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.

⇒ Each application has to implement its own runtime configuration strategy.
⇒ Unnecessary and redundant implementation effort
Table of Contents

- Requirements of the Runtime Configuration Registry
- Existing Runtime Configuration Implementation
- Design of the new RIOT Registry
  - Architecture
  - Components
  - API
  - External Configuration Managers Integration Example
- Future Work
Table of Contents

- Requirements of the Runtime Configuration Registry
- Existing Runtime Configuration Implementation
- Design of the new RIOT Registry
  - Architecture
  - Components
  - API
  - External Configuration Managers Integration Example
- Future Work
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
Persistent Configurations

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

- Requirements of the Runtime Configuration Registry

- Existing Runtime Configuration Implementation

- Design of the new RIOT Registry
  - Architecture
  - Components
  - API
  - External Configuration Managers Integration Example

- Future Work
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

- Requirements of the Runtime Configuration Registry
- Existing Runtime Configuration Implementation
- Design of the new RIOT Registry
  - Architecture
  - Components
  - API
  - External Configuration Managers Integration Example
- Future Work
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](https://github.com/RIOT-OS/RIOT/pull/10622)
- [github.com/RIOT-OS/RIOT/pull/10799](https://github.com/RIOT-OS/RIOT/pull/10799)
### 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**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>u8</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
</tr>
<tr>
<td>schemas</td>
<td>Array&lt;Configuration Schema&gt;</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Configuration Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>parameters</td>
</tr>
<tr>
<td>groups</td>
</tr>
<tr>
<td>instances</td>
</tr>
<tr>
<td>mapping</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>name</td>
</tr>
<tr>
<td>parameters</td>
</tr>
<tr>
<td>groups</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;Enum&gt; Parameter Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
</tr>
<tr>
<td>uint8</td>
</tr>
<tr>
<td>int8</td>
</tr>
<tr>
<td>float32</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Schema Instance</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>u16</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
</tr>
<tr>
<td>data</td>
<td>void *</td>
</tr>
<tr>
<td>commit_cb</td>
<td>callback</td>
</tr>
</tbody>
</table>

- **id**: Parameter ID or Group ID
- **name**: String
- **data**: Pointer to void
- **commit_cb**: Callback function
  - **scope**: Instance or Group or Parameter
  - **id**: Parameter ID or Group ID
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

<table>
<thead>
<tr>
<th>Storage Instance</th>
<th>storage</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>void *(fs_mount etc.)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>storage</th>
<th>function(Schema Instance, Schema Parameter, buf, buf_len)</th>
</tr>
</thead>
<tbody>
<tr>
<td>load</td>
<td>Storage Instance</td>
</tr>
<tr>
<td>save</td>
<td>Storage Instance</td>
</tr>
<tr>
<td>save</td>
<td>Schema Instance</td>
</tr>
<tr>
<td>save</td>
<td>Schema Parameter</td>
</tr>
<tr>
<td>save</td>
<td>Registry Value</td>
</tr>
<tr>
<td>save</td>
<td>Storage Instance</td>
</tr>
</tbody>
</table>
Core API

Core
- get
- set
- commit
- export

Core Setup
- add_namespace
- add_schema
- _instance
registry_get(
    &instance,
    &parameter,
    &value_ptr,
    )

Get pointer to parameter value from Schema Instance

Write pointer of parameter value to value_ptr
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)

instance->commit_cb(GROUP, group->id)

instance->commit_cb(PARAMETER, parameter->id)

Configuration changes take effect

Modules/drivers apply changes
Core API: Export

registry_export(export_cb)

Foreach CN as cn
registry_export_namespace(cn, export_cb)

Foreach CS in cn as cs
registry_export_schema(cs, export_cb)

Foreach SI in cs as si
registry_export_instance(si, export_cb)

Foreach CG in cs as cg
registry_export_group(instance, group, export_cb)

Foreach CP in cs as cp
registry_export_parameter(instance, group, export_cb)

Configuration objects are exported
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_cbt,
)

Foreach stored configuration as
{&instance,
 &parameter,
 &value}

load_cbt(
&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

RIOT Device

LwM2M Server

LwM2M / COAP
set 3420 / 0 / 0 to #ff00ff

COAP 404

COAP 4??

COAP 204

receive

send 404

send 4??

send 204

Found object 3420

Yes

No

Convert LwM2M data to match the RIOT Registry

convert hex color to rgb uint8 values:
#ff00ff => 255, 0, 255

results

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)

RIOT Registry

LwM2M Client

Object 3420

send 4??

COAP 4??

COAP 204

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)
Table of Contents

• Requirements of the Runtime Configuration Registry

• Existing Runtime Configuration Implementation

• Design of the new RIOT Registry
  • Architecture
  • Components
  • API
  • External Configuration Managers Integration Example

• Future Work
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