RTEMS Specification and Qualification Toolchain for DDR/1

embedded brains GmbH

July 1, 2020

Overview

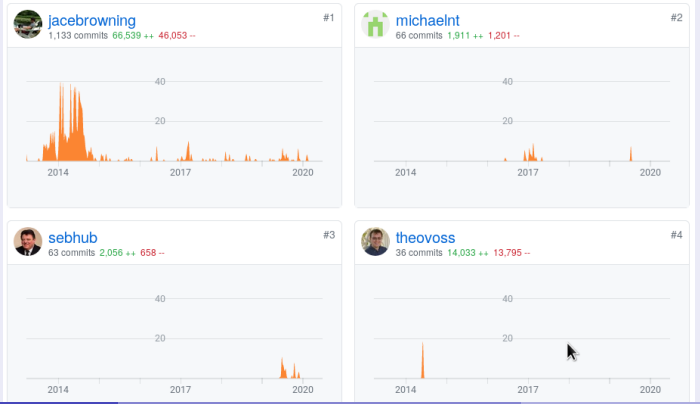
Topics of this Presentation

- Doorstop Removal
- Qualification Toolchain
- New Build System
- Glossary of Terms
- Application Configuration Options
- Interface Specification
- Requirements
- Validation
- Action Requirements
- TODO

Doorstop Removal - Well known tool

What is Doorstop?

- Doorstop was intended to be used as the requirements management tool for this activity.
- We know a bit of Doorstop. We spent some time to learn about it and improve it.
- Top four contributors to Doorstop:



Doorstop Removal - Why?

Why was Doorstop removed from the project?

- The UIDs in combination with the document hierarchy lead to duplication, e.g. a/b/c/a-b-c-d.yml. You have the path (a/b/c) also in the file name (a-b-c). You cannot have relative UIDs in links (e.g. ../parent-req) . The specification items may contain multiple requirements, e.g. min/max attributes. There is no way to identify them.
- The links cannot have custom attributes, e.g. role, enabled-by.
- The links are ordered by Doorstop alphabetically by UID.
- Its primary use case is requirements management. The standard attributes are specific to requirements.
- Inside a document (directory) items are supposed to have a common type (set of attributes).
- The verification of the items is quite limited.

Doorstop replacement

Doorstop was replaced by custom developed tools using the best ideas from Doorstop.

Qualification Toolchain - Driving Principle

Don't **R**epeat **Y**ourself

Qualification Toolchain - Ramp Up

Tasks Done

- Discuss and write Python guidelines for the RTEMS Software Engineering manual
- Project setup
  - Gitlab repositories (provided by ESA)
  - Virtual Python 3 environment
  - yapf code formatter
  - flake8, mypy, and pylint analysers
  - pytest unit tests with code coverage report (branch coverage)
  - Gitlab CI pipeline (provided by ESA)
- Basic infrastructure modules
  - Specification item class
  - Item cache
  - YAML include feature

Qualification Toolchain - Specification Item

What are specification items?

Specification items are dictionaries (a set of key-value pairs). The value types are:

- nothing (null)
- scalars: booleans, integer, floating-point numbers, strings
- lists
- dictionaries

Maps directly to Python data structures.

Example

```
minimum: 123
text: |
  Foo bar.
valid-values:
- 1
- 123
```

Qualification Toolchain - Item Cache

Reading YAML files is quite slow in Python

```
$ rm -r cache
$ time ./spec2doc.py

real    0m10.041s
user    0m9.830s
sys     0m0.049s
```

Do we have to wait 10 seconds each time we invoke some of the tools?

```
$ time ./spec2doc.py

real    0m0.408s
user    0m0.378s
sys     0m0.029s
```

No, because there is an item cache.

Qualification Toolchain - Item UIDs

Absolute UID

Defined by a leading / and the path of directories from the specification base directory to the file of the item separated by / characters and the file name without the `.yml` extension. For example, the item in file `spec/req/rtems/tasks/ident.yml` has the UID `spec:req/rtems/tasks/ident`.

Relative UID

Defined by the path of directories from the file containing the source specification item to the file of the destination item separated by / characters and the file name of the destination item without the `.yml` extension. For example, the relative UID from `spec:/if/rtems/status/code` to `spec:/if/rtems/status/header` is `spec:header`.

UID Character Set

By convention, UID parts are restricted to the characters `[a-zA-Z0-9_-]`.

Qualification Toolchain - Item Links

Link Attributes

- Link target item UID
- Link role
- Additional role-specific attributes

Link Order

The link order is defined by the list element order.

Example: rtems_status_code

```
links:
- role: interface-placement
  uid: header
- role: interface-ingroup
  uid: /groups/api/classic/status
- role: interface-enumerator
  uid: successful
- role: interface-enumerator
  uid: task-exitted
```

Qualification Toolchain - Item Types

Item Types Defined by Items

- Specification item types are defined by specification items
- These items document the specification types - `spec2doc.py`
- These items verify that the specification items are in line with the type specification - `specverify.py`
- Specification items that specify specification items are verified by specification items that specify specification items
- Types are recursively defined
- Type refinement is done by an arbitrary attribute at an arbitrary nesting level

Specification Type Items (183 Items)

https://git.rtems.org/sebh/rtems-qual.git/tree/spec/spec

Generated Specification Item Documentation

https://docs.rtems.org/branches/master/eng/req/items.html

Qualification Toolchain - Item Types Example

Example

```
SPDX-License-Identifier: CC-BY-SA-4.0 OR BSD-2-Clause
copyrights:
- Copyright (C) 2020 embedded brains GmbH (http://www.embedded-brains.de)
enabled-by: true
links:
- role: spec-member
  uid: root
- role: spec-refinement
  spec-key: type
  spec-value: requirement
  uid: root
spec-description: null
spec-example: null
spec-info:
  dict:
    attributes:
      rationale:
        description: |
          If the value is present, then it shall state the rationale or
          justification of the requirement.
        spec-type: optional-str
      references:
        description: null
        spec-type: requirement-reference-list
      requirement-type:
        description: |
          It shall be the requirement item type.
        spec-type: name
      text:
        description: |
          It shall state the requirement.
        spec-type: requirement-text
    description: |
      This set of attributes specifies a requirement.
    mandatory-attributes: all
spec-name: Requirement Item Type
spec-type: requirement
type: spec
```

Explanations

- The type is a member of the specification type hierarchy defined by the `spec:root` item.
- The type refines the `spec:root` item through the `type` attribute if the value is `requirement`.
- Items of this type are dictionaries with four mandatory attributes: `rationale`, `references`, `requirement-type`, and `text`.
- How can this type specification be simplified?
- What can be omitted?

Qualification Toolchain - Context-Sensitive Substitution

Reference Items and Values in Text

- `${uid:/key/path}`
- `${.:relative/key/path}`
- Can be used to generate artificial attributes, e.g. `${/glossary/task:/plural}`

Example

```
This can be changed later with the ${set-priority:/name} directive using the
returned semaphore identifier.
```

Sphinx Output

```
This can be changed later with the
:ref:‘rtems_semaphore_set_priority() <InterfaceRtemsSemaphoreSetPriority>‘
directive using the returned semaphore identifier.
```

Doxygen Output

```
This can be changed later with the rtems_semaphore_set_priority() directive
using the returned semaphore identifier.
```

New Build System

Pre-Qualified Build

- Removal of Doorstop simplified the build specification (user defined link order)
- 2089 build specification items
  - Source files
  - Target files
  - Flags
  - Options
  - Configuration files
  - Libraries
  - Test programs
- Support to build potentially pre-qualified `start.o`, `librtemsbsp.a` and `librtemscpu.a`
- Support to build `librtemsbspextra.a` and `librtemscpuextra.a`
- Support to create pre-qualified only workspace (not a five minute job to get to this point)
  - Used to verify a pre-qualified only build
  - Used to generate the SDD only for the pre-qualified sources (239 header files, 268 C source files, 8 assembler source files)

Glossary of Terms

Project-Wide Glossary of Terms

- Prototype use case to ramp up the qualification toolchain
- Specification items for glossary groups
- Specification items for glossary terms
- Generation of a project-wide glossary of terms for the RTEMS Classic API Guide
- Generation of a document-specific glossary of terms: only terms used in a document show up in the document glossary (how many documents in ESA activities have you seen which include a bunch of terms in each document?)

Application Configuration Options

Tame The Beast

- Every application needs to deal with `<rtems/confdefs.h>`
- Traditional `<rtems/confdefs.h>` was a nighmare, just look at the pre-processed header file in RTEMS 4.10
- Application configuration and `<rtems/confdefs.h>` was refactored and cleaned up in more than a hundred patches
- Now every application configuration option is specified in a specification item (173 items)
- Documentation for the RTEMS Classic API Guide is generated: https://docs.rtems.org/branches/master/c-user/config/index.html

Interface Specification - Interface Types

Domain - used to group interfaces at the highest level

- API
- Implementation
- C Language
- Compiler
- Build Options
- User

Interfaces

- Compound (struct, union)
- Container
- Define
- Enum
- Enumerator
- Forward declaration
- Function
- Group
- Header file
- Macro
- Typedef
- Unspecified (non-API interfaces)
- Variable

Interface Specification - Classic API

Classic API

- Complete specification of the interfaces of <rtems.h> and header files included by this header (778 items)
- Non-API interfaces are unspecified:

spec:/if/rtems/basedefs/compiler-memory-barrier

```
definition:
  default: |
    do { } while ( 0 )
  variants:
  - definition: |
      ${/if/compiler/asm:/name}
        volatile( "" ::: "memory" )
    enabled-by:
    - defined(${/if/compiler/gnuc:/name})
name: RTEMS_COMPILER_MEMORY_BARRIER
```

spec:/if/compiler/gnuc

```
interface-type: unspecified
name: __GNUC__
type: interface
```

- Header files with documentation in Doxygen markup are generated
- Documentation of directives in the RTEMS Classic API Guide is generated

Requirements

Requirement Attributes

requirement-type: functional

- action
- capability
- dependability-function
- function
- operational
- safety-function

requirement-type: non-functional

- build-configuration
- constraint
- design
- documentation
- interface
- interface-requirement
- maintainability
- performance
- portability
- quality
- reliability
- resource
- safety

text	the statement of the requirement
rationale	optional text to state the rationale or justification of the requirement
references	references to entities external to the specification, for example software architecture elements, C language functions

Validation

Validation Items

method the validation method is

- `by-analysis`
- `by-inspection`
- `by-review-of-design`

text provides the validation evidence depending on the validation method

Test Case Items

Test case items are organized in test suites. They contain several attributes which allow generation of test plans and test code.

Action Requirement Items

They contain test code aligned with the specification.

Action Requirements - Motivating Example (1)

Individual requirement items for rtems_task_ident(name, node, *id)

- If the `id` parameter is `NULL`, then the directive shall return `RTEMS_INVALID_ADDRESS`.
- If the `node` parameter is invalid, then the directive shall return `RTEMS_INVALID_NODE`.
- If no object with the specified name exists, then the directive shall return `RTEMS_INVALID_NAME`.
- ...

Potential issues

- What happens if `id` is `NULL` and no object with the specified name exists?
- How many variants have we forgotten?
- Are there duplicates?

Action Requirements - Motivating Example (2)

Obtain a Mutex

- `rtems_semaphore_obtain()`
- `rtems_mutex_lock()`
- `pthread_mutex_lock()`
- `mtx_lock()`
- `std::lock_guard<std::mutex>`
- ...

Specification Issue

All mutex obtain API variants just obtain a mutex with a shared implementation below the API layer. How can we reuse a generic mutex obtain specification and validation tests?

Action Requirements - The Specification Workhorse

What is an action?

An action is a function call or macro code execution which produces a defined set of post-condition states for a set of pre-condition states.

What needs to be specified?

- The set of pre-conditions, each with a set of states.
- Test code to prepare the pre-condition states.
- The set of post-conditions, each with a set of states.
- Test code to check the post-condition states.
- The transition map which specifies a set of post-conditions states for each variation of pre-condition states.

Benefits

- The qualification toolchain ensures that all variations of pre-condition states have exactly one set of post-conditions specified to support verification activities.
- Test code generation which covers the complete transition map.
- Test runner generation to allow test/specification building blocks.

Action Requirements - Example (1)

Status paint(void *data, Option option)

The paint() function paints some data.

Pre-Conditions (Data, Option)

Data	NullPtr	The data parameter shall be `NULL`.
	Valid	The data parameter shall reference a valid data buffer.
Option	Red	The option parameter shall be RED.
	Green	The option parameter shall be GREEN.

Post-Conditions (Status, Data)

Status	Success	The status shall be SUCCESS.
	Error	The status shall be ERROR.
Data	Unchanged	The data shall be unchanged by the action.
	Red	The data shall be red.
	Green	The data shall be green.

Action Requirements - Example (2)

Transition Map: Status paint(void *data, Option option)

Pre-Conditions		Post-Conditions	
Data	Option	Status	Data
NullPtr	Red	Error	Unchanged
NullPtr	Green	Error	Unchanged
Valid	Red	Success	Red
Valid	Green	Success	Green

TODO - Specification

Items

- Add documentation of Classic API to interface specification items
- Fine tune RTEMS sources for space profile
- Write high-level specification items (action requirements) for space profile functions
- Add requirements and validation tests
- Polish Doxygen markup for SDD
- Add specification types for hardware interfaces
- Specify hardware interfaces for low-level drivers
- Plan, design, and implement performance tests
- Plan, design, and implement resource tests

TODO - Qualification Toolchain

Items

- Revist QT specification
- Revist QT architecture
- Write a QT user manual
- Specify and document QT configuration
- Build the first Qualification Data Package
  - RTEMS tools via the RTEMS Source Builder (RSB)
  - RTEMS sources as single branch Git repositories
  - Pre-qualified build and deployment
- Incremental build of QDPs
- Add a test plan generator
- Add a test executor
- Add a test report generator with code coverage information

Questions?