A Runtime Configuration Registry for RIOT

Lasse Jonas Rosenow

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG

September 19, 2023

Motivation

Many applications in the IoT use parameters that need to be changed at runtime.

- Authentication credentials
- Sampling rate of a measurement
- Color of an LED

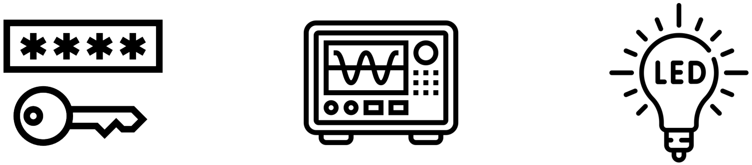


Figure: Flaticon.com

Problem

RIOT does not provide an API for runtime parameters.

$\Rightarrow$ Each application has to implement its own runtime configuration strategy.

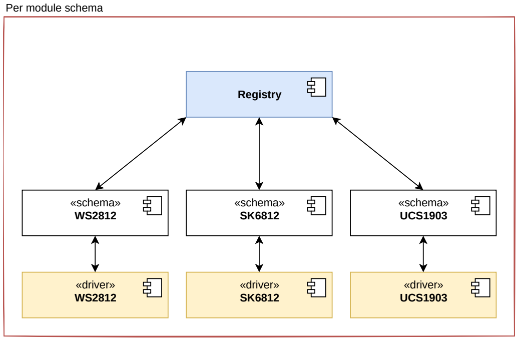
$\Rightarrow$ Unnecessary and redundant implementation effort

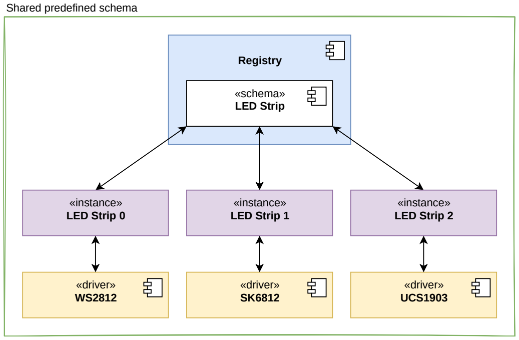
Table of Contents

Table of Contents

Shared Configuration Schemas

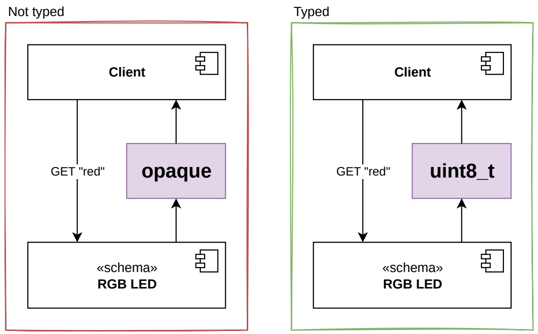
- Modules / drivers of the same kind must share their Configuration Schema => Consistent API
- Configuration parameter values must be stored in "instances" of the Configuration Schema





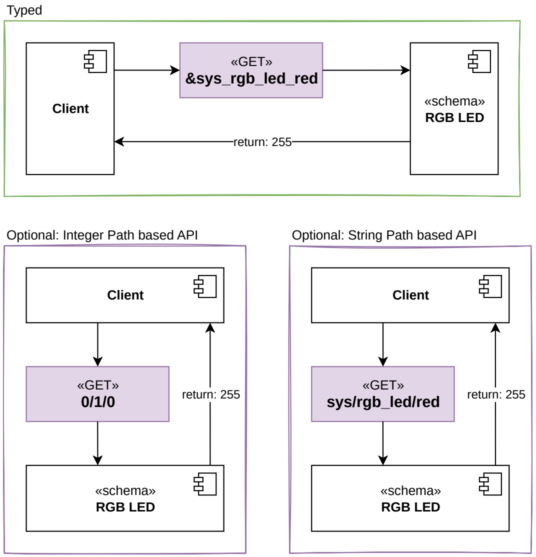
Typed Configuration Parameters

- Expose configuration parameter types
- Guaranteeing a specific type makes the API more robust
- Important for (External) Configuration Managers
- Parameter values are stored in the according programming language type



Pointer based API

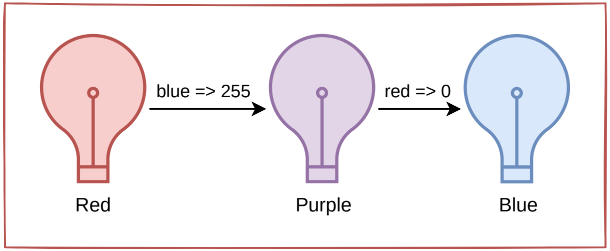
- Core API consumes Pointers to Structs such as Configuration Schemas, Configuration Parameters etc.
- Optional:
  - Integer Path based API
  - String Path based API



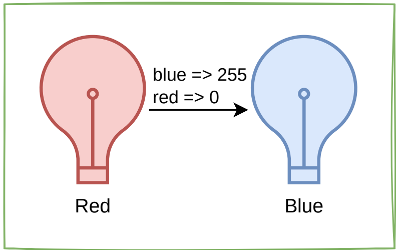
Transactionally Commit Configuration Changes

- Apply multiple new configuration parameter values at the same time
- For example an RGB LED:
  - 3 configuration parameters (r, g, b)
  - Not applying all 3 new values at the same time can cause an "undesired color" in between

Commit per parameter



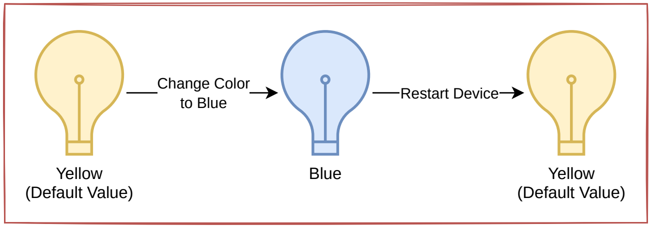
Transactional Commit



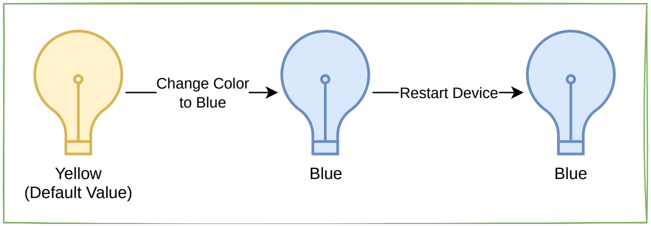
Persistent Configurations

- Optionally recover configurations after a device restart
- Optionally write configuration values to a non-volatile storage

Not persisted configurations



Persisted configurations



Integration with External Configuration Managers

- Allow external Configuration Managers to read and update configurations
- Support for:
  - LwM2M schema mapping
  - Custom CLI
  - Custom CoAP based API
  - Custom MQTT based API

Table of Contents

Apache Mynewt: Config

- Configuration Management Module of the Mynewt Operating System developed by Apache
- API
  - Get
  - Set
  - Commit
  - Export
  - Load
  - Save
- Configurations are identified by unique “String Paths”
- Configuration Values are encoded as “Strings”
- Each module provides the “String Path” (de)serialization and internal logic via a “Handler” interface for each API function

Comparing our Requirements to “Apache Mynewt: Config”

- Pro
  - Persist configurations
  - Transactionally apply multiple configuration changes
  - (String Path based API)
- Contra
  - Only String Path based API
    - No Integer Path based API
    - No Pointer based API
  - No shared Configuration Schemas (Per module Configuration Schemas)
  - No parameter types (everything is a string)

Table of Contents

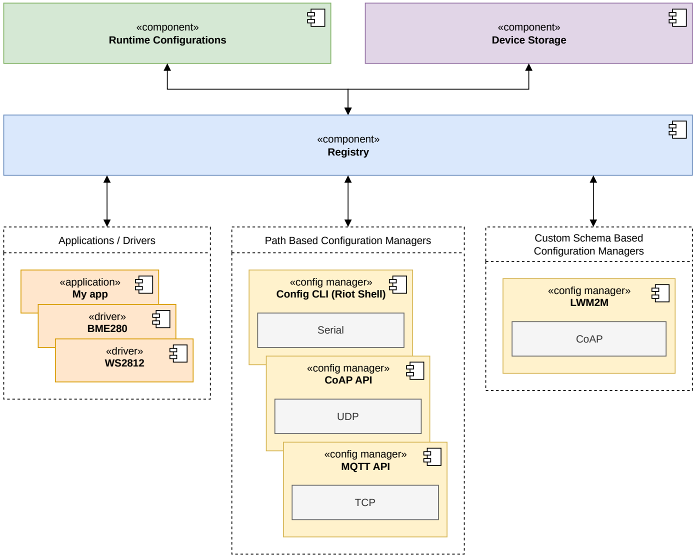
RIOT Registry

Registry to manage configurations of RIOT modules and applications

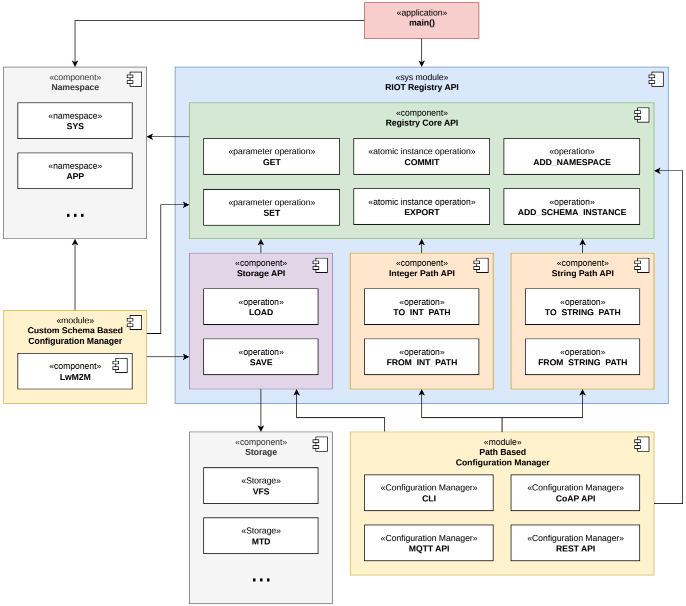
Based on “Apache Mynewt: Config” and its RIOT adaption by Leandro and José et al:

- `github.com/RIOT-OS/RIOT/pull/10622`
- `github.com/RIOT-OS/RIOT/pull/10799`

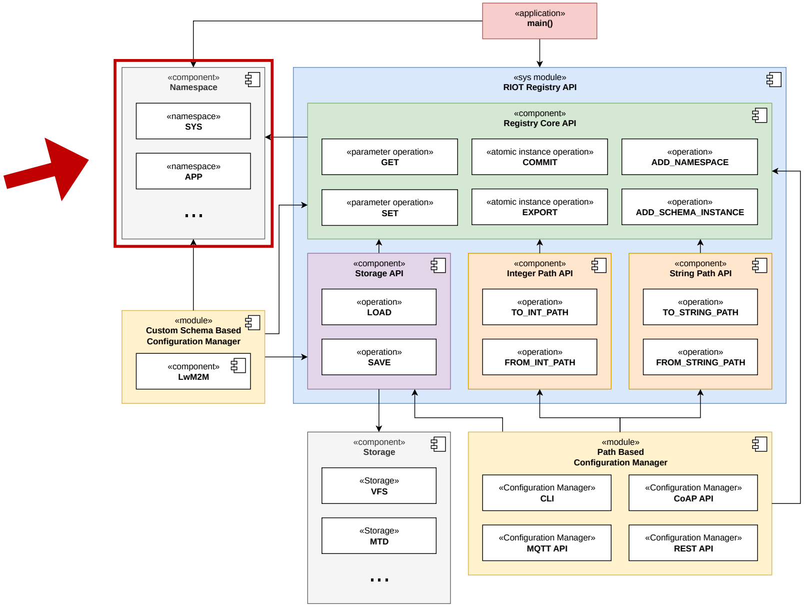
Architecture



Components


«application» main()
«component» Namespace
«namespace» SYS
«namespace» APP
...
«sys module» RIOT Registry API
«component» Registry Core API
«parameter operation» GET
«atomic instance operation» COMMIT
«operation» ADD_NAMESPACE
«parameter operation» SET
«atomic instance operation» EXPORT
«operation» ADD_SCHEMA_INSTANCE
«component» Storage API
«operation» LOAD
«operation» SAVE
«component» Integer Path API
«operation» TO_INT_PATH
«operation» FROM_INT_PATH
«component» String Path API
«operation» TO_STRING_PATH
«operation» FROM_STRING_PATH
«module» Custom Schema Based Configuration Manager
«component» LwM2M
«component» Storage
«Storage» VFS
«Storage» MTD
...
«module» Path Based Configuration Manager
«Configuration Manager» CLI
«Configuration Manager» CoAP API
«Configuration Manager» MQTT API
«Configuration Manager» REST API

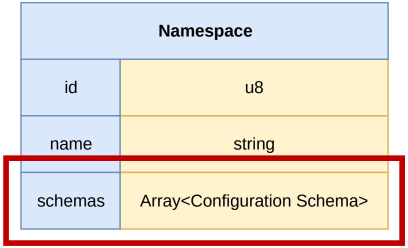
Component: Namespace



«application» main()
«component» Namespace
«namespace» SYS
«namespace» APP
...
«sys module» RIOT Registry API
«component» Registry Core API
«parameter operation» GET
«atomic instance operation» COMMIT
«operation» ADD_NAMESPACE
«parameter operation» SET
«atomic instance operation» EXPORT
«operation» ADD_SCHEMA_INSTANCE
«component» Storage API
«operation» LOAD
«operation» SAVE
«component» Integer Path API
«operation» TO_INT_PATH
«operation» FROM_INT_PATH
«component» String Path API
«operation» TO_STRING_PATH
«operation» FROM_STRING_PATH
«module» Custom Schema Based Configuration Manager
«component» LwM2M
«component» Storage
«Storage» VFS
«Storage» MTD
...
«module» Path Based Configuration Manager
«Configuration Manager» CLI
«Configuration Manager» CoAP API
«Configuration Manager» MQTT API
«Configuration Manager» REST API

Component: Namespace

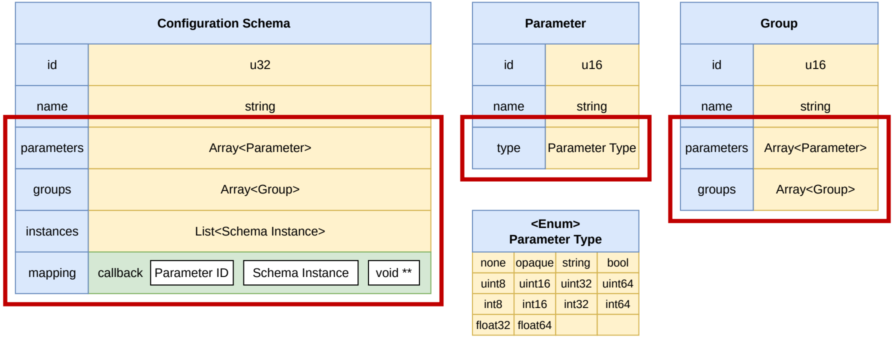
- Separates Configuration Schemas into different domains (e.g. SYS or APP)
- Prevents collisions between predefined SYS Configuration Schemas and user defined custom Configuration Schemas
- Contains Configuration Schemas



Namespace	
id	u8
name	string
schemas	Array<Configuration Schema>

Component: Configuration Schema

- Interface between the RIOT Registry and modules / apps - configurations
- Defines configurable parameters and their types
- Does not contain parameter values, but points to a list of instances



Configuration Schema	
id	u32
name	string
parameters	Array<Parameter>
groups	Array<Group>
instances	List<Schema Instance>
mapping	callback Parameter ID Schema Instance void **

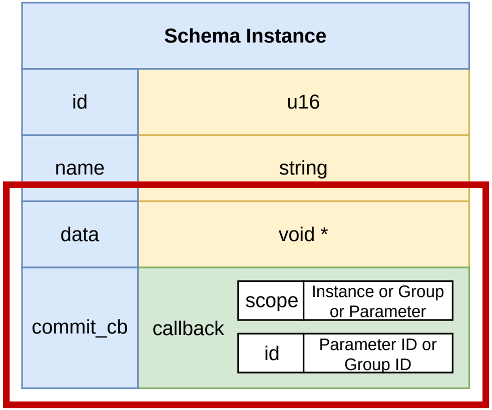
Parameter	
id	u16
name	string
type	Parameter Type

Group	
id	u16
name	string
parameters	Array<Parameter>
groups	Array<Group>

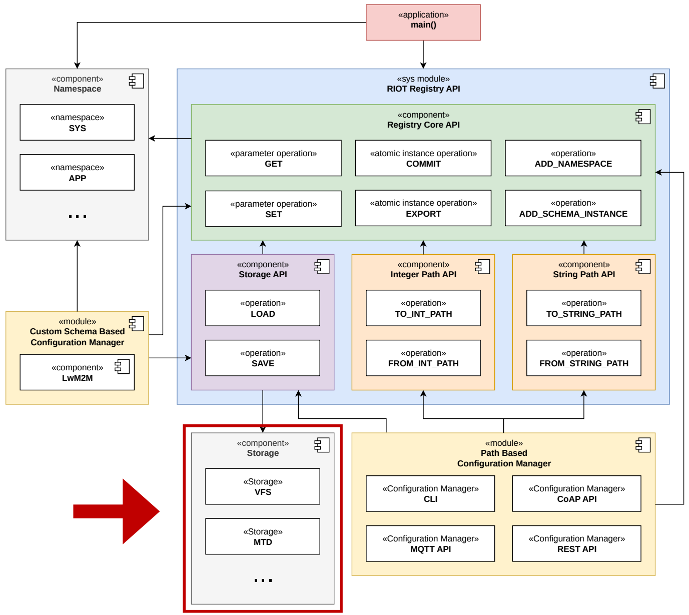
<Enum> Parameter Type			
none	opaque	string	bool
uint8	uint16	uint32	uint64
int8	int16	int32	int64
float32	float64		

Component: Schema Instance

- Contains the configuration values
- Implemented by a module or driver that needs to expose runtime configurations
- The "commit_cb" function is called by the RIOT Registry to inform that a Configuration Parameter has changed



Component: Storage


«application» main()
«component» Namespace
«namespace» SYS
«namespace» APP
...
«sys module» RIOT Registry API
«component» Registry Core API
«parameter operation» GET
«atomic instance operation» COMMIT
«operation» ADD_NAMESPACE
«parameter operation» SET
«atomic instance operation» EXPORT
«operation» ADD_SCHEMA_INSTANCE
«component» Storage API
«operation» LOAD
«operation» SAVE
«component» Integer Path API
«operation» TO_INT_PATH
«operation» FROM_INT_PATH
«component» String Path API
«operation» TO_STRING_PATH
«operation» FROM_STRING_PATH
«module» Custom Schema Based Configuration Manager
«component» LwM2M
«component» Storage
«Storage» VFS
«Storage» MTD
...
«module» Path Based Configuration Manager
«Configuration Manager» CLI
«Configuration Manager» CoAP API
«Configuration Manager» MQTT API
«Configuration Manager» REST API

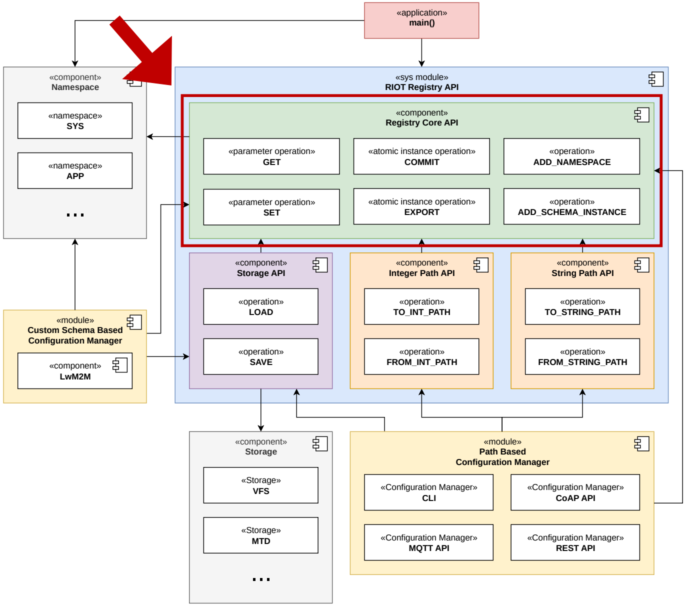
Component: Storage

- Load/save configurations from/to storage
- Data conversion to a suitable format such as CBOR or JSON
- Read from multiple Storages
- Write to only one Storage

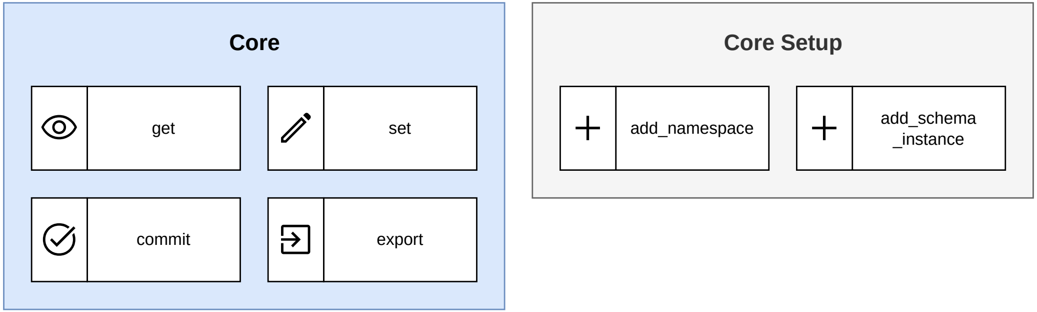
Storage Instance	
storage	Storage
data	void * (fs_mount etc.)

Storage			
load	callback	Storage Instance	function(Schema Instance, Schema Parameter, buf, buf_len)
save_start	callback	Storage Instance	
save	callback	Storage Instance Schema Parameter	Schema Instance Registry Value
save_end	callback	Storage Instance	

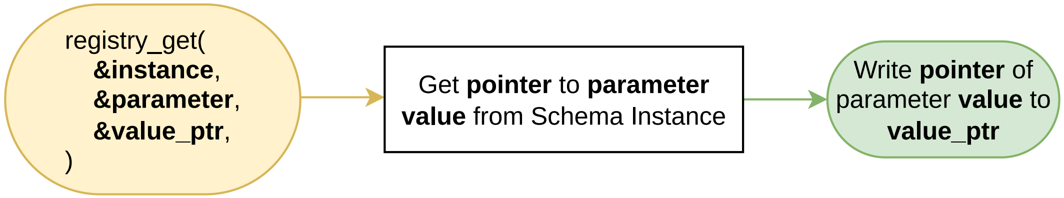
Core API


«application» main()
«component» Namespace
«namespace» SYS
«namespace» APP
...
«sys module» RIOT Registry API
«component» Registry Core API
«parameter operation» GET
«atomic instance operation» COMMIT
«operation» ADD_NAMESPACE
«parameter operation» SET
«atomic instance operation» EXPORT
«operation» ADD_SCHEMA_INSTANCE
«component» Storage API
«operation» LOAD
«operation» SAVE
«component» Integer Path API
«operation» TO_INT_PATH
«operation» FROM_INT_PATH
«component» String Path API
«operation» TO_STRING_PATH
«operation» FROM_STRING_PATH
«module» Custom Schema Based Configuration Manager
«component» LwM2M
«component» Storage
«Storage» VFS
«Storage» MTD
...
«module» Path Based Configuration Manager
«Configuration Manager» CLI
«Configuration Manager» CoAP API
«Configuration Manager» MQTT API
«Configuration Manager» REST API

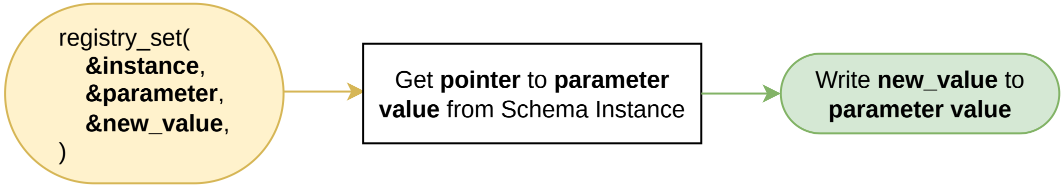
Core API


Core
get
set
commit
export
Core Setup
add_namespace
add_schema
_instance

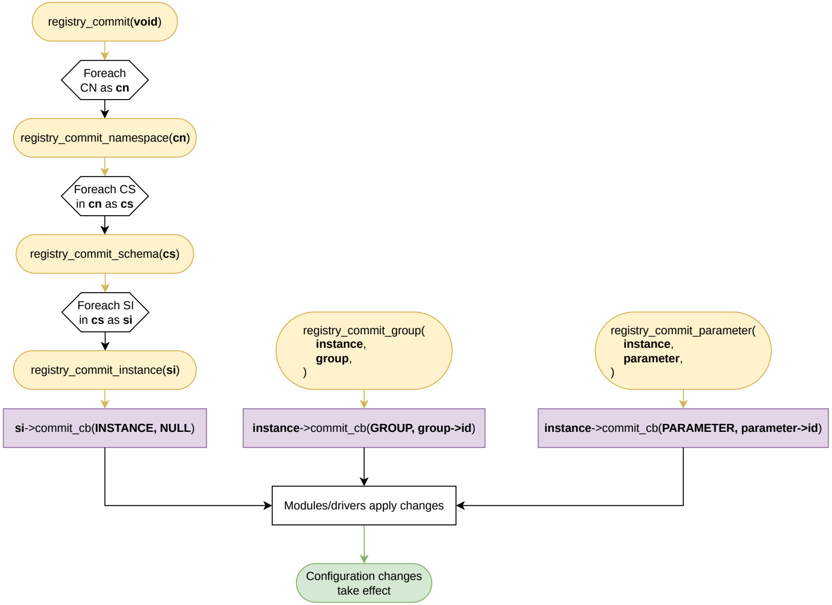
Core API: Get


registry_get(
&instance,
&parameter,
&value_ptr,
)
Get **pointer** to **parameter value** from Schema Instance
Write **pointer** of parameter **value** to **value_ptr**

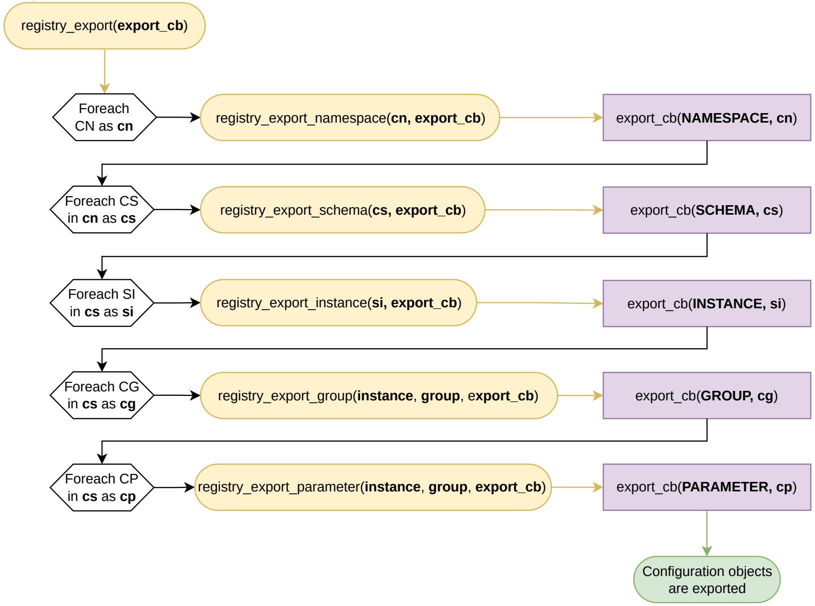
Core API: Set


registry_set(
&instance,
&parameter,
&new_value,
)
Get pointer to parameter value from Schema Instance
Write new_value to parameter value

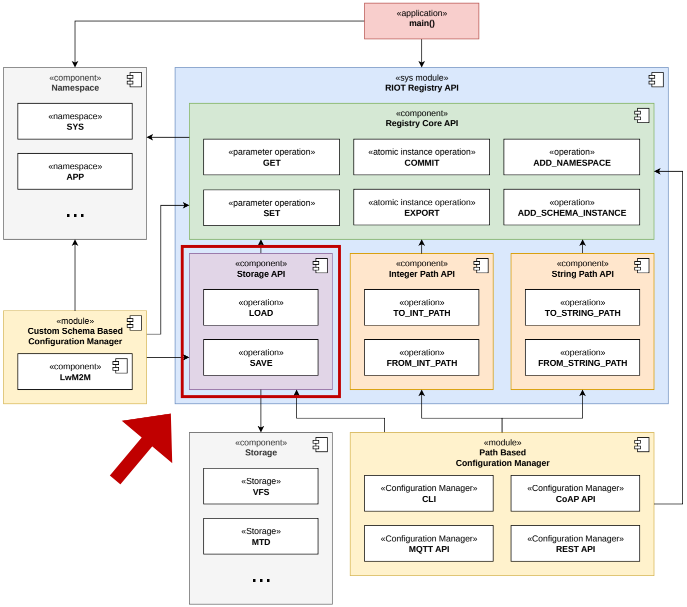
Core API: Commit


registry_commit(**void**)
Foreach CN as **cn**
registry_commit_namespace(**cn**)
Foreach CS in **cn** as **cs**
registry_commit_schema(**cs**)
Foreach SI in **cs** as **si**
registry_commit_instance(**si**)
si->commit_cb(**INSTANCE, NULL**)
registry_commit_group(**instance**, **group**,)
instance->commit_cb(**GROUP, group->id**)
registry_commit_parameter(**instance**, **parameter**,)
instance->commit_cb(**PARAMETER, parameter->id**)
Modules/drivers apply changes
Configuration changes take effect

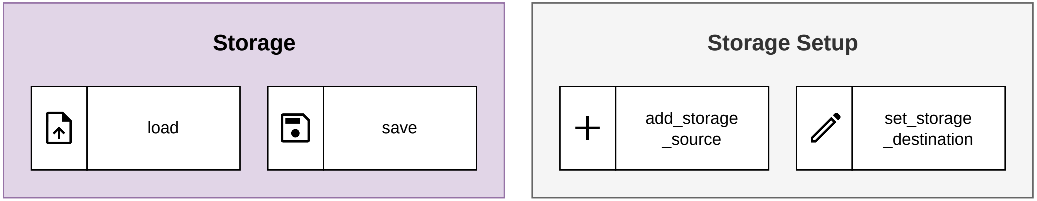
Core API: Export


registry_export(**export_cb**)
Foreach CN as **cn**
registry_export_namespace(**cn, export_cb**)
export_cb(**NAMESPACE, cn**)
Foreach CS in **cn** as **cs**
registry_export_schema(**cs, export_cb**)
export_cb(**SCHEMA, cs**)
Foreach SI in **cs** as **si**
registry_export_instance(**si, export_cb**)
export_cb(**INSTANCE, si**)
Foreach CG in **cs** as **cg**
registry_export_group(**instance**, **group**, **export_cb**)
export_cb(**GROUP, cg**)
Foreach CP in **cs** as **cp**
registry_export_parameter(**instance**, **group**, **export_cb**)
export_cb(**PARAMETER, cp**)
Configuration objects are exported

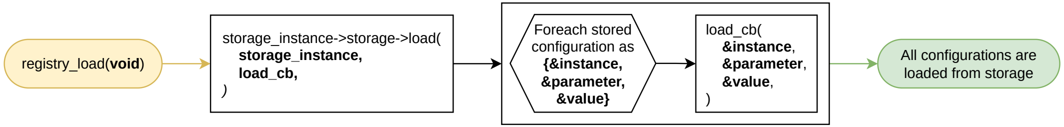
Storage API


«application» main()
«component» Namespace
«namespace» SYS
«namespace» APP
...
«sys module» RIOT Registry API
«component» Registry Core API
«parameter operation» GET
«atomic instance operation» COMMIT
«operation» ADD_NAMESPACE
«parameter operation» SET
«atomic instance operation» EXPORT
«operation» ADD_SCHEMA_INSTANCE
«component» Storage API
«operation» LOAD
«operation» SAVE
«component» Integer Path API
«operation» TO_INT_PATH
«operation» FROM_INT_PATH
«component» String Path API
«operation» TO_STRING_PATH
«operation» FROM_STRING_PATH
«module» Custom Schema Based Configuration Manager
«component» LwM2M
«component» Storage
«Storage» VFS
«Storage» MTD
...
«module» Path Based Configuration Manager
«Configuration Manager» CLI
«Configuration Manager» CoAP API
«Configuration Manager» MQTT API
«Configuration Manager» REST API

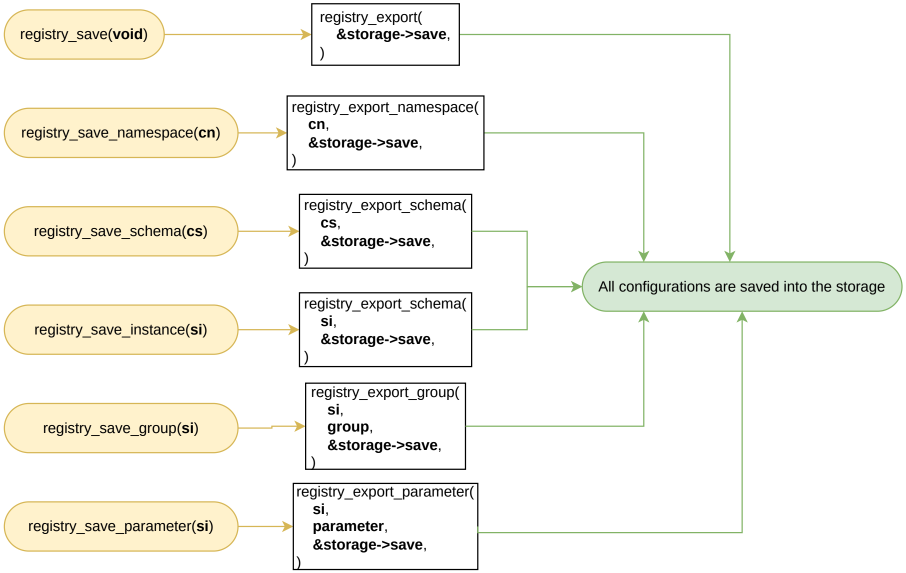
Storage API


Storage
load
save
Storage Setup
add_storage _source
set_storage _destination

Storage API: Load


registry_load(void)
storage_instance->storage->load(
storage_instance,
load_cb,
)
Foreach stored configuration as {&instance, &parameter, &value}
load_cb(
&instance,
&parameter,
&value,
)
All configurations are loaded from storage

Storage API: Save


registry_save(**void**)
registry_save_namespace(**cn**)
registry_save_schema(**cs**)
registry_save_instance(**si**)
registry_save_group(**si**)
registry_save_parameter(**si**)
registry_export(**&storage->save**,)
registry_export_namespace(**cn**, **&storage->save**,)
registry_export_schema(**cs**, **&storage->save**,)
registry_export_schema(**si**, **&storage->save**,)
registry_export_group(**si**, **group**, **&storage->save**,)
registry_export_parameter(**si**, **parameter**, **&storage->save**,)
All configurations are saved into the storage

External Configuration Manager Example: LwM2M Schema Mapping

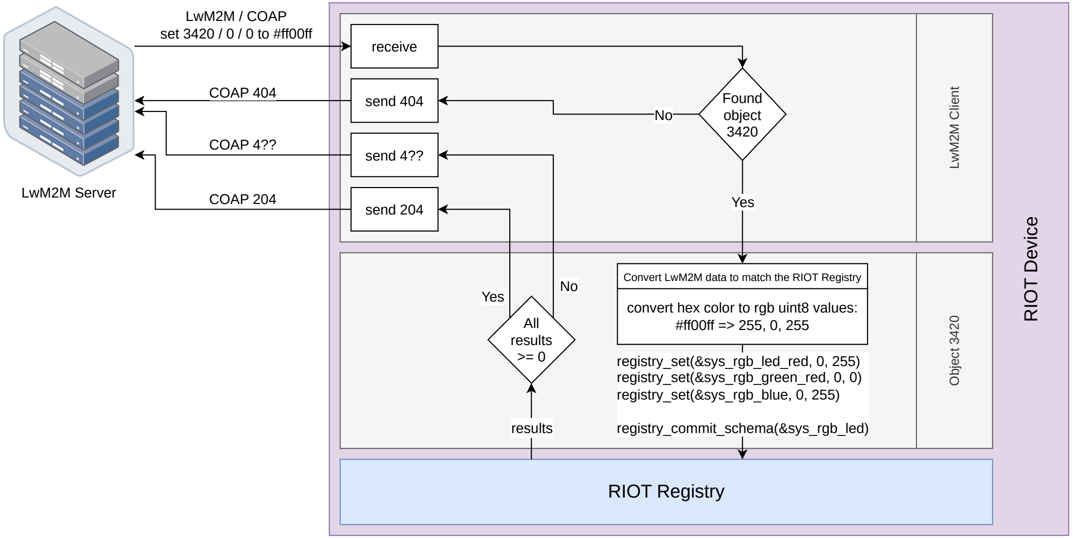

LwM2M / COAP
set 3420 / 0 / 0 to #ff00ff
LwM2M Server
COAP 404
COAP 4??
COAP 204
receive
send 404
send 4??
send 204
Found object 3420
No
Yes
LwM2M Client
Convert LwM2M data to match the RIOT Registry
convert hex color to rgb uint8 values:
#ff00ff => 255, 0, 255
registry_set(&sys_rgb_led_red, 0, 255)
registry_set(&sys_rgb_green_red, 0, 0)
registry_set(&sys_rgb_blue, 0, 255)
registry_commit_schema(&sys_rgb_led)
All results >= 0
Yes
No
results
Object 3420
RIOT Registry
RIOT Device

Table of Contents

Future Work

- Python Code Generator to Generate Namespaces / Schemas from JSON or YAML Files
- External Configuration Manager Implementation
- Specification of Sys Configuration Schemas
- Integration of the RIOT Registry into RIOT Modules and Drivers

Thank You!

github.com/RIOT-OS/RIOT/pull/19895