Linaro connect
Hong Kong 2018

Devicetree Schema Documentation and Validation

Grant Likely - Arm

devicetree
.org

Devicetree Schema Documentation and Validation

The problem: too easy to get devicetree wrong

- Data must be encoded in very specific ways
- Toolchain provides little validation
- No checks against documented schema (aka. bindings)
  - Schemas are loosely structure prose
  - Not machine readable
- Steep learning curve

Project Goals

- Define a DT schema language
  - Human friendly
  - Machine readable
  - Include binding documentation
- Better tooling
  - Validate DTS files at build time
  - Validate DT Schema files are in the correct format
  - Useful error and warning messages
- Leverage existing technology
  - Use existing schema validation framework
    - Extended to handle quirks of DT
  - Don’t write a lot of code!
  - Don’t define an entirely new language!
- Generate Specification Documentation from Schema files

Prototype Implementation

http://github.com/robherring/yaml-bindings

Based on:

- Python3
- YAML 1.2 file format
- JSON Schema vocabulary
  - As implemented in Python jsonschema library
- Sphinx-Doc extension

~250 lines of python code

~250 lines of metaschema files

~250 lines of schema files

YAML File Format

http://www.yaml.org

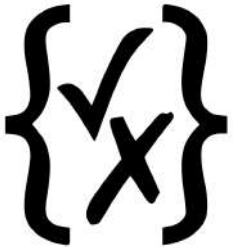
- Human friendly
- Portable
- Structured
- Simple transcode DTS to YAML
- Extensible with custom datatypes
- Superset of JSON


```
%YAML 1.2
---
YAML: YAML Ain't Markup Language

What It Is: YAML is a human friendly data serialization
  standard for all programming languages.

YAML Resources:
  YAML 1.2 (3rd Edition): http://yaml.org/spec/1.2/spec.html
  YAML 1.1 (2nd Edition): http://yaml.org/spec/1.1/
  YAML 1.0 (1st Edition): http://yaml.org/spec/1.0/
  YAML Issues Page: https://github.com/yaml/yaml/issues
  YAML Mailing List: yaml-core@lists.sourceforge.net
  YAML IRC Channel: "#yaml on irc.freenode.net"
  YAML Cookbook (Ruby): http://yaml4r.sourceforge.net/cookbook/
(local)
  YAML Reference Parser: http://ben-kiki.org/ypaste/

Projects:
  C/C++ Libraries:
  - libyaml       # "C" Fast YAML 1.1
  - Syck          # (dated) "C" YAML 1.0
  - yaml-cpp      # C++ YAML 1.2 implementation
 Python:
  - PyYAML        # YAML 1.1, pure python and libyaml binding
  - ruamel.yaml   # YAML 1.2, update of PyYAML with round-tripping
```

JSON Schema Vocabulary

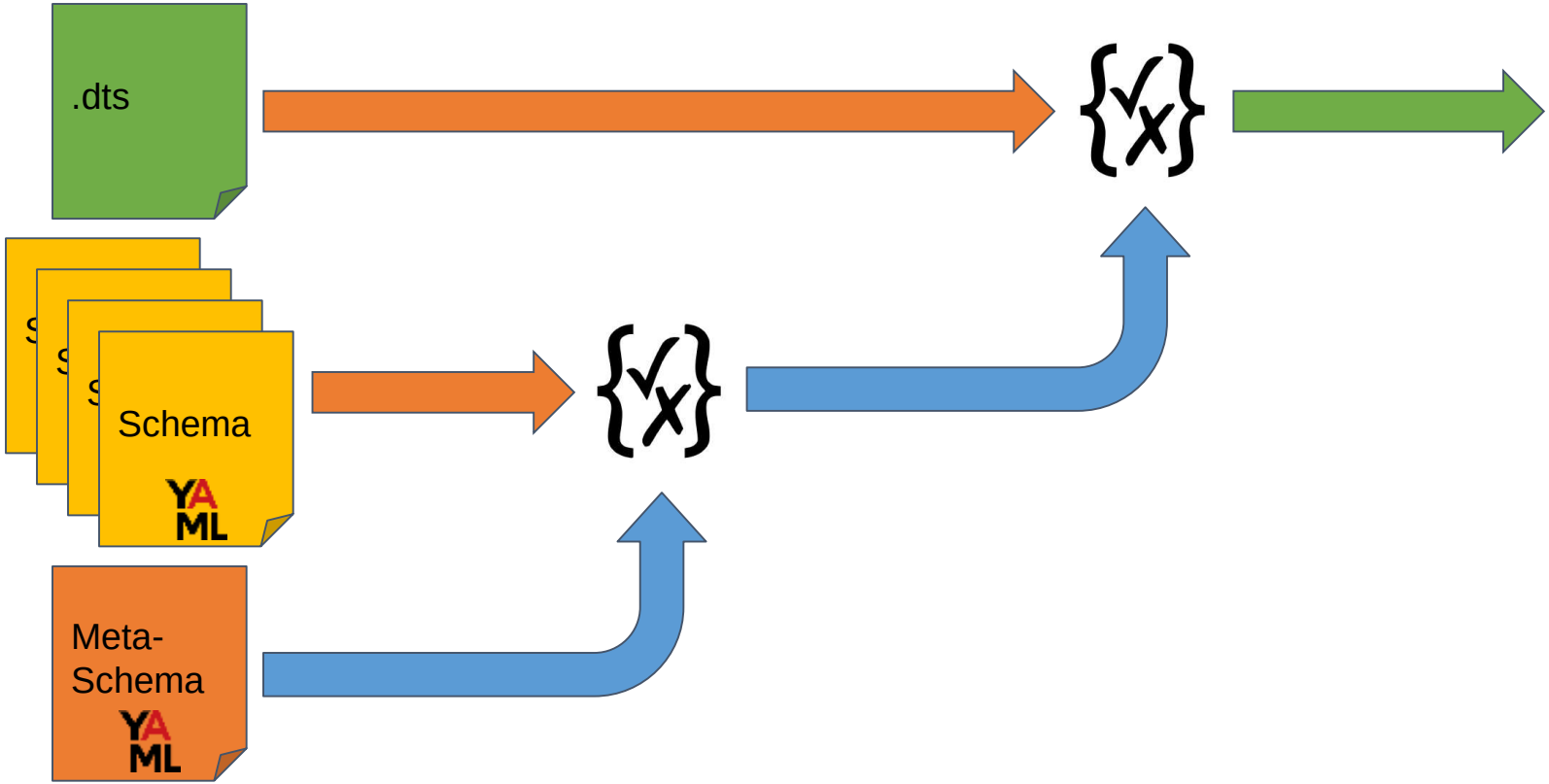
http://json-schema.org

- Intended for and encoded in JSON
  - Works fine with YAML
- Defines vocabulary and processing model for validation
  - Can extend with DT specific vocabulary
- JSON Schema nodes apply constraints on target document



```
{
    "title": "Person",
    "type": "object",
    "properties": {
        "firstName": {
            "type": "string"
        },
        "lastName": {
            "type": "string"
        },
        "age": {
            "description": "Age in years",
            "type": "integer",
            "minimum": 0
        }
    },
    "required": ["firstName", "lastName"]
}
```

DT Schema Conceptual Model



Examples -- (best to go look at project)

Meta Schema

```
%YAML 1.2
---
$id: "http://devicetree.org/meta-schemas/core.yaml#"
$schema: "http://json-schema.org/draft-06/schema#"
description: "Metaschema for devicetree binding documentation"

allOf: [ { $ref: "http://json-schema.org/draft-06/schema#" } ]

[...]

properties:
  $id:
    pattern: 'http://devicetree.org/(test/)?schemas/.*\.yaml#'
  $schema:
    const: "http://devicetree.org/meta-schemas/core.yaml#"

[...]

Required: [$id, $schema, version, title, maintainers, description]

additionalProperties: false
```

Schema

```
%YAML 1.2
---
$id: http://devicetree.org/schemas/root-node.yaml#
$schema: http://devicetree.org/meta-schemas/core.yaml#

title: Common root node
description: |
  Common properties always required in the root node of the tree

maintainers:
  - Device Tree <dt@kernel.org>

select:
  required: ["$path"]
  properties:
    $path: {enum: ["/"]}

properties:
  compatible: {}
  model:
    type: string
  "#address-cells": {}
  "#size-cells": {}
  memory: {}
  aliases: { "$ref": "http://devicetree.org/schemas/aliases.yaml" }
```

Generate Documentation

```
title: /aliases Node
description: |
  A devicetree may have an aliases node (``/aliases``) that defines one or more
  alias properties.

patternProperties:
  "^[a-z][a-z0-9\\-]*$": { type: string }
additionalProperties: false

examples:
  example1: |
    aliases {
      serial0 = "/simple-bus@fe000000/serial@11c500";
      ethernet0 = "/simple-bus@fe000000/ethernet@31c000";
    };

maintainers:
  - Devicetree Specification Mailing List <devicetree-spec@vger.kernel.org>
```

Demonstration

- Devicetree Validation

```
$ ./dt-validate.py test/juno.cpp.yaml
```

- Devicetree Schema Validation

```
$ ./tools/dt-doc-validate test/schemas/good-example.yaml
$ ./tools/dt-doc-validate test/schemas/bad-example.yaml
```

- Running Testcases

```
$ make test
```

Next Steps

- Start requiring binding files to be YAML encoded
  - Can be enforced in kernel
- Add DTSchema support into DTC
  - Add YAML output filter to DTC, or
  - Update libfdt Python bindings to expose JSON object model
- Get working with DTS files in kernel
- Flush out core schema files
- Decide where code should live
  - Separate Repo?
  - In DTC repo?
  - In Devicetree-schema repo?
  - In kernel?

DTSchema hacking Tuesday and Wednesday mornings in Core Hacking Room

Thank You

#HKG18

HKG18 keynotes and videos on: connect.linaro.org

For further information: www.linaro.org