Writing Devicetree Bindings in json-schema

Devicetree bindings are written using json-schema vocabulary. Schema files are written in a JSON-compatible subset of YAML. YAML is used instead of JSON as it is considered more human readable and has some advantages such as allowing comments (Prefixed with '#').

Also see :ref:`example-schema`.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\devicetree\bindings\[linux-master][Documentation][devicetree][bindings]writing-schema.rst`, **line 11**); ***backlink***
>
> Unknown interpreted text role "ref".

Schema Contents

Each schema doc is a structured json-schema which is defined by a set of top-level properties. Generally, there is one binding defined per file. The top-level json-schema properties used are:

$id
: A json-schema unique identifier string. The string must be a valid URI typically containing the binding's filename and path. For DT schema, it must begin with "http://devicetree.org/schemas/". The URL is used in constructing references to other files specified in schema "$ref" properties. A $ref value with a leading '/' will have the hostname prepended. A $ref value with only a relative path or filename will be prepended with the hostname and path components of the current schema file's '$id' value. A URL is used even for local files, but there may not actually be files present at those locations.

$schema
: Indicates the meta-schema the schema file adheres to.

title
: A one-line description on the contents of the binding schema.

maintainers
: A DT specific property. Contains a list of email address(es) for maintainers of this binding.

description
: Optional. A multi-line text block containing any detailed information about this binding. It should contain things such as what the block or device does, standards the device conforms to, and links to datasheets for more information.

select
: Optional. A json-schema used to match nodes for applying the schema. By default, without 'select', nodes are matched against their possible compatible-string values or node name. Most bindings should not need select.

allOf
: Optional. A list of other schemas to include. This is used to include other schemas the binding conforms to. This may be schemas for a particular class of devices such as I2C or SPI controllers.

properties
: A set of sub-schema defining all the DT properties for the binding. The exact schema syntax depends on whether properties are known, common properties (e.g. 'interrupts') or are binding/vendor-specific properties.

A property can also define a child DT node with child properties defined under it.

For more details on properties sections, see 'Property Schema' section.

patternProperties
: Optional. Similar to 'properties', but names are regex.

required
: A list of DT properties from the 'properties' section that must always be present.

examples
: Optional. A list of one or more DTS hunks implementing the binding. Note: YAML doesn't allow leading tabs, so spaces must be used instead.

Unless noted otherwise, all properties are required.

Property Schema

The 'properties' section of the schema contains all the DT properties for a binding. Each property contains a set of constraints using json-schema vocabulary for that property. The properties schemas are what are used for validation of DT files.

For common properties, only additional constraints not covered by the common, binding schema need to be defined such as how many values are valid or what possible values are valid.

Vendor-specific properties will typically need more detailed schema. With the exception of boolean properties, they should have a reference to a type in schemas/types.yaml. A "description" property is always required.

The Devicetree schemas don't exactly match the YAML-encoded DT data produced by dtc. They are simplified to make them more compact and avoid a bunch of boilerplate. The tools process the schema files to produce the final schema for validation. There are currently 2 transformations the tools perform.

The default for arrays in json-schema is they are variable-sized and allow more entries than explicitly defined. This can be restricted by defining 'minItems', 'maxItems', and 'additionalItems'. However, for DeviceTree Schemas, a fixed size is desired in most cases, so these properties are added based on the number of entries in an 'items' list.

The YAML Devicetree format also makes all string values an array and scalar values a matrix (in order to define groupings) even when only a single value is present. Single entries in schemas are fixed up to match this encoding.

Testing

Dependencies

The DT schema project must be installed in order to validate the DT schema binding documents and validate DTS files using the DT schema. The DT schema project can be installed with pip:

```
pip3 install dtschema
```

Note that 'dtschema' installation requires 'swig' and Python development files installed first. On Debian/Ubuntu systems:

```
apt install swig python3-dev
```

Several executables (dt-doc-validate, dt-mk-schema, dt-validate) will be installed. Ensure they are in your PATH (~/.local/bin by default).

Running checks

The DT schema binding documents must be validated using the meta-schema (the schema for the schema) to ensure they are both valid json-schema and valid binding schema. All of the DT binding documents can be validated using the `dt_binding_check` target:

```
make dt_binding_check
```

In order to perform validation of DT source files, use the `dtbs_check` target:

```
make dtbs_check
```

Note that `dtbs_check` will skip any binding schema files with errors. It is necessary to use `dt_binding_check` to get all the validation errors in the binding schema files.

It is possible to run both in a single command:

```
make dt_binding_check dtbs_check
```

It is also possible to run checks with a subset of matching schema files by setting the `DT_SCHEMA_FILES` variable to a specific schema file or pattern.

```
make dt_binding_check DT_SCHEMA_FILES=trivial-devices.yaml
make dt_binding_check DT_SCHEMA_FILES=/gpio/
make dtbs_check DT_SCHEMA_FILES=trivial-devices.yaml
```

json-schema Resources

JSON-Schema Specifications

Using JSON Schema Book

Annotated Example Schema

Also available as a separate file: :download:`example-schema.yaml`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\devicetree\bindings\[linux-master][Documentation][devicetree][bindings]writing-schema.rst, line 176); *backlink*

Unknown interpreted text role "download".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\devicetree\bindings\[linux-master][Documentation][devicetree][bindings]writing-schema.rst, line 178)

Unknown directive type "literalinclude".

```
.. literalinclude:: example-schema.yaml
```