST be serialized as a JSON name within an Action object.	Y	td-actions		5		3	8
td-action-objects	Syntax	The type of the members `input` and `output` MUST be serialized as a JSON object.	Y	td-actions		7		4	11
td-action-arrays	Syntax	The type of the members `forms`, `scopes`, and `security` MUST be serialized as a JSON array.	Y	td-actions		7		4	11
td-action-semantic	Syntax	Definitions within the `actions` member MAY have additional semantic annotations based on JSON-LD 1.1 keywords.	N	td-actions		5		3	8
td-events	Syntax	In such a case these variables in the URI MUST be collected in the JSON-object based `uriVariables` member with the associated (unique) variables names as JSON names.	Y			2		6	8
td-event-names	Syntax	Each optional vocabulary term as defined in the class `Event`, as well as its two superclasses `InteractionPattern` and `DataSchema`, MUST be serialized as a JSON name within an Event object.	Y	td-events		2		6	8
td-event-objects	Syntax	The type of the members `subscription`, `data`, and `cancellation` MUST be serialized as a JSON object.	Y	td-events		3		8	11

ID	Category	Assertion	Req	Parent	Context	Results			
						P	F	N	T
td-event-arrays	Syntax	The type of the members `forms`, `required`, and `enum`, `scopes`, and `security` MUST be serialized as a JSON array.	Y	td-events		3		8	11
td-event-semantic	Syntax	Definitions within the `events` member MAY have additional semantic anotations based on JSON-LD 1.1 keywords.	N	td-events		2		6	8
td-forms	Syntax	Each mandatory and optional vocabulary term as defined in the class `Form`, MUST be serialized as a JSON name.	Y			6		2	8
td-event-response-arrays	Syntax	If required, the type of the member `response` MUST be serialized as a JSON array.	Y					7	7
td-form-protocolbindings	Syntax	If required, `forms` MAY be supplemented with protocol-specific vocabulary terms identified with a prefix.	N	td-forms		2		6	8
td-form-contenttype	Syntax	When a Thing Description instance is processed and interpreted by a JSON-LD 1.1 processor, each `forms` (array) entry MUST contain a `contentType` due to the open-world assumption of Linked Data.	Y	td-forms		4	2	2	8
td-forms-response	Syntax	If the `response` member is used in `forms` it MUST contain the `contentType` as defined in class `Response`.	Y			1		6	7
td-links	Syntax	Each mandatory and optional vocabulary term as defined in the class `Link`, MUST be serialized as a JSON name.	Y			3		5	8
td-security	Security	Each mandatory and optional vocabulary term as defined in the class `SecurityScheme`, MUST be serialized as a JSON name.	Y			4	2	2	8
td-security-mandatory	Security	Every Thing MUST have at least one security scheme specified at the top level.	Y	td-security		4	2	2	8
td-security-overrides	Security	Security schemes MAY also be specified at the interaction and form levels. In this case, definitions at the lower levels override (completely replace) the definitions at the higher level.	N	td-security		3	2	2	7
td-security-binding	Security	If a Thing requires a specific access mechanism for a resource, that mechanism MUST be specified in the Thing Description's security scheme configuration for that resource.	Y	td-security		1	2	3	6
td-security-no-extras	Security	If a Thing does not require a specific access mechanism for a resource, that mechanism MUST NOT be specified in the Thing Description's security scheme configuration for that resource.	Y	td-security		1		5	6

ID	Category	Assertion	Req	Parent	Context	Results			
						P	F	N	T
td-version	Syntax	The version container MAY be used to provide additional application and/or device specific version information based on terms from non-TD namespaces.	N			4		7	11
td-media-type	Syntax	The media type `application/td+json` MUST be also associated with the JSON-LD context `http://www.w3.org/ns/td`.	Y	td-forms		6		2	8
td-readOnly-observable-default	Syntax	If the name terms `readOnly` and/or `observable` are not present within a `properties` definition, the default value defined in MUST be assumed.	Y	td-properties		5	4		9
td-content-type-default	Syntax	If the `contentType` name term is not present within a `forms` definition, the default value as defined in MUST be assumed.	Y			2	3	3	8
td-jsonld-preprocessing-context	Preprocessing	Before starting JSON-LD 1.1 processing of a Thing Description instance, there MUST be a `@context` name from JSON-LD 1.1 as defined in Section .	Y			3		5	8
td-jsonld-preprocessing-prefix	Preprocessing	Before starting JSON-LD 1.1 processing of a Thing Description instance, all external vocabulary terms used in the Thing Description MUST provide their context URIs as prefix or within the `@context` member.	Y			3		5	8
td-jsonld-preprocessing-defaults	Preprocessing	Before starting JSON-LD 1.1 processing of a Thing Description instance, all mandatory vocabulary terms as defined in Section that are missing from the instance MUST be inserted explicitly with their default value.	Y			3		5	8
td-vocab-support	Vocabulary	`support`: Provides information about the TD maintainer (e.g., author, link or telephone number to get support, etc). MAY be included. Type: `anyURI`.	N	td-vocabulary		3		5	8
td-vocab-description	Vocabulary	`description`: Provides additional (human-readable) information. MAY be included. Type: `string`.	N	td-vocabulary	td-vocab-properties td-vocab-actions td-vocab-events	4		4	8
td-vocab-links	Vocabulary	`links`: Provides Web links to arbitrary resources that relate to the specified Thing Description. MAY be included. Type: array of `Link`.	N	td-vocabulary		4		4	8

ID	Category	Assertion	Req	Parent	Context	Results			
						P	F	N	T
td-vocab-securityDefinitions	Vocabulary	`securityDefinitions`: Set of named security configurations (definitions only). Not actually applied unless names are used in a security section. MAY be included. Type: `SecurityScheme`.	N	td-vocabulary		4	1	4	9
td-vocab-lastModified	Vocabulary	`lastModified`: Provides information when the TD instance was last modified. MAY be included. Type: `string`.	N	td-vocabulary		3		5	8
td-vocab-properties	Vocabulary	`properties`: Data schema nested definitions. MAY be included. Type: `DataSchema`.	N	td-vocabulary	td-vocab-properties	8			8
td-vocab-base	Vocabulary	`base`: Define the base URI that is valid for all defined local interaction resources. All other URIs in the TD must then be resolved using the algorithm defined in [[!RFC3986]]. MAY be included. Type: `anyURI`.	N	td-vocabulary		5		3	8
td-vocab-name	Vocabulary	`name`: Name for query, header, or cookie parameters. MAY be included. Type: `string`.	N	td-vocabulary		8			8
td-vocab-version	Vocabulary	`version`: Provides version information. MAY be included. Type: `Versioning`.	N	td-vocabulary		4		4	8
td-vocab-actions	Vocabulary	`actions`: All Action-based interaction patterns of the Thing. MAY be included. Type: `Action`.	N	td-vocabulary		7		1	8
td-vocab-events	Vocabulary	`events`: All Event-based interaction patterns of the Thing. MAY be included. Type: `Event`.	N	td-vocabulary		5		3	8
td-vocab-created	Vocabulary	`created`: Provides information when the TD instance was created. MAY be included. Type: `string`.	N	td-vocabulary		3		5	8

ID	Category	Assertion	Req	Parent	Context	Results			
						P	F	N	T
td-vocab-security	Vocabulary	`security`: Set of security definition names, chosen from those defined in securityDefinitions. These must all be satisfied for access to resources at or below the current level, if not overridden at a lower level. MAY be included. Type: array of `string`.	N	td-vocabulary	td-vocab-forms td-vocab-properties td-vocab-actions td-vocab-events	4	5		9
td-vocab-forms	Vocabulary	`forms`: Indicates one or more endpoints from which an interaction pattern is accessible. MUST be included. Type: array of `Form`.	Y	td-vocabulary	td-vocab-properties td-vocab-actions td-vocab-events	8			8
td-vocab-scopes	Vocabulary	`scopes`: Set of authorization scope identifiers, provided as an array. These are provided in tokens returned by an authorization server and associated with forms in order to identify what resources a client may access and how. MAY be included. Type: array of `string`.	N	td-vocabulary	td-securityDefinitions			9	9
td-vocab-title	Vocabulary	`title`: Provides a human-readable title (e.g., display a text for UI representation) of the interaction pattern. MAY be included. Type: `string`.	N	td-vocabulary		6	1	1	8
td-vocab-uriVariables	Vocabulary	`uriVariables`: Define URI template variables as collection based on DataSchema declarations. MAY be included. Type: `DataSchema`.	N	td-vocabulary	td-vocab-properties td-vocab-actions td-vocab-events	1		7	8
td-vocab-observable	Vocabulary	`observable`: Indicates whether a remote servient can subscribe to ("observe") the Property, to receive change notifications or periodic updates (true/false). MAY be included. Default: `false`. Type: `boolean`.	N	td-vocabulary	td-vocab-properties	5		3	8
td-vocab-safe	Vocabulary	`safe`: Signals if the action is safe (=true) or not. Used to signal if there is no internal state (cf. resource state) is changed when invoking an Action. In that case responses can be cached as example. MUST be included. Default: `false`. Type: `boolean`.	Y	td-vocabulary	td-vocab-actions			8	8

ID	Category	Assertion	Req	Parent	Context	Results			
						P	F	N	T
td-vocab-output	Vocabulary	`output`: Used to define the output data schema of the action. MAY be included. Type: `DataSchema`.	N	td-vocabulary	td-vocab-actions	1		7	8
td-vocab-input	Vocabulary	`input`: Used to define the input data schema of the action. MAY be included. Type: `DataSchema`.	N	td-vocabulary	td-vocab-actions	6		2	8
td-vocab-idempotent	Vocabulary	`idempotent`: Signals if the action is idempotent (=true) or not. Informs if the action can be called repeatedly with the same outcome. . MUST be included. Default: `false`. Type: `boolean`.	Y	td-vocabulary	td-vocab-actions			8	8
td-vocab-cancellation	Vocabulary	`cancellation`: Defines any data that needs to be passed to cancel a subscription, e.g., a specific message to remove a Webhook. MAY be included. Type: `DataSchema`.	N	td-vocabulary	td-vocab-events			8	8
td-vocab-data	Vocabulary	`data`: Defines the data schema of the Event instance messages pushed by the Thing. MAY be included. Type: `DataSchema`.	N	td-vocabulary	td-vocab-events	3		5	8
td-vocab-subscription	Vocabulary	`subscription`: Defines data that needs to be passed upon subscription, e.g., filters or message format for setting up Webhooks. MAY be included. Type: `DataSchema`.	N	td-vocabulary	td-vocab-events			8	8
td-vocab-contentType	Vocabulary	`contentType`: Assign a content type based on a media type [[!MEDIATYPES]] (e.g., 'application/json) and (optional) parameters (e.g., 'charset=utf-8'). MAY be included. Type: `string`.	N	td-vocabulary	td-vocab-forms	3	2	3	8
td-vocab-subprotocol	Vocabulary	`subprotocol`: Indicates the exact mechanism by which an interaction will be accomplished for a given protocol when there are multiple options. For example, for HTTP and Events, it indicates which of several available mechanisms should be used for asynchronous notifications. MAY be included. Type: `string` (one of `longpoll`).	N	td-vocabulary	td-vocab-forms	6		2	8

ID	Category	Assertion	Req	Parent	Context	Results			
						P	F	N	T
td-vocab-op	Vocabulary	`op`: Indicates the expected result of performing the operation described by the form. For example, the Property interaction allows get and set operations. The protocol binding may contain a form for the get operation and a different form for the set operation. The op attribute indicates which form is which and allows the client to select the correct form for the operation required. MAY be included. Type: `string` (one of `readproperty`, `writeproperty`, `observeproperty`, `invokeaction`, `subscribeevent`, or `unsubscribeevent`).	N	td-vocabulary	td-vocab-forms	7		1	8
td-vocab-href	Vocabulary	`href`: URI of the endpoint where an interaction pattern is provided. MUST be included. Type: `anyURI`.	Y	td-vocabulary	td-vocab-forms td-vocab-links	8			8
td-vocab-response	Vocabulary	`response`: This optional term can be used if, e.g., the output communication metadata differ from input metdata (e.g., output contentType differ from the input contentType). The response name contains metadata that is only valid for the reponse messages. MAY be included. Type: `Response`.	N			2		5	7
td-vocab-type	Vocabulary	`type`: Assignment of JSON-based data types compatible with JSON Schema (one of boolean, integer, number, string, object, array, or null). MAY be included. Type: `string` (one of `object`, `array`, `string`, `number`, `integer`, `boolean`, or `null`).	N	td-vocabulary	td-vocab-input td-vocab-output td-vocab-subscription td-vocab-data td-vocab-cancellation td-vocab-properties	7		1	8
td-vocab-rel	Vocabulary	`rel`: Indicates the relation to an other Thing. MAY be included. Type: `string`.	N	td-vocabulary	td-vocab-links	4		4	8

ID	Category	Assertion	Req	Parent	Context	Results P	F	N	T
td-vocab-anchor	Vocabulary	`anchor`: By default, the context of a link is the URL of the representation it is associated with, and is serialised as a URI. When present, the anchor parameter overrides this with another URI, such as a fragment of this resource, or a third resource (i.e., when the anchor value is an absolute URI). MAY be included. Type: `anyURI`.	N	td-vocabulary	td-vocab-links			8	8
td-vocab-instance	Vocabulary	`instance`: Provides a version identicator of this TD instance. MUST be included. Type: `string`.	Y	td-vocabulary	td-vocab-input td-vocab-output td-vocab-subscription td-vocab-data td-vocab-cancellation td-vocab-properties	3		5	8
td-vocab-unit	Vocabulary	`unit`: Provides unit information that is used, e.g., in international science, engineering, and business. MAY be included. Type: `DataSchema`.	N	td-vocabulary	td-vocab-input td-vocab-output td-vocab-subscription td-vocab-data td-vocab-cancellation td-vocab-properties			8	8
td-vocab-writeOnly	Vocabulary	`writeOnly`: Boolean value that indicates whether a property interaction / value is write only (=true) or not (=false). MUST be included. Default: `false`. Type: `boolean`.	Y	td-vocabulary	td-vocab-enum		1	7	8
td-vocab-enum	Vocabulary	`enum`: Restricted set of values provided as an array. MAY be included. Type: array of any type.	N	td-vocabulary	td-vocab-input td-vocab-output td-vocab-subscription td-vocab-data td-vocab-cancellation td-vocab-properties	3		5	8
td-vocab-const	Vocabulary	`const`: Provides a constant value. MAY be included. Type: any type.	N	td-vocabulary	td-vocab-input td-vocab-output td-vocab-subscription td-vocab-data td-vocab-cancellation td-vocab-properties			8	8
td-vocab-oneOf	Vocabulary	`oneOf`: Used to ensure that the data is valid against one of the specified schemas in the array. MAY be included. Type: array of `DataSchema`.	N	td-vocabulary	td-vocab-input td-vocab-output td-vocab-subscription td-vocab-data td-vocab-cancellation td-vocab-properties			8	8

ID	Category	Assertion	Req	Parent	Context	Results			
						P	F	N	T
td-vocab-readOnly	Vocabulary	`readOnly`: Boolean value that indicates whether a property interaction / value is read only (=true) or not (=false). MUST be included. Default: `false`. Type: `boolean`.	Y	td-vocabulary	td-vocab-enum		1	7	8
td-vocab-minItems	Vocabulary	`minItems`: Defines the minimum number of items that have to be in the array. MAY be included. Type: `unsignedInt`.	N	td-vocabulary	td-vocab-items			8	8
td-vocab-maxItems	Vocabulary	`maxItems`: Defines the maximum number of items that have to be in the array. MAY be included. Type: `unsignedInt`.	N	td-vocabulary	td-vocab-items			8	8
td-vocab-items	Vocabulary	`items`: Used to define the characteristics of an array. MAY be included. Type: `DataSchema`.	N	td-vocabulary	td-vocab-enum	1		7	8
td-vocab-maximum	Vocabulary	`maximum`: Specifies a maximum numeric value. Only applicable for associated number or integer types. MAY be included. Type: `integer`.	N	td-vocabulary	td-vocab-items	4		4	8
td-vocab-minimum	Vocabulary	`minimum`: Specifies a minimum numeric value. Only applicable for associated number or integer types. MAY be included. Type: `integer`.	N	td-vocabulary	td-vocab-items	4		4	8
td-vocab-required	Vocabulary	`required`: Defines which members of the object type are mandatory. MAY be included. Type: array of `string`.	N	td-vocabulary	td-vocab-items			8	8
td-vocab-scheme	Vocabulary	`scheme`: Identification of security mechanism being configured. MUST be included. Type: `string` (one of `nosec`, `basic`, `cert`, `digest`, `bearer`, `pop`, `psk`, `public`, `oauth2`, or `apikey`).	Y	td-vocabulary	td-securityDefinitions	9			9
td-vocab-proxy	Vocabulary	`proxy`: URI of the proxy server this security configuration provides access to. If not given, the corresponding security configuration is for the endpoint. MAY be included. Type: `anyURI`.	N	td-vocabulary	td-securityDefinitions		1	8	9

ID	Category	Assertion	Req	Parent	Context	Results P	F	N	T
td-vocab-in	Vocabulary	`in`: Specifies the location of security authentication information (one of header, query, body, or cookie). MAY be included. Default: `header`. Type: `string`.	N	td-vocabulary	td-securityDefinitions			9	9
td-vocab-qop	Vocabulary	`qop`: Quality of protection (one of auth or auth-int). MAY be included. Default: `auth`. Type: `string`.	N	td-vocabulary	td-securityDefinitions			9	9
td-vocab-authorization	Vocabulary	`authorization`: URI of the authorization server. MAY be included. Type: `anyURI`.	N	td-vocabulary	td-securityDefinitions		2	7	9
td-vocab-format	Vocabulary	`format`: Specifies format of security authentication information (one of jwt, jwe, or jws). MAY be included. Default: `jwt`. Type: `string`.	N	td-vocabulary	td-securityDefinitions	2		7	9
td-vocab-alg	Vocabulary	`alg`: Encoding, encryption, or digest algorithm (one of MD5, ES256, or ES512-256). MAY be included. Default: `ES256`. Type: `string`.	N	td-vocabulary	td-securityDefinitions	2		7	9
td-vocab-identity	Vocabulary	`identity`: Pre-shared key identity. MAY be included. Type: `string`.	N	td-vocabulary	td-securityDefinitions			9	9
td-vocab-refresh	Vocabulary	`refresh`: URI of the refresh server. MAY be included. Type: `anyURI`.	N	td-vocabulary	td-securityDefinitions			9	9
td-vocab-token	Vocabulary	`token`: URI of the token server. MAY be included. Type: `anyURI`.	N	td-vocabulary	td-securityDefinitions			9	9
td-vocab-flow	Vocabulary	`flow`: Authorization flow (one of implicit, password, client, or code). MAY be included. Default: `implicit`. Type: `string`.	N	td-vocabulary	td-securityDefinitions			9	9
client-data-schema	Client	A WoT Thing acting as a client when interacting with another Thing MUST generate data payloads according to the data schemas given in the target Thing's WoT Thing Description.	Y			6			6
client-uri-template	Client	A WoT Thing acting as a client when interacting with another Thing MUST generate URIs according to the URI templates, base URLs, and form href parameters given in the target Thing's WoT Thing Description.	Y			3		3	6

8.2 Interoperability results

This section documents specific pairs of implementations for which interoperability has been demonstrated. Each row represents a producer and each column a consumer. If a pairing has at least one recorded instance of interoperation, and no recorded failures, then it is marked as a "Pass". If a pairing also has at least one recorded instance that uses a security configuration identified in WoT Security Best Practices then it is marked as "Secure". However, if such a pairing has *any* reported failures, no matter how many other successful pairings it has, including secure pairings, it is marked as a "Fail". Note that since implementations refer to code bases that may be used for many services or devices, each successful pairing reported may in practice correspond to a number of successful service or device pairings.

Producers \ Consumers	**Siemens AG node-wot**	**Panasonic Corporation Browser Based Client**	**Panasonic Corporation Node-RED Based Client**
Intel Corporation OCF Metadata Translator	Pass	Secure	Secure
Intel Corporation WebSpeak	Not Impl	Secure	Secure
Intel Corporation Simple Web Camera	Not Impl	Secure	Fail

Appendices

Appendix A - Test Specifications

td-vocabulary:
Test Method: Parent

This assertion will be considered satisfied if and only if all assertions that it is a parent of are satisfied. These all have ID's of the form `td-vocab-*`.

td-unique-identifiers:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester

td-jsonld-keywords:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- EXAMPLE EXISTS only for @context
- COUNTER EXAMPLE CANNOT EXIST

Perhaps should be checked as a JSON-LD file; it is not clear that is the intent. Assertion seems ambiguous. Other assertions constrain the location and values associated with these keywords. In particular `td-context` restricts it, so this could be considered a parent of that assertion.

td-string-type:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- ASSERTION NOT CLEAR: Does this mean that strings we have in TD such as "type":"number" must be "type" and not (type) or (type is number) etc.?
- EXAMPLE DOES NOT EXIST
- COUNTER EXAMPLE DOES NOT EXIST

Can be checked with a JSON Schema. However, the actual assertion wording is ambiguous. Should probably be "Vocabulary terms that identify values that use simple types...". On the other hand, this rule may be included in more general rules that:

1. A JSON TD MUST be serialized as a JSON file, and
2. Values associated with vocabulary terms MUST have the type given in the table.

td-integer-type:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- ASSERTION NOT CLEAR: does this mean that strings we have in TD such as "type":123 must be 123 and not (1 2 3)?
- EXAMPLE DOES NOT EXIST
- COUNTER EXAMPLE DOES NOT EXIST

See comments under td-string-type.

td-number-type:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- ASSERTION NOT CLEAR: does this mean that strings we have in TD such as "type":123 must be 123 and not (1 2 3)?
- EXAMPLE DOES NOT EXIST
- COUNTEREXAMPLE DOES NOT EXIST

See comments under td-string-type.

td-context:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- ASSERTION NOT CLEAR: Can you have an @context key without this context file?
- IF YES THEN TESTED WRONGLY

Assertion should be revised to say that if the @context is used, it MUST have the given value (or the primary, without a prefix). Assuming this is what is meant, it can be checked with a JSON Schema.

td-context-jsonld:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester

td-additional-contexts:
Test Method: JSON Schema

- TEST EXISTS in AssertionTester
- EXAMPLE EXISTS BUT NOT EXPLICITLY DONE FOR THIS ASSERTION

td-context-toplevel:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- EXAMPLE EXISTS BUT NOT EXPLICITELY DONE FOR THIS ASSERTION

- COUNTER EXAMPLE DOESN'T EXIST

td-objects:
Test Method: JSON Schema

NO TEST EXISTS

-
- EXAMPLE EXISTS BUT NOT EXPLICITLY DONE FOR THIS ASSERTION
- COUNTER EXAMPLE DOESN'T EXIST

td-arrays:
Test Method: JSON Schema

- NO TEST EXISTS
- EXAMPLE EXISTS BUT NOT EXPLICITLY DONE FOR THIS ASSERTION
- COUNTER EXAMPLE DOESN'T EXIST

td-properties:
Test Method: JSON Schema

- NO TEST EXISTS
-
- TDS THAT DON'T RESPECT THIS ASSERTION ARE VALIDATED

td-property-names:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- ASSERTION NOT CLEAR

td-property-objects:
Test Method: JSON Schema

- TEST EXISTS in AssertionTester
- ASSERTION NOT CLEAR: Sounds wrong since items can be in an array in JSON Schema. **Currently test does NOT include `items`, and restriction on items MAY be wrong in assertion.**

td-property-arrays:
Test Method: JSON Schema

- TEST EXISTS IN AssertionTester
- TEST IS INCOMPLETE: does not check for some of the elements (required and enum) specified in the assertion. The master JSON Schema also does not check for these.

td-property-semantic:
Test Method: JSON Schema?

- NO TEST EXISTS
- CANNOT BE TESTED
- EXAMPLE EXISTS

td-property-defaults:
Test Method: Unknown

- NO TEST EXISTS

td-actions:
Test Method: JSON Schema; Parent?

- NO TEST EXISTS

td-action-names:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- ASSERTION NOT CLEAR

td-action-objects:
Test Method: JSON Schema

- TEST EXISTS in AssertionTester
- EXAMPLE DOES NOT EXIST

td-action-arrays:
Test Method: JSON Schema

- TEST EXISTS IN AssertionTester
- Assertion does not include elements mentioned in td-property-arrays and td-event-arrays: required and enum. Is this correct or an oversight?
- EXAMPLE EXISTS
- COUNTER EXAMPLE DOES NOT EXIST

td-action-semantic:
Test Method: JSON Schema?

- NO TEST EXISTS
- CANNOT BE TESTED

td-events:
Test Method: JSON Schema; Parent?

- NO TEST EXISTS
- HOW TO TEST COUNTER EXAMPLE?
- WHAT IS A COUNTER EXAMPLE?

td-event-names:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- ASSERTION NOT CLEAR

td-event-objects:
Test Method: JSON Schema

- TEST EXISTS in AssertionTester
- EXAMPLE DOES NOT EXIST

td-event-arrays:
Test Method: JSON Schema

- TEST EXISTS IN AssertionTester
- TEST IS INCOMPLETE: does not check for some of the elements (required and enum) specified in the assertion. The master JSON Schema also does not check for these.
- EXAMPLE DOES NOT EXIST EXPLICITLY FOR THIS

td-event-semantic:
Test Method: JSON Schema?

- NO TEST EXISTS
- CANNOT BE TESTED

td-forms:
Test Method: JSON Schema

- HOW TO TEST COUNTEREXAMPLE?
- WHAT IS A COUNTEREXAMPLE?

td-event-response-arrays:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- EXAMPLE DOES NOT EXIST EXPLICITLY FOR THIS

td-form-protocolbindings:
Test Method: JSON Schema?

- NO TEST EXISTS
- IF JSON Schema not possible, CONSIDER TESTING WITH SHEX

td-form-contenttype:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester

td-forms-response:
Test Method: JSON Schema

- NO TEST EXISTS

td-links:
Test Method: Unknown

- HOW TO TEST COUNTEREXAMPLE?
- WHAT IS A COUNTEREXAMPLE?

td-security:
Test Method: JSON Schema; Parent?

- NO TEST EXISTS
- HOW TO TEST COUNTEREXAMPLE?
- WHAT IS A COUNTEREXAMPLE?

td-security-mandatory:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- EXAMPLE EXISTS
- COUNTEREXAMPLE DOESN'T EXIST

td-security-overrides:
Test Method: JSON Schema

- NO TEST EXISTS

td-security-binding:
Test Method: Manual

- For HTTP: Test with Postman and/or inspection of headers with curl.
- For CoAP: Unknown.
- PENETRATION TEST

td-security-no-extras:
Test Method: Manual

- For HTTP: Test with Postman and/or inspection of headers with curl.
- For CoAP: Unknown.
- PENETRATION TEST

td-version:
Test Method: JSON Schema

- TEST EXISTS in AssertionTester
- EXAMPLE EXISTS
- **Test exists, but may be incorrect or out of date!**

td-media-type:
Test Method: JSON Schema

- NO TEST EXISTS; NEEDS SCHEMA IN AssertionTester
- ASSERTION NOT CLEAR

td-readOnly-observable-default:
Test Method: Unknown

- NO TEST EXISTS

td-content-type-default:
Test Method: Unknown

- NO TEST EXISTS

td-jsonld-preprocessing-context:
Test Method: Unknown

- NO TEST EXISTS
- HOW TO DISGUISH WHETHER IT SHOULD BE A JSON-LD WHICH LACKS THE @CONTEXT?

td-jsonld-preprocessing-prefix:
Test Method: Unknown

- NO TEST EXISTS

td-jsonld-preprocessing-defaults:
Test Method: Unknown

- NO TEST EXISTS

td-vocab-support: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-description: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-links: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-securityDefinitions: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-lastModified: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-properties: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-base: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-name: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-version: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-actions: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-events: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-created: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-security: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-forms: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-scopes: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-title: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-uriVariables: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-observable: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-safe: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-output: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-input: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-idempotent: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-cancellation: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-data: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-subscription: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-contentType: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-subprotocol: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-op: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-href: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-response: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-type: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-rel: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-anchor: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-instance: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-unit: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-writeOnly: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-enum: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-const: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-oneOf: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-readOnly: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-minItems: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-maxItems: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-items: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-maximum: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-minimum: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-required: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-scheme: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-proxy: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-in: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-qop: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-authorization: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-format: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-alg: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-identity: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-refresh: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-token: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

td-vocab-flow: ***(table)***
Test Method: JSON Schema

- NO TEST EXISTS

client-data-schema: ***(extra)***
Test Method: Unknown

- NO TEST EXISTS

client-uri-template: ***(extra)***
Test Method: Unknown

- NO TEST EXISTS

Appendix B - Acknowledgements

The Web of Things Working Group would like to acknowledge the contributions of several individuals:

- TBD
