Drupal 8 configuration schema cheat sheet

1.5 - April 9. 2015.

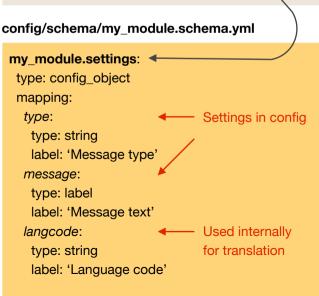
Configuration schema in Drupal 8 is used to describe the structure of configuration files. It is then applied to:

- Typecast configuration to ensure type consistency (see StorableConfigBase::castValue())
- Automated persistence of configuration entity properties (see ConfigEntityBase::toArray())
- Automated generation of the configuration translation user interface (see the core module)

A simple example

config/install/my module.settings.yml





Basic schema types

Core provides the following data types. Contributed modules may define new base types. More are defined in core.data types.schema.yml.

Scalar types

boolean

integer

float

string

uri

email

List types

mapping: known keys

sequence: unknown keys

Common subtypes

label: short & translatable

text: long & translatable

config object: object root

config_entity: entity root

Subtyping

All of config schema is about subtyping from existing types. The simple example earlier is subtyping config object (which is a subtype of *mapping*) further defining keys with their own types.

The config object or config entity types are for the root of config schema definitions as appropriate.

Types route, filter, mail, etc. are provided for common complex internal data structures.

Dynamic type with [%parent]

Exact types may not be known ahead of time and may depend on the data. Schema allows to define types based on the data as well. Let's say the type of message may depend on the type value: either a list of messages or a simple warning message. Let's use 'multiple' for the list case and keep 'warning' for the single line message.

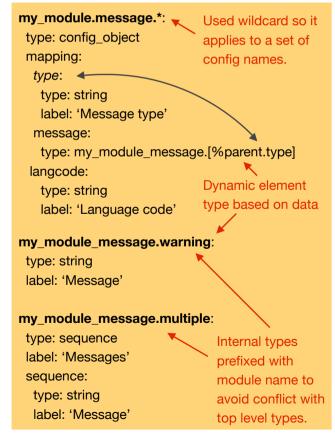
config/install/my module.message.single.yml

type: warning message: 'Hello!' langcode: en

config/install/my module.message.multiple.yml

type: multiple message: - 'Hello!' - 'Hi!' langcode: en

config/schema/my module.schema.yml



Chaining is possible as %parent.%parent.type, etc.

Dynamic type with [type]

If the data to vary your type by is under the data to be typed, that is when [type] becomes useful.

config/install/my_module.message.single.yml

```
message:
type: warning
value: 'Hello!'
langcode: en
```

config/install/my module.message.multiple.yml

```
message:
type: multiple
value:
- 'Hello!'
- 'Hi!'
langcode: en
```

config/schema/my_module.schema.yml

```
my_module.message.*:
                           Use the type key
type: config_object
                           under the message
mapping:
 message:
  type: my module message.[type]
 [...]
my_module_message.warning:
type: mapping
                                Need to define
                                type and value
my module message.multiple:
                                keys as
type: mapping
                                appropriate.
 [...]
```

You may also define a **my_module_message_base** base type that includes common keys like 'type' and extend from that with any custom keys per type.

Dynamic type with [%key]

config/install/my_module.messages.yml

```
messages:

'single:1': 'Hello!'

'single:2': 'Hi!'

'multiple:1':

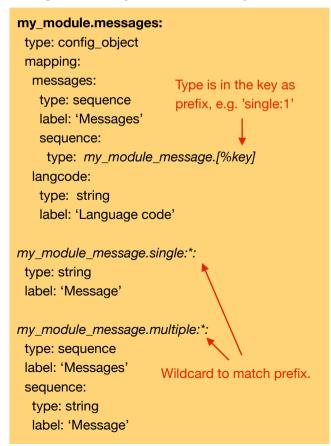
- 'Good morning!'

- 'Good night!'

langcode: en
```

This is now a list of arbitrary message elements.

config/schema/my_module.schema.yml



Schema debugging

To debug configuration schemas use the Configuration Inspector module (http://drupal.org/project/config_inspector) which helps you find schema mismatches with active configuration and inspect how your schema is applied to your configuration.

Schema testing

- All TestBase deriving tests in core now use \$strictConfigSchema = TRUE which results in strict scheme adherence testing for all configuration saved. Only opt out of this if you really need to. Your schema should match your data and pass this test.
- Use SchemaCheckTestTrait in your test to check for specific config files only.

More documentation

See https://www.drupal.org/node/1905070 for even more configuration schema documentation and examples.

Issues?

- For issues with core configuration schemas, tag them with 'Configuration schema' and 'Configuration system' and pick the appropriate module as component.
- For issues with the configuration schema system itself, use the 'configuration system' component and also tag with 'Configuration schema'.

Created by Gábor Hojtsy

https://www.drupal.org/user/4166/contact