TD Assertions

How WoT implementers help the TD specification!

Ege Korkan and Michael McCool

Goal of Testing

The WoT Working Group needs to verify the implementability of each specification before it can be published as a W3C Recommendation.

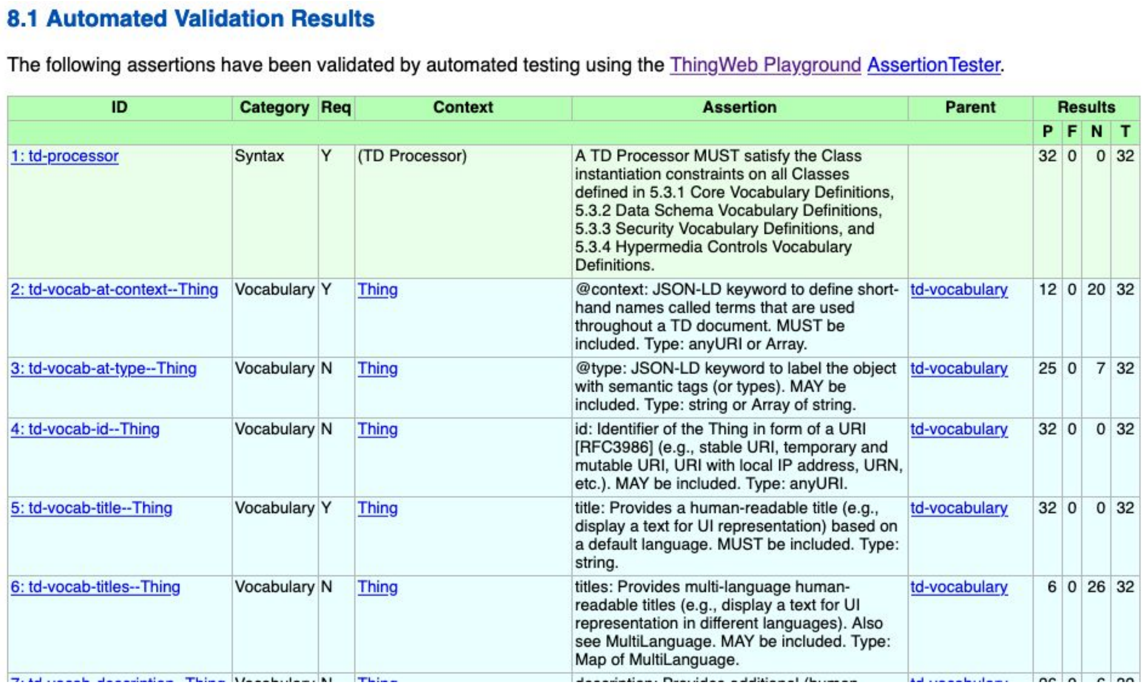
- Each specification has a set of “assertions” (statements including words like MUST, MAY, SHOULD, etc.) about each optional and required feature.
- For the Thing Description specification, we look at set of TDs submitted by participants for each implementation and do one of two things:
  1. Run a tool to verify as many features as possible automatically.
  2. For those that cannot be verified automatically, ask for “manual” validation from participants by submission of a CSV file.

Result of Testing

Testing results in a detailed **Implementation Report**:

https://w3c.github.io/wot-thing-description/testing/report11.html

- Each implementation needs a (mostly) distinct code base.
- There can be multiple implementations from the same organization.



8.1 Automated Validation Results

The following assertions have been validated by automated testing using the ThingWeb Playground AssertionTester.

ID	Category	Req	Context	Assertion	Parent	Results			
						P	F	N	T
1: td-processor	Syntax	Y	(TD Processor)	A TD Processor MUST satisfy the Class instantiation constraints on all Classes defined in 5.3.1 Core Vocabulary Definitions, 5.3.2 Data Schema Vocabulary Definitions, 5.3.3 Security Vocabulary Definitions, and 5.3.4 Hypermedia Controls Vocabulary Definitions.		32	0	0	32
2: td-vocab-at-context--Thing	Vocabulary	Y	Thing	@context: JSON-LD keyword to define short-hand names called terms that are used throughout a TD document. MUST be included. Type: anyURI or Array.	td-vocabulary	12	0	20	32
3: td-vocab-at-type--Thing	Vocabulary	N	Thing	@type: JSON-LD keyword to label the object with semantic tags (or types). MAY be included. Type: string or Array of string.	td-vocabulary	25	0	7	32
4: td-vocab-id--Thing	Vocabulary	N	Thing	id: Identifier of the Thing in form of a URI [RFC3986] (e.g., stable URI, temporary and mutable URI, URI with local IP address, URN, etc.). MAY be included. Type: anyURI.	td-vocabulary	32	0	0	32
5: td-vocab-title--Thing	Vocabulary	Y	Thing	title: Provides a human-readable title (e.g., display a text for UI representation) based on a default language. MUST be included. Type: string.	td-vocabulary	32	0	0	32
6: td-vocab-titles--Thing	Vocabulary	N	Thing	titles: Provides multi-language human-readable titles (e.g., display a text for UI representation in different languages). Also see MultiLanguage. MAY be included. Type: Map of MultiLanguage.	td-vocabulary	6	0	26	32

More than 300 lines like this!

Goal of Testfests

The Working Group verifies the implementability of each specification using events called **Testfest**.

For TD, during a Testfest we ask implementers indicate what features specified by the TD specification have been implemented.

However, new test results can be submitted at any time with a PR; it is not necessary to wait for a Testfest.

TODO

- ~~Link to spec text from implementation report at the slide titles (Cris)~~
- ~~Put a slide about the structure of the assertion explanation slides~~
  - ~~Say additional description~~
- ~~Adding TD examples (some from Daniel: sec-body-*)~~
  - Adding payload examples
- ~~Show implementation report~~
- Motivation (why is this important):
  - Mention that we can change the spec based on this
  - W3C Process requires this
  - Implementability of the specification
  - Checking interoperability among implementations
- ~~Add "more" -> Need 1 more implementation~~
- ~~Change "Resolution” to “Developer Instructions”~~

Goal of this Event

We have realized that some of our assertions are not very self-explanatory.

It is not always possible to have multiple sentences explaining an assertion in the specification. We want to use this slideset to provide additional description on how to implement such features. This way, you can start submitting your TDs and implementation results.

If features are not implemented, we have **at risk** features.

$$\downarrow$$

?

At Risk: What does it imply?

When an assertion (i.e. a feature) has less than 2 implementations, it cannot be part of the final W3C Recommendation.

- **Example:** We introduced the multipleOf term.
- We need at least 2 different implementations (at least 1 TD from each) that have this term in them in order for this feature to be included in the specification.

When we publish a Candidate Recommendation, if there is a **lack of implementation**, that assertion becomes **at risk** and is highlighted yellow in the specification.

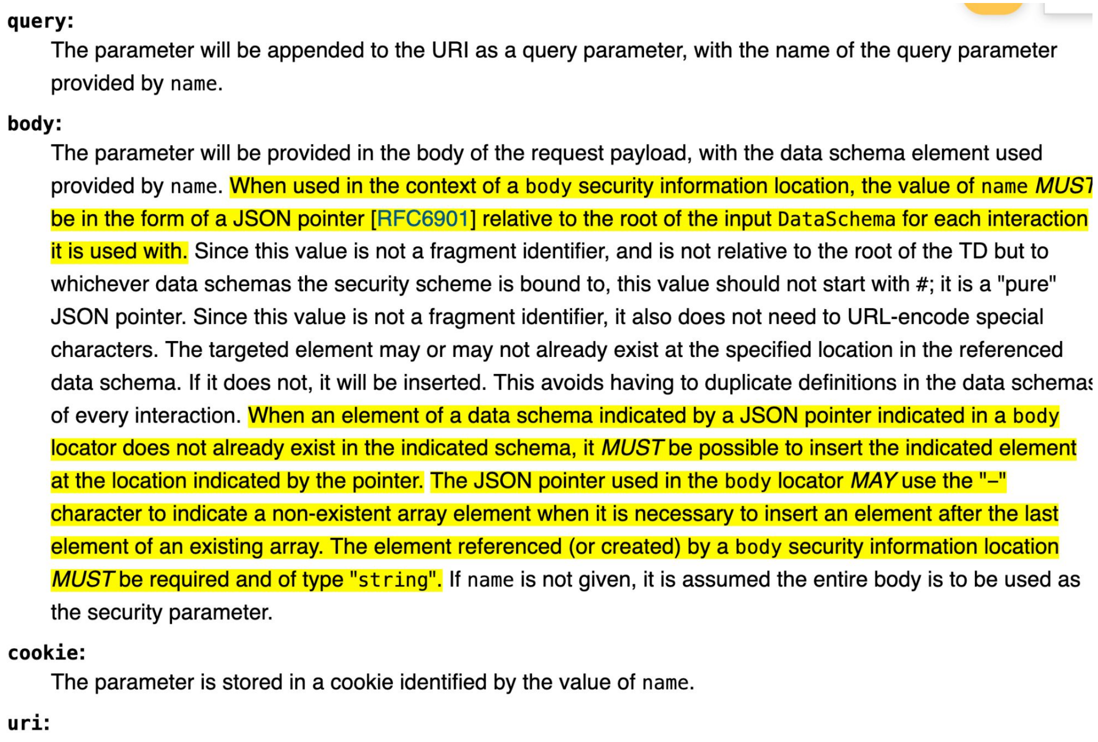
- If an assertion is still at risk by the time of transition to Proposed Recommendation, we have to **remove** the assertion (and the feature).

Assertion Context

It is not always possible to understand an assertion by itself.

Please also look at the context.

The context usually also includes additional explanatory (informative) text.



`query`:

The parameter will be appended to the URI as a query parameter, with the name of the query parameter provided by `name`.

`body`:

The parameter will be provided in the body of the request payload, with the data schema element used provided by `name`. When used in the context of a `body` security information location, the value of `name` *MUST* be in the form of a JSON pointer [RFC6901] relative to the root of the input `DataSchema` for each interaction it is used with. Since this value is not a fragment identifier, and is not relative to the root of the TD but to whichever data schemas the security scheme is bound to, this value should not start with #; it is a "pure" JSON pointer. Since this value is not a fragment identifier, it also does not need to URL-encode special characters. The targeted element may or may not already exist at the specified location in the referenced data schema. If it does not, it will be inserted. This avoids having to duplicate definitions in the data schemas of every interaction. When an element of a data schema indicated by a JSON pointer indicated in a `body` locator does not already exist in the indicated schema, it *MUST* be possible to insert the indicated element at the location indicated by the pointer. The JSON pointer used in the `body` locator *MAY* use the "–" character to indicate a non-existent array element when it is necessary to insert an element after the last element of an existing array. The element referenced (or created) by a `body` security information location *MUST* be required and of type `"string"`. If `name` is not given, it is assumed the entire body is to be used as the security parameter.

`cookie`:

The parameter is stored in a cookie identified by the value of `name`.

`uri`:

Danger Zone!

Please note that we are looking at the remaining 5% of the features after we have already done significant testing.

These are sometimes difficult to implement, and sometimes are only applicable to special cases.

How to Contribute

Instructions: Please follow our the GitHub Readme at https://github.com/w3c/wot-testing/tree/main/events/2023.03.DevMtg

It is simply about uploading TDs and CSV files

Deadline: 17 May 2023

This is the planned Recommendation transition date minus 2 weeks so that we have time to fix the specification.

- If we do not get implementation evidence by then we have to start work to remove these assertions from the specification.

If you need help with submitting results, contact Ege Korkan and Michael McCool.

There will be another event to collect inputs in the week of April 24th.

Overall List of At Risk Features in TD

From https://github.com/w3c/wot-thing-description/blob/main/testing/atrisk.csv

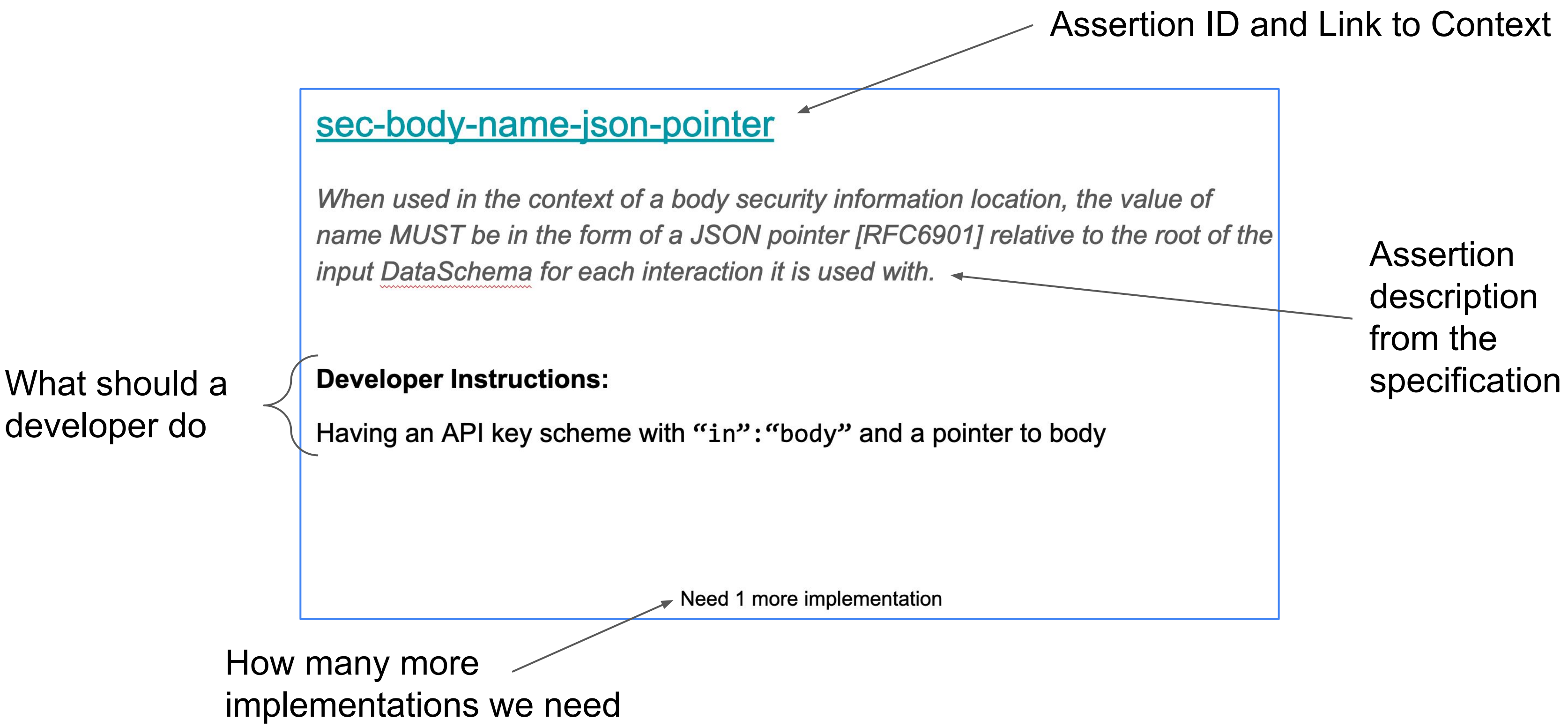
- td-producer-mixed-direction
- sec-body-name-json-pointer
- sec-body-name-json-pointer-creatable
- sec-body-name-json-pointer-array
- sec-body-name-json-pointer-type
- td-security-uri-variables-distinct
- td-security-oauth2-client-flow
- td-security-oauth2-client-flow-no-auth
- **td-security-oauth2-device-flow (2)**
- tm-derivation-validity (resolved)
- tm-versioning-1
- security-mutual-auth-td (resolved)
- security-server-auth-td (1)
- **security-context-secure-fetch (2)**
- security-oauth-limits
- security-static-context (resolved)
- security-remote-context (1)
- privacy-immutable-id-as-property (1)

Some of these need 1 more and **some need 2 more implementations.**
Corresponding slides include the number

Assertion Categories

- API Key in Body
  - sec-body-name-json-pointer
  - sec-body-name-json-pointer-creatable
  - sec-body-name-json-pointer-array
  - sec-body-name-json-pointer-type
- OAuth2
  - td-security-oauth2-client-flow
  - td-security-oauth2-client-flow-no-auth
  - td-security-oauth2-device-flow
  - security-oauth-limits
- Thing Models
  - tm-derivation-validity
  - tm-versioning-1
- Secure Context Handling
  - security-static-context
  - security-remote-context
  - security-context-secure-fetch
- TD Retrieval
  - security-mutual-auth-td
  - security-server-auth-td
- Others
  - td-security-uri-variables-distinct
  - td-producer-mixed-direction
  - privacy-immutable-id-as-property

Structure of each Assertion Explanation


Assertion ID and Link to Context
Assertion description from the specification
What should a developer do
How many more implementations we need

sec-body-name-json-pointer

When used in the context of a body security information location, the value of name MUST be in the form of a JSON pointer [RFC6901] relative to the root of the input DataSchema for each interaction it is used with.

Developer Instructions:

Having an API key scheme with “`in`”:“`body`” and a pointer to body

Need 1 more implementation

API Key in the body assertions

TD 1.1 has introduced a new API key mechanism where the key can be sent as part of the message payload (body).

Following 4 assertions are about this.

sec-body-name-json-pointer

When used in the context of a body security information location, the value of name MUST be in the form of a JSON pointer [RFC6901] relative to the root of the input DataSchema for each interaction it is used with.

Developer Instructions:

Having an API key scheme with `“in”:“body”` and a pointer to body

Need 1 more implementation

sec-body-name-json-pointer-type

The element referenced (or created) by a body security information location MUST be required and of type "string".

Developer Instructions:

When pointing to a key with the `name` keyword, that key should have type string

Need 1 more implementation

Example TD

```
{ "@context": "https://www.w3.org/2022/wot/td/v1.1", "title": "sec-body-name-json-pointer sample",
    "securityDefinitions": {
        "apikey_body": {
            "scheme": "apikey",
            "in": "body",
            "name": "/keyLocation"
        } },
    "security": [ "apikey_body" ],
    "actions": {
        "moveTo": {
            "input": {
                "type": "object",
                "properties": {
                    "x": { "type": "integer" },
                    "y": { "type": "integer" },
                    "keyLocation": { "type": "string" }
                } },
            "forms": [ {
                    "href": "http://localhost:8080/actions/moveTo"
                } ] } } }
```

sec-body-name-json-pointer-creatable

When an element of a data schema indicated by a JSON pointer indicated in a body locator does not already exist in the indicated schema, it MUST be possible to insert the indicated element at the location indicated by the pointer.

Developer Instructions:

If when pointing to a key in a payload schema, if the target does not exist in the schema, then the API key should be inserted during the generation of the payload.

Note: Without this feature the key would need to be added to all payload data schemas.

Need 1 more implementation

Example TD and Payload

```
{ "@context": "https://www.w3.org/2022/wot/td/v1.1", "title": "sec-body-name-json-pointer-creatable sample",
    "securityDefinitions": {
        "apikey_body": {
            "scheme": "apikey",
            "in": "body",
            "name": "/keyLocationToCreate"
        } },
    "security": [ "apikey_body" ],
    "actions": {
        "moveTo": {
            "input": {
                "type": "object",
                "properties": {
                    "x": { "type": "integer" },
                    "y": { "type": "integer" }
                } },
            "forms": [ {
                    "href": "http://localhost:8080/actions/moveTo"
                } ] } } }
```

Payload

```
{
  "x": 12,
  "y": 100,
  "keyLocationToCreate": "mySecret"
}
```

sec-body-name-json-pointer-array

The JSON pointer used in the body locator MAY use the "-" character to indicate a non-existent array element when it is necessary to insert an element after the last element of an existing array.

Developer Instructions:

Pointer having value “-” means that payload (which should be an array) should have the API key concatenated with it as the last element.

Note: Without this feature the key would need to be added to all payload data schemas and would not be flexible enough for arrays with varying length

Need 1 more implementation

Example TD and Payload

```
{ "@context": "https://www.w3.org/2022/wot/td/v1.1", "title": "sec-body-name-json-pointer-array sample",
    "securityDefinitions": {
        "apikey_body": {
            "scheme": "apikey",
            "in": "body",
            "name": "-"
        } },
    "security": [ "apikey_body" ],
    "actions": {
        "moveIn": {
            "input": {
                "type": "array",
                "items": [
                  { "type": "integer" },
                  { "type": "integer" }
                ],
                "additionalItems": { "type": "string" }
            },
            "forms": [ {
                    "href": "http://localhost:8080/actions/moveIn"
                } ] } } }
```

Payload

```
[
  12, 100, "mySecret"
]
```

OAuth2

These are about how to configure the OAuth2 security scheme field inside securityDefinitions

The code flow is very *detailed*, and after that, we keep on removing some steps in the other flows to make it more simple. Hence for some, we need only the token server URL and for some, we need authorization and token server URL.

td-security-oauth2-client-flow

For the client flow ***`token`*** *MUST be included.*

Developer Instructions:

TDs always need the **`token`** key set when the OAuth2 client flow is used. Of course this means OAuth2's client flow needs to be implemented also. This indicates the URL of the token server.

Need 1 more implementation

Example TD

```
{
   "title": "myThing",
   "@context": "https://www.w3.org/2022/wot/td/v1.1",
   "securityDefinitions":{
       "oauth2_sc":{
           "scheme":"oauth2",
           "flow": "client",
           "token":"https://dev-37050809.okta.com/oauth2/aus7n8imnqQY5YWWq5d7"
       }
   },
   "security":"oauth2_sc",
   // ...
}
```

td-security-oauth2-client-flow-no-auth

For the client flow `authorization` MUST NOT be included.

Developer Instructions:

Do **not** put a value for `authorization` in the security scheme when using the OAuth2 client flow. In other OAuth2 flows this value would point to an authorization server but this is not used in the client flow. Of course this means OAuth2's client flow needs to be implemented also.

Need 1 more implementation

Example TD

```
{
   "title": "mygardenthing",
   "@context": "https://www.w3.org/2022/wot/td/v1.1",
   "securityDefinitions":{
       "oauth2_sc":{
           "scheme":"oauth2",
           "flow": "client",
           "token":"https://dev-37050809.okta.com/oauth2/aus7n8imnqQY5YWWq5d7",
           ~~"authorization":"https://myAuth.server.com/myId"~~
       }
   },
   "security":"oauth2_sc",
   // ...
}
```

td-security-oauth2-device-flow

*For the device flow both **`authorization`** and **`token`** MUST be included.*

Developer Instructions:

In the OAuth2 device flow, values for both the authorization and token keys should be present at the same time in the security scheme. Of course this means OAuth2's device flow needs to be implemented.

Need 2 new implementations

Example TD

```
{
   "title": "mygardenthing",
   "@context": "https://www.w3.org/2022/wot/td/v1.1",
   "securityDefinitions":{
       "oauth2_sc":{
           "scheme":"oauth2",
           "flow": "device",
           "token":"https://dev-37050809.okta.com/oauth2/aus7n8imnqQY5YWWq5d7",
           "authorization":"https://myAuth.server.com/myId"
       }
   },
   "security":"oauth2_sc",
   // ...
}
```

security-oauth-limits

To limit the scope and duration of access to Things, tokens SHOULD be used to manage access.

Developer Instructions:

This assertion is made in the context of managing limited duration accesses, i.e. providing a visitor with a temporary pass. Tokens are ideal for this situation since they can be given expiry times. Tokens can be used directly with the bearer security scheme or as part of OAuth2 flows.

In fact, ANY implementation that uses bearer tokens, as long as those tokens are given expiry times, can be considered as satisfying this assertion.

Need 1 new implementation

Thing Models

A Thing Model mainly describes interaction affordances such as the Properties, Actions, and Events and common metadata. This paradigm can be compared with abstract class or interface definition (~Thing Model) in object-oriented programming to create objects (~Thing Descriptions).

tm-derivation-validity

When a Thing Descriptions is instantiated by relying on a Thing Model, it SHOULD be valid according to that Thing Model.

Developer Instructions:

TM to TD generators that are doing their job correctly should pass this: the TD needs to be an instance of the TM.

Need 0 more implementations

Example TM and its valid TD

Simple TM

```
{
  // ...
  "@type": "tm:ThingModel",
  "title": "Smart Pump",
  "id": "urn:example:{{RANDOM_ID_PATTERN}}",
  "description": "Smart Pump live plant and 
simulator",
  "version" : {"model" : "1.0.0" },
  "properties" : {
    "temp" : { "type" : "string" //... } }
       // ...
}
```

Needs to be equal

Needs to be following pattern

Needs to be equal

Needs to be equal

Simple TD instance

```
{
  // ...
  "@type": "Thing",
  "title": "Smart Pump",
  "id": "urn:example:123-321-123-321",
  "description": "Smart Pump live plant and simulator",
  "version" : {"instance": "1.0.0", "model": "1.0.0" },
  "properties" : {
    "temp" : { "type" : "string" //... } }
  "links" : [{ // ...}],
       // ...
}
```

See also Example 66 (TM) Example 67 (TD)

Example TM and its **invalid** TD

Simple TM

```
{
  // ...
  "@type": "tm:ThingModel",
  "title": "Smart Pump",
  "id": "urn:example:{{RANDOM_ID_PATTERN}}",
  "description": "Smart Pump live plant and
simulator",
  "version" : {"model" : "1.0.0" },
  "properties" : {
    "temp" : { "type" : "string" //... } }
       // ...
}
```

Simple TD instance

```
{
  // ...
  "@type": "Thing",
  "title": "Smart Pump 1",
  "id": "urn:mac:123-321-123-321",
  "description": "Smart Pump live plant and simulator",
  "version" : {"instance": "1.0.0", "model": "1.0.0" },
  "properties" : {
    "temp" : { "type" : "number" //... } }
  "links" : [{ // ...}],
       // ...
}
```

tm-versioning-1

When the Thing Model definitions change over time, this SHOULD be reflected in the version container.

Developer Instructions:

If you version TMs, you are doing this.

Need 1 more implementation

Two Versions of a TM

TM Version 1.0.0

```
{
  // ...
  "@type": "tm:ThingModel",
  "title": "Smart Pump",
  "id": "urn:example:{{RANDOM_ID_PATTERN}}",
  "description": "Smart Pump live plant and
simulator",
  "version" : {"model" : "1.0.0" },
  "properties" : {
    "temp" : { "type" : "string" //... } }
       // ...
}
```

TM Version 1.0.1

```
{
  // ...
  "@type": "tm:ThingModel",
  "title": "Smart Pump {{INCREMENT}}",
  "id": "urn:example:{{RANDOM_ID_PATTERN}}",
  "description": "Smart Pump live plant and
simulator",
  "version" : {"model" : "1.0.1" },
  "properties" : {
    "temp" : { "type" : "string" //... } }
       // ...
}
```

Secure Context Handling

These are about how implementations should fetch and/or manage ontologies.

security-static-context

Constrained implementations SHOULD use vetted versions of their supported context extensions managed statically or as part of a secure update process.

Developer Instructions:

Implementations that read TDs should generally avoid downloading context extensions, as it is expensive and a possible privacy risk, and should instead support a fixed set of extensions "baked into" their code, UNLESS the system can support full JSON-LD/RDF processing.

If you don't support RDF processing then you probably implement this assertion - possibly trivially, if your implementation does not support ANY extensions. However, any vocabulary that uses a prefix to support a protocol, e.g. **`htv`**, can be considered as supporting a vocabulary extension.

Need 0 more implementations

security-remote-context

Constrained implementations SHOULD NOT follow links to remote contexts.

Developer Instructions:

This is actually very similar to security-static-context: it just means don’t fetch links to context files listed in the @context unless you absolutely have to. If the context can be recognized by the URL, it’s better for the implementation to have the meaning of that context “baked into” it rather than fetching the definition. The extension vocabularies associated with contexts need to be “fixed” for this reason.

If a Consumer has a built-in understanding of any extension vocabulary, for instance even for a protocol binding such as htv, then it probably implements this.

Need 1 new implementation

security-context-secure-fetch

If it is necessary to fetch a context definition file, an implementation SHOULD first attempt to use HTTP over TLS even when only an HTTP URL is given.

Developer Instructions:

Even if your implementation sees http:// it should first attempt to use https://, that is TLS, to fetch context definitions. This is just because context links are often conventionally given as http, but we still want to fetch them securely whenever possible (and necessary).

Compare with the previous assertion: ***only systems that do RDF processing need to even worry about this.*** At present, this mostly means implementations of Thing Description Directories supporting SPARQL as part of WoT Discovery.

Need 2 new implementations

TD Retrieval

Secure retrieval of TDs

These relate to fetching TDs from external sources, e.g. web servers. Note that in the next iteration of the standards these may be moved to Discovery and they overlap with some assertions there.

security-mutual-auth-td

Thing Descriptions SHOULD be obtained only through mutually authenticated secure channels.

Developer Instructions:

This means that “mutual” TLS should (ideally) be used when fetching a TD. In this case both the server and the client need to present certificates to each other. Note that on the public internet often only one-way TLS is used, identifying the server to the client. But identifying the client to the server provides better security, and this is often used in internal corporate web servers. If a web server requests a client certificate the browser will ask the user to provide one.

Need 0 more implementations

security-server-auth-td

In cases where the Consumer is associated with a person, e.g. browsers, TDs MAY be obtained through a channel where only the TD provider is authenticated.

Developer Instructions:

This may seem to conflict with security-mutual-auth-td, but the trouble with mutual TLS is that it reveals (in fact, *proves*) the identity of the client. So in situations where the client wants to protect their identity they should NOT use client certificates. This is in fact the normal mode of operation of the web on the open internet - it is an example of security and privacy sometimes being in conflict.

However, any implementation that does not use a client certificate when fetching a TD (but ***does*** use https (TLS) to authenticate the server) satisfies this assertion.

Need 1 new implementation

Others

Assertions that did not fit into a specific category

privacy-immutable-id-as-property

Ideally, any required immutable identifiers SHOULD only be made available via affordances, such as a property, whose value can only be obtained after appropriate authentication and authorization, and managed separately from the TD identifier.

Developer Instructions:

Some systems or contexts (e.g. medical) may require devices to have immutable ids, but these can be privacy risks. Rather than using such ids directly as the id of the TD, consider having the immutable id of the Thing retrievable via a property affordance and use a different id (which can be updated if necessary) for the TD. Property affordances can be protected with additional authorization controls, unlike the content of the TD.

To satisfy this assertion you just need a property that returns an immutable id (like the MAC address).

Need 1 more implementation

Example TD

```
{
 "@context": "https://www.w3.org/2022/wot/td/v1.1",
 "title": "OxygenConcentrator",
 "description": "An internet connected portable oxygen concentrator",
 "securityDefinitions": {"basic_sc": {"scheme": "basic"}},
 "security": "basic_sc",
 "properties": {
   "thingId": {
     "type": "string","readOnly": true
     //...
   },
   "ownerInformation": {
     "type": "object",
     "properties": {
       "name": {"type": "string"},
       "userId": {"type": "string"}
     },
     "readOnly": true,
     //...
   }
 }
}
```

no id field

Private data behind affordances that need Basic Auth

td-security-uri-variables-distinct

The names of URI variables declared in a Security Scheme MUST be distinct from all other URI variables declared in the TD.

Developer Instructions:

Any uriVariables declared in security schemes can’t conflict with any others declared elsewhere in the TD.

Note that this is trivially satisfied if there are no other uriVariables declared outside of a security scheme, also.

Need 2 new implementations

Example TD

```
{
   "@context": "http://www.w3.org/ns/td",
   // ...
   "uriVariables": {
       "apiversion": {
           "type": "string", "enum":["v1","v2.1"]
       }
   },
   "properties": {
       "weather": {
           // ...
           "uriVariables": {
               "city": {"type": "string"}
           },
           "forms": [{"href": "http://example.org/{apiversion}/weather/{city}" }]
       }
   }
}
```

They should be different

td-producer-mixed-direction

TD producers SHOULD attempt to provide mixed direction strings in a way that can be displayed successfully by a naive user agent.

Developer Instructions:

Mixed direction strings may occur when text from two languages are mixed in one string, for example including an English word (like a product name) in the middle of Hebrew text. Very often systems need to guess at the rendering direction and use various rules to do so, but generally the first character is what determines the overall direction. **Strings should be worded so the first character gives the correct direction for the entire string.**

The real problem is generating examples for this. If your native language is Arabic or Hebrew, please consider submitting TDs with mixed language strings. Tests should then check if these are displayed correctly, e.g. in browsers generating dashboards from these TDs.

If you are a native speaker of a language with Right To Left writing, please try implementing this.

Need 1 more implementation

Example

Correct TD here