Muen System Specification

Adrian-Ken Rueegsegger, Reto Buerki

v0.6, September 2, 2021

無縁

Copyright © 2021 codelabs GmbH
Copyright © 2021 secunet Security Networks AG

Further publications, reprints, duplications or recordings - no matter in which form, of the entire document or parts of it - are only permissible with the prior consent of codelabs GmbH or secunet Security Networks AG.

Contents

1 Introduction **5**

2 System Policy **6**
2.1 Policy Format 6

3 System Integration **8**

4 Tau0 Concept **11**
4.1 Static Mode of Operation 11
4.2 Dynamic 11

5 Toolchain **13**
5.1 Overview 13
5.2 Core Tools 16
5.3 Additional Tools 22

6 Policy Validation **25**
6.1 Device Domains 25
6.2 Scheduling 25
6.3 Configuration 26
6.4 Kernel 26
6.5 Memory 26
6.6 Platform 28
6.7 Model Specific Registers (MSR) 28
6.8 Files 28
6.9 Devices 29
6.10 Subjects 29
6.11 Hardware 30
6.12 Events 31

7 Policy Structure **32**
7.1 Policy Schema Documentation 32

8 Appendix **91**
8.1 Annotated Example Policy 91

9 Bibliography **125**

List of Figures

3.1 System Integration 9

5.1 Build process 14

Listings

5.1 $\tau 0$ Command Stream . . . 19
8.1 Demo System (VT-d) . . . 91

Chapter 1

Introduction

The Muen system policy is a description of a component-based system running on top of the Muen Separation Kernel (SK). It defines what hardware resources are present, how many active components (called subjects) the system is composed of, how they interact and which system resources they are allowed to access. The contents a Muen system policy is composed of are outlined in chapter 2.

A system integrator specifies and configures such a component-based system at integration time in XML format. The Muen toolchain transforms the system description in multiple steps to the final system description, resolving abstractions which exist to make life simpler and less error-prone to the integrator. Additionally, the toolchain also creates various build artifacts which are incorporated into the system image. Chapter 3 gives an overview of the system integration process.

The Muen SK can be regarded as a policy enforcement engine, in the sense that it has no knowledge about the actual content of the generated data structures and in consequence the policy. For example, it knows nothing about the contents of subject page tables which define a subject's address space, nor does it know anything about its own page tables. In fact, these structures are not even mapped into the kernel.

The most important and final step in the integration of a Muen system is the actual generation of the data structures which guarantee subject isolation and the composition of the final system image. This step is performed by a trusted system composer called (static) $\tau 0$ (Tau Zero). The concept of $\tau 0$ is introduced in chapter 4.

Section 5 explains every tool and the system image composer in detail. It also presents the usage of each tool. Section 6 then outlines all semantic checks performed on the system policy primarily by the validation tool, but also by other tools in the toolchain.

Finally section 7 specifies the XML schema and structure of the source format of the Muen system policy. Explanations and examples illustrate how to configure a component-based system with the Muen SK.

Chapter 2

System Policy

The Muen policy specifies the following properties of a system:

- Configuration values
- Hardware resources
- Platform description
- Physical memory regions
- Device domains
- Events
- Communication channels
- Components
- Subjects
- Scheduling plans

The policy serves as a static description of a Muen system. Since all aspects of the system are fixed at integration time the policy is very well suited for automated as well as manual validation prior to system execution.

The details of each property above is outlined with examples in the XSD-schema of the format source policy in section 7.

2.1 Policy Format

The system policy is specified in XML. There are currently three different policy formats:

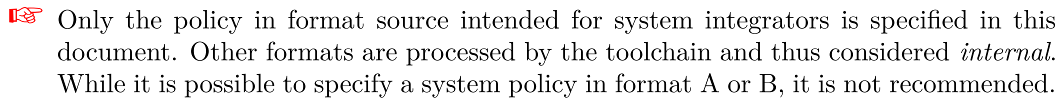
- Source Format
- Format A
- Format B

The motivation to have several policy formats is to provide abstractions and a compact way for users to specify a system in format source while simultaneously facilitate traceability as well as reduced complexity of tools operating on the policy formats A and B.

The implementation of such tools is simplified by the absence of higher-level abstractions in the latter formats which would make the extraction of input data more involved.

Furthermore, the final format B must specify every aspect of the system explicitly, e.g. all attributes have a concrete value assigned, something which would be very tedious and repetitive and that burden should not be put on an integrator.

The following sections give more detail about each policy format.



☞ Only the policy in format source intended for system integrators is specified in this document. Other formats are processed by the toolchain and thus considered *internal*. While it is possible to specify a system policy in format A or B, it is not recommended.

2.1.1 Source Format

The user-specified policy is written in the so called *source format*. Constructs such as channels provide abstractions to simplify the specification of component-based systems. Many XML elements and attributes are optional and are *expanded* during later steps of the policy compilation process.

Kernel and $\tau 0$ subject (4.2) resources are not part of the source format since they are automatically added as part of the policy expansion step.

The use of configuration values enables parametrization of the system policy.

The policy in source format is specified in detail in section 7, while appendix 8.1 provides an annotated example policy illustrating the various policy elements.

2.1.2 Format A

Format A is a processed version of the source format where all inclusions of external files are resolved and abstractions such as channels have been deconstructed into their constituent parts. For example, a channel is expanded to a physical memory region and the corresponding writer and reader subject mappings with the appropriate access rights. Optional associated events have been automatically created and correctly linked with the designated subjects.

In this format all implicit elements, such as for example automatically generated page table memory regions, are specified. The kernel and $\tau 0$ configuration is also declared as part of format A.

The only optional attributes are addresses of physical memory regions.

2.1.3 Format B

Format B is equivalent to Format A except that all physical memory regions have a fixed location (i.e. their physical address is set).

Chapter 3

System Integration

A Muen system defined via the system policy is transformed and integrated by various tools to generate a bootable system image.

The directed graph 3.1 on page 9 illustrates the process.

At the top, the graph shows how configuration and build parameters are applied to the following constituents of the system policy:

- Hardware description (static)
  Contains manually specified devices by the integrator, e.g. common hardware like I/O ports of a PC speaker. Such devices are not automatically collected by the hardware configuration generator.
- Hardware description (generated)
  Hardware description extracted from a running Linux system by the `mugenhwcfg` tool (section 5.3.3).
- Platform description
  Common names and abstractions to form a unifying view over different hardware configurations. Additionally, platform-specific configuration values can be provided here.
- System description
  Specification of an actual component-based system running on the Muen SK.

These combined inputs form the parameterized system policy in format source, which can be used by components to extract system information. Such information might be for example the log channel count of a debug server subject, or whether a specific debug facility has been enabled by the system integrator.

The CSPECs mechanism outlined in the *Muen Component Specification* document [1] can be used by components to generate source specifications (e.g. in SPARK/Ada) from the component description. Furthermore, a component might expand its own component description with information extracted from system information, or it might use the `mucbinsplit` tool (section 5.3.5) to automatically fill in the memory regions provided by its binary after compilation. The expanded component description is then merged with the system policy for further processing.

After all component descriptions have been merged into the system policy, it is expanded by the expander tool (section 5.2.3). This step transforms the system policy from format source to format A. Abstractions like directed channels are now resolved to basic shared memory mechanisms and events, non-present optional attributes are added and set to default values.

The allocator tool (5.2.4) then loops over all physical memory regions which have no address assigned and places them in memory by allocating a region and thus a physical start address from the usable pool. The usable pool information is extracted from the allocatable memory block list (7.1.12) in the system policy. This process transforms the policy to format B where all elements must be present and attributes specified.

The policy is then checked for consistency and configuration errors by the validator tool (5.2.5). If a misconfiguration is found, the user is informed and the build aborts. The extensive checks performed by the validator tool are listed in section 6. If no error is found, the system policy is then ready to be used for three subsequent steps:

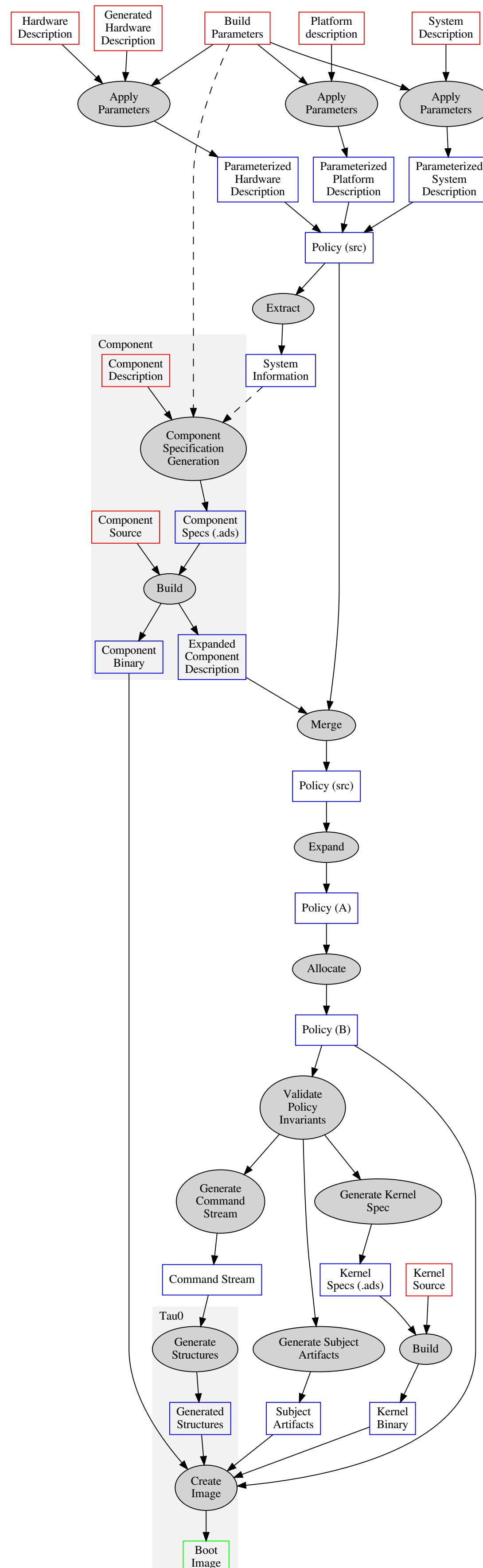


Figure 3.1: System Integration

- Generate kernel specifications (SPARK/Ada source files)
- Generate structures for subjects
- Create a command stream for $\tau 0$

The kernel source specifications contain tables and constants which represent the policy that is compiled into the kernel as part of the kernel build process and enforced at runtime.

An example of generated subject structures are ACPI tables, which are mapped into a Linux VM to announce the available hardware resources.

The command stream generator (5.2.7) generates instructions in XML format for the $\tau 0$ system composer explained in the following section.

Chapter 4

Tau0 Concept

The XML command stream together with the other build artifacts like subject structures or the kernel binary is provided as input to the trusted system composer $\tau 0$. Its task is to compose a system image while making sure that certain invariants are not violated. The $\tau 0$ concept is a mechanism to gradually increase the flexibility of a component based system while keeping a high level of assurance regarding the correctness of isolation enforcement.

There are two modes of operation for $\tau 0$:

- static
- dynamic

In the static scenario, the task of $\tau 0$ is to construct a bootable system image by assembling the input files and generating data structures such as page tables, all while checking that invariants necessary for correct isolation are valid. An example for such an invariant is that no subject memory mapping may reference a memory region containing paging structures.

For the dynamic case, the goal is to have a trusted $\tau 0$ subject with additional privileges to interact with the Muen SK over a special $\tau 0$ interface. This will allow $\tau 0$ to change certain clearly defined aspects of the system state at runtime. A potential use-case would be to set up a new subject, assign resources like memory and devices to it and then instruct the kernel to schedule it.

Since it must be guaranteed that a dynamic system is as secure as the static one, $\tau 0$ must be developed with the same care as the kernel itself, meaning it must be written in SPARK/Ada and security properties which provide hard isolation must be formally proven. This process is currently ongoing.

4.1 Static Mode of Operation

The static variant is the one which is currently implemented. $\tau 0$ runs at integration time and assembles the system image by constructing the data structures guaranteeing isolation and merging in the build artifacts of the other Muen tools, like untrusted ACPI data structures for Linux VMs.

Static $\tau 0$ fulfills its task by creating the system image in memory while processing the commands from the command stream. See listing 5.1 on page 19 for an example command stream.

$\tau 0$ is written in SPARK/Ada and it applies memory typization to formally prove aspects of the system. Command processing starts from a well-known good state and it is enforced that each system state transition resulting from a new command input results in a good state again by showing that invariants hold after the transition. If not, the command is rejected and the build aborts.

See the project README or the webpage[1] for more information about the current state of $\tau 0$.

4.2 Dynamic

While the system image is composed by the static variant of $\tau 0$, the goal is to run the same code as $\tau 0$ subject at runtime. Note that this is not yet implemented but planned as a way

[1] https://muen.sk/tau0.html

forward to achieve more dynamic systems while having the same assurance about security and safety properties.

The dynamic $\tau0$ running as subject will reconstruct the system state defined at integration time and continue to process commands starting from there. Depending on the system use case, commands might be sent to dynamic $\tau0$ by a special control subject.

The dynamic variant can be divided into multiple sub-variants, depending on how much dynamic system behavior is allowed. For example, the initial dynamic variant might only allow entity construction, not destruction.

Chapter 5

Toolchain

5.1 Overview

While the previous section 3 presented an overview of the system integration process and section 4 introduced the $\tau 0$ concept, this section focuses on the detailed description of the tools forming the Muen toolchain.

The tool-based processing of the Muen system policy can be divided into the following steps:

- Policy merging
- Components build
- Components specification merging
- Policy compilation
- Policy validation
- Structure generation
- Command stream generation for $\tau 0$
- Image generation by $\tau 0$

Following the Unix philosophy "A program should do only one thing and do it well" each of the tools only performs a specific task. They work in conjunction to process a user-defined policy and build a bootable system image. Figure 5.1 presents another illustration of the policy processing, this time laying the focus on the tools. The following sections explain each processing step while section 5.2 describes each tool separately.

5.1.1 Policy Merging

The Merger tool outlined in section 5.2.1 is responsible to merge XML files stored at different locations on the file system into one system policy in format source.

The tool reads a system configuration in XML format to locate the following files:

- System policy
- Hardware specification
- Additional hardware specification
- Platform specification

The tool also provides an implementation of the XML XInclude mechanism[1]. Using includes, the policy writer is able to separate and organize the system policy as desired. Instead of specifying the whole policy in one file, subject specifications can for example be split into separate files, or

[1] http://www.w3.org/TR/xinclude-11/

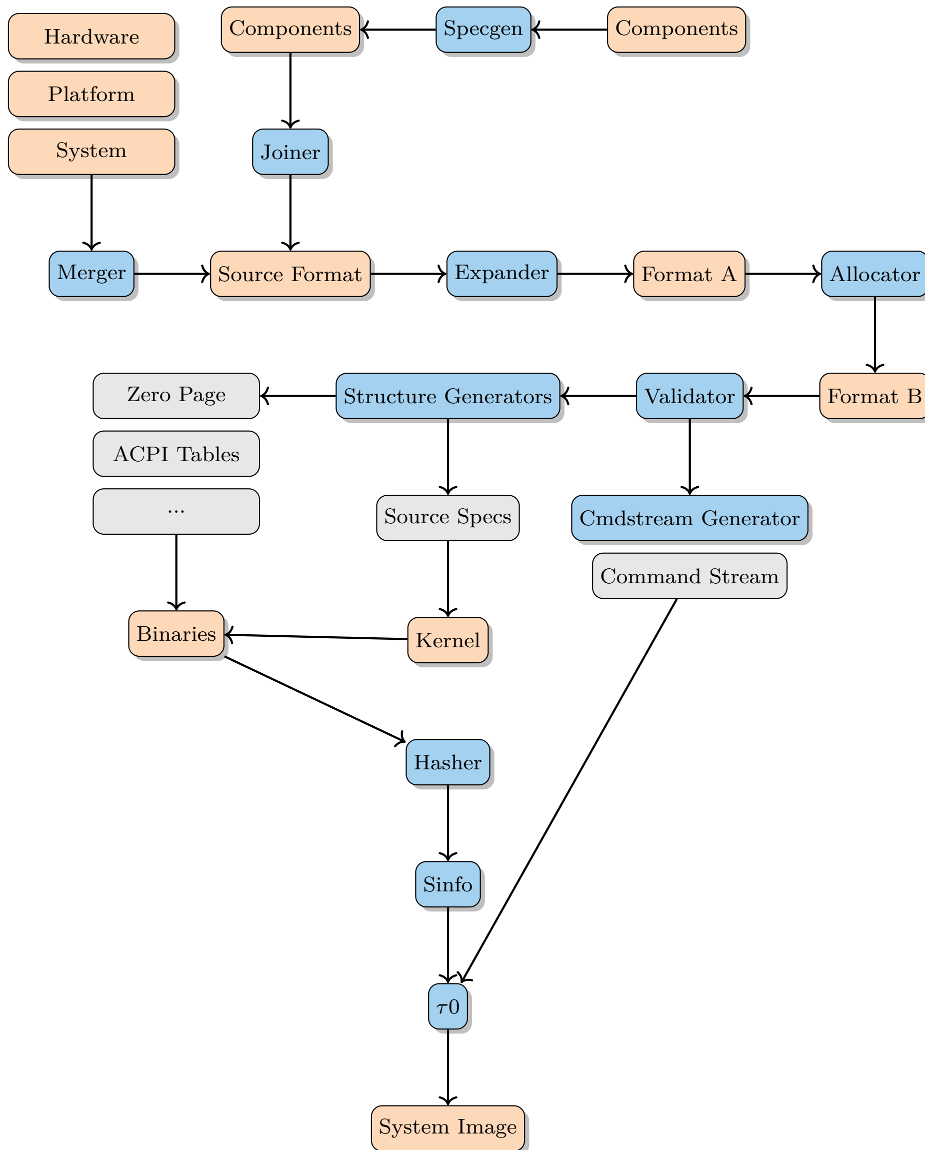


Figure 5.1: Build process

common parts shared by different system descriptions can be extracted, thus modularizing and simplifying the policy specification task.

After the merge step, the resulting policy is well formatted to minimize the difference in the generated policies resulting from the subsequent tasks. This allows the user to review (`diff`) and therefore verify the results of each policy compilation task.

Expressions can be used to formulate (nested) boolean terms using the numeric equality/inequality and logical operators. They are evaluated to boolean config values prior to processing conditionals.

The use of conditionals enables selective activation of parts of the source policy depending on the value of a given config variable. This allows flexible parameterization of a system during policy compilation by setting the value of a config variable or formulating an appropriate boolean expression.

Config variable substitution enables the policy writer to set the value of attributes to those of referenced config values. Attributes that start with a dollar sign followed by a config value name are substituted by the value of the config variable.

5.1.2 Components Build

After hardware, platform and high-level system policy are merged into a single source policy file, components may extract relevant information. For example an XSL transformation (XSLT) script could extract the I/O port of a specific device and create a corresponding configuration value based on it, which is then included in the component specification.

The `mucgenspec` tool described in 5.2.9 implements the blue *Specgen* task shown in figure 5.1. It is used to process component specifications and, similar to the policy merger, supports conditionals, expressions and configuration value substitutions. It also generates Ada/SPARK packages containing constants derived from the declared component resources and config values. These constants can be used to reliably address specific or configurable resources in the source code.

After the component specification has been processed, the component source code is compiled into a binary.

The `mucbinsplit` tool described in 5.3.5 can be used to extract ELF sections of the component binary into separate files. It automatically extends the component specification by adding a corresponding memory region with the appropriate access rights (e.g. executable, writable) for text, rodata, data, bss and stack sections.

5.1.3 Components Specification Merging

The processed component specifications are merged into the system source policy by the Muen component specification joiner tool described in section 5.2.2.

This step is optional as static component specifications which need no processing can also be manually specified in the system policy directly.

5.1.4 Policy Compilation

Policy compilation encompasses the tasks involved to transform the policy from source format to format A and finally to format B, which is the fully expanded format with no implicit properties.

The Expander tool takes care of completing the user-specified policy with additional information and resolving abstractions only available in format source to their corresponding low-level constructs.

For example, the concept of *channels* only exists in format source. Therefore a channel specified in format source must be expanded to shared memory regions with optional associated events in format A. Also, the Expander tool inserts specifications for the Muen kernel itself so the user is lifted from that burden. Generally, the aim of the expansion task is to make the life of a policy writer as easy as possible by expanding all information which can be derived automatically. Section 5.2.3 explains the Expander tool in detail.

The result of the expansion task is a policy in format A which is the input for the Allocator tool. This tool is responsible to assign physical memory addresses to all memory regions which are not already explicitly placed in memory. By querying the hardware section of the policy, the tool is aware of the total amount of available RAM on a specific system and allocates regions of it for memory elements with no explicit physical address. The Allocator tool also implements optimization strategies to keep the resulting system image as small as possible. For example, file-backed memory regions (e.g. a memory region storing a component executable) are preferably placed in lower physical regions. See section 5.2.4 for a description of the Allocator tool.

After the allocation task is complete, the policy is stored in format B. This format states all system properties explicitly and is used as input for the Validation step.

5.1.5 Policy Validation

Before structures required to pack the final system image are generated, the policy must be thoroughly validated to catch errors in the system specification. Such errors might range from overlapping memory, undefined resource references to incomplete scheduling plans etc. The Validator task performs checks that assure the policy in format B is sound and free from higher-level errors that are not covered by XML schemata restrictions.

It is important to always run the Validator as the system could otherwise exhibit unexpected behavior. This is especially true if a policy writer decides to specify the system directly in format

B which is also possible but not advised. Section 5.2.5 explains the usage of the Validator tool, while section 6 outlines all performed checks.

It should be noted that correct memory typization and all invariants enforced by $\tau0$ when constructing the system image cannot be bypassed, since the checks are inherent to the generation of the bootable image file.

5.1.6 Structure Generation

The structure generation step encompasses various tools which extract information from a policy in format B and generate files in different formats.

While some generated files are directly linked into the Muen kernel (i.e. Source Specifications, see 5.2.9), most of them are subject-related. Depending on the subjects included in the actual system policy, the following subject structures are generated:

- MSR store regions
- Sinfo regions
- Regions for Linux VMs
  - ACPI tables
  - Linux zero-page (ZP) regions
- Regions for MirageOS/Solo5[2] unikernels
  - Solo5 boot info

As these structures do not affect isolation between subjects or subjects and the kernel, they are not generated by $\tau0$ but only included as binary data via XML command stream and build artifacts.

The structure generator tools are explained in section 5.2.9.

5.1.7 Image Creation

The system image composer assembles the final system image. This task is performed by $\tau0$ static introduced in the previous section 4.1. The usage of it is specified in 5.2.8.

5.2 Core Tools

This section describes the tools which form the core of the Muen toolchain.

5.2.1 Policy Merger

The merger tool `mucfgmerge` combines user-provided system policy files into a single XML document.

Name
: `mucfgmerge`

Input
: System configuration as XML, Colon-separated list of include paths

Output
: System policy in format source (merged)

This tool reads the system configuration and merges the specified system policy, hardware and platform files into a single file. It evaluates boolean expressions, resolves conditional parts of the policy and substitutes attribute configuration value references. Included files are inserted at the corresponding locations in the merged file. The XML content is re-formatted so changes to the policy by subsequent build steps can be manually reviewed or visualized by diffing the files.

It also merges the platform configuration section (if any) into the global configuration section, removing the platform configuration section in the process.

[2] https://github.com/Solo5/solo5

5.2.2 Component Specification Joiner

The Muen component specification joiner adds component XML specifications to the component section of a specified system policy and writes the result to a designated output file. Each given component/library specification is loaded and validated against the component specification XML schema. If it is correct the content is added to the components section of the system policy specified as input file. If the given system policy does not yet contain a components section, it is created. The result is written to the file specified by the `-o` parameter. In-place processing is supported by passing in the same value for input and output file.

Name
: `mucfgcjoin`

Input
: System policy in format source, comma-separated list of component specs

Output
: System policy in format source (joined)

5.2.3 Expander

The expander completes the user-provided system policy by creating or deriving additional configuration elements.

Name
: `mucfgexpand`

Input
: System policy in format source

Output
: System policy in format A (expanded)

The Expander performs the following actions:

- Pre-check the system policy to make sure it is sound
- Expand channels
- Expand device resources
- Expand device isolation domains
- Expand kernel sections
- Expand minimal $\tau 0$ subject
- Expand additional memory regions
- Expand hardware-/platform-related information
- Expand additional subject information
- Expand profile-specific information
- Expand scheduling information
- Post-check resulting policy

5.2.4 Allocator

The Allocator is responsible to assign a physical address to all global memory regions.

Name
: `mucfgalloc`

Input
: System policy in format A

Output
: System policy in format B (allocated)

First, the Allocator initializes the physical memory view of the system based on the physical memory blocks specified in the XML hardware section. It then reserves memory that is occupied by pre-allocated memory elements (i.e. memory regions with a physical address or device memory). Finally it places all remaining memory regions in physical memory. In order to reduce the size of the final system image file-backed memory regions are placed at the start of memory.

5.2.5 Validator

The Validator performs additional checks that go beyond the basic restrictions imposed by the XML schema validation. For example it checks that the hardware provides an IOMMU device and that all references to subjects are resolvable. See 6 for a complete list of all executed checks. The tool aborts with a non-zero exit status and an explanatory message to the user if checks fail.

Name
: `mucfgvalidate`

Input
: System policy in format B

Output
: None, raises exception on error

5.2.6 Hasher

The `mucfgmemhashes` tool is used to add memory integrity hashes to a given policy.

Name
: `mucfgmemhashes`

Input
: System policy in format B

Output
: System policy in format B with memory integrity hashes

The tool appends a hash to all memory regions with fill and file content. It must run after all files have been generated by the structure generator tools.

The actual hash is generated using the SHA-256 algorithm and is intended to be used to verify the integrity of memory regions during runtime.

Note that no hashes are generated for sinfo memory regions. Since the hash information is exported via sinfo, and the sinfo region is itself part of the memory information of a subject, this hash would be self-referential.

The tool also replaces all occurrences of `hashRef` elements. A hash reference element instructs the tool to copy the hash element of the referenced memory region after message digest generation.

From an abstract point of view, the `hashRef` element is a way to link multiple memory regions by declaring that the hash of the content is the same. The hash may serve as an indicator on how to reconstruct the (initial) content of a memory region. This mechanism is used by e.g. the subject loader (SL) during subject init and reset operation. The subject loader expansion step remaps writable memory regions of the loadee (the subject under loader control) to SL and replaces the original regions with new ones containing a hash reference to the associated physical memory region. This way SL is able to determine the intended content of the target memory region by looking up the region in its sinfo page using the hash value as key.

5.2.7 Tau0 Command Stream Generator

The `mugentau0cmds` tool creates an XML command stream for $\tau 0$ to let it compose the system image specified by the system policy given as input.

The tool reads the policy in format B and translates it to a sequence of commands as shown in the following listing:

```xml
<tau0>
 <commands>
  <addMemoryBlock address="16#0000#" size="157"/>
  <addMemoryBlock address="16#0010_0000#" size="130816"/>
  <addMemoryBlock address="16#2020_0000#" size="130564"/>
  <addMemoryBlock address="16#4000_5000#" size="453932"/>
  <addMemoryBlock address="16#bae9_f000#" size="256"/>
  <addMemoryBlock address="16#baf9_f000#" size="96"/>
  <addMemoryBlock address="16#0001_0000_0000#" size="3401216"/>
  <addMemoryBlock address="16#ba3b_a000#" size="23"/>
  <addMemoryBlock address="16#bb80_0000#" size="16896"/>
  <addProcessor id="0" apicId="0"/>
  <addProcessor id="1" apicId="2"/>
  <addIoapic sid="16#f0f8#"/>
  <createLegacyDevice device="0"/>
  <addIOPortRangeDevice device="0" from="16#03c0#" to="16#03df#"/>
  <addMemoryDevice device="0" caching="WC" address="16#000a_0000#" size="32"/>
  <activateDevice device="0"/>
  <createLegacyDevice device="1"/>
  <addIOPortRangeDevice device="1" from="16#0060#" to="16#0060#"/>
  <addIOPortRangeDevice device="1" from="16#0064#" to="16#0064#"/>
  <addIRQDevice device="1" irq="1"/>
  <addIRQDevice device="1" irq="12"/>
  <activateDevice device="1"/>
  <createLegacyDevice device="2"/>
  <addIOPortRangeDevice device="2" from="16#0070#" to="16#0071#"/>
  <activateDevice device="2"/>
  <createLegacyDevice device="3"/>
  <addIOPortRangeDevice device="3" from="16#0cf9#" to="16#0cf9#"/>
  <addIOPortRangeDevice device="3" from="16#0404#" to="16#0404#"/>
  <activateDevice device="3"/>
  <createLegacyDevice device="4"/>
  <addMemoryDevice device="4" caching="UC" address="16#fec0_0000#" size="1"/>
  <activateDevice device="4"/>
```

Listing 5.1: $\tau 0$ Command Stream

As $\tau 0$ strictly enforces certain invariants, the system must be constructed in a way not to violate these invariants. For example, before memory can be typed as being a VT-d root table, this memory must be cleared. Otherwise the memory typing model of $\tau 0$ is violated.

The `mugentau0cmds` tool must take this into consideration when iterating over the resources specified in the input system policy and generating commands which instruct $\tau 0$ to create the specified system.

Name
: `mugentau0cmds`

Input
: System policy in format B

Output
: XML command stream for $\tau 0$ static

5.2.8 Tau0 Static

The $\tau 0$ static component serves as an image composer during integration. The concept and motivation of this approach is described in chapter 4.

Name
: `tau0_main`

Input
: XML command stream, input directory containing build artifacts

Output
: Muen system image

Output format
: Command Stream Loader (CSL) image[3], bootable by any compliant bootloader.

If a command is received which violates a constraint enforced by $\tau 0$ static, the tool aborts system image construction, displays an error message and returns with a non-zero exit status.

[3] https://www.codelabs.ch/download/bsbsc-spec.pdf

5.2.9 Structure Generators

These tools do not change the policy and use it read-only.

MSR Stores Generator

Generate MSR store for each subject with MSR access.

Name
: `mugenmsrstore`

Input
: System policy in format B

Output
: MSR store files of subjects in binary format

Output format
: Intel SDM Vol. 3C, "24.8.2 VM-Entry Controls for MSRs" and Intel SDM Vol. 3C, "24.7.2 VM-Exit Controls for MSRs".

The tool generates MSR stores for each subject. The MSR store is used to save/load MSR values of registers not implicitly handled by hardware on subject exit/resumption.

MSR stores are used by hardware (VT-x) to enforce isolation of MSR (i.e. subjects that have access to the same MSRs cannot transfer data via these registers).

ACPI Tables

Generate ACPI tables for all Linux subjects.

Name
: `mugenacpi`

Input
: System policy in format B

Output
: ACPI tables of all Linux subjects

Output format
: Advanced Configuration and Power Interface (ACPI) Specification[4]

ACPI tables are used to announce available hardware to VM subjects. A set of tables consists of an RSDP, XSDT, FADT and DSDT table. See the ACPI specification for more information about a specific table.

Linux Zero Pages

Generate Zero Pages for all Linux subjects.

Name
: `mugenzp`

Input
: System policy in format B

Output
: Zero pages of all Linux subjects

Output format
: Linux Boot Protocol[5]
: Zero Page[6]

The so-called Zero Page (ZP) exports information required by the boot protocol of the Linux kernel on the x86 architecture. The kernel uses the provided information to retrieve settings about its runtime environment:

[4] http://www.acpi.info/DOWNLOADS/ACPIspec50.pdf
[5] https://www.kernel.org/doc/Documentation/x86/boot.txt
[6] https://www.kernel.org/doc/Documentation/x86/zero-page.txt

- Type of bootloader
- Map of physical memory (e820 map)
- Address and size of initial ramdisk(s)
- Kernel command line parameters

Solo5 Boot Info

Generate Solo5 boot info structures for MirageOS unikernels[7] running on the Solo5 platform.

Name
: `mugensolo5`

Input
: System policy in format B

Output
: Solo5 boot info for all MirageOS subjects

Output format
: struct hvt_boot_info[8]

The boot info structure exports information required by Solo5. The unikernel uses the provided information to retrieve settings about its runtime environment:

- Memory size in bytes
- Address of end of unikernel
- CPU cycle counter frequency, Hz
- Address of command line (C string)
- Address of application manifest

Kernel Source Specifications

Generate source specifications used by kernel.

Name
: `mugenspec`

Input
: System policy in format B

Output
: Source specifications in SPARK, C and GPR format

Gathers data from the system policy to generate various source files in SPARK, C and GNAT project file (GPR) format. Created output includes constant values for memory addresses, device resources, scheduling plans, etc. See the description of the `Skp` package hierarchy in the Muen Kernel Specification document [2] for the exact information these packages provide.

Component Source Specifications

Process component description and generate source specifications from it. Write processed description to specified output file.

Name
: `mucgenspec`

Input
: Component description in XML, colon-separated list of include paths

[7] https://mirage.io
[8] https://github.com/Solo5/solo5/blob/master/include/solo5/hvt_abi.h

Output
: Component source specifications in SPARK, processed component description in XML

The component spec generation tool processes the given component description by evaluating XIncludes, boolean expressions and resolving conditional parts. Furthermore, it performs substitutions of attributes with configuration values.

It also generates Ada/SPARK packages containing constants of the declared logical component resources. The generated specifications can be used in the component source code to access the declared resources.

The resulting processed component description is written to the given output location.

Subject Info (sinfo)

Generate subject information data for each subject.

Name
: `mugensinfo`

Input
: System policy in format B

Output
: Subject info data in binary format

Output format
: As specified in [1] and `common/musinfo/musinfo.ads`

The Sinfo page is used to export subject information data extracted from the system policy to subjects. Currently, information about available memory regions, communication channels, events, vectors and assigned PCI devices is provided.

5.3 Additional Tools

This section lists additional helper tools which simplify the process of generating and validating a Muen system.

5.3.1 Kernel ELF Checker

The `mucheckelf` tool enforces that the format of a given Muen kernel ELF binary matches the kernel memory layout specified in a system policy. Furthermore, the ELF kernel entry point is compared to the expected value.

Size, VMA (Virtual Memory Address) and permissions of binary ELF sections are validated against kernel memory regions defined in the policy. The following table lists the correspondence of ELF section names to logical kernel memory region names.

ELF Section	Memory Name
.text	kernel_text
.data	kernel_data
.rodata	kernel_ro
.bss	kernel_bss
.globaldata	kernel_global$_{d}ata$

5.3.2 Stack Usage Checker

The `mucheckstack` tool statically calculates the worst-case stack usage of a native Ada/SPARK component or the Muen kernel compiled with the -fcallgraph-info switch[9].

The tool takes a GNAT project file and a stack limit in bytes as input. All control-flow information (.ci) files found in the object directory of the main project and all of its dependencies are parsed. Once the control-flow graph is constructed the maximum stack usage of each subprogram

[9] https://www.adacore.com/uploads/technical-papers/Stack_Analysis.pdf

is calculated and checked against the user-specified limit. The tool exits with a failure if a stack usage exceeding the limit is detected.

Note that the tool is not applicable to arbitrary software projects as it does not handle dynamic/unbounded stack usage and recursion. In the context of the Muen project these cases can not occur since they are prohibited by the following restriction pragmas:

- No_Recursion
- No_Secondary_Stack
- No_Implicit_Dynamic_Code

Additionally, the `-Wstack-usage` compiler switch warns about potential unbounded stack usage.

5.3.3 Hardware Config Generator

The `mugenhwcfg`[10] tool has been created to automate the process of gathering all necessary hardware information. To collect data for a new target hardware all that is required is to run the tool on a common Linux distribution[11]. See the project README for more information.

Name
`mugenhwcfg`

Input
None

Output
Hardware description in `output.xml`

The tool is implemented in a way to extract as much information from the system and generate a hardware configuration even if problems are encountered. The aim is to assist the integrator as much as possible in writing a hardware configuration for the target hardware.

Therefore, the tool only fails with a non-zero exit status and no output if essential required data can not be extracted from the system. Other problems are reported in the potentially incomplete `output.xml` file as XML comments, making the encountered problems on the actual machine evident. The following snippet provides an example of such a warning comment in the header of the generated `output.xml` file:

```
* WARNING *: Unable to resolve device class 0c80. Please update pci.ids
(-u) and try again
```

The comments should be rather self-explanatory. In this case, the problem is only a minor issue since the tool was simply unable to resolve a device class number to a human-readable string.

The next example has more consequences:

```
* WARNING *: Skipping invalid IRQ resource for device 0000:00:1f.3: None
```

This has the effect that no IRQ resource is appended in the specification of the device exhibiting this problem. While the device can still be assigned to a subject, it is missing the IRQ element and as a result the IRQ resource itself. It can be assumed that this leads to problems with the driver interacting with the device. For proper operation, it is the policy writer's task to rectify the hardware specification by determining the correct configuration manually.

[10] https://git.codelabs.ch/?p=muen/mugenhwcfg.git
[11] https://github.com/roburio/mugenhwcfg-live

5.3.4 Scheduling Plan Generator

The `mugenschedcfg`[12] tool generates scheduling plans for Muen based on a given scheduling configuration. The configuration allows the user to specify the following scheduling properties:

- Number of CPU cores
- The tick rate of the CPUs
- Security constraints to meet
  - Same CPU domains
  - Simultaneous execution domains
- Subject specifications
- Score functions
- Number of plans to generate
- Plans
  - Weighting of plan importance
  - Levels
  - Subjects of a plan
  - Chains with throughput metric

Consult the project's README and example plans on how to use the tool.

5.3.5 Component Binary Splitter

The `mucbinsplit` tool splits component binaries into multiple files, one per ELF section.

Name
`mucbinsplit`

Input
Component description in XML, Component ELF binary

Output
Binary files corresponding to ELF sections, processed component description in XML

The component binary splitter tool processes component binaries and creates a separate file for each ELF section. The component XML description is extended by adding a file-backed memory region for each ELF section with the appropriate virtual mapping address, size and access rights. The RIP value is set to the ELF entry point of the component binary.

The resulting processed component description is written to the given output location while the binary section files are written to the specified output path.

[12] https://git.codelabs.ch/?p=muen/mugenschedcfg.git

Chapter 6

Policy Validation

Prior to operate on the policy, any tool outlined in the toolchain section 5.2 checks all required preconditions by running *validator* procedures. For example a tool accessing physical devices via subject logical device references will execute a validator checking such references for validity.

Before the policy is used to generate system structures like sinfo regions, or the command stream for $\tau 0$, the expanded policy in format B is validated by executing a comprehensive set of checks. This is done by the `mucfgvalidate` tool outlined in section 5.2.5.

The following sections list the various checks executed by `mucfgvalidate` and the other Muen build tools in the toolchain.

6.1 Device Domains

The following checks are performed to guarantee that IOMMU device domains are correctly configured in the system policy.

- Validate that domain device references are unique.
- Validate that no virtual memory regions of a domain overlap.
- Validate that domain memory references are unique.
- Validate that domain memory referenced by subjects is mapped at the same virtual address.
- Validate memory type of physical memory referenced by domains.
- Validate that each device referenced by a device domain is a PCI device.
- Validate that each device domain has a physical PT memory region.
- Validate that each PCI bus has a physical VT-d context memory region.

6.2 Scheduling

The following checks are performed to verify the correctness of the scheduling configuration in the system policy.

- Validate that each major frame specifies the same number of CPUs.
- Validate subject references.
- Validate that subjects are scheduled on the correct logical CPU.
- Validate tick counts in major frame.
- Validate that barrier IDs do not exceed barrier count and are unique.
- Validate that barrier sizes do not exceed the number of logical CPUs.
- Validate that the barrier sizes and count of a major frame corresponds to the minor frame synchronization points.
- Validate that minor frame barrier references are valid.

6.3 Configuration

The following checks are performed to guarantee correctness of configuration options in the system policy.

- Validate config variable name uniqueness.
- Check that all booleans defined in config contain a valid value.
- Check that all integers defined in config contain a valid value.
- Check that all expression config variable references are valid.
- Check that all integers defined in expressions contain a valid value.
- Check that all booleans defined in expressions contain a valid value.
- Check that all conditional config variable references are valid.

6.4 Kernel

The following kernel-specific checks are performed on the policy.

- Validate that all CPU-local data section virtual addresses are equal.
- Validate that all CPU-local BSS section virtual addresses are equal.
- Validate that all global data section virtual addresses are equal and that the expected number of mappings exists.
- Validate that all stack virtual addresses are equal.
- Validate that all crash audit mappings exist and that their virtual addresses are equal.
- Validate that every kernel has a stack and interrupt stack region mapped and both regions are guarded by unmapped pages below and above.
- Validate that all IOMMU memory-mapped IO regions are consecutive.
- Validate that each active CPU has a memory section.
- Validate that no virtual memory regions of the kernel overlap.
- Validate that the system board is referenced in the kernel logical devices section and that it provides a logical reset port.
- Validate that the debug console device and its resources matches the kernel diagnostics device specified in the platform section.

6.5 Memory

The following checks are performed to verify that the memory is correctly configured in the system policy.

- Validate that a VMXON region exists for every specified kernel.
- Validate size of VMXON regions.
- Validate that VMXON regions are in low-mem.
- Validate that all VMXON regions are consecutive.
- Validate that a VMCS region exists for each declared subject.
- Validate size of VMCS regions.

- Validate that physical memory region names are unique.
- Validate that physical memory referenced by logical memory exists.
- Validate that all physical memory addresses are page aligned.
- Validate that all virtual memory addresses are page aligned.
- Validate that all memory region sizes are multiples of page size.
- Validate kernel or subject entities encoded in physical memory names (e.g. linux|zp or kernel_0|vmxon).
- Validate that no physical memory regions overlap.
- Validate that an uncached crash audit region is present.
- Validate that crash audit region is located after system image.
- Validate that a kernel data region exists for every CPU.
- Validate that a kernel BSS region exists for every CPU.
- Validate that a kernel stack region exists for every CPU.
- Validate that a kernel interrupt stack region exists for every CPU.
- Validate that a kernel PT region exists for every CPU.
- Validate that kernel PT regions are in the first 4G.
- Validate that scheduling group info regions are mapped by the kernel running subjects of that scheduling group. Also verify that the kernel mapping is at the expected virtual location.
- Validate that a subject state memory region with the expected size exists for every subject.
- Validate that a subject interrupts memory region with the expected size exists for every subject.
- Validate that memory of type kernel is only mapped by kernel or Tau0.
- Validate that memory of type system is not mapped by any entity.
- Validate that memory of type 'device' (e.g. device_rmrr) is only mapped by device domains.
- Validate that subject state memory regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that subject interrupts memory regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that subject MSR store memory regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that subject timed event memory regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that subject VMCS regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that subject FPU state regions are mapped by the kernel running that subject. Also verify that the kernel mapping is at the expected virtual location.
- Validate that a subject FPU state memory region with the expected size exists for every subject.
- Validate that a subject timed event memory region with the expected size exists for every subject.

- Validate that a subject I/O Bitmap region with the expected size exists for every subject.
- Validate that a subject MSR Bitmap region with the expected size exists for every subject.
- Validate that a subject MSR store memory region exists for each subject that accesses MSR registers not managed by VMCS.
- Validate that a subject pagetable memory region exists for each subject.
- Validate that a scheduling group info memory region exists for each scheduling group.
- Validate that subjects map the scheduling group info region of the scheduling group they belong to.
- Validate size of VT-d root table region.
- Validate size of VT-d context table region.
- Validate that a VT-d root table region exists if domains are present.
- Validate that a VT-d interrupt remapping table region exists.

6.6 Platform

The following checks are performed to verify the correctness of the platform configuration in the system policy.

- Validate that physical devices referenced by device aliases exist.
- Validate that physical device resources referenced by device aliases exist.
- Validate that physical devices referenced by device classes exist.
- Validate that subject devices that reference an alias only contain resources provided by the device alias.
- Validate that the physical device and resources referenced by the kernel diagnostics device exists.
- Validate that the kernel diagnostics device resources match the requirements of the specified diagnostics type.

6.7 Model Specific Registers (MSR)

The following checks are performed to verify Model Specific Register (MSR) specifications in the system policy.

- Validate that all MSR start addresses are smaller than end addresses.
- Validate that subject MSRs are in the allowed list:
  - IA32_SYSENTER_CS/ESP/EIP
  - IA32_DEBUGCTL
  - IA32_EFER/STAR/LSTAR/CSTAR/FMASK
  - IA32_FS_BASE/GS_BASE/KERNEL_GS_BASE

6.8 Files

The following file-specific checks are performed.

- Check existence of files referenced in XML policy.
- Check if files fit into corresponding memory region.

6.9 Devices

The following checks are performed to guarantee that hardware devices are correctly configured in the system policy.

- Validate that devices referenced by logical devices exist.
- Validate that device names (including device aliases/classes) are unique.
- Validate that all physical IRQs are unique.
- Validate that physical device IRQs referenced by logical IRQs exist.
- Validate that ISA IRQs fulfill their constraints.
- Validate that PCI LSI IRQs fulfill their constraints.
- Validate that PCI MSI IRQs fulfill their constraints.
- Validate that PCI MSI IRQs are consecutive.
- Validate that physical IRQ names are unique per device.
- Validate that all I/O start ports are smaller than end ports.
- Validate that physical I/O ports referenced by logical I/O ports exist.
- Validate that all physical I/O ports are unique.
- Validate that physical I/O port names are unique per device.
- Validate that device memory names are unique per device.
- Validate that device memory referenced by logical device memory exists.
- Validate that PCI device bus, device, function triplets are unique.
- Validate that logical device references of each subject do not refer to the same physical device.
- Validate that PCI device reference bus, device, function triplets are unique per subject.
- Validate that all device references specifying a bus, device, function triplet are references to physical PCI devices.
- Validate that all device references to PCI multi-function devices belong to the same subject and have the same logical device number.
- Validate that all device references not specifying a bus, device, function triplet are references to physical legacy (non-PCI) devices.
- Validate that all logical PCI devices specify bus number zero.
- Validate that all IOMMU memory-mapped IO regions have a size of 4K.

6.10 Subjects

The following checks are performed to verify the correctness of the subject configuration in the system policy.

- Validate subject name uniqueness.
- Validate subject CPU ID.
- Validate uniqueness of global subject IDs.
- Validate per-CPU uniqueness of local subject IDs.
- Validate memory types of memory mappings (ie. allow access by subjects).

- Validate that no subject references an IOMMU device.
- Validate that all subjects are runnable, i.e. directly referenced in the scheduling plan or target of a switch event of a subject that is itself scheduled.
- Validate that subject scheduling group IDs match values as determined by the scheduling plan and handover events.
- Validate that logical names of subject devices are unique.
- Validate that IRQ vector numbers of PCI device references with MSI enabled are consecutive.
- Validate that logical names of subject unmask IRQ events conform to the naming scheme (`unmask_irq_$IRQNR`) and that the unmask number matches the physical IRQ.
- Validate that no virtual memory regions of a subject overlap.
- Validate that multiple initramfs regions are consecutive.
- Validate that no subject has write access to the crash audit region.
- Validate that subject device mmconf mappings are correct.
- Validate that shared PCI devices specify the same PCI element.
- Validate that the VMX controls conform to the checks specified in Intel SDM Vol. 3C, "26.2.1 Checks on VMX Controls".
- Validate that the Pin-Based VM-Execution controls meet the requirements for the execution of Muen.
- Validate that the Processor-Based VM-Execution Controls meet the requirements for the execution of Muen.
- Validate that the secondary Processor-Based VM-Execution Controls meet the requirements for the execution of Muen.
- Validate that the VM-Exit Controls meet the requirements for the execution of Muen.
- Validate that the VM-Entry Controls meet the requirements for the execution of Muen.
- Validate that the VMX CR0 guest/host masks meet the requirements for the execution of Muen.
- Validate that the VMX CR4 guest/host masks meet the requirements for the execution of Muen.
- Validate that the VMX Exception bitmap meet the requirements for the execution of Muen.

6.11 Hardware

The following checks are performed on the hardware section of the policy.

- Validate that memory regions fit into available hardware memory.
- Validate that no memory blocks overlap.
- Validate that the size of memory blocks is a multiple of page size.
- Validate that PCI config space address is specified if PCI devices are present.
- Validate that the hardware provides enough physical CPU cores.
- Validate that the processor CPU sub-elements are correct.
- Validate that at least one I/O APIC device is present.

- Validate that all I/O APICs have a valid source ID capability.
- Validate that at least one and at most eight IOMMU devices are present.
- Validate that all IOMMUs have the AGAW capability set correctly and that multiple IOMMUs specify the same value.
- Validate that all IOMMUs have correct register offset capabilities.
- Check that the hardware contains a system board device providing the expected configuration.

6.12 Events

The following checks are performed to guarantee that events are correctly configured in the system policy.

- Check that all physical event names are unique.
- Check that each global event has associated sources and one target.
- Check subject event references.
- Validate that there are no self-references in subject's event notification entries.
- Validate that notification entries switch to a subject running on the same core and in the same scheduling group.
- Validate that target subjects of IPI notification entries run on different logical CPUs.
- Validate that target event IDs are unique.
- Validate that source event IDs are unique per group.
- Check source event ID validity.
- Check that source event IDs of the VMX Exit group are all given or a default is specified.
- Check that self events provide a target action.
- Check that kernel-mode events have an action specified.
- Check that system-related actions are only used with kernel-mode events.
- Check that level-triggered IRQs have a corresponding unmask IRQ event.

Chapter 7

Policy Structure

7.1 Policy Schema Documentation

7.1.1 systemType

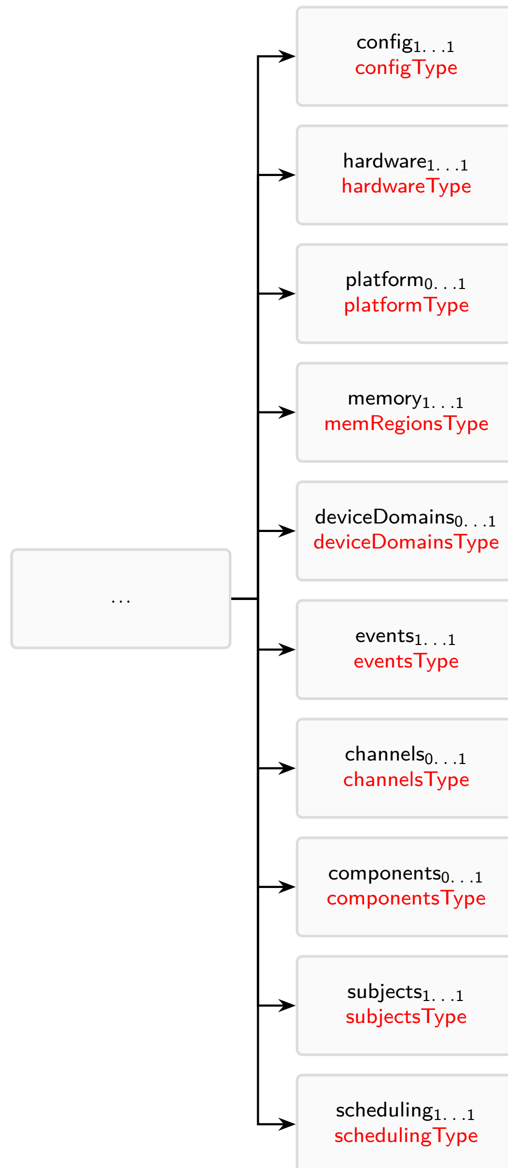
A Muen system policy specifies all hardware resources such as physical memory, devices, CPU time, etc and how these resources are accessed by the separation kernel, the subjects and devices.

The `system` section is the top-level element in the Muen system policy. It contains various sub-elements which specify all aspects of a concrete system.

This is the *source format* of the Muen system policy. It allows for abstractions, such as channels, which are broken down into their constituent parts by the toolchain in format A and B accordingly.

See line 3 and following in listing 8.1 on page 91 for an annotated system policy example.

Structure



7.1.2 configType

The purpose of a config section is to specify configuration values which parameterize a system or a component. It allows to declare boolean, string and integer values. The following sections in the system policy provide support for configuration values:

- System
- Platform
- Component

During the build process, configuration values provided by the platform are merged into the global system configuration. Component configuration values allow the parameterization of component-local functionality.

Besides component parameterization, configuration options can be used in `if` conditionals, as shown in the following example.

```
<if variable="xhcidbg_enabled" value="true">
 ...
</if>
```

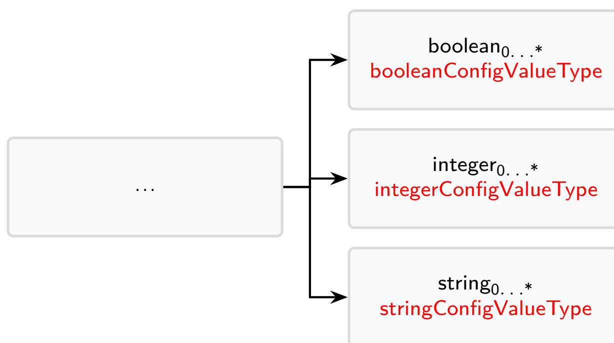
A second use case is XML attribute value expansion as follows:

```
1    <channel name="debuglog" size="$logchannel_size"/>
```

The `size` attribute value is not specified directly, but parameterized via an integer configuration option.

See line 17 in listing 8.1 for an example config section.

Structure



7.1.3 booleanConfigValueType

Configuration option for values in boolean format.

Attributes

Name	Type	Use
name	nameType	optional
Name of the configuration option.		
value	booleanType	optional
Value of the configuration option.		

7.1.4 nameType

Base: xs:string

The `nameType` is used to give (unique) names to elements.

Restrictions

minimal length = 1, maximal length = 63

7.1.5 booleanType

Base: xs:string

Boolean type.

Restrictions

values:

- true
- false

7.1.6 integerConfigValueType

Configuration option for values in integer format.

Attributes

Name	Type	Use
name	nameType	optional
Name of the configuration option.		
value	xs:integer	optional
Value of the configuration option.		

7.1.7 stringConfigValueType

Configuration option for values in string format.

Attributes

Name	Type	Use
name	nameType	optional
Name of the configuration option.		
value	xs:string	optional
Value of the configuration option.		

7.1.8 hardwareType

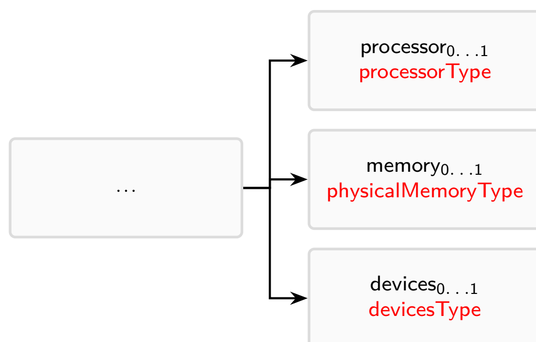
Systems running the Muen SK perform static resource allocation at integration time. This means that all available hardware resources of a target machine must be defined in the system policy in order for these resources to be allocated to subjects.

The `hardware` element is the top-level element of the hardware specification in the system policy. Information provided by a hardware description includes the amount of available physical memory blocks including reserved memory regions (RMRR), the number of physical CPU cores and hardware device resources.

The Muen toolchain provides a handy tool to automate the cumbersome process of gathering hardware resource data from a running Linux system: `mugenhwcfg`[1].

See line 79 in listing 8.1 for an example hardware section.

Structure



7.1.9 processorType

The `processor` element specifies the number of CPU cores, the processor speed in MHz and the Intel VMX preemption timer rate.

Since Intel CPUs can have arbitrary APIC identifiers, the APIC IDs of all physical CPUs are enumerated here. The APIC ID is required for interrupt and IPI routing.

[1]https://git.codelabs.ch/?p=muen/mugenhwcfg.git

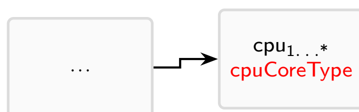
See line 98 in listing 8.1 for an example processor element. The `cpu` elements must fulfill the following constraints to be valid:

- A node exists for every physical core of the system
- The optional `cpuId` attribute of all elements must be consecutive
- If specified, a node with `cpuId` value `0` must exist
- A node with `apicId` value `0` must exist and, if specified, it must have a `cpuId` value within the active CPU range, i.e. the BSP is part of the system scheduling plan
- All `apicId` attributes must have even numbers

Attributes

Name	Type	Use
cpuCores	xs:positiveInteger	required
Number of available CPU cores. Note that this value designates physical, hardware cores, not Hyper-Threading (HT) *threads*. HT is disabled on Muen.		
speed	xs:positiveInteger	required
Tick rate of CPU cores in MHz.		
vmxTimerRate	vmxTimerRateType	required
The VMX-premption timer counts down at a rate proportional to that of the timestamp counter (TSC). This value specifies this proportion, see Intel SDM Vol. 3C, "25.5.1 VMX-Preemption Timer" for more details.		

Structure



7.1.10 cpuCoreType

Specification of one physical CPU core.

Attributes

Name	Type	Use
apicId	xs:unsignedByte	required
CPU local APIC ID, see Intel SDM Vol. 3A, "10.4.6 Local APIC ID".		
cpuId	xs:unsignedByte	optional
Unique CPU ID.		

7.1.11 vmxTimerRateType

Base: xs:nonNegativeInteger
VMX-preemption timer count down rate.

Restrictions

value $\leq$31

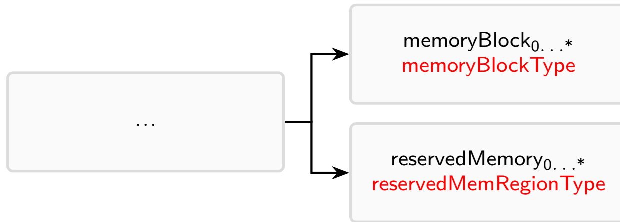
7.1.12 physicalMemoryType

The hardware `memory` element specifies the available physical memory blocks including reserved memory regions (RMRR, see Intel VT-d Specification, "8.4 Reserved Memory Region Reporting Structure").

Only memory blocks reported by the BIOS E820 map as non-*reserved* must be configured in this section, e.g. *usable* or *ACPI NVS*, *ACPI data*.

See line 110 in listing 8.1 for an example memory element.

Structure



7.1.13 memoryBlockType

Base: memoryBlockBaseType
Consecutive block of memory provided by the hardware.

Attributes

Name	Type	Use
name	nameType	required
Name of memory block.		
physicalAddress	word64Type	required
Start address of memory block.		
size	memorySizeType	required
Size of memory block.		
allocatable	booleanType	optional
Indication to a physical memory allocator that this block allows allocation of physical memory regions. If this attribute is false, an allocator should only place fixed memory regions in this range, i.e. memory regions with the physicalAddress attribute set by the integrator. Note that host physical memory below 1 MiB is considered special, the attribute must be set to false. Only unmapped memory of type *system* is allowed in that special memory block.		

7.1.14 word64Type

Base: xs:string
64-bit machine word.

Restrictions

Pattern = 16#[0-9a-fA-F]4(_([0-9a-fA-F]4))0,3#

7.1.15 memorySizeType

Base: word64Type < xs:string
The `memorySizeType` is used to declare memory sizes.

Restrictions

no restriction

7.1.16 reservedMemRegionType

Base: memoryBlockBaseType
A `reservedMemory` element is a special memory block declaration. It specifies a reserved memory region as outlined in the Intel VT-d Specification, "8.4 Reserved Memory Region Reporting Structure" (RMRR).

Reserved memory regions are BIOS allocated memory ranges that may be DMA targets for certain legacy device use-cases. Devices that require access to such a region refer to it by name.

See line 126 in listing 8.1 for an example RMRR element.

Attributes

Name	Type	Use
name	nameType	required
Name of memory block.		
physicalAddress	word64Type	required
Start address of memory block.		
size	memorySizeType	required
Size of memory block.		

7.1.17 devicesType

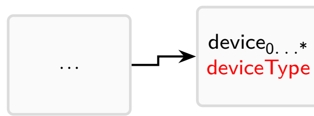
The `devices` element enumerates all devices provided by the hardware platform. Different kinds of devices, be it PCI(e) or legacy (non-PCI), can be declared in this section.

See line 139 in listing 8.1 for an example devices enumeration.

Attributes

Name	Type	Use
pciConfigAddress	word64Type	optional
Physical base address of the PCI configuration space region.		

Structure



7.1.18 deviceType

Base: deviceBaseType

The `device` element specifies a physical device and its associated resources. There are three main device resource types:

- IRQ
- I/O port range
- Memory

The presence of a PCI element indicates whether the device is a PCI or a legacy device.

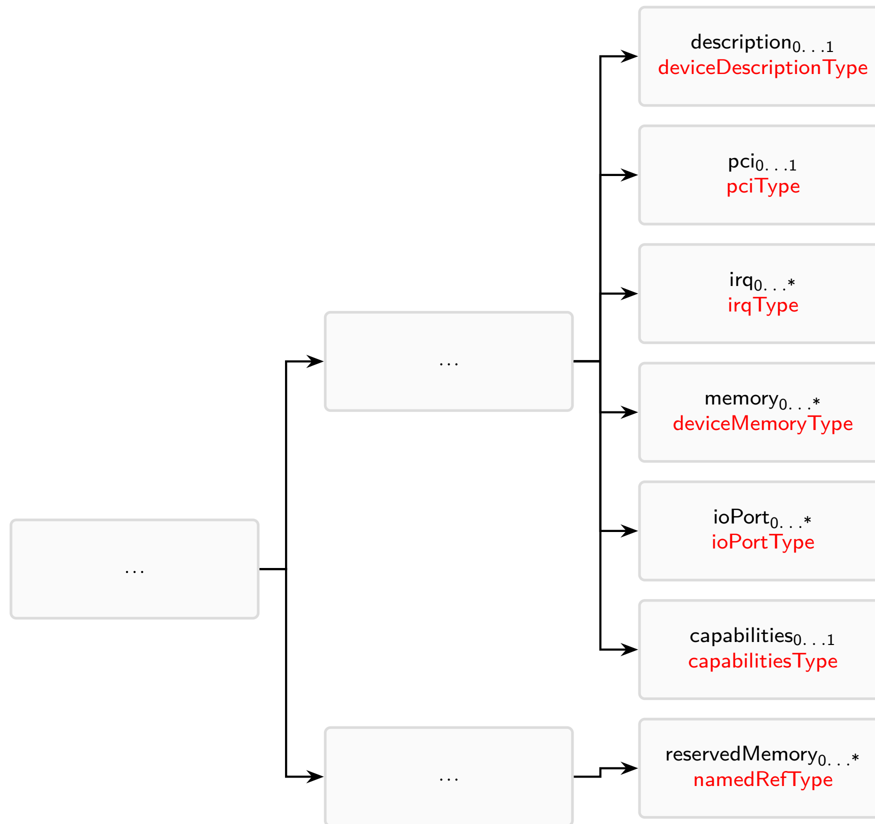
Capabilities can be used to convey additional device-specific information. The base address of the memory mapped PCI config space is defined by the `pciConfigAddress` attribute.

See line 145 in listing 8.1 for an example device declaration.

Attributes

Name	Type	Use
name	nameType	required
Unique device name.		

Structure



7.1.19 namedRefType

The `namedRefType` is used to reference a named element in the policy.

Attributes

Name	Type	Use
ref	nameType	required
Name of referenced element.		

7.1.20 deviceDescriptionType

Base: xs:string
Device description (free text).

Restrictions

no restriction

7.1.21 pciType

Base: pciAddressType
PCI(e) devices are specified using the `pci` element.
The element provides the following information:

- PCI device address (BDF)
- Identification
- IOMMU group information

The location of the PCI device in the PCI topology is specified by the Bus, Device, Function triplet (BDF).

See line 279 in listing 8.1 for an example PCI element declaration.

Attributes

Name	Type	Use
bus PCI Bus number.	byteType	required
device PCI Device number.	pciDeviceNumberType	required
function PCI Function number.	pciFunctionNumberType	required

Structure

7.1.22 deviceIdentificationType

The `identification` element specifies the PCI device class, device, revision and vendor ID.

For more information, consult the PCI Local Bus Specification, "Configuration Space Decoding".

See line 294 in listing 8.1 for an example PCI identification.

Attributes

Name	Type	Use
classcode PCI device class.	word16Type	required
vendorId PCI vendor ID.	word16Type	required
deviceId PCI device ID.	word16Type	required
revisionId PCI device revison ID.	byteType	required

7.1.23 word16Type

Base: word64Type < xs:string

16-bit machine word.

Restrictions

length = 8

7.1.24 byteType

Base: xs:string

Machine octet (8-bits).

Restrictions

Pattern = 16#[0-9a-fA-F]2#

7.1.25 iommuGroupType

Devices in the same IOMMU group cannot be properly isolated from each other because they may perform inter-device transactions directly, without going through the IOMMU.

Note that this information is currently not used by the toolchain. It is a hint to the system integrator whether two devices can be properly isolated from each other or not.

See line 303 in listing 8.1 for an example IOMMU group declaration.

Attributes

Name	Type	Use
id	xs:nonNegativeInteger	required
IOMMU group number.		

7.1.26 pciDeviceNumberType

Base: xs:string
PCI Device number.

Restrictions

Pattern = 16#[0|1][0-9a-fA-F]#

7.1.27 pciFunctionNumberType

Base: xs:nonNegativeInteger
PCI Function number.

Restrictions

value ≤ 7

7.1.28 irqType

The `irq` element specifies a device IRQ resource.

The specified IRQ number is one of:

- Legacy IRQ (ISA)
  Range `0 .. 15`.
- PCI INTx IRQ, line-signaled
  Range `0 .. Max_LSI_IRQ`, whereas `Max_LSI_IRQ` is defined by the hardware I/O APIC configuration `gsi_base` + `max_redirection_entry` of I/O APIC with `max(gsi_base)`. `gsi_base` and `max_redirection_entry` are I/O APIC device capabilities.

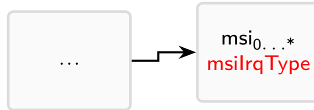
`msi` sub-elements are present if the device supports MSI interrupts. The element count designates the number of supported MSI interrrupts.

See line 182 in listing 8.1 for an example device IRQ declaration.

Attributes

Name	Type	Use
name	nameType	required
Name of device IRQ resource.		
number	irqNumberType	required
Legacy or PCI line-based IRQ.		

Structure



7.1.29 msiIrqType

There are two different interrupt types which devices may trigger: legacy/PCI LSI IRQs and Message Signaled Interrupts (MSI). The legacy/PCI LSI IRQ is specified by the number attribute of the `irq` element. For MSIs, each `msi` element defines an MSI IRQ that may be assigned to subjects. Each MSI may be individually routed.

See line 345 in listing 8.1 for example device MSI elements.

Attributes

Name	Type	Use
name Name of MSI resource.	nameType	required

7.1.30 irqNumberType

Base: xs:nonNegativeInteger

IRQ number. High IRQs are reserved for kernel usage.

Restrictions

value ≤ 220

7.1.31 deviceMemoryType

Base: memoryBlockBaseType

A device `memory` element specifies a memory region which is used to interact with the associated device.

For PCI devices, the specified region is programmed into one device BAR (Base Address Register) by system firmware. See the PCI Local Bus Specification or the PCI Express Base Specification for more details.

See line 163 in listing 8.1 for an example device memory declaration.

Attributes

Name	Type	Use
name Name of memory block.	nameType	required
physicalAddress Start address of memory block.	word64Type	required
size Size of memory block.	memorySizeType	required
caching Device memory caching type.	cachingType	required

7.1.32 cachingType

Base: xs:string

Memory caching type, see Intel SDM Vol. 3A, "11.3 Methods of Caching Available".

- Strong Uncacheable (UC)
- Write Combining (WC)

- Write Through (WT)
- Write Back (WB)
- Write Protected (WP)

Restrictions

Pattern = UC|WC|WT|WB|WP

7.1.33 ioPortType

The `ioPort` element specifies a device I/O port resource from `start` octet up to and including `end` octet. A single byte-accessed port is designated by specifying the same `start` and `end` values.

See line 173 in listing 8.1 for an example device IRQ declaration.

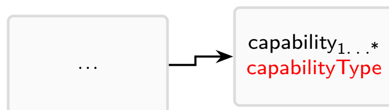
Attributes

Name	Type	Use
name Name of I/O port resource.	nameType	required
start Start port of this resource.	word16Type	required
end End port of this resource.	word16Type	required

7.1.34 capabilitiesType

List of device capabilities.

Structure



7.1.35 capabilityType

Base: xs:string

A device `capability` is used to assign additional information to a device. Such a capability might be used by the Muen toolchain to perform certain actions on devices with a given capability (e.g. `ioapic`). A system integrator may use this facility to define its own capabilities used by custom tools.

A capability element can have an optional value.

See line 235 in listing 8.1 for example capabilities.

Attributes

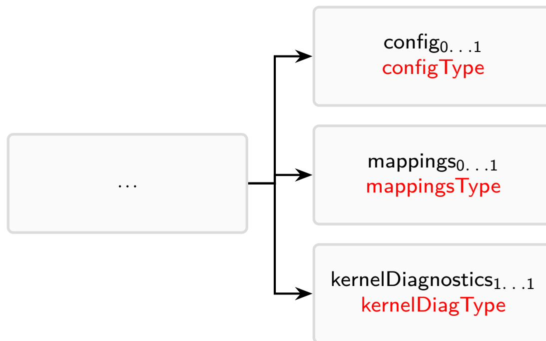
Name	Type	Use
name Capability name (free text).	xs:string	required

7.1.36 platformType

To enable an uniform view of the hardware resources across different physical machines from the system integrators perspective, the platform description layer is interposed between the hardware resource description and the rest of the system policy. This allows to build a Muen system for different physical target machines using the same system policy.

See line 501 in listing 8.1 for an example platform section.

Structure

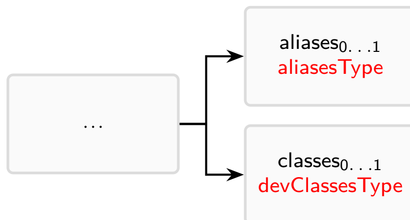


7.1.37 mappingsType

Platform device alias and class mappings section. Used to assign a stable name to a hardware device or to group (multiple) devices under a given name.

See line 510 in listing 8.1 for an example platform mappings section.

Structure



7.1.38 aliasesType

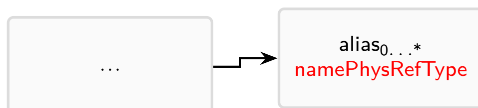
Aliases are a renaming mechanism for physical hardware devices and their resources. By using alias names in the system policy references to concrete hardware resources can be avoided. Additionally, aliases may be used to define a device which only contains a subset of the resources of the physical device. This can be achieved by only renaming the resources that the device alias should export.

See line 516 in listing 8.1 for an example aliases section.

Aliases are resolved in the following system policy sections.

- `/system/subjects/subject/component/map`
- `/system/subjects/subject/devices/device`
- `/system/deviceDomains/domain/devices/device`

Structure



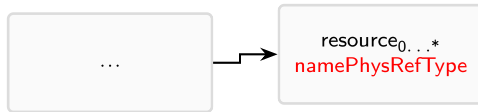
7.1.39 namePhysRefType

Named resource reference. Used for device aliases and device alias resource references.

Attributes

Name	Type	Use
name Alias name.	nameType	required
physical Reference to physical device or device resource.	nameType	required

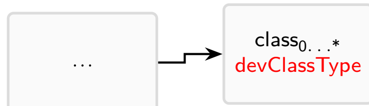
Structure



7.1.40 devClassesType

The `classes` element specifies a list of device classes.

Structure



7.1.41 devClassType

Device classes enable the grouping of devices and allow referencing all devices by a single name. This simplifies the process of assigning multiple devices to a subject.

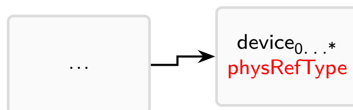
Note: A device class may contain an arbitrary number of devices, including zero.

See line 548 in listing 8.1 for a device class example.

Attributes

Name	Type	Use
name Device class name.	nameType	required

Structure



7.1.42 physRefType

Reference to physical device or physical device resource.

Attributes

Name	Type	Use
physical Physical resource name (device or resource sub-element).	nameType	required

7.1.43 kernelDiagType

The debug build Muen SK can be instructed to output debugging information during runtime. The platform diagnostics device specifies which device the kernel is to use for this purpose.

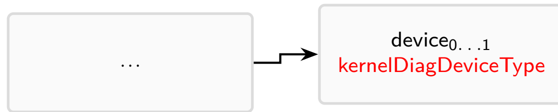
The presence of this device and the necessary resources are checked by the validator tool.

See line 568 in listing 8.1 for an example platform diagnostic device configuration.

Attributes

Name	Type	Use
type Specifies the type of diagnostics device to use.	kernelDiagKindType	required

Structure



7.1.44 kernelDiagDeviceType

Reference to physical device for `uart` and `vga` diagnostic device type.

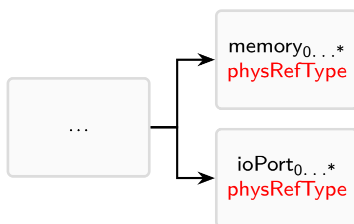
If an UART device is referenced via type `uart`, an I/O port resource must be provided. If a VGA device is referenced via type `vga`, a memory resource must be provided (both checked via validator).

Attributes

Name	Type	Use
physical	nameType	required

Name of physical device to use for kernel diagnostics output.

Structure



7.1.45 kernelDiagKindType

Base: xs:string

Type of diagnostics device. While `none` disables kernel diagnostics output, `uart` specifies an Universal Asynchronous Receiver-Transmitter serial device. `hsuart` is a High-Speed UART with memory mapped I/O.

`vga` outputs the kernel diagnostics information to a VGA console, which is mainly useful for initial bring-up of a new hardware platform with no UART device.

Restrictions

values:

- none
- uart
- hsuart
- vga

7.1.46 memRegionsType

This section declares all physical memory regions (RAM) and thus the physical memory layout of the system. Regions declared in this section can be assigned to subjects and device domains.

Memory regions are defined by the following attributes:

- Name
- Caching type
- Size
- Physical address*

- Alignment*
- Memory type*

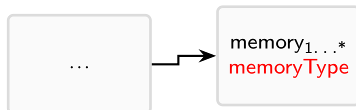
Attributes with an asterisk are optional. While alignment and memory type are set to a default value if not specified, the physical address is filled in by the allocator tool, which allocates all memory regions and finalizes the physical memory layout.

Additionally, the content of a region can be declared as backed by a file or filled with a pattern.

Note: The caching type is an attribute of the physical memory region by design to avoid inconsistent typing, even though the Intel Page Attribute Table (PAT) mechanism allows to set it for each memory mapping, see Intel SDM Vol. 3A, "11.12.4 Programming the PAT".

See line 582 in listing 8.1 for an example memory region section.

Structure



7.1.47 memoryType

Base: physicalMemBaseType < memoryBaseType

The `memoryType` specifies a physical memory region by name, size and caching.

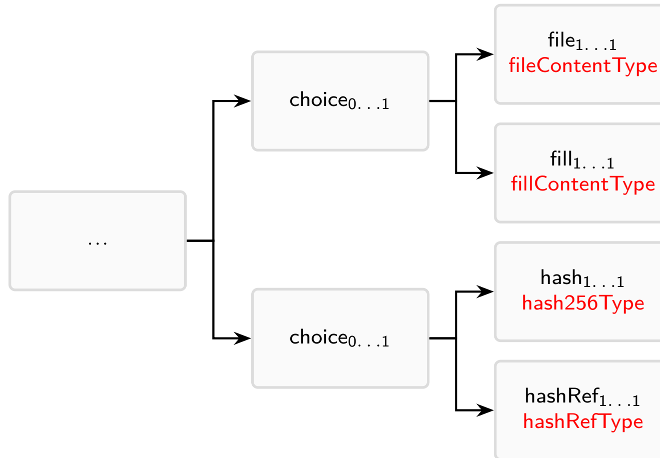
If no explicit physical address is specified for the region, the `mucfgalloc` tool will allocate a free one in usable memory, honoring the optional `alignment` attribute.

If no explicit `alignment` attribute is specified, it is set to `16#1000#` by the expander. If no explicit `type` attribute is specified, it is set to `subject` by the expander.

Attributes

Name	Type	Use	Description
size	memorySizeType	required	Size of region. Must be a multiple of page size (4K). Enforced by validator.
name	nameType	required	Name of region.
caching	cachingType	required	Caching type to use for memory region.
type	subjectMemoryKindType	optional	Optional subject memory type.
alignment	alignmentType	optional	Alignment the physical address of the memory region must honor (checked by the validator tool).
physicalAddress	word64Type	optional	Physical address of memory region.

Structure



7.1.48 subjectMemoryKindType

Base: memoryKindType < xs:string

Subject memory type to categorize memory assigned to a subject. The validator tool checks that a subject only maps memory regions of types outlined in this section (6.10).

Also used by build tools to lookup certain elements by type. For example, the `mugenzp` tool looks for subject memory of type `subject_zeropage` to process all Linux zero-pages in the policy.

The following memory types are currently supported:

- `subject`
  Generic subject memory, used e.g. for RAM regions of VM subjects. The `mugenzp` tool used for Linux VMs (5.2.9) exports such regions as `E820_RAM` in the ZP E820 memory map.

- `subject_info`
  Subject info (sinfo) region provided to all subjects. The sinfo region is used to query information about the execution environment. The file backing of this region is created by the `mugensinfo` tool (5.2.9).

- `subject_state`
  Subject execution state. Mapped into the SK kernel executing the given subject, kernels running on other CPUs have no access. Accessible by subject monitors running on the same CPU if specified in the policy. Validator enforces that each subject has an associated `subject_state` region and that it is mapped at the expected virtual address in the executing kernel (6.10).

- `subject_binary`
  Subject executable as a whole or separate subject executable regions (text, rodata, data, bss, stack) with access rights (writable/executable). The `mucbinsplit` tool automatically creates a component `provides` section with separate binary regions and associated backing files from a component binary (5.3.5).

- `subject_channel`
  Physical memory region used as shared channel between two subjects. The expander tool transforms channels in system policy source format to memory regions with this type in system policy format A/B, as described in section 7.1.65.

- `subject_crash_audit`
  Memory region used by crash audit facility to store system crash information into slots, see [2]. This information is preserved after a crash by performing a system warm start. Validators enforce that

  - Region is present and uncached, 6.5
  - Region does not overlap with image, 6.5

  - Kernel mappings are present and correct, 6.4
  - No subject has write access to this region, 6.10

- `subject_initrd`
  Physical memory of this type designates an *initial ramdisk*. This memory type is mostly used by Linux VMs. If multiple initrd regions are mapped into a subject, they must be adjacent (6.10).

  The `mugenzp` tool (5.2.9) extracts the virtual address and size of a subject-mapped region of this memory type and stores the values in the generated Linux zero-page (ZP) backing file.

- `subject_bios`
  Indicates to subjects that the memory region is reserved for BIOS/firmware and must not be used as regular RAM.

- `subject_acpi_*`
  Indicates to subjects that the memory region contains an ACPI table. See the ACPI specification for more information about RSDP, XSDT, FADT and DSDT ACPI tables. The `mugenzp` tool (5.2.9) exports such regions as `E820_ACPI` in the ZP E820 memory map.

- `subject_zeropage`
  Indicates to Linux subjects that the memory region contains a zero-page. See the Linux kernel Zero Page documentation for more information.

- `subject_solo5_boot_info`
  Indicates to a VM running Solo5/Mirage that the memory region contains a boot info structure. The file-backing of such a region may be created using the `mugensolo5` tool (5.2.9).

- `subject_device`
  Designates a memory region which is allowed to be added to a subject and a device domain. The difference to the `subject` memory type is that the region is not exported as `E820_RAM` but `E820_RESERVED` to Linux subjects. Therefore, such a region is useful to implement custom drivers without interference from Linux DMA zone handling.

- `subject_timed_event`
  Region designates a subject timed event page, as described in [1]. The expander tool creates a physical memory region for each subject and maps it into the associated subject and the SK kernel executing this subject.

Restrictions

values:

- subject
- subject_info
- subject_state
- subject_binary
- subject_channel
- subject_crash_audit
- subject_initrd
- subject_bios
- subject_acpi_rsdp
- subject_acpi_xsdt
- subject_acpi_fadt

- subject_acpi_dsdt
- subject_zeropage
- subject_solo5_boot_info
- subject_device
- subject_timed_event

7.1.49 alignmentType

Base: word64Type < xs:string

Memory alignment constraint for memory region. Taken into account by the allocator tool and checked by the validator.

Restrictions

values:

- 16#1000#
- 16#0020_0000#
- 16#4000_0000#

7.1.50 fileContentType

The `file` child element designates a file-backed memory region.

The `filename` attribute specifies the name of the file to use as content for the physical memory region, the `offset` attribute is `none` by default but can be customized to include a partial file.

See line 662 in listing 8.1 for a file-backed memory region example. The following checks on the file content are performed.

- If `offset` is `none`, the size of the file must be less than the memory region size.
- If `offset` is not `none`, the offset must be less than the file size. The file size is *not* checked but the memory region size is used as upper bound.

Attributes

Name	Type	Use
filename	xs:string	required
Filename of file to (partially) include. Note that the actual file processed by the toolchain also depends on the working directory passed as command line option to the specific tool.		
offset	optionalOffsetType	required
Read file offset in bytes.		

7.1.51 optionalOffsetType

Optional file offset value in bytes.

Restrictions

Union of

- word64Type
- noneType

7.1.52 fillContentType

The `fill` element designates a memory region which is initialized with the given pattern.

See line 616 in listing 8.1 for a file-backed memory region example.

Attributes

Name	Type	Use
pattern	byteType	required
Fill pattern (hex).		

7.1.53 hash256Type

The `hash` child element of a memory region designates a 256-bit hash over the memory content.

The Mucfgmemhashes tool in the Muen toolchain generates such a hash-sum for every content-backed memory region in a given policy.

Attributes

Name	Type	Use
value	optionalHashType	required
256-Bits message digest over file-backed memory content.		

7.1.54 optionalHashType

Allows the specification of a hash digest or `none`.

Restrictions

Union of

- hash256DigestType
- noneType

7.1.55 hashRefType

The optional `hashRef` child element of a physical memory region instructs the Mucfgmemhashes tool to copy the hash element of the referenced memory region after message digest generation.

From an abstract point of view, the `hashRef` element is a way to link multiple memory regions by declaring that the hash of the content is the same. This concept is e.g. used by the subject loader mechanism to restore writable memory regions to their initial state.

Attributes

Name	Type	Use
memory	nameType	required
Name of referenced physical memory region.		

7.1.56 deviceDomainsType

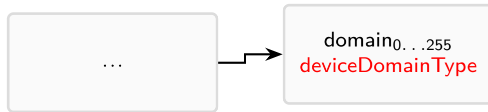
The physical memory accessible by PCI devices is specified by so called device domains. Such domains define memory mappings of physical memory regions for one or multiple devices. Device references select a subset of hardware devices provided by the hardware/platform. Devices may be referenced by device name, alias or device class.

Device references can optionally set the `mapReservedMemory` attribute so RMRR regions referenced by the device are also mapped into the device domain.

Device domains are isolated from each other by the use of Intel VT-d.

See line 678 in listing 8.1 for a device domain example.

Structure



7.1.57 deviceDomainType

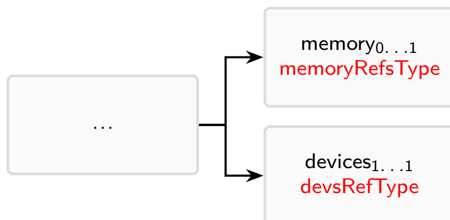
A device domain allows referenced devices access to the specified memory regions. It also provides handling for reserved memory region reporting (RMRR), see Intel VT-d Specification, "8.4 Reserved Memory Region Reporting Structure".

Attributes

Name	Type	Use
name	nameType	required

Name of the device domain.

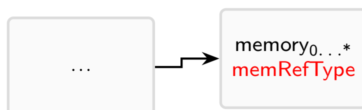
Structure



7.1.58 memoryRefsType

List of physical memory region references.

Structure



7.1.59 memRefType

A `memory` element maps a physical memory region into the address space of a device domain or subject entity. The region will be accessible to the entity at the specified `virtualAdress` with permissions defined by the `executable` and `writable` attributes.

See line 694 in listing 8.1 for an example of such a mapping.

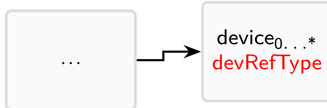
Attributes

Name	Type	Use
virtualAddress	word64Type	required
Address in entity address space where the physical memory region is mapped.		
physical	nameType	required
Name of referenced physical memory region.		
logical	nameType	required
Logical name of mapping.		
writable	booleanType	required
Defines if the mapped memory is writable.		
executable	booleanType	required
Defines if the memory region contents are executable by the processor.		

7.1.60 devsRefType

Device domain device references.

Structure



7.1.61 devRefType

Device domain device reference. Referenced devices gain access to memory regions of device domain.

Attributes

Name	Type	Use
logical	nameType	required
Logical name in this context.		
physical	nameType	required
Physical device or device alias to include in given device domain.		
mapReservedMemory	booleanType	optional
Whether to automatically map RMRR memory associated with device.		

7.1.62 eventsType

Events are an activity caused by a subject (source) that impacts a second subject (target) or is directed at the kernel. Events are declared globally and have a unique name to be unambiguous. An event must have a single source and one target.

Subjects can use events to either deliver an interrupt, hand over execution to or reset the state of a target subject. The first kind of event provides a basic notification mechanism and enables the implementation of event-driven services. The second type facilitates suspension of execution of the source subject and switching to the target. Such a construct is used to pass the thread of execution on to a different subject, e.g. invocation of a debugger subject if an error occurs in the source subject. The third kind is used to faciliate the restart of subjects.

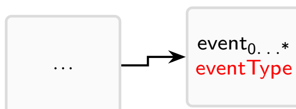
An event can also have the same source and target, which is called *self* event. Such events are useful to implement para-virtualized timers in VM subjects for example.

Kernel events are special in that they are targeted at the kernel. The currently supported events are system reboot and shutdown.

For documentation about linking physical events to source and target subjects, see section 7.1.124.

See line 720 in listing 8.1 for an example events section.

Structure



7.1.63 eventType

The `eventType` specifies an event by name and mode.

The following event modes are currently supported:

- `asap`
  The asap event is an abstraction to state that the event should be delivered as soon as possible, depending on the CPU of the target subject. If the target runs on another CPU core, this mode is expanded to mode *ipi*, which is only available in policy formats A and B,

instructing the kernel to preempt the kernel running the target subject and inject the event immediately. If the target subject runs on the same core as the source subject, the mode is expanded to mode *async*.

- `async`
  Async events trigger no preemption at the target subject. The event is marked as pending in the target subject's pending event table and inserted on the next VM exit/entry cycle of the target subject.

- `self`
  An event can also have the same source and target, which is called a self event. Such events are useful to implement para-virtualized timers in VM subjects for example. A subject sends itself a delayed event, using the timed event mechanism. Note that a self event must always have a target action assigned, which is checked by the validator.

- `switch`
  The switch mode facilitates suspension of execution of the source subject and switching to the target. This can only happen between subjects running on the same core. Such a construct is used to pass the thread of execution on to a different subject, e.g. invocation of a debugger subject if an error occurs in the source subject.

- `kernel`
  These kinds of events are directed at the kernel an thus only specify a source since the target is the kernel. They are used to enable specific subjects to unmask level-triggered IRQs and trigger a system reboot, poweroff or explicit panic (crash audit slot allocation and reboot).

See line 744 in listing 8.1 for an example global event declaration.

Attributes

Name	Type	Use
name Name of the event.	nameType	required
mode Mode of the event.	eventModeType	required

7.1.64 eventModeType

Base: xs:string
Event mode.
See 7.1.63 for details about the supported event modes.

Restrictions

values:

- asap
- async
- self
- switch
- kernel

7.1.65 channelsType

Inter-subject communication is specified by so called channels. These channels represent directed information flows since they have a single writer and possibly multiple readers. Optionally a channel can have an associated notification event (doorbell interrupt).

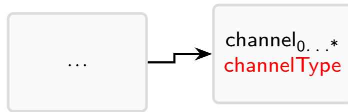
Channels are declared globally and have an unique name to be unambiguous.

Note that channels are a policy source format abstraction. The toolchain resolves this concept into memory regions and events as well as the appropriate subject mappings.

For documentation about linking physical channels to subjects see section 7.1.132. For documentation about declaring requested channels in components see section 7.1.97. For information how to map a physical channel with a logical component channel at subject level, see section 7.1.139.

See line 808 in listing 8.1 for an example channel section.

Structure



7.1.66 channelType

The `channel` element declares a physical channel.

Besides the `name` and `size` of the channel, the optional `hasEvent` attribute can be set to declare that the given channel requests an associated event. The expander tool will then automatically create a global event of the requested event type.

See line 822 in listing 8.1 for an example channel declaration.

Attributes

Name	Type	Use
name Channel name.	nameType	required
size Size of the channel in bytes. Must be a multiple of page size (4K). Enforced by validator.	memorySizeType	required
hasEvent Associated event type (if any).	eventModeType	optional

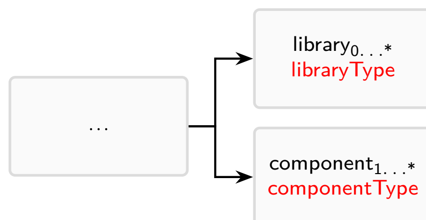
7.1.67 componentsType

The `components` element holds a list of components and component libraries.

Note that components are a policy source format abstraction. The toolchain resolves this concept into subjects by adding the appropriate memory regions, events and devices.

See line 857 in listing 8.1 for an example components section.

Structure



7.1.68 libraryType

A component library is a specialized component specification which is used to share common resources required for library code to operate. Component libraries can be included by multiple components in order to share functionality. An example is a logging service provided by a dedicated component, whereas the logging client is provided as a library with a shared memory channel for the actual log messages.

A component specification declares library dependencies to request the library resources from the system through the inclusion of the library specification in the `depends` section. This way components inherit the resources of libraries.

On the source code level, a library is included by mechanisms provided by the respective programming language. Note that the component library code is *not* shared between components but lives in the isolated execution environment of a subject instantiating the component (i.e. statically linked libraries).

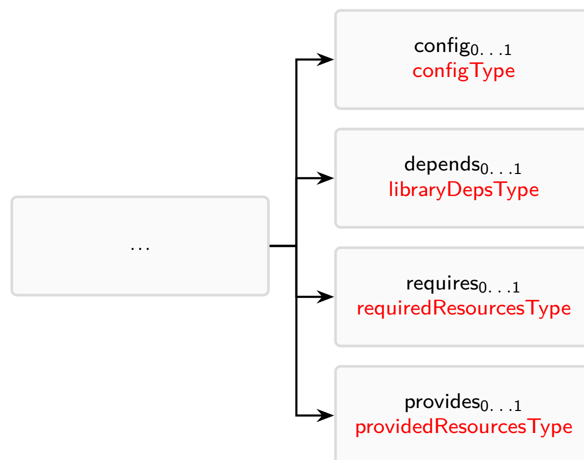
Libraries can request the same resources as ordinary components. A subject instantiating the component must also map the resources requested by libraries the component depends on.

See line 866 in listing 8.1 for example library specifications.

Attributes

Name	Type	Use
name	nameType	required
Component/library name.		

Structure



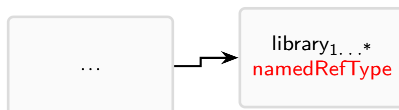
7.1.69 libraryDepsType

Components and libraries are allowed to declare dependencies to other libraries. All resources required by the included library are merged with the ones specified by the component or library. Libraries can depend on other libraries.

A subject realizing this component must correctly map all component and library resource requirements to physical resources in order to fulfill the expectations.

See line 1011 in listing 8.1 for an example dependency section.

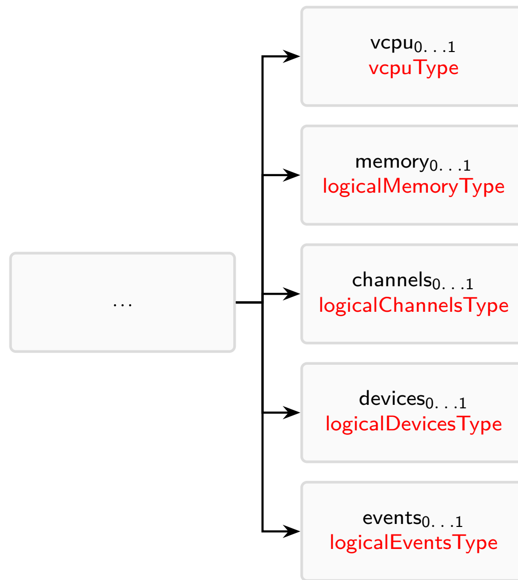
Structure



7.1.70 requiredResourcesType

Declaration of resources a component or library requires to operate.

Structure

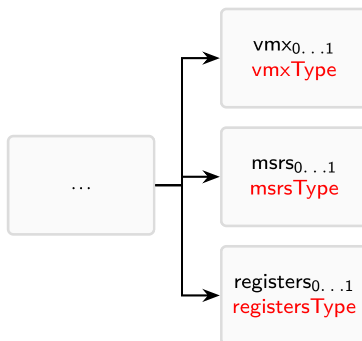


7.1.71 vcpuType

The `vcpu` element controls the execution behavior of the virtual CPU (vCPU). A default vCPU profile is selected by the component profile, but CPU execution settings can be customized both at component and subject level.

See line 938 in listing 8.1 for an example on how to customize a vCPU profile.

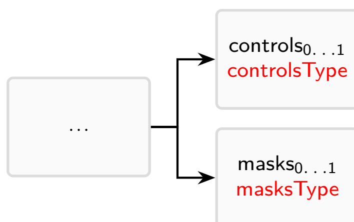
Structure



7.1.72 vmxType

Controls Intel VMX vCPU settings.

Structure



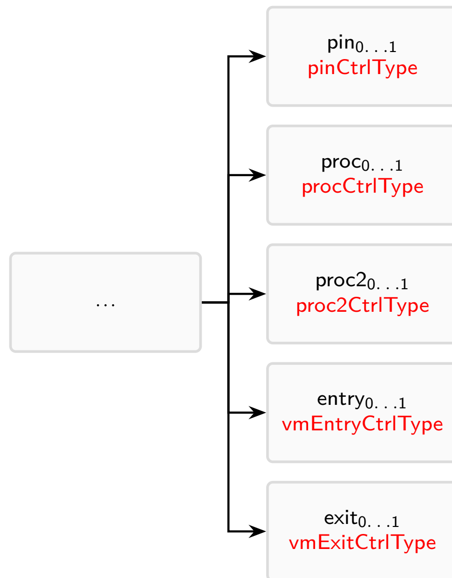
7.1.73 controlsType

Configures the following Intel VMX settings:

- Pin-Based VM-Execution Controls

- Primary Processor-Based VM-Execution Controls
- Secondary Processor-Based VM-Execution Controls
- VM-Entry Controls
- VM-Exit Controls

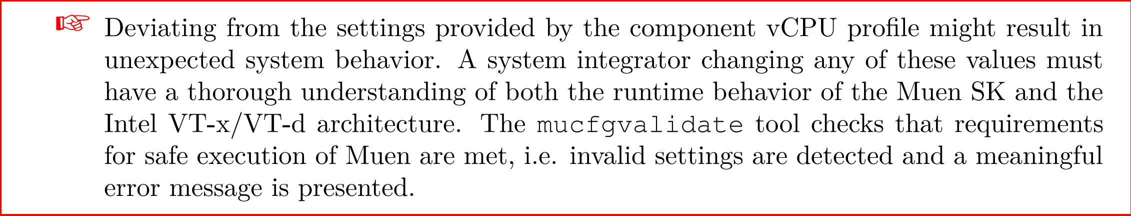
Structure



7.1.74 pinCtrlType

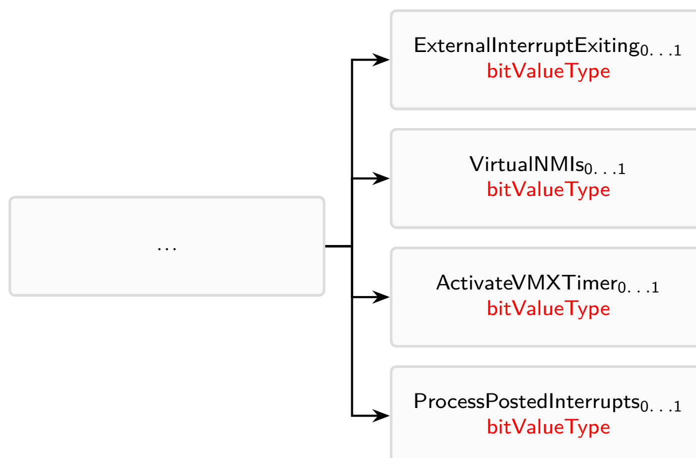
Configures Intel VMX pin-based VM-execution controls. These controls constitute a 32-bit vector that governs the handling of asynchronous events (for example: interrupts) while running in VMX non-root mode.

See Intel SDM Vol. 3C, "24.6.1 Pin-Based VM-Execution Controls" for more details and the meaning of the different bit-fields.



☞ Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator changing any of these values must have a thorough understanding of both the runtime behavior of the Muen SK and the Intel VT-x/VT-d architecture. The `mucfgvalidate` tool checks that requirements for safe execution of Muen are met, i.e. invalid settings are detected and a meaningful error message is presented.

Structure



7.1.75 bitValueType

Base: xs:nonNegativeInteger

The value of one bit, either 1 (True) 0 (False).

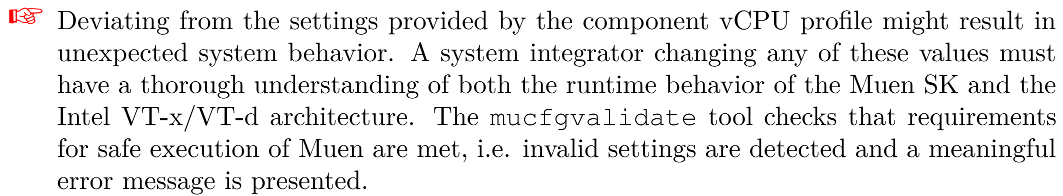
Restrictions

$0 \leq 1$

7.1.76 procCtrlType

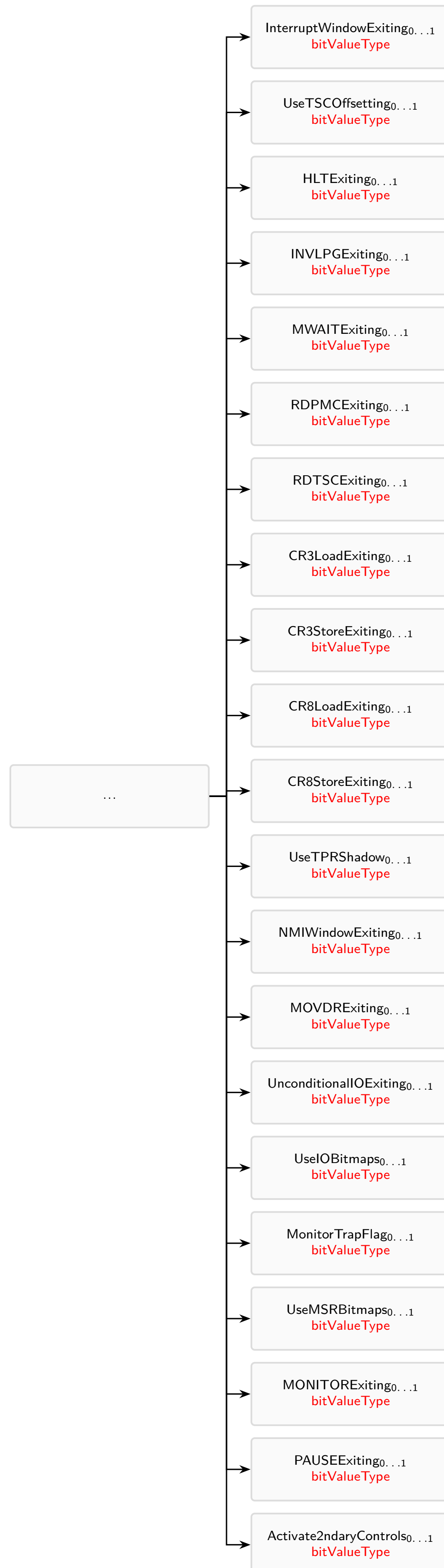
The processor-based VM-execution controls constitute two 32-bit vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions. These are the *primary processor-based* VM-execution controls and the *secondary processor-based* VM-execution controls.

The `proc` element configures the primary processor-based VM-execution controls, see Intel SDM Vol. 3C, "24.6.2 Processor-Based VM-Execution Controls" for more details and the meaning of the different bit-fields.



☞ Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator changing any of these values must have a thorough understanding of both the runtime behavior of the Muen SK and the Intel VT-x/VT-d architecture. The `mucfgvalidate` tool checks that requirements for safe execution of Muen are met, i.e. invalid settings are detected and a meaningful error message is presented.

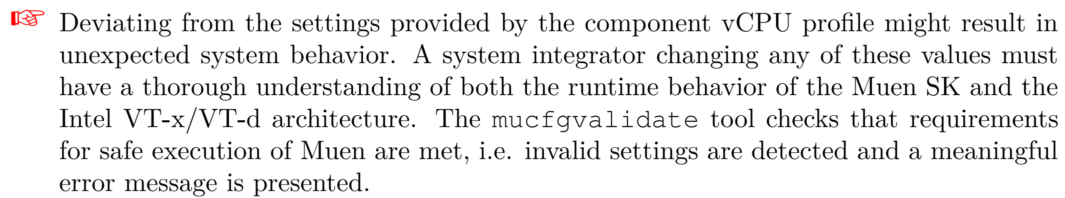
Structure



7.1.77 proc2CtrlType

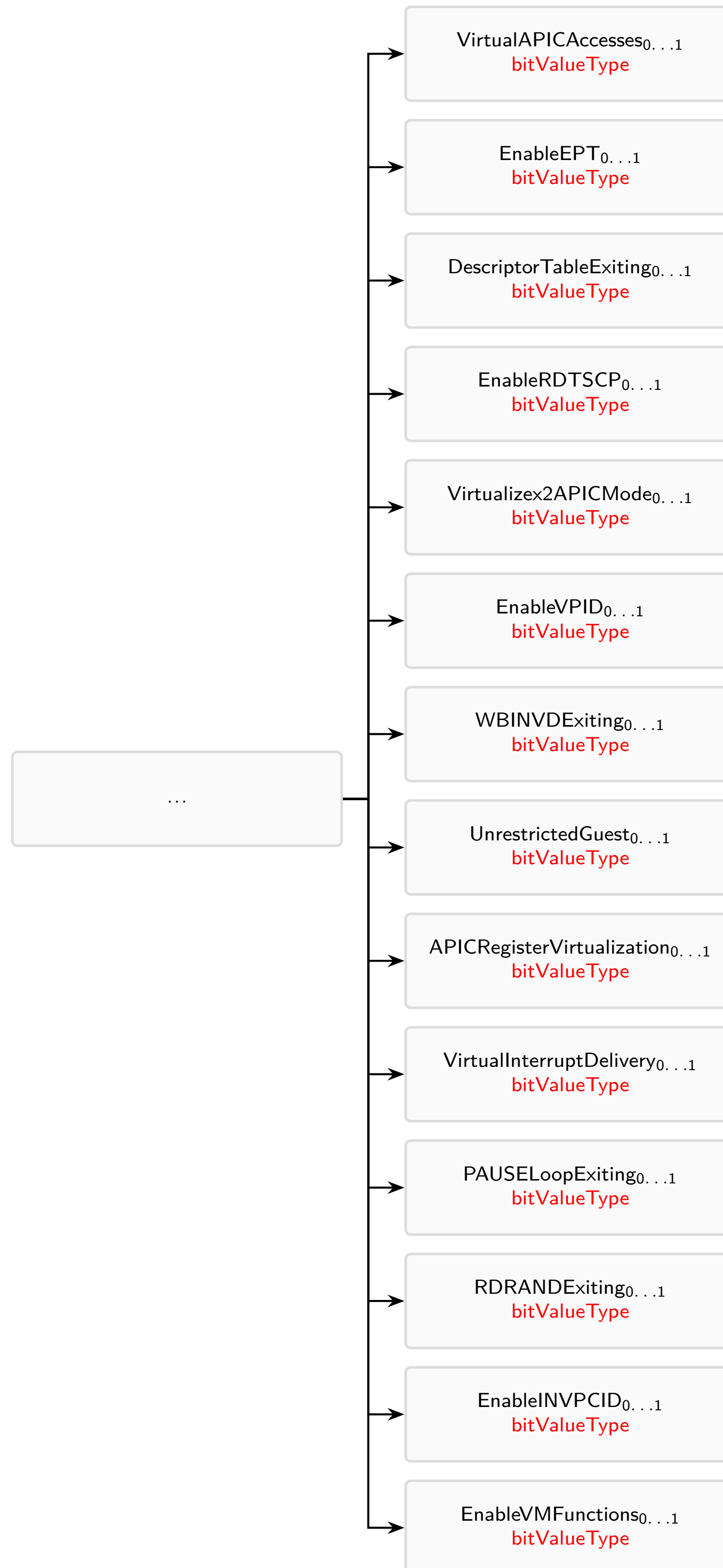
The processor-based VM-execution controls constitute two 32-bit vectors that govern the handling of synchronous events, mainly those caused by the execution of specific instructions. These are the *primary processor-based* VM-execution controls and the *secondary processor-based* VM-execution controls.

The `proc2` element configures the secondary processor-based VM-execution controls, see Intel SDM Vol. 3C, "24.6.2 Processor-Based VM-Execution Controls" for more details and the meaning of the different bit-fields.



☞ Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator changing any of these values must have a thorough understanding of both the runtime behavior of the Muen SK and the Intel VT-x/VT-d architecture. The `mucfgvalidate` tool checks that requirements for safe execution of Muen are met, i.e. invalid settings are detected and a meaningful error message is presented.

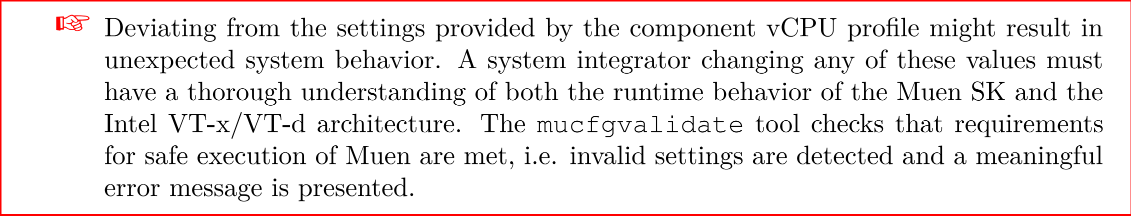
Structure



7.1.78 vmEntryCtrlType

Configures Intel VMX VM-entry controls. These controls constitute a 32-bit vector that governs the basic operation of VM entries.

See Intel SDM Vol. 3C, "24.8.1 VM-Entry Controls" for more details and the meaning of the different bit-fields.

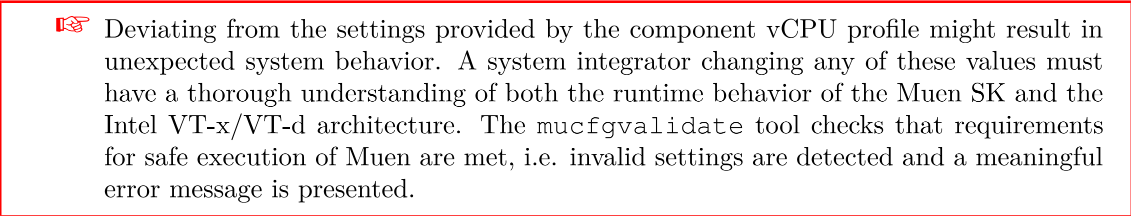

Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator changing any of these values must have a thorough understanding of both the runtime behavior of the Muen SK and the Intel VT-x/VT-d architecture. The `mucfgvalidate` tool checks that requirements for safe execution of Muen are met, i.e. invalid settings are detected and a meaningful error message is presented.

Structure

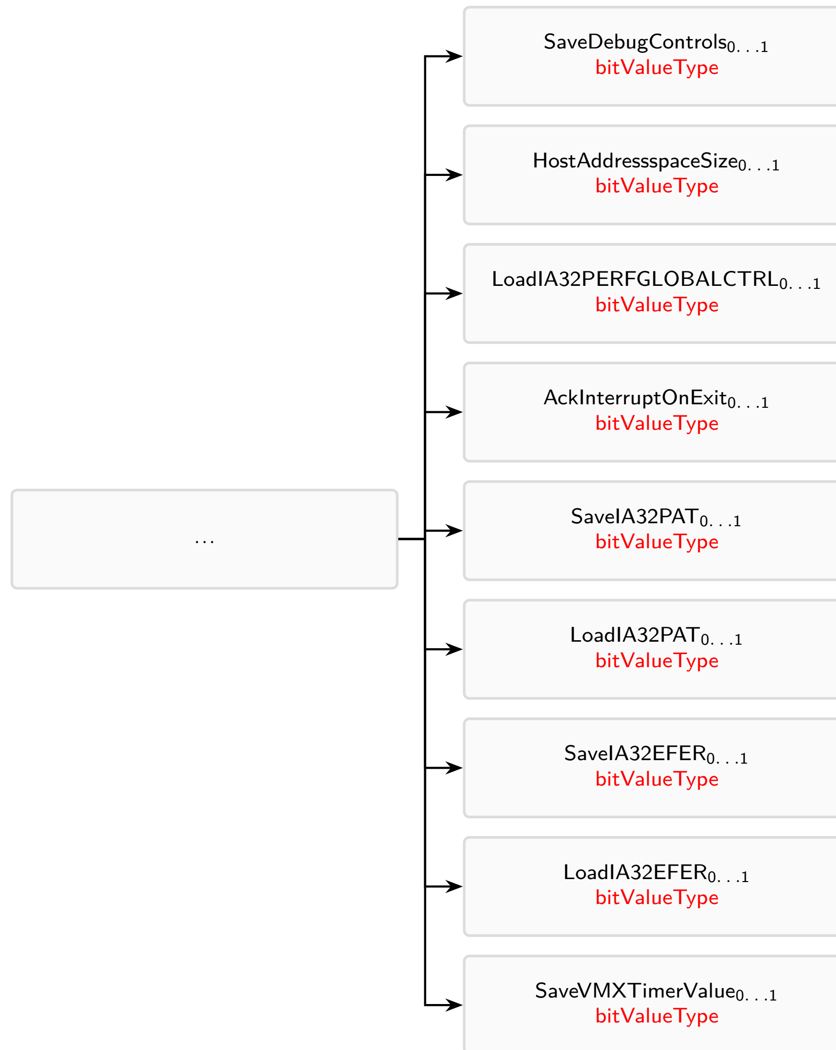
7.1.79 vmExitCtrlType

Configures Intel VMX VM-exit controls. These controls constitute a 32-bit vector that governs the basic operation of VM exits.

See Intel SDM Vol. 3C, "24.7.1 VM-Exit Controls" for more details and the meaning of the different bit-fields.


Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator changing any of these values must have a thorough understanding of both the runtime behavior of the Muen SK and the Intel VT-x/VT-d architecture. The `mucfgvalidate` tool checks that requirements for safe execution of Muen are met, i.e. invalid settings are detected and a meaningful error message is presented.

Structure



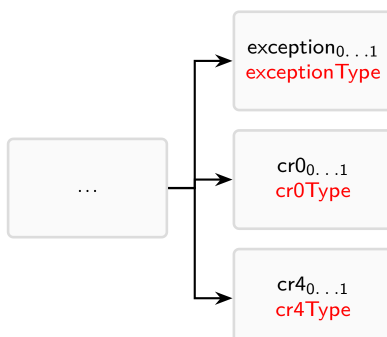
7.1.80 masksType

The `masks` element configures the Intel VMX CR0/CR4 guest/host masks and the guest/host exception bitmap.

In general, bits set to 1 in a guest/host mask correspond to bits *owned* by the host, causing a VM exit if the associated event occurs.

Reading from host owned bits in CR0/CR4 does not result in a VM exit but the value of the CR0/CR4 read shadow is returned instead (see Intel SDM Vol. 3C, "24.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4").

Structure

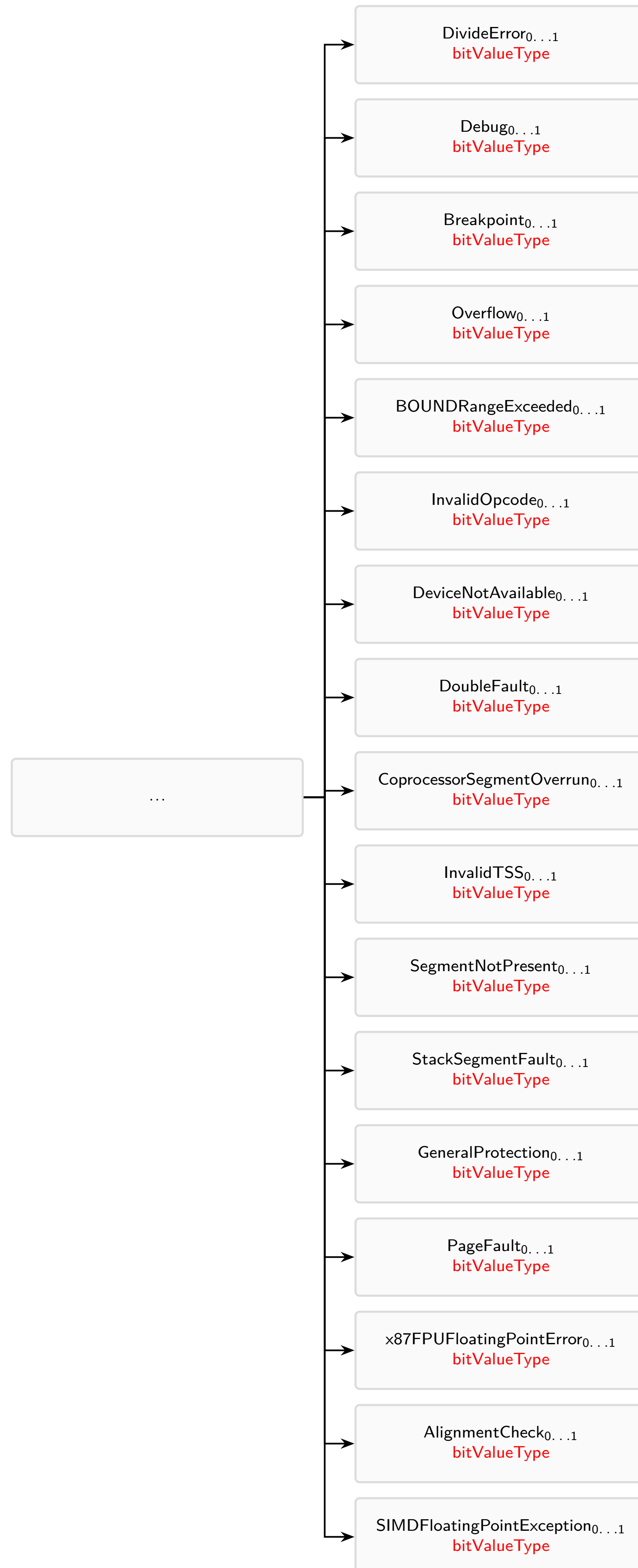


7.1.81 exceptionType

Configures Intel VMX exception bitmap. The exception bitmap is a 32-bit field that contains one bit for each exception. When an exception occurs, its vector is used to select a bit in this field. If the bit is 1, the exception causes a VM exit. If the bit is 0, the exception is delivered normally through the IDT, using the descriptor corresponding to the exceptions vector.

See Intel SDM Vol. 3C, "24.6.3 Exception Bitmap" for more details on the exception bitmap configuration.

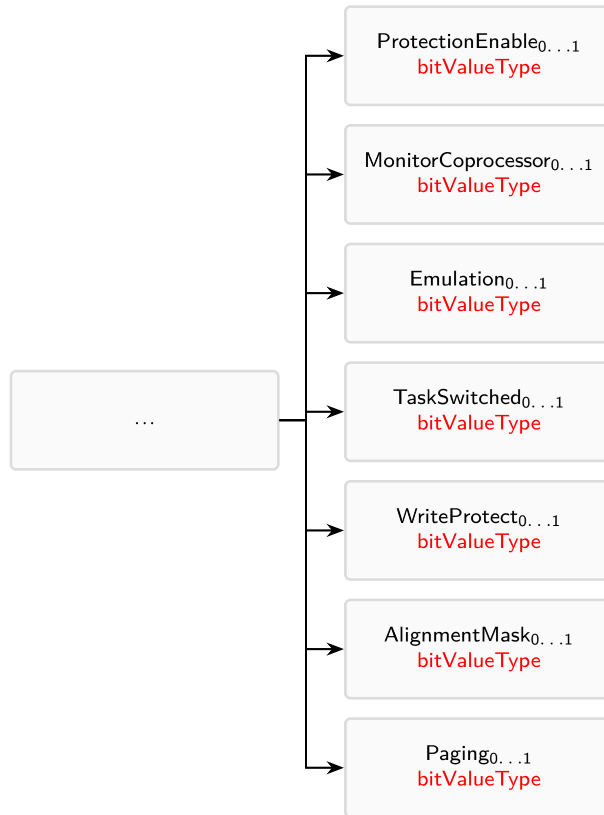
Structure



7.1.82 cr0Type

Allows to set initial values of the CR0 control register or bits in the CR0 guest/host ownership mask.

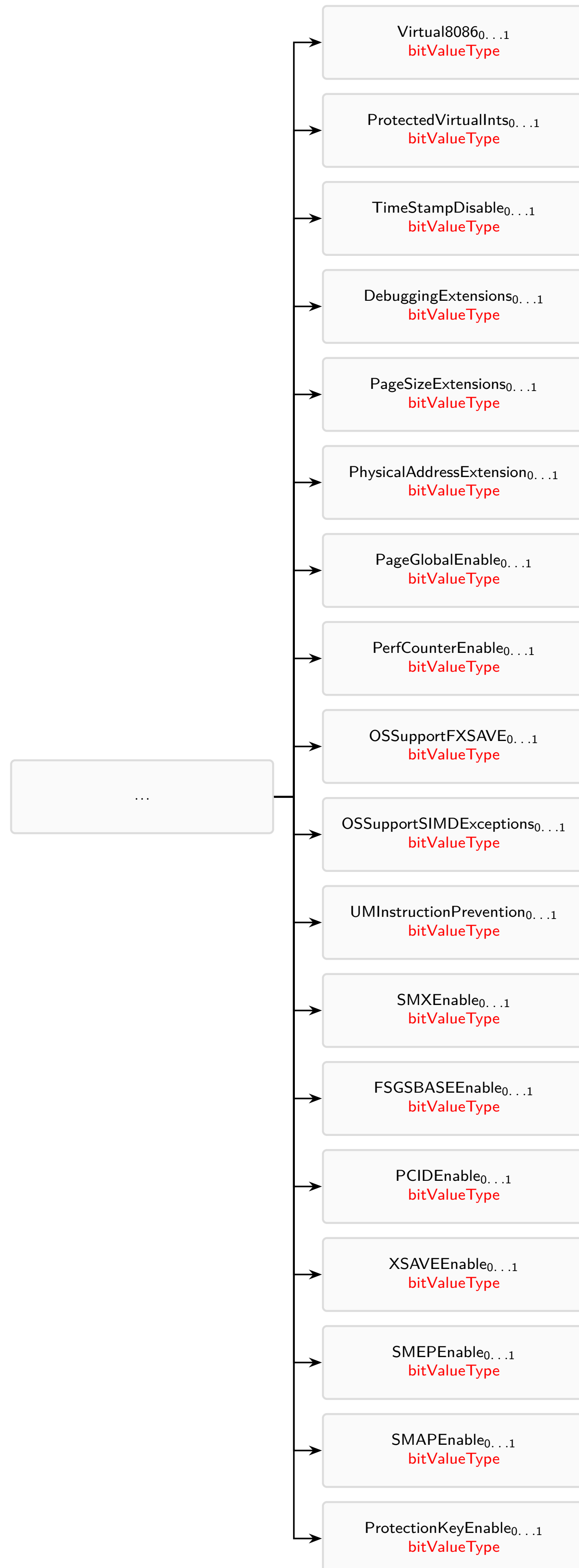
Structure



7.1.83 cr4Type

Allows to set initial values of the CR4 control register or bits in the CR4 guest/host ownership mask.

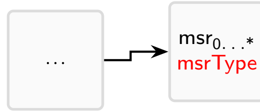
Structure



7.1.84 msrsType

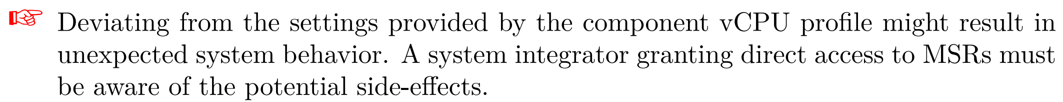
List of model-specific registers (MSRs) a subject is allowed to access. The settings in this section are translated to the MSR bitmaps of the associated subject (as described by Intel SDM Vol. 3C, "24.6.9 MSR-Bitmap Address").

Structure



7.1.85 msrType

An `msr` element allows a subject direct access to the specified model-specific register (MSR).


Deviating from the settings provided by the component vCPU profile might result in unexpected system behavior. A system integrator granting direct access to MSRs must be aware of the potential side-effects.

Attributes

Name	Type	Use
start MSR start address.	msrAddressType	required
end MSR end address.	msrAddressType	required
mode MSR access permissions.	msrModeType	required

7.1.86 msrAddressType

Base: xs:string
Start/end address value for MSRs in the low or high range:

- Low : `16#0000_0000# .. 16#0000_1fff#`
- High : `16#C000_0000# .. 16#C000_1fff#`

See also Intel SDM Vol. 3C, "24.6.9 MSR-Bitmap Address".

Restrictions

Pattern = 16#([cC0]000_)?[01]([0-9a-fA-F]3)#

7.1.87 msrModeType

Base: xs:string
MSR access rights.

Restrictions

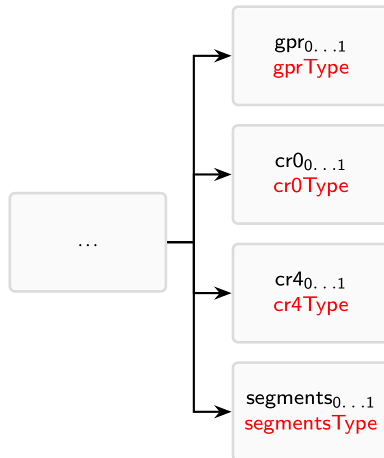
values:

- r
- w
- rw

7.1.88 registersType

The `registers` element specifies the initial value of general-purpose (GPR), CR0/CR4 and segment registers.

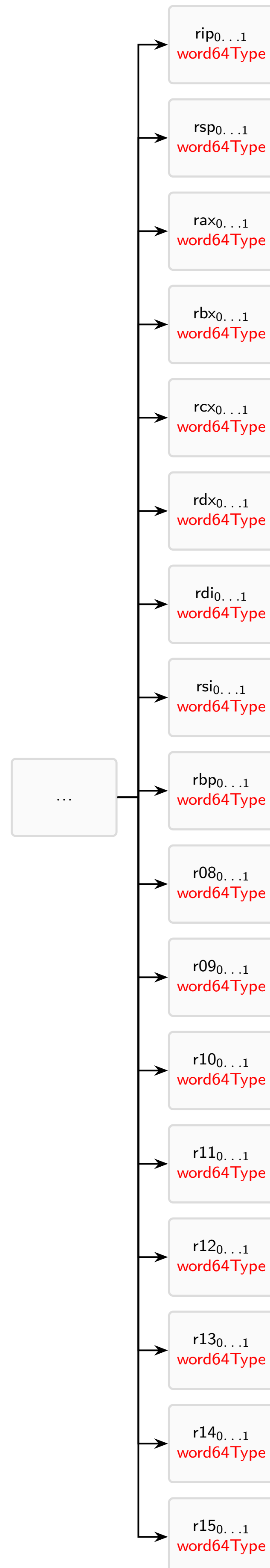
Structure



7.1.89 gprType

The `gpr` element specifies the initial values of subject general-purpose registers (GPRs).

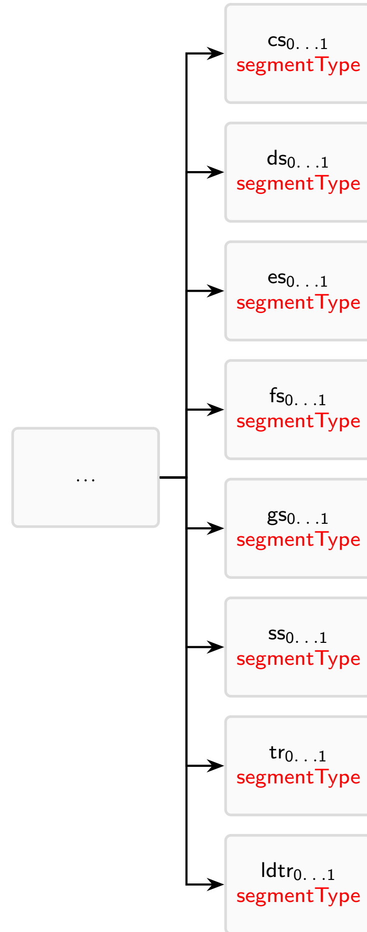
Structure



7.1.90 segmentsType

The `segments` element specifies the initial values of subject segment registers.

Structure



7.1.91 segmentType

Initial value of a segment register, including hidden part. See Intel SDM Vol. 3A, "3.4.3 Segment Registers" for more details on segment registers.

Attributes

Name	Type	Use
selector	word16Type	required
Segment selector value.		
base	word64Type	required
Segment base address.		
limit	word32Type	required
Segment limit.		
access	word32Type	required
Segment access information.		

7.1.92 word32Type

Base: word64Type < xs:string
32-bit machine word.

Restrictions

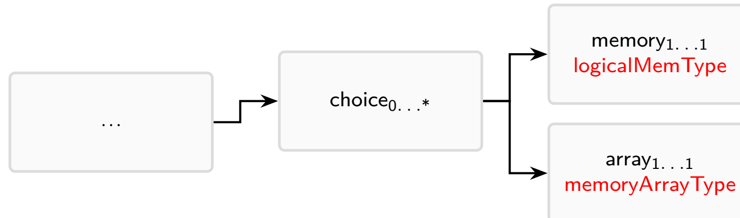
value ≤ 13

7.1.93 logicalMemoryType

In this section, components can specify expected memory mappings with given access rights and region size.

See line 960 in listing 8.1 for an example specification.

Structure



7.1.94 logicalMemType

The `memory` element requests a memory region with the specified `size` and permissions from the system. The region is expected to be placed at the address given via the `virtualAddress` attribute.

See line 965 in listing 8.1 for an example specification.

Attributes

Name	Type	Use
size	word64Type	required
Size of memory in bytes. Must be a multiple of page size (4K).		
virtualAddress	word64Type	required
Expected address of memory mapping.		
logical	nameType	required
Logical name of mapping.		
writable	booleanType	required
Defines if the mapped memory is writable.		
executable	booleanType	required
Defines if the memory region contents are executable by the processor.		

7.1.95 memoryArrayType

The memory array abstraction simplifies the declaration of consecutive memory mappings with a given base address, region size and `executable` and `writable` attributes. The child elements declare the number of expected regions.

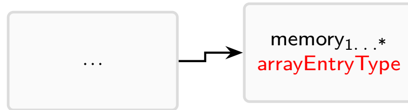
Attributes

Name	Type	Use
logical	nameType	required
Logical name of mapping.		
writable	booleanType	required
Defines if the mapped memory is writable.		
executable	booleanType	required
Defines if the memory region contents are executable by the processor.		
virtualAddressBase	word64Type	required
Expected address of memory mapping.		
elementSize	word64Type	required
Size of one array element in bytes. Must be a multiple of page size (4K).		

(continuation)

Name	Type	Use

Structure



7.1.96 arrayEntryType

Array entries specify the number of array elements and assign a logical name to each element.

See line 1130 in listing 8.1 for an example array entry declaration.

Attributes

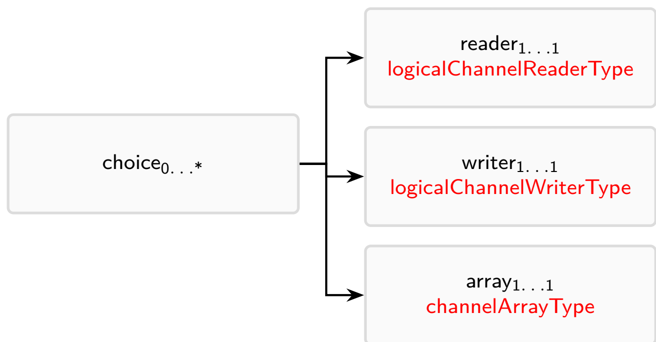
Name	Type	Use
logical	nameType	required
Logical name of array entry.		

7.1.97 logicalChannelsType

Components and libraries use the `channels` sub-section of `requires` to specify expected communication channels.

See line 891 in listing 8.1 for an example specification.

Structure



7.1.98 logicalChannelReaderType

The `reader` element requests a read-only channel of the specified size, address and optional notification vector.

See line 896 in listing 8.1 for an example channel reader specification.

Attributes

Name	Type	Use
logical	nameType	required
Logical name of reader channel.		
virtualAddress	word64Type	required
Expected address of channel memory mapping.		
size	word64Type	required
Expected size of channel. Must be a multiple of page size (4K).		
vector	vectorType	optional
Notification vector.		

7.1.99 vectorType

Base: xs:nonNegativeInteger
Vector number.

Restrictions

value ≤255

7.1.100 logicalChannelWriterType

The `writer` element requests a channel with write permissions of the specified size, address and optional notification event number.

See line 911 in listing 8.1 for an example channel writer specification.

Attributes

Name	Type	Use
logical Logical name of writer channel.	nameType	required
virtualAddress Expected address of channel memory mapping.	word64Type	required
size Expected size of channel. Must be a multiple of page size (4K).	word64Type	required
event Notification event number.	xs:nonNegativeInteger	optional

7.1.101 channelArrayType

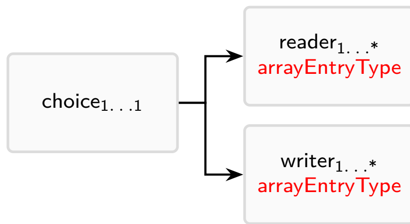
The channel array abstraction simplifies the declaration of consecutive channel mappings with a given base address, channel size and optional event/vector bases. The child elements declare the number of expected channels and either the `reader` or `writer` role.

See line 1123 in listing 8.1 for an example specification.

Attributes

Name	Type	Use
logical Logical channel array name.	nameType	required
eventBase The eventBase attribute specifies the event number of the first element in the array. This number is incremented for all further elements in the array (eventBase + 1). Note that this attribute is only taken into consideration for a writer array.	xs:nonNegativeInteger	optional
vectorBase The vectorBase attribute specifies the vector number of the first element in the array. This number is incremented for all further elements in the array (vectorBase + 1). Note that this attribute is only taken into consideration for a reader array.	vectorType	optional
virtualAddressBase Expected address of memory mapping.	word64Type	required
elementSize Size of one array element in bytes. Must be a multiple of page size (4K).	word64Type	required

Structure

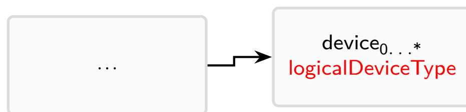


7.1.102 logicalDevicesType

The `devices` sub-section of the `requires` section is used to specify expected devices with their associated resources.

See line 1035 in listing 8.1 for an example specification.

Structure



7.1.103 logicalDeviceType

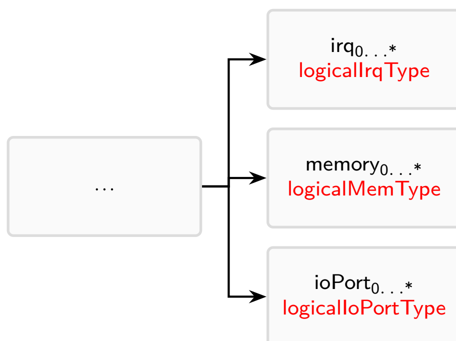
A `device` element specifies an expected logical device with its resources. Possible resources are `irq`, `memory` and `ioPort`.

See line 1040 in listing 8.1 for an example specification.

Attributes

Name	Type	Use
logical	nameType	required
Logical device name.		

Structure



7.1.104 logicalIrqType

An `irq` element of a logical device reference requests an IRQ with given number from the system policy. The specified number will be injected when the device requires attention for the associated logical function.

See line 1045 in listing 8.1 for an example IRQ reference.

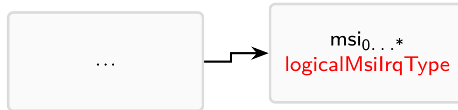
Attributes

Name	Type	Use
logical	nameType	required
Logical name of IRQ resource.		

(continuation)

Name	Type	Use
vector	vectorType	required
Expected IRQ number.		

Structure



7.1.105 logicalMsiIrqType

The presence of `msi` child elements of an `irq` device resource specifies that the component expects the device to be operated in MSI mode. The number of elements defines the expected MSI vector number count to be provided by the referenced device.

Attributes

Name	Type	Use
logical	nameType	required
Name of MSI resource.		

7.1.106 logicalIoPortType

The `ioPort` element requests a device I/O port resource with given range `start .. end` from the system.

See line 1054 in listing 8.1 for an example I/O port reference.

Attributes

Name	Type	Use
logical	nameType	required
Logical I/O port name.		
start	word16Type	required
I/O port start address.		
end	word16Type	required
I/O port end address.		

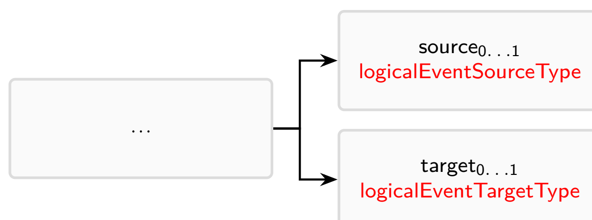
7.1.107 logicalEventsType

The `events` sub-section of the `requires` section is used to specify expected events with optional event actions.

A component can specify both source as well as target events.

See line 1150 in listing 8.1 for an example specification.

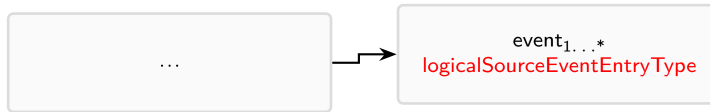
Structure



7.1.108 logicalEventSourceType

Specifies expected source events.

Structure



7.1.109 logicalSourceEventEntryType

Base: baseLogicalEventType

An entry in the component's source event list.

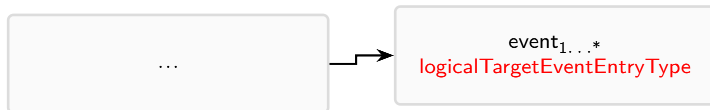
Attributes

Name	Type	Use
logical Logical name of event.	nameType	required
id ID of source event.	xs:nonNegativeInteger	required

7.1.110 logicalEventTargetType

Specifies expected event targets.

Structure



7.1.111 logicalTargetEventEntryType

Base: baseLogicalEventType

An entry in the component's target event list.

Attributes

Name	Type	Use
logical Logical name of event.	nameType	required
id ID of target event entry.	xs:nonNegativeInteger	optional

7.1.112 providedResourcesType

Components usually come in the form of an executable file. To this end, the `provides` section specifies the memory regions of the component binary executable with their content.

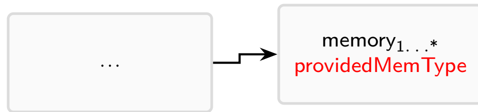
From a security perspective, it is often desirable to provide the different binary section as separate memory regions with the appropriate access rights, i.e. only the text section is executable, rodata is not writable and so on.

Memory specified in this sections are expanded to mapped physical regions for each subject that instantiates this component.

Note: the Mucbinsplit tool can be used to extract these section from an ELF binary into separate files and automatically add the corresponding memory elements to the component specification.

See line 1064 in listing 8.1 for an example `provides` section.

Structure



7.1.113 providedMemType

Base: memoryBaseType

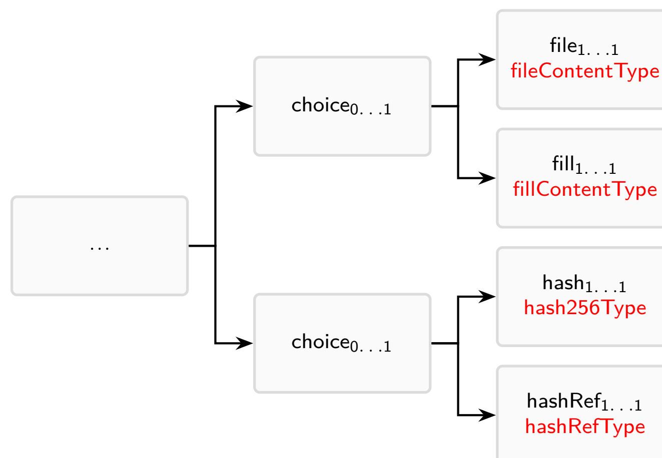
A `memory` element in the `provides` section declares memory region provided by the component. Mostly used to provide (a part) of the component binary.

See line 1082 in listing 8.1 for an example specification.

Attributes

Name	Type	Use
size	memorySizeType	required
Size of region. Must be a multiple of page size (4K). Enforced by validator.		
virtualAddress	word64Type	required
Virtual address in component address space.		
type	subjectMemoryKindType	optional
Memory type (e.g. subject_binary).		
logical	nameType	required
Logical name of mapping.		
writable	booleanType	required
Defines if the mapped memory is writable.		
executable	booleanType	required
Defines if the memory region contents are executable by the processor.		

Structure



7.1.114 componentType

Base: libraryType

A component is a piece of software which shall be executed by the SK. Components represent the building blocks of a component-based system and can be regarded as templates for executable entities instantiated by subjects.

The specification of a component declares the *binary program* by means of (file-backed memory) regions. It also specifies the component's view of the expected execution environment. A component may request the following resources from the system:

- Logical channels
- Logical memory regions

- Logical devices
- Logical events

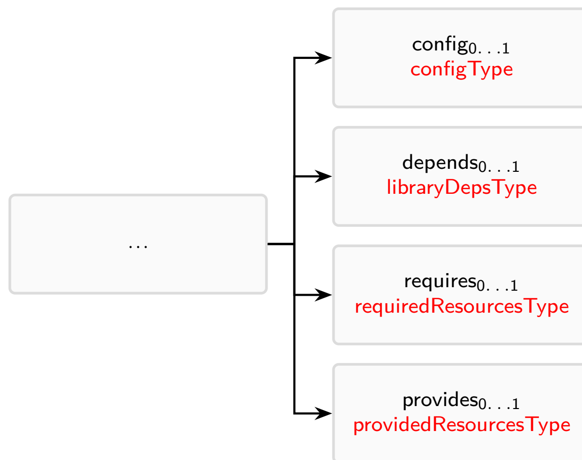
Components are identified by name and specify a profile. The profile controls the settings of the virtual CPU (vCPU).

See line 988 in listing 8.1 for an example component.

Attributes

Name	Type	Use
name Component/library name.	nameType	required
profile Component profile.	componentProfileType	required

Structure



7.1.115 componentProfileType

Base: xs:string

The component profile defines default vCPU settings and triggers profile specific actions in the expander tool. The following actions are performed for the 'linux' profile.

- Add Linux zero-page (ZP, generated by Mugenzp)
- Add ACPI table regions (generated by Mugenacpi)
- Append sinfo address to boot parameters (`muen_sinfo`)
- Add dummy legacy BIOS regions (start address `16#000c_0000#`)
- Invalidate guest state of Linux SMP emulation sibling subjects

Restrictions

values:

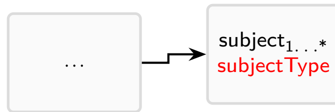
- native
- vm
- linux

7.1.116 subjectsType

The `subjects` element holds a list of subjects.

See line 1768 in listing 8.1 for an example subjects section.

Structure



7.1.117 subjectType

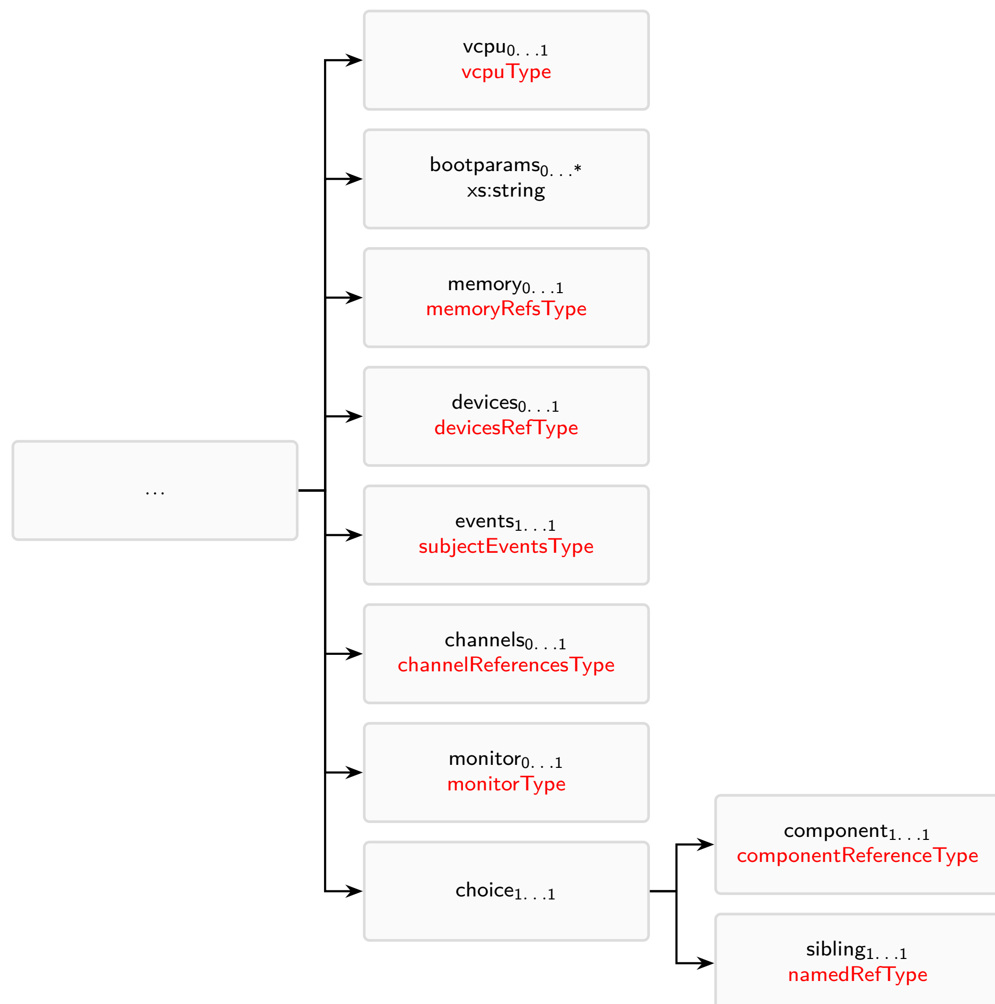
A subject is an instance of a component, i.e. an active component in the system policy that may be scheduled. Its specification references a component and maps all requested logical resources to physical resources provided by the system. Additional resources to the ones requested by the component can be specified here. This enables specialization of the base component specification.

See line 1772 in listing 8.1 for an example subject declaration.

Attributes

Name	Type	Use
name	nameType	required
Unique subject name.		

Structure

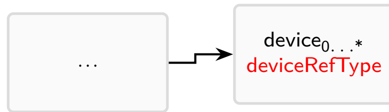


7.1.118 devicesRefType

List of device references. Used to grant a subject access to hardware devices and their resources.

See line 2099 in listing 8.1 for example device references.

Structure



7.1.119 deviceRefType

The `device` element allows a subject access to devices referenced via the `physical` attribute.

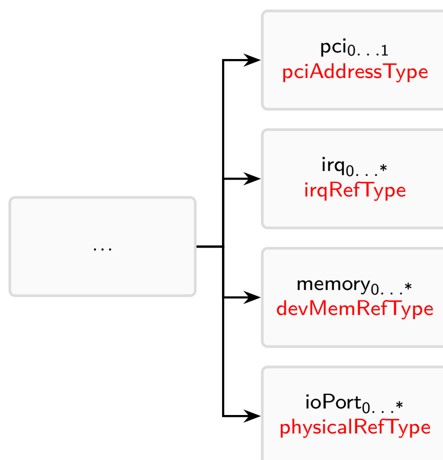
For PCI devices only a single virtual bus is provided (bus 0). The `pci` element may be used to place the device at a specific location (BDF). If no other logical device resources of the device are specified, then the expander tool will map all physical devices resources into the subject. When logical device resources are explicitly specified, then only access to those are actually granted. The physical attribute must be either a reference to an existing physical device, device alias or device class. Validators check that this is the case.

See line 2104 in listing 8.1 for an example reference.

Attributes

Name	Type	Use
logical Logical device name.	nameType	required
physical Name of physical device to reference.	nameType	required

Structure



7.1.120 pciAddressType

PCI Bus, Device, Function triplet (BDF).

Attributes

Name	Type	Use
bus PCI Bus number.	byteType	required
device PCI Device number.	pciDeviceNumberType	required
function PCI Function number.	pciFunctionNumberType	required

7.1.121 irqRefType

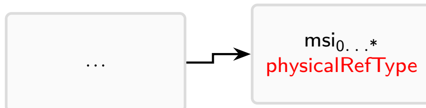
The device `irq` element assigns the referenced physical IRQ to the subject, i.e. if the device triggers the referenced physical IRQ, the specified `vector` number will be injected into the subject by the SK.

The presence of `msi` sub-elements enforces MSI mode (the default for MSI-capable devices and automatic device resource expansion).

Attributes

Name	Type	Use
logical	nameType	required
Logical IRQ name.		
physical	nameType	required
Name of physical device IRQ.		
vector	vectorType	optional
Vector to inject into subject if device triggers IRQ. Will be allocated by the expander if none is specified.		

Structure



7.1.122 physicalRefType

References a physical resource given by the `physical` attribute, and assigns a `logical` name to it.

Attributes

Name	Type	Use
logical	nameType	required
Logical name for resource reference.		
physical	nameType	required
Name of physical resource.		

7.1.123 devMemRefType

The device `memory` element maps the device memory region referenced via the `physical` attribute into the subject address space at address `virtualAddress`. The `executable`, `writable` attributes define the access permissions for the subject.

See line 2119 in listing 8.1 for an example device memory reference.

Attributes

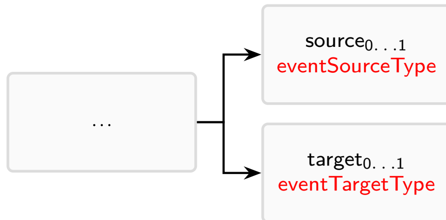
Name	Type	Use
virtualAddress	word64Type	optional
Address of mapping in subject address space. If none is specified, an identity mapping is applied by the expander tool.		
physical	nameType	required
Name of referenced physical memory region.		
logical	nameType	required
Logical name of mapping.		
writable	booleanType	required
Defines if the mapped memory is writable.		
executable	booleanType	required
Defines if the memory region contents are executable by the processor.		

7.1.124 subjectEventsType

The subject `events` element specifies all events originating from or directed at this subject. The physical attribute is the name of a event defined in the global events section.

See line 1781 in listing 8.1 for an example subject events section.

Structure



7.1.125 eventSourceType

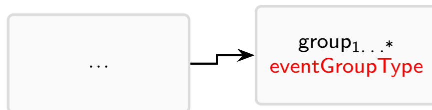
The event `source` element specifies events that are allowed to be triggered by the associated subject.

Source events are divided into two groups: `vmx_exit` and `vmcall`. For event group `vmx_exit` the id attribute specifies the trap number while in the `vmcall` group it designates the hypercall number.

The `vmx_exit` group is translated to a lookup table for handling VMX exit traps as defined by Intel SDM Vol. 3D, "Appendix C VMX Basic Exit Reasons". The `vmcall` group on the other hand is translated into a lookup table to handle hypercalls.

See line 1787 in listing 8.1 for an example event source section.

Structure



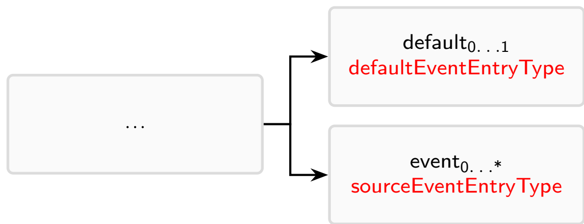
7.1.126 eventGroupType

Source event group element. Currently, two groups are supported: `vmcall` for hypercalls and `vmx_exit` for all other supported traps.

Attributes

Name	Type	Use
name	eventGroupNameType	required
Name of event group.		

Structure



7.1.127 defaultEventEntryType

Base: baseDefaultEventType

The `default` element entry can be used to specify an event which should be added for all event ids that have not been explicitly specified.

See line 1802 in listing 8.1 for a default source event example.

Attributes

Name	Type	Use
physical	nameType	required
Global event reference.		

7.1.128 sourceEventEntryType

Base: baseEventWithIDType < baseEventType

A source `event` entry specifies a source event node, i.e. it registers a handler for the given event `id`. These IDs, depending on the event group, are either hypercall numbers or VMX basic exit reasons.

It is possible to assign event actions to event source entries. Currently supported source event actions are `system_reboot` and `system_poweroff`, which both have the kernel itself as endpoint.

See line 1855 in listing 8.1 for a source event entry example.

Attributes

Name	Type	Use
logical	nameType	required
Logical event name.		
physical	nameType	required
Physical event name.		
id	xs:nonNegativeInteger	required
ID of event.		

7.1.129 eventGroupNameType

Base: xs:string

Supported event groups.

Restrictions

values:

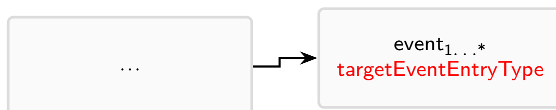
- vmx_exit
- vmcall

7.1.130 eventTargetType

The event `target` element specifies events that the subject is an *endpoint* of.

See line 1870 in listing 8.1 for an example event target section.

Structure



7.1.131 targetEventEntryType

Base: baseEventType

The `event` element in the target section specifies one event endpoint by referencing a physical event and assigning a logical name to it.

See line 1875 in listing 8.1 for an example event endpoint.

Attributes

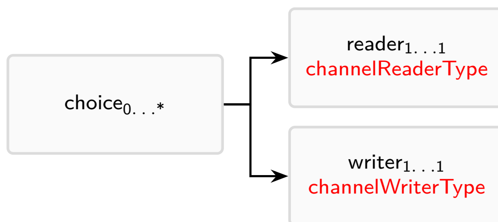
Name	Type	Use
logical	nameType	required
Logical event name.		
physical	nameType	required
Physical event name.		
id	xs:nonNegativeInteger	optional
Event ID.		

7.1.132 channelReferencesType

The `channel` section of a subject declares references to communication channels. The referenced channels become accessible to the requesting subject either as reader or writer endpoint.

See line 2233 in listing 8.1 for an example section.

Structure



7.1.133 channelReaderType

A channel `reader` element references a global communication channel as reader endpoint, i.e. the channel is mapped read-only into the subject address space.

See line 2239 in listing 8.1 for an example reader declaration.

Attributes

Name	Type	Use
logical	nameType	required
Logical name of reader channel.		
physical	nameType	required
Name of physical channel.		
virtualAddress	word64Type	required
Address of mapping in subject address space.		
vector	vectorType	optional
Associated vector. Must be set if a physical channel with hasEvent mode != switch is referenced (enforced by validator).		

7.1.134 channelWriterType

A channel `writer` element references a global communication channel as writer endpoint, i.e. the channel is mapped with write permissions into the subject address space.

See line 2246 in listing 8.1 for an example writer declaration.

Attributes

Name	Type	Use
logical	nameType	required
Logical name of writer channel.		
physical	nameType	required
Name of physical channel.		

(continuation)

Name	Type	Use
virtualAddress	word64Type	required
Address of mapping in subject address space.		
event	xs:nonNegativeInteger	optional
Associated event number. Must be set if a physical channel with hasEvent attribute is referenced.		

7.1.135 monitorType

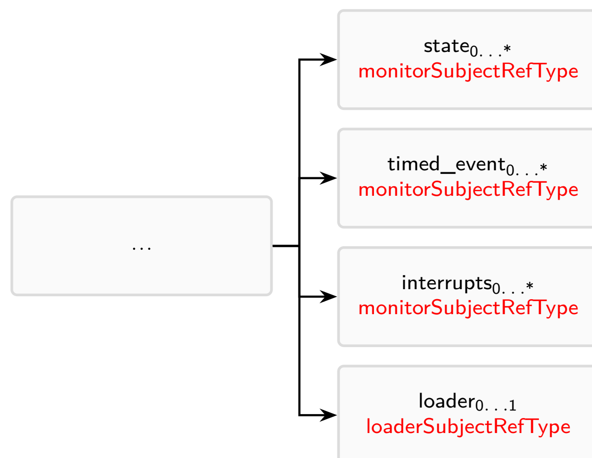
The monitor abstraction enables subjects to request access to certain data of another subject specified by name. Possible child elements are:

- State
- Timed_Events
- Interrupts
- Loader

See the Muen Component Specification document for details about these subject monitor interfaces.

See line 1886 in listing 8.1 for an example monitor section.

Structure



7.1.136 monitorSubjectRefType

Base: loaderSubjectRefType

Give subject monitor (SM) access to the referenced subject state.

Attributes

Name	Type	Use
subject	nameType	required
Name of monitored subject.		
logical	nameType	required
Logical name of state mapping.		
virtualAddress	word64Type	required
Address to map requested subject address space.		
writable	booleanType	required
Whether or not the given state is mapped writable into the SM.		

7.1.137 loaderSubjectRefType

The `loader` mechanism effectively puts the loaded subject denoted by the `subject` attribute under loader control, as it is not able to start without the help of the loader.

In more detail, the `loader` monitor element instructs the expander tool to map all memory regions of the referenced subject into the address space of the monitor subject, using the specified `virtualAddress` as offset in the address space of the loader.

If a memory region of the loaded subject is writable and file-backed, the region is replaced with an empty region and linked via the `hashRef` mechanism to the original region which is mapped into the loader.

The state of the loaded subject is then invalidated by clearing the `CR4.VMXE` bit in the initial subject CR4 register value. If such a subject is scheduled by the kernel, a VMX exit *VM-entry failure due to invalid guest state* (33) occurs. See Intel SDM Vol. 3C, "23.7 Enabling and Entering VMX Operation" and Intel SDM Vol. 3C, "23.8 Restrictions on VMX Operation" for more details. This trap is linked to the loader via normal VMX event handling. After handover, the loader initializes the memory regions replaced by the expander with the designated content.

All information required to *load* the loaded subject is provided to the loader subject via its own sinfo API. Memory regions prefixed with `monitor_sinfo_` provide access to the sinfo regions of the loaded subjects. Regions prefixed with `monitor_state_` specify memory regions containing the subject register state of the loaded subject.

The difference between the `monitor_sinfo_` memory region address in the loader and the address of the `sinfo` memory region in the target sinfo information denotes the `virtualAddress` offset attribute of the `loader` element in the policy. This information combined is enough to fully construct the initial state of the loaded subject, or to reset a subject to its initial state on demand.

The loader may also optionally check the hashes of the restored regions, as this information is provided via the sinfo mechanism as well.

See line 1903 in listing 8.1 for an example loader element.

Attributes

Name	Type	Use
subject	nameType	required
Name of monitored subject.		
logical	nameType	required
Logical name of state mapping.		
virtualAddress	word64Type	required
Address to map requested subject address space.		

7.1.138 componentReferenceType

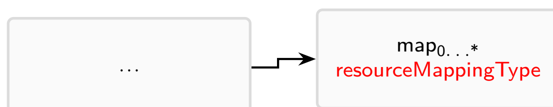
The `component` reference element specifies which component this subject instantiates. All logical resources required by the component must be mapped to physical resources of the appropriate type. Validators make sure that all requirements are satisfied and that no mapping has been omitted.

See line 1812 in listing 8.1 for an example component reference.

Attributes

Name	Type	Use
ref	nameType	required
Name of referenced component.		

Structure



7.1.139 resourceMappingType

The map element maps a physical resource provided by the system with a resource requested by the referenced component.

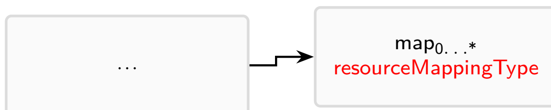
This element allows recursion to map child resources as well (e.g. device memory, I/O ports etc).

See line 1828 in listing 8.1 for an example mapping.

Attributes

Name	Type	Use
logical	nameType	required
Name of logical resource requested by the component.		
physical	nameType	required
Physical name of resource.		

Structure



7.1.140 schedulingType

The Muen SK implements a fixed, cyclic scheduler. The scheduling element is used to specify such a static plan by means of a major frame. A major frame consist of an arbitrary number of minor frames. Minor frames in turn specify a duration in number of ticks a subject is scheduled before preemption in terms of the tick rate.

The tickRate attribute has the unit Hertz (Hz) and specifies the number of clock ticks per second. The ticks attribute of minor frames is expressed in terms of this tick rate. As an example: if we want to declare the minor frame duration in terms of microseconds (10^{-6}) then a tick rate of 1000000 must be used.

The duration of a major frame must be the same on each CPU, meaning the sum of all minor frame ticks for any given CPU must be identical. However, different major frames can have arbitrary length.

The Tau0 subject designates to the kernel which major frame is the currently active one. At the end of each major frame, the kernel determines the active major frame and switches to that scheduling plan for the duration of the major frame.

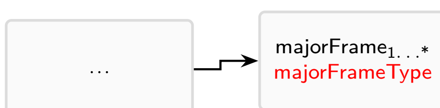
All subjects which can hand over execution to another subject via a switch event form a so called scheduling group. Membership to a scheduling group is determined by the specified switch events and how they link subjects together. Minor frames designate the subject that is to be executed for the given amount of ticks. The subject name identifies the *initial* subject of a minor frame but any member of the scheduling group of the given subject may be executed during that minor frame.

See line 2310 in listing 8.1 for an example scheduling plan.

Attributes

Name	Type	Use
tickRate	xs:positiveInteger	required
Scheduling clock ticks in Hz.		

Structure

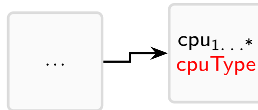


7.1.141 majorFrameType

A major frame consists of a sequence of minor frames for a given CPU. When the end of a major frame is reached, all CPUs synchronize and the scheduler starts over from the beginning using the first minor frame again. This means that major frames are repeated in a cyclic fashion until a different major frame is designated via the Tau0 interface.

See line 2342 in listing 8.1 for an example major frame.

Structure



7.1.142 cpuType

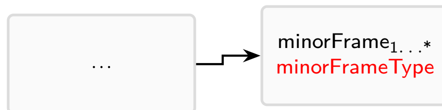
The cpu element is used to specify major frames for each CPU of the system.

See line 2350 in listing 8.1 for an example cpu element.

Attributes

Name	Type	Use
id ID of CPU.	xs:nonNegativeInteger	required

Structure



7.1.143 minorFrameType

A minor frame specifies the number of scheduling ticks a subject is allowed to run on the CPU specified by the parent cpu element.

See line 2355 in listing 8.1 for an example minor frame.

Attributes

Name	Type	Use
subject Name of scheduled subject.	nameType	required
ticks Number of scheduling ticks in minor frame.	xs:positiveInteger	required

Chapter 8

Appendix

8.1 Annotated Example Policy

````xml
<?xml version='1.0' encoding='utf-8'?>
<system>
 <!--
  A Muen system policy specifies all hardware resources such as physical
  memory, devices, CPU time, etc and how these resources are accessed by
  the separation kernel, the subjects and devices.

  The `system` section is the top-level element in the Muen system policy.
  It contains various sub-elements which specify all aspects of a concrete
  system.

  This is the *source format* of the Muen system policy. It allows for
  abstractions, such as channels, which are broken down into their
  constituent parts by the toolchain in format A and B accordingly.
 -->
 <config>
  <!--
   The purpose of a config section is to specify configuration values which
   parameterize a system or a component. It allows to declare boolean,
   string and integer values. The following sections in the system policy
   provide support for configuration values:

   - System

   - Platform

   - Component

   During the build process, configuration values provided by the platform
   are merged into the global system configuration. Component configuration
   values allow the parameterization of component-local functionality.

   Besides component parameterization, configuration options can be used in
   `if` conditionals, as shown in the following example.

   ``` xml
   <if variable="xhcidbg_enabled" value="true">
   ...
   </if>

   ```

   A second use case is XML attribute value expansion as follows:

   ``` xml
   <channel name="debuglog" size="$logchannel_size"/>

   ```

   The `size` attribute value is not specified directly, but parameterized
   via an integer configuration option.
  -->
   <boolean name="xhcidbg_enabled" value="true"/>
   <boolean name="dbgserver_serial_enabled" value="true"/>
   <boolean name="dbgserver_sink_pcspkr" value="false"/>
   <boolean name="dbgserver_sink_shmem" value="false"/>
   <boolean name="linux_debug" value="false"/>
   <boolean name="ahci_drv_enabled" value="false"/>
   <boolean name="pciconf_emulation_enabled" value="true"/>
   <boolean name="pciconf_emulation_xhci_enabled" value="false"/>
   <boolean name="ahci_supported" value="true"/>
````

```
   <boolean name="serial_supported" value="true"/>
   <boolean name="hsuart_supported" value="false"/>
   <boolean name="xhcidbg_supported" value="false"/>
   <boolean name="uefi_gop_rmrr_access" value="false"/>
   <boolean name="ahci_drv_active" value="false"/>
   <boolean name="dbgserver_sink_serial" value="true"/>
   <boolean name="dbgserver_sink_xhcidbg" value="false"/>
   <string name="pciconf_emulation_nic_devid" value="16#01#"/>
   <string name="pciconf_emulation_nic_physdev" value="ethernet_controller_1"/>
   <string name="logchannel_size" value="16#0002_0000#"/>
   <string name="system" value="xml/demo_system_vtd.xml"/>
   <string name="hardware" value="hardware/lenovo-t430s.xml"/>
   <string name="additional_hardware" value="hardware/common_hardware.xml"/>
   <string name="platform" value="platform/lenovo-t430s.xml"/>
   <string name="igd_opregion_address" value="16#baf5_5000#"/>
  </config>
  <hardware>
   <!--
    Systems running the Muen SK perform static resource allocation at
    integration time. This means that all available hardware resources of a
    target machine must be defined in the system policy in order for these
    resources to be allocated to subjects.

    The `hardware` element is the top-level element of the hardware
    specification in the system policy. Information provided by a hardware
    description includes the amount of available physical memory blocks
    including reserved memory regions (RMRR), the number of physical CPU
    cores and hardware device resources.

    The Muen toolchain provides a handy tool to automate the cumbersome
    process of gathering hardware resource data from a running Linux system:
    `mugenhwcfg`\[1\].

    1.  https://git.codelabs.ch/?p=muen/mugenhwcfg.git
   -->
   <processor cpuCores="2" speed="2893" vmxTimerRate="5">
    <!--
     The `processor` element specifies the number of CPU cores, the processor
     speed in MHz and the Intel VMX preemption timer rate.

     Since Intel CPUs can have arbitrary APIC identifiers, the APIC IDs of
     all physical CPUs are enumerated here. The APIC ID is required for
     interrupt and IPI routing.
    -->
    <cpu apicId="0"/>
    <cpu apicId="2"/>
   </processor>
   <memory>
    <!--
     The hardware `memory` element specifies the available physical memory
     blocks including reserved memory regions (RMRR, see Intel VT-d
     Specification, "8.4 Reserved Memory Region Reporting Structure").

     Only memory blocks reported by the BIOS E820 map as non-*reserved* must
     be configured in this section, e.g. *usable* or *ACPI NVS*, *ACPI data*.
    -->
    <memoryBlock allocatable="false" name="System RAM" physicalAddress="16#0000#" size="16#0009_d000#
     "/>
    <memoryBlock allocatable="true" name="System RAM" physicalAddress="16#0010_0000#" size="16#1
     ff0_0000#"/>
    <memoryBlock allocatable="true" name="System RAM" physicalAddress="16#2020_0000#" size="16#1
     fe0_4000#"/>
    <memoryBlock allocatable="true" name="System RAM" physicalAddress="16#4000_5000#" size="16#6
     ed2_c000#"/>
    <memoryBlock allocatable="false" name="ACPI Non-volatile Storage" physicalAddress="16#bae9_f000#"
      size="16#0010_0000#"/>
    <memoryBlock allocatable="false" name="ACPI Tables" physicalAddress="16#baf9_f000#" size="16#0006
     _0000#"/>
    <memoryBlock allocatable="true" name="System RAM" physicalAddress="16#0001_0000_0000#" size="
     16#0003_3e60_0000#"/>
    <reservedMemory name="rmrr1" physicalAddress="16#ba3b_a000#" size="16#0001_7000#">
     <!--
      A `reservedMemory` element is a special memory block declaration. It
      specifies a reserved memory region as outlined in the Intel VT-d
      Specification, "8.4 Reserved Memory Region Reporting Structure" (RMRR).

      Reserved memory regions are BIOS allocated memory ranges that may be DMA
      targets for certain legacy device use-cases. Devices that require access
      to such a region refer to it by name.
     -->
    </reservedMemory>
    <reservedMemory name="rmrr2" physicalAddress="16#bb80_0000#" size="16#0420_0000#"/>
   </memory>
   <devices pciConfigAddress="16#f800_0000#">
    <!--
     The `devices` element enumerates all devices provided by the hardware
     platform. Different kinds of devices, be it PCI(e) or legacy (non-PCI),
```

```
    can be declared in this section.
   -->
   <device name="vga">
    <!--
     The `device` element specifies a physical device and its associated
     resources. There are three main device resource types:

     - IRQ

     - I/O port range

     - Memory

     The presence of a PCI element indicates whether the device is a PCI or a
     legacy device.

     Capabilities can be used to convey additional device-specific
     information. The base address of the memory mapped PCI config space is
     defined by the `pciConfigAddress` attribute.
    -->
    <memory caching="WC" name="buffer" physicalAddress="16#000a_0000#" size="16#0002_0000#">
     <!--
      A device `memory` element specifies a memory region which is used to
      interact with the associated device.

      For PCI devices, the specified region is programmed into one device BAR
      (Base Address Register) by system firmware. See the PCI Local Bus
      Specification or the PCI Express Base Specification for more details.
     -->
    </memory>
    <ioPort end="16#03df#" name="ports" start="16#03c0#">
     <!--
      The `ioPort` element specifies a device I/O port resource from `start`
      octet up to and including `end` octet. A single byte-accessed port is
      designated by specifying the same `start` and `end` values.
     -->
    </ioPort>
   </device>
   <device name="ps2">
    <irq name="kbd_irq" number="1">
     <!--
      The `irq` element specifies a device IRQ resource.

      The specified IRQ number is one of:

      - Legacy IRQ (ISA)ă
      Range `0 .. 15`.

      - PCI INTx IRQ, line-signaledă
      Range `0 .. Max_LSI_IRQ`, whereas `Max_LSI_IRQ` is defined by the
      hardware I/O APIC configuration `gsi_base` + `max_redirection_entry`
      of I/O APIC with `max(gsi_base)`. `gsi_base` and
      `max_redirection_entry` are I/O APIC device capabilities.

      `msi` sub-elements are present if the device supports MSI interrupts.
      The element count designates the number of supported MSI interrrupts.
     -->
    </irq>
    <irq name="mouse_irq" number="12"/>
    <ioPort end="16#0060#" name="port_60" start="16#0060#"/>
    <ioPort end="16#0064#" name="port_64" start="16#0064#"/>
   </device>
   <device name="cmos_rtc">
    <ioPort end="16#0071#" name="ports" start="16#0070#"/>
   </device>
   <device name="pcspeaker">
    <ioPort end="16#0061#" name="Port_61" start="16#0061#"/>
    <ioPort end="16#0043#" name="Port_42_43" start="16#0042#"/>
   </device>
   <device name="system_board">
    <!--
     The system board must provide a reset and pm1a_cnt port as well as
     the pm1a_cnt_slp_typ capability. The presence of this device and
     the necessary resources are checked by the Mucfgvalidate tool.  The
     resources are used by the kernel for system reboot and poweroff.
    -->
    <ioPort end="16#0cf9#" name="reset" start="16#0cf9#"/>
    <ioPort end="16#0404#" name="pm1a_cnt" start="16#0404#"/>
    <capabilities>
     <capability name="systemboard"/>
     <capability name="pm1a_cnt_slp_typ">7168</capability>
    </capabilities>
   </device>
   <device name="ioapic_1">
    <!--
     The I/O Advanced Programmable Interrupt Controller (I/O APIC) is
     used by the kernel for interrupt routing of legacy IRQs.  The
```

```
    presence of this device and the necessary resources are checked by
    the validator tool.
   -->
   <memory caching="UC" name="mem1" physicalAddress="16#fec0_0000#" size="16#1000#"/>
   <capabilities>
    <capability name="ioapic">
     <!--
      A device `capability` is used to assign additional information to a
      device. Such a capability might be used by the Muen toolchain to perform
      certain actions on devices with a given capability (e.g. `ioapic`). A
      system integrator may use this facility to define its own capabilities
      used by custom tools.

      A capability element can have an optional value.
     -->
    </capability>
    <capability name="gsi_base">0</capability>
    <capability name="max_redirection_entry">23</capability>
    <capability name="source_id">16#f0f8#</capability>
   </capabilities>
  </device>
  <device name="iommu_1">
   <!--
    This device specifies an Intel VT-d DMA and interrupt remapping
    hardware. It is used by the Muen SK to implement device separation
    by means of device domains, see below.  The capabilities define
    specific properties of the IOMMU, such as Guest Address Width,
    Fault Register Offset etc. Refer to the Intel VT-d Specification,
    "10.4 Register Descriptions".
   -->
   <memory caching="UC" name="mmio" physicalAddress="16#fed9_0000#" size="16#1000#"/>
   <capabilities>
    <capability name="iommu"/>
    <capability name="agaw">39</capability>
    <capability name="fr_offset">512</capability>
    <capability name="iotlb_invalidate_offset">264</capability>
   </capabilities>
  </device>
  <device name="iommu_2">
   <memory caching="UC" name="mmio" physicalAddress="16#fed9_1000#" size="16#1000#"/>
   <capabilities>
    <capability name="iommu"/>
    <capability name="agaw">39</capability>
    <capability name="fr_offset">512</capability>
    <capability name="iotlb_invalidate_offset">264</capability>
   </capabilities>
  </device>
  <device name="host_bridge_1">
   <description>Intel Corporation 3rd Gen Core processor DRAM Controller</description>
   <pci bus="16#00#" device="16#00#" function="0">
    <!--
     PCI(e) devices are specified using the `pci` element.

     The element provides the following information:

     - PCI device address (BDF)

     - Identification

     - IOMMU group information

     The location of the PCI device in the PCI topology is specified by the
     Bus, Device, Function triplet (BDF).
    -->
    <identification classcode="16#0600#" deviceId="16#0154#" revisionId="16#09#" vendorId="16#8086#
   ">
     <!--
      The `identification` element specifies the PCI device class, device,
      revision and vendor ID.

      For more information, consult the PCI Local Bus Specification,
      "Configuration Space Decoding".
     -->
    </identification>
    <iommuGroup id="0">
     <!--
      Devices in the same IOMMU group cannot be properly isolated from each
      other because they may perform inter-device transactions directly,
      without going through the IOMMU.

      Note that this information is currently not used by the toolchain. It is
      a hint to the system integrator whether two devices can be properly
      isolated from each other or not.
     -->
    </iommuGroup>
   </pci>
   <memory caching="UC" name="mmconf" physicalAddress="16#f800_0000#" size="16#1000#"/>
```

```
    </device>
    <device name="vga_compatible_controller_1">
     <description>Intel Corporation 3rd Gen Core processor Graphics Controller</description>
     <pci bus="16#00#" device="16#02#" function="0">
      <identification classcode="16#0300#" deviceId="16#0166#" revisionId="16#09#" vendorId="16#8086#
"/>
      <iommuGroup id="1"/>
     </pci>
     <irq name="irq1" number="16">
      <msi name="msi1"/>
     </irq>
     <memory caching="UC" name="mem1" physicalAddress="16#d000_0000#" size="16#0040_0000#"/>
     <memory caching="WC" name="mem2" physicalAddress="16#c000_0000#" size="16#1000_0000#"/>
     <memory caching="WC" name="mem3" physicalAddress="16#000c_0000#" size="16#0002_0000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f801_0000#" size="16#1000#"/>
     <ioPort end="16#603f#" name="ioport1" start="16#6000#"/>
     <reservedMemory ref="rmrr2">
      <!--
       This device specifies that it requires access to the reserved
       memory range (RMRR) with the given name.
      -->
     </reservedMemory>
    </device>
    <device name="usb_controller_1">
     <description>Intel Corporation 7 Series/C210 Series Chipset Family USB xHCI Host Controller</
description>
     <pci bus="16#00#" device="16#14#" function="0">
      <identification classcode="16#0c03#" deviceId="16#1e31#" revisionId="16#04#" vendorId="16#8086#
"/>
      <iommuGroup id="2"/>
     </pci>
     <irq name="irq1" number="16">
      <msi name="msi1">
       <!--
        There are two different interrupt types which devices may trigger:
        legacy/PCI LSI IRQs and Message Signaled Interrupts (MSI). The
        legacy/PCI LSI IRQ is specified by the number attribute of the `irq`
        element. For MSIs, each `msi` element defines an MSI IRQ that may be
        assigned to subjects. Each MSI may be individually routed.
       -->
      </msi>
      <msi name="msi2"/>
      <msi name="msi3"/>
      <msi name="msi4"/>
      <msi name="msi5"/>
      <msi name="msi6"/>
      <msi name="msi7"/>
      <msi name="msi8"/>
     </irq>
     <memory caching="UC" name="mem1" physicalAddress="16#d252_0000#" size="16#0001_0000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f80a_0000#" size="16#1000#"/>
     <reservedMemory ref="rmrr1"/>
    </device>
    <device name="communication_controller_1">
     <description>Intel Corporation 7 Series/C216 Chipset Family MEI Controller #1</description>
     <pci bus="16#00#" device="16#16#" function="0">
      <identification classcode="16#0780#" deviceId="16#1e3a#" revisionId="16#04#" vendorId="16#8086#
"/>
      <iommuGroup id="3"/>
     </pci>
     <irq name="irq1" number="16">
      <msi name="msi1"/>
     </irq>
     <memory caching="UC" name="mem1" physicalAddress="16#d253_5000#" size="16#1000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f80b_0000#" size="16#1000#"/>
    </device>
    <device name="serial_controller_1">
     <description>Intel Corporation 7 Series/C210 Series Chipset Family KT Controller</description>
     <pci bus="16#00#" device="16#16#" function="3">
      <identification classcode="16#0700#" deviceId="16#1e3d#" revisionId="16#04#" vendorId="16#8086#
"/>
      <iommuGroup id="3"/>
     </pci>
     <irq name="irq1" number="19">
      <msi name="msi1"/>
     </irq>
     <memory caching="UC" name="mem1" physicalAddress="16#d253_c000#" size="16#1000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f80b_3000#" size="16#1000#"/>
     <ioPort end="16#60b7#" name="ioport1" start="16#60b0#"/>
    </device>
    <device name="ethernet_controller_1">
     <description>Intel Corporation 82579LM Gigabit Network Connection (Lewisville)</description>
     <pci bus="16#00#" device="16#19#" function="0">
      <identification classcode="16#0200#" deviceId="16#1502#" revisionId="16#04#" vendorId="16#8086#
"/>
      <iommuGroup id="4"/>
     </pci>
```

```
   <irq name="irq1" number="20">
    <msi name="msi1"/>
   </irq>
   <memory caching="UC" name="mem1" physicalAddress="16#d250_0000#" size="16#0002_0000#"/>
   <memory caching="UC" name="mem2" physicalAddress="16#d253_b000#" size="16#1000#"/>
   <memory caching="UC" name="mmconf" physicalAddress="16#f80c_8000#" size="16#1000#"/>
   <ioPort end="16#609f#" name="ioport1" start="16#6080#"/>
  </device>
  <device name="usb_controller_2">
   <description>Intel Corporation 7 Series/C216 Chipset Family USB Enhanced Host Controller #2</
   description>
   <pci bus="16#00#" device="16#1a#" function="0">
    <identification classcode="16#0c03#" deviceId="16#1e2d#" revisionId="16#04#" vendorId="16#8086#
   "/>
    <iommuGroup id="5"/>
   </pci>
   <irq name="irq1" number="16"/>
   <memory caching="UC" name="mem1" physicalAddress="16#d253_a000#" size="16#1000#"/>
   <memory caching="UC" name="mmconf" physicalAddress="16#f80d_0000#" size="16#1000#"/>
   <reservedMemory ref="rmrr1"/>
  </device>
  <device name="audio_device_1">
   <description>Intel Corporation 7 Series/C216 Chipset Family High Definition Audio Controller</
   description>
   <pci bus="16#00#" device="16#1b#" function="0">
    <identification classcode="16#0403#" deviceId="16#1e20#" revisionId="16#04#" vendorId="16#8086#
   "/>
    <iommuGroup id="6"/>
   </pci>
   <irq name="irq1" number="22">
    <msi name="msi1"/>
   </irq>
   <memory caching="UC" name="mem1" physicalAddress="16#d253_0000#" size="16#4000#"/>
   <memory caching="UC" name="mmconf" physicalAddress="16#f80d_8000#" size="16#1000#"/>
  </device>
  <device name="usb_controller_3">
   <description>Intel Corporation 7 Series/C216 Chipset Family USB Enhanced Host Controller #1</
   description>
   <pci bus="16#00#" device="16#1d#" function="0">
    <identification classcode="16#0c03#" deviceId="16#1e26#" revisionId="16#04#" vendorId="16#8086#
   "/>
    <iommuGroup id="11"/>
   </pci>
   <irq name="irq1" number="23"/>
   <memory caching="UC" name="mem1" physicalAddress="16#d253_9000#" size="16#1000#"/>
   <memory caching="UC" name="mmconf" physicalAddress="16#f80e_8000#" size="16#1000#"/>
   <reservedMemory ref="rmrr1"/>
  </device>
  <device name="isa_bridge_1">
   <description>Intel Corporation QM77 Express Chipset LPC Controller</description>
   <pci bus="16#00#" device="16#1f#" function="0">
    <identification classcode="16#0601#" deviceId="16#1e55#" revisionId="16#04#" vendorId="16#8086#
   "/>
    <iommuGroup id="12"/>
   </pci>
   <memory caching="UC" name="mmconf" physicalAddress="16#f80f_8000#" size="16#1000#"/>
  </device>
  <device name="sata_controller_1">
   <description>Intel Corporation 7 Series Chipset Family 6-port SATA Controller [AHCI mode]</
   description>
   <pci bus="16#00#" device="16#1f#" function="2">
    <identification classcode="16#0106#" deviceId="16#1e03#" revisionId="16#04#" vendorId="16#8086#
   "/>
    <iommuGroup id="12"/>
   </pci>
   <irq name="irq1" number="19">
    <msi name="msi1"/>
   </irq>
   <memory caching="UC" name="mem1" physicalAddress="16#d253_8000#" size="16#1000#"/>
   <memory caching="UC" name="mmconf" physicalAddress="16#f80f_a000#" size="16#1000#"/>
   <ioPort end="16#60af#" name="ioport1" start="16#60a8#"/>
   <ioPort end="16#60bf#" name="ioport2" start="16#60bc#"/>
   <ioPort end="16#60a7#" name="ioport3" start="16#60a0#"/>
   <ioPort end="16#60bb#" name="ioport4" start="16#60b8#"/>
   <ioPort end="16#607f#" name="ioport5" start="16#6060#"/>
  </device>
  <device name="smbus_1">
   <description>Intel Corporation 7 Series/C216 Chipset Family SMBus Controller</description>
   <pci bus="16#00#" device="16#1f#" function="3">
    <identification classcode="16#0c05#" deviceId="16#1e22#" revisionId="16#04#" vendorId="16#8086#
   "/>
    <iommuGroup id="12"/>
   </pci>
   <irq name="irq1" number="18"/>
   <memory caching="UC" name="mem1" physicalAddress="16#d253_4000#" size="16#1000#"/>
   <memory caching="UC" name="mmconf" physicalAddress="16#f80f_b000#" size="16#1000#"/>
   <ioPort end="16#efbf#" name="ioport1" start="16#efa0#"/>
```

```
    </device>
    <device name="network_controller_1">
     <description>Intel Corporation Centrino Advanced-N 6205 [Taylor Peak]</description>
     <pci bus="16#03#" device="16#00#" function="0">
      <identification classcode="16#0280#" deviceId="16#0085#" revisionId="16#34#" vendorId="16#8086#
     "/>
      <iommuGroup id="13"/>
     </pci>
     <irq name="irq1" number="17">
      <msi name="msi1"/>
     </irq>
     <memory caching="UC" name="mem1" physicalAddress="16#d1c0_0000#" size="16#2000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f830_0000#" size="16#1000#"/>
    </device>
    <device name="system_peripheral_1">
     <description>Ricoh Co Ltd PCIe SDXC/MMC Host Controller</description>
     <pci bus="16#04#" device="16#00#" function="0">
      <identification classcode="16#0880#" deviceId="16#e823#" revisionId="16#04#" vendorId="16#1180#
     "/>
      <iommuGroup id="14"/>
     </pci>
     <irq name="irq1" number="18">
      <msi name="msi1"/>
     </irq>
     <memory caching="UC" name="mem1" physicalAddress="16#d140_0000#" size="16#1000#"/>
     <memory caching="UC" name="mmconf" physicalAddress="16#f840_0000#" size="16#1000#"/>
    </device>
   </devices>
  </hardware>
  <platform>
   <!--
    To enable an uniform view of the hardware resources across different
    physical machines from the system integrators perspective, the platform
    description layer is interposed between the hardware resource
    description and the rest of the system policy. This allows to build a
    Muen system for different physical target machines using the same system
    policy.
   -->
   <mappings>
    <!--
     Platform device alias and class mappings section. Used to assign a
     stable name to a hardware device or to group (multiple) devices under a
     given name.
    -->
    <aliases>
     <!--
      Aliases are a renaming mechanism for physical hardware devices and their
      resources. By using alias names in the system policy references to
      concrete hardware resources can be avoided. Additionally, aliases may be
      used to define a device which only contains a subset of the resources of
      the physical device. This can be achieved by only renaming the resources
      that the device alias should export.
     -->
     <alias name="serial_device_1" physical="serial_controller_1">
      <resource name="ioport1" physical="ioport1"/>
     </alias>
     <alias name="nic_1" physical="ethernet_controller_1">
      <resource name="irq1" physical="irq1">
       <resource name="msi1" physical="msi1"/>
      </resource>
      <resource name="mem1" physical="mem1"/>
      <resource name="mem2" physical="mem2"/>
     </alias>
     <alias name="storage_controller" physical="sata_controller_1"/>
     <alias name="ahci_controller" physical="sata_controller_1">
      <resource name="irq1" physical="irq1">
       <resource name="msi1" physical="msi1"/>
      </resource>
      <resource name="ahci_registers" physical="mem1"/>
      <resource name="mmconf" physical="mmconf"/>
     </alias>
    </aliases>
    <classes>
     <!--
      The `classes' element specifies a list of device classes.
     -->
     <class name="desktop_devices">
      <!--
       Device classes enable the grouping of devices and allow referencing all
       devices by a single name. This simplifies the process of assigning
       multiple devices to a subject.

       Note: A device class may contain an arbitrary number of devices,
       including zero.
      -->
      <device physical="audio_device_1"/>
      <device physical="ethernet_controller_1"/>
```

```
      <device physical="network_controller_1"/>
      <device physical="sata_controller_1"/>
      <device physical="system_peripheral_1"/>
     </class>
     <class name="additional_nics">
      <device physical="network_controller_1"/>
     </class>
    </classes>
   </mappings>
   <kernelDiagnostics type="uart">
    <!--
     The debug build Muen SK can be instructed to output debugging
     information during runtime. The platform diagnostics device specifies
     which device the kernel is to use for this purpose.

     The presence of this device and the necessary resources are checked by
     the validator tool.
    -->
    <device physical="serial_controller_1">
     <ioPort physical="ioport1"/>
    </device>
   </kernelDiagnostics>
  </platform>
  <memory>
   <!--
    This section declares all physical memory regions (RAM) and thus the
    physical memory layout of the system. Regions declared in this section
    can be assigned to subjects and device domains.

    Memory regions are defined by the following attributes:

    - Name

    - Caching type

    - Size

    - Physical address\*

    - Alignment\*

    - Memory type\*

    Attributes with an asterisk are optional. While alignment and memory
    type are set to a default value if not specified, the physical address
    is filled in by the allocator tool, which allocates all memory regions
    and finalizes the physical memory layout.

    Additionally, the content of a region can be declared as backed by a
    file or filled with a pattern.

    Note: The caching type is an attribute of the physical memory region by
    design to avoid inconsistent typing, even though the Intel Page
    Attribute Table (PAT) mechanism allows to set it for each memory
    mapping, see Intel SDM Vol. 3A, "11.12.4 Programming the PAT".
   -->
   <memory caching="WB" name="control_example" size="16#1000#">
    <fill pattern="16#00#">
     <!--
      The `fill` element designates a memory region which is initialized with
      the given pattern.
     -->
    </fill>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="control_sm_1" size="16#1000#">
    <fill pattern="16#00#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="control_sm_2" size="16#1000#">
    <fill pattern="16#ff#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="control_time" size="16#1000#">
    <fill pattern="16#ff#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="control_linux_1" size="16#1000#">
    <fill pattern="16#ff#"/>
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="status_example" size="16#1000#">
    <hash value="none"/>
   </memory>
   <memory caching="WB" name="status_sm_1" size="16#1000#">
    <hash value="none"/>
   </memory>
```

```
645    <memory caching="WB" name="status_sm_2" size="16#1000#">
        <fill pattern="16#00#"/>
647     <hash value="none"/>
       </memory>
649    <memory caching="WB" name="status_time" size="16#1000#">
        <fill pattern="16#00#"/>
651     <hash value="none"/>
       </memory>
653    <memory caching="WB" name="status_linux_1" size="16#1000#">
        <fill pattern="16#00#"/>
655     <hash value="none"/>
       </memory>
657    <memory caching="WB" name="slot_control_1" size="16#1000#">
        <fill pattern="16#00#"/>
659    </memory>
       <memory caching="WB" name="initramfs" size="16#00e0_0000#" type="subject_initrd">
661     <file filename="initramfs.cpio.gz" offset="none">
         <!--
663       The 'file' child element designates a file-backed memory region.

665       The 'filename' attribute specifies the name of the file to use as
          content for the physical memory region, the 'offset' attribute is 'none'
667       by default but can be customized to include a partial file.
         -->
669     </file>
       </memory>
671    <memory caching="WB" name="nic_linux|ram" size="16#1000_0000#"/>
       <memory caching="WB" name="nic_linux|lowmem" size="16#0008_0000#"/>
673    <memory caching="WB" name="storage_linux|ram" size="16#1000_0000#"/>
       <memory caching="WB" name="storage_linux|lowmem" size="16#0008_0000#"/>
675    <memory caching="UC" name="crash_audit" physicalAddress="16#0001_0000_0000#" size="16#1000#" type=
         "subject_crash_audit"/>
      </memory>
677   <deviceDomains>
       <!--
679     The physical memory accessible by PCI devices is specified by so called
        device domains. Such domains define memory mappings of physical memory
681     regions for one or multiple devices. Device references select a subset
        of hardware devices provided by the hardware/platform. Devices may be
683     referenced by device name, alias or device class.

685     Device references can optionally set the 'mapReservedMemory' attribute
        so RMRR regions referenced by the device are also mapped into the device
687     domain.

689     Device domains are isolated from each other by the use of Intel VT-d.
       -->
691    <domain name="nic_domain">
        <memory>
693      <memory executable="false" logical="dma1" physical="nic_linux|lowmem" virtualAddress="16#0002
         _0000#" writable="true">
          <!--
695        A 'memory' element maps a physical memory region into the address space
           of a device domain or subject entity. The region will be accessible to
697        the entity at the specified 'virtualAdress' with permissions defined by
           the 'executable' and 'writable' attributes.
699       -->
         </memory>
701      <memory executable="false" logical="dma2" physical="nic_linux|ram" virtualAddress="16#0100_0000#
         " writable="true"/>
        </memory>
703     <devices>
         <device logical="first_nic" physical="ethernet_controller_1"/>
705      <device logical="additional_nics" physical="additional_nics"/>
        </devices>
707    </domain>
       <domain name="storage_domain">
709     <memory>
         <memory executable="false" logical="dma1" physical="storage_linux|lowmem" virtualAddress="
         16#0002_0000#" writable="true"/>
711      <memory executable="false" logical="dma2" physical="storage_linux|ram" virtualAddress="16#0100
         _0000#" writable="true"/>
        </memory>
713     <devices>
         <device logical="storage_controller" physical="storage_controller"/>
715      <device logical="xhci" physical="usb_controller_1"/>
        </devices>
717    </domain>
      </deviceDomains>
719   <events>
       <!--
721     Events are an activity caused by a subject (source) that impacts a
        second subject (target) or is directed at the kernel. Events are
723     declared globally and have a unique name to be unambiguous. An event
        must have a single source and one target.
725
        Subjects can use events to either deliver an interrupt, hand over
```

```
     execution to or reset the state of a target subject. The first kind of
     event provides a basic notification mechanism and enables the
     implementation of event-driven services. The second type facilitates
     suspension of execution of the source subject and switching to the
     target. Such a construct is used to pass the thread of execution on to a
     different subject, e.g. invocation of a debugger subject if an error
     occurs in the source subject. The third kind is used to faciliate the
     restart of subjects.

     An event can also have the same source and target, which is called
     *self* event. Such events are useful to implement para-virtualized
     timers in VM subjects for example.

     Kernel events are special in that they are targeted at the kernel. The
     currently supported events are system reboot and shutdown.
    -->
    <event mode="switch" name="resume_linux_1">
     <!--
      The `eventType` specifies an event by name and mode.

      The following event modes are currently supported:

      - `asap`ă
      The asap event is an abstraction to state that the event should be
      delivered as soon as possible, depending on the CPU of the target
      subject. If the target runs on another CPU core, this mode is
      expanded to mode *ipi*, which is only available in policy formats A
      and B, instructing the kernel to preempt the kernel running the
      target subject and inject the event immediately. If the target
      subject runs on the same core as the source subject, the mode is
      expanded to mode *async*.

      - `async`ă
      Async events trigger no preemption at the target subject. The event
      is marked as pending in the target subjects pending event table and
      inserted on the next VM exit/entry cycle of the target subject.

      - `self`ă
      An event can also have the same source and target, which is called a
      self event. Such events are useful to implement para-virtualized
      timers in VM subjects for example. A subject sends itself a delayed
      event, using the timed event mechanism. Note that a self event must
      always have a target action assigned, which is checked by the
      validator.

      - `switch`ă
      The switch mode facilitates suspension of execution of the source
      subject and switching to the target. This can only happen between
      subjects running on the same core. Such a construct is used to pass
      the thread of execution on to a different subject, e.g. invocation
      of a debugger subject if an error occurs in the source subject.

      - `kernel`ă
      These kinds of events are directed at the kernel an thus only
      specify a source since the target is the kernel. They are used to
      enable specific subjects to unmask level-triggered IRQs and trigger
      a system reboot, poweroff or explicit panic (crash audit slot
      allocation and reboot).
     -->
    </event>
    <event mode="switch" name="resume_linux_2"/>
    <event mode="switch" name="trap_to_sm_1"/>
    <event mode="switch" name="trap_to_sm_2"/>
    <event mode="switch" name="load_linux_1"/>
    <event mode="switch" name="start_linux_1"/>
    <event mode="switch" name="reset_linux_1"/>
    <event mode="switch" name="reset_linux_2"/>
    <event mode="async" name="reset_sm_1"/>
    <event mode="async" name="reset_slot_1"/>
    <event mode="async" name="serial_irq4_linux_1"/>
    <event mode="async" name="serial_irq4_linux_2"/>
    <event mode="self" name="timer_linux_1"/>
    <event mode="self" name="timer_linux_2"/>
    <event mode="self" name="reboot_linux_1"/>
    <event mode="kernel" name="system_reboot"/>
    <event mode="kernel" name="system_poweroff"/>
    <event mode="kernel" name="system_panic"/>
    <event mode="switch" name="example_yield"/>
    <event mode="self" name="example_self"/>
   </events>
   <channels>
    <!--
     Inter-subject communication is specified by so called channels. These
     channels represent directed information flows since they have a single
     writer and possibly multiple readers. Optionally a channel can have an
     associated notification event (doorbell interrupt).

```

```
   Channels are declared globally and have an unique name to be
   unambiguous.

   Note that channels are a policy source format abstraction. The toolchain
   resolves this concept into memory regions and events as well as the
   appropriate subject mappings.
  -->
  <channel hasEvent="asap" name="input_events" size="16#1000#">
   <!--
    The 'channel' element declares a physical channel.

    Besides the 'name' and 'size' of the channel, the optional 'hasEvent'
    attribute can be set to declare that the given channel requests an
    associated event. The expander tool will then automatically create a
    global event of the requested event type.
   -->
  </channel>
  <channel hasEvent="asap" name="virtual_input_1" size="16#1000#"/>
  <channel hasEvent="asap" name="virtual_input_2" size="16#1000#"/>
  <channel hasEvent="asap" name="virtual_console_1" size="16#0001_0000#"/>
  <channel hasEvent="asap" name="virtual_console_2" size="16#0001_0000#"/>
  <channel name="time_info" size="16#1000#"/>
  <channel name="debuglog_subject1" size="16#0002_0000#"/>
  <channel name="debuglog_subject2" size="16#0002_0000#"/>
  <channel name="debuglog_subject3" size="16#0002_0000#"/>
  <channel name="debuglog_subject4" size="16#0002_0000#"/>
  <channel name="debuglog_subject5" size="16#0002_0000#"/>
  <channel name="debuglog_subject6" size="16#0002_0000#"/>
  <channel name="debuglog_subject7" size="16#0002_0000#"/>
  <channel hasEvent="switch" name="nic_dm_request" size="16#1000#"/>
  <channel hasEvent="switch" name="nic_dm_response" size="16#1000#"/>
  <channel hasEvent="switch" name="storage_dm_request" size="16#1000#"/>
  <channel hasEvent="switch" name="storage_dm_response" size="16#1000#"/>
  <channel name="debuglog_controller" size="16#0002_0000#"/>
  <channel name="testchannel_1" size="16#1000#"/>
  <channel name="testchannel_2" size="16#1000#"/>
  <channel name="testchannel_3" size="16#0010_0000#"/>
  <channel name="testchannel_4" size="16#0010_0000#"/>
  <channel name="debuglog_example" size="16#0002_0000#"/>
  <channel hasEvent="switch" name="example_request" size="16#1000#"/>
  <channel hasEvent="switch" name="example_response" size="16#1000#"/>
 </channels>
 <components>
  <!--
   The 'components' element holds a list of components and component
   libraries.

   Note that components are a policy source format abstraction. The
   toolchain resolves this concept into subjects by adding the appropriate
   memory regions, events and devices.
  -->
  <library name="libmutime">
   <!--
    A component library is a specialized component specification which is
    used to share common resources required for library code to operate.
    Component libraries can be included by multiple components in order to
    share functionality. An example is a logging service provided by a
    dedicated component, whereas the logging client is provided as a library
    with a shared memory channel for the actual log messages.

    A component specification declares library dependencies to request the
    library resources from the system through the inclusion of the library
    specification in the 'depends' section. This way components inherit the
    resources of libraries.

    On the source code level, a library is included by mechanisms provided
    by the respective programming language. Note that the component library
    code is *not* shared between components but lives in the isolated
    execution environment of a subject instantiating the component (i.e.
    statically linked libraries).

    Libraries can request the same resources as ordinary components. A
    subject instantiating the component must also map the resources
    requested by libraries the component depends on.
   -->
   <requires>
    <channels>
     <!--
      Components and libraries use the 'channels' sub-section of 'requires' to
      specify expected communication channels.
     -->
     <reader logical="time_info" size="16#1000#" virtualAddress="16#000f_ffd0_0000#">
      <!--
       The 'reader' element requests a read-only channel of the specified size,
       address and optional notification vector.
      -->
     </reader>
```

```
901   </channels>
     </requires>
903 </library>
    <library name="libmudebuglog">
905  <config>
      <string name="logchannel_size" value="16#0002_0000#"/>
907  </config>
     <requires>
909   <channels>
       <writer logical="debuglog" size="16#0002_0000#" virtualAddress="16#000f_fff0_0000#">
911     <!--
         The `writer` element requests a channel with write permissions of the
913      specified size, address and optional notification event number.
        -->
915    </writer>
      </channels>
917  </requires>
    </library>
919 <library name="libmudm">
     <requires>
921   <channels>
       <writer event="8" logical="dm_pciconf_req" size="16#1000#" virtualAddress="16#2000_0000#"/>
923    <reader logical="dm_pciconf_res" size="16#1000#" virtualAddress="16#2000_1000#"/>
      </channels>
925  </requires>
    </library>
927 <library name="libmuinit">
     <depends>
929   <library ref="libmucontrol"/>
     </depends>
931 </library>
    <library name="muinit">
933  <depends>
      <library ref="libmuinit"/>
935  </depends>
     <requires>
937   <vcpu>
       <!--
939     The `vcpu` element controls the execution behavior of the virtual CPU
        (vCPU). A default vCPU profile is selected by the component profile, but
941     CPU execution settings can be customized both at component and subject
        level.
943    -->
       <registers>
945     <gpr>
         <rip>16#0010_0000#</rip>
947     </gpr>
       </registers>
949   </vcpu>
     </requires>
951  <provides>
      <memory executable="true" logical="muinit" size="16#9000#" type="subject_binary" virtualAddress=
      "16#0010_0000#" writable="false">
953    <file filename="muinit" offset="none"/>
      </memory>
955  </provides>
    </library>
957 <library name="libxhcidbg">
     <requires>
959   <memory>
       <!--
961     In this section, components can specify expected memory mappings with
        given access rights and region size.
963    -->
       <memory executable="false" logical="xhci_dma" size="16#0004_1000#" virtualAddress="16#0100_0000
      #" writable="true">
965     <!--
         The `memory` element requests a memory region with the specified `size`
967      and permissions from the system. The region is expected to be placed at
         the address given via the `virtualAddress` attribute.
969     -->
       </memory>
971   </memory>
      <devices>
973    <device logical="xhci">
        <memory executable="false" logical="xhci_registers" size="16#0001_0000#" virtualAddress="16#
      e000_0000#" writable="true"/>
975    </device>
      </devices>
977  </requires>
    </library>
979 <library name="libmucontrol">
     <requires>
981   <memory>
       <memory executable="false" logical="control" size="16#1000#" virtualAddress="16#000f_ffff_3000#
      " writable="false"/>
```

```
983       <memory executable="false" logical="status" size="16#1000#" virtualAddress="16#000f_ffff_2000#"
           writable="true"/>
        </memory>
985     </requires>
       </library>
987    <component name="ps2_drv" profile="native">
        <!--
989      A component is a piece of software which shall be executed by the SK.
         Components represent the building blocks of a component-based system and
991      can be regarded as templates for executable entities instantiated by
         subjects.
993
         The specification of a component declares the *binary program* by means
995      of (file-backed memory) regions. It also specifies the components view
         of the expected execution environment. A component may request the
997      following resources from the system:

999      - Logical channels

1001     - Logical memory regions

1003     - Logical devices

1005     - Logical events

1007     Components are identified by name and specify a profile. The profile
         controls the settings of the virtual CPU (vCPU).
1009    -->
        <depends>
1011     <!--
          Components and libraries are allowed to declare dependencies to other
1013      libraries. All resources required by the included library are merged
          with the ones specified by the component or library. Libraries can
1015      depend on other libraries.

1017      A subject realizing this component must correctly map all component and
          library resource requirements to physical resources in order to fulfill
1019      the expectations.
         -->
1021     <library ref="libmudebuglog"/>
        </depends>
1023    <requires>
         <vcpu>
1025      <registers>
           <gpr>
1027        <rip>16#0020_0000#</rip>
           </gpr>
1029      </registers>
         </vcpu>
1031     <channels>
          <writer event="1" logical="input_events" size="16#1000#" virtualAddress="16#0005_0000#"/>
1033     </channels>
         <devices>
1035      <!--
           The `devices` sub-section of the `requires` section is used to specify
1037       expected devices with their associated resources.
          -->
1039      <device logical="ps2">
           <!--
1041        A `device` element specifies an expected logical device with its
            resources. Possible resources are `irq`, `memory` and `ioPort`.
1043       -->
           <irq logical="kbd_irq" vector="49">
1045        <!--
             An `irq` element of a logical device reference requests an IRQ with
1047         given number from the system policy. The specified number will be
             injected when the device requires attention for the associated logical
1049         function.
            -->
1051       </irq>
           <irq logical="mouse_irq" vector="60"/>
1053       <ioPort end="16#0060#" logical="port_60" start="16#0060#">
            <!--
1055         The `ioPort` element requests a device I/O port resource with given
             range `start .. end` from the system.
1057        -->
           </ioPort>
1059       <ioPort end="16#0064#" logical="port_64" start="16#0064#"/>
          </device>
1061     </devices>
        </requires>
1063    <provides>
         <!--
1065      Components usually come in the form of an executable file. To this end,
          the `provides` section specifies the memory regions of the component
1067      binary executable with their content.
```

```
1069     From a security perspective, it is often desirable to provide the
         different binary section as separate memory regions with the appropriate
1071     access rights, i.e. only the text section is executable, rodata is not
         writable and so on.
1073
         Memory specified in this sections are expanded to mapped physical
1075     regions for each subject that instantiates this component.

1077     Note: the Mucbinsplit tool can be used to extract these section from an
         ELF binary into separate files and automatically add the corresponding
1079     memory elements to the component specification.
        -->
1081    <memory executable="true" logical="text" size="16#3000#" type="subject_binary" virtualAddress="
        16#0020_0000#" writable="false">
         <!--
1083      A 'memory' element in the 'provides' section declares memory region
          provided by the component. Mostly used to provide (a part) of the
1085      component binary.
         -->
1087     <file filename="ps2_drv_text" offset="none"/>
         <hash value="16#844431db86ff4090e65de03f3dc5c0f7f03f5905a227aa6636a688b7e07bf872#"/>
1089    </memory>
        <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary" virtualAddress
        ="16#0020_5000#" writable="false">
1091     <file filename="ps2_drv_rodata" offset="none"/>
         <hash value="16#232bd0e2a4f9b4f51ebe60d6617513b5db259105824a15d4e42ef5a63561f307#"/>
1093    </memory>
        <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress="
        16#0020_6000#" writable="true">
1095     <file filename="ps2_drv_data" offset="none"/>
         <hash value="16#7bc0583376e16a39fa960b38b9c2467bb4130639f85da34b52dfe34e0453ae65#"/>
1097    </memory>
        <memory executable="false" logical="bss" size="16#2000#" type="subject_binary" virtualAddress="
        16#0020_3000#" writable="true">
1099     <fill pattern="16#00#"/>
        </memory>
1101    <memory executable="false" logical="stack" size="16#4000#" type="subject_binary" virtualAddress=
        "16#1000#" writable="true">
         <fill pattern="16#00#"/>
1103    </memory>
       </provides>
1105  </component>
      <component name="vt" profile="native">
1107   <depends>
        <library ref="libmudebuglog"/>
1109   </depends>
       <requires>
1111    <vcpu>
         <registers>
1113      <gpr>
           <rip>16#0020_0000#</rip>
1115      </gpr>
         </registers>
1117    </vcpu>
        <memory>
1119     <memory executable="false" logical="slot_control_1" size="16#1000#" virtualAddress="16#0007
        _0000#" writable="true"/>
        </memory>
1121    <channels>
         <array elementSize="16#0001_0000#" logical="console" vectorBase="34" virtualAddressBase="
        16#0010_0000#">
1123      <!--
           The channel array abstraction simplifies the declaration of consecutive
1125       channel mappings with a given base address, channel size and optional
           event/vector bases. The child elements declare the number of expected
1127       channels and either the 'reader' or 'writer' role.
          -->
1129      <reader logical="NIC Linux">
           <!--
1131        Array entries specify the number of array elements and assign a logical
            name to each element.
1133       -->
          </reader>
1135      <reader logical="Storage Linux"/>
         </array>
1137     <array elementSize="16#1000#" eventBase="1" logical="input_devices" virtualAddressBase="16#0005
        _0000#">
          <writer logical="input_device_1"/>
1139      <writer logical="input_device_2"/>
         </array>
1141     <reader logical="input_events" size="16#1000#" vector="48" virtualAddress="16#0006_0000#"/>
        </channels>
1143    <devices>
         <device logical="vga">
1145      <memory executable="false" logical="buffer" size="16#0002_0000#" virtualAddress="16#000a_0000#
        " writable="true"/>
          <ioPort end="16#03df#" logical="ports" start="16#03c0#"/>
```

```
      </device>
     </devices>
     <events>
      <!--
       The `events` sub-section of the `requires` section is used to specify
       expected events with optional event actions.

       A component can specify both source as well as target events.
      -->
      <source>
       <event id="30" logical="shutdown">
        <system_poweroff>
         <!--
          An example of a source event action directed at the kernel. If this
          event is triggered by the associated subject, the system will power
          off.
         -->
        </system_poweroff>
       </event>
       <event id="31" logical="reboot">
        <system_reboot/>
       </event>
      </source>
     </events>
    </requires>
    <provides>
     <memory executable="true" logical="text" size="16#a000#" type="subject_binary" virtualAddress="
     16#0020_0000#" writable="false">
      <file filename="vt_text" offset="none"/>
      <hash value="16#49737e6dbd00e6e3e6bfae08b4141b9d0013f6bfd7d24d76a1d59e0098858ee4#"/>
     </memory>
     <memory executable="false" logical="rodata" size="16#4000#" type="subject_binary" virtualAddress
     ="16#0020_c000#" writable="false">
      <file filename="vt_rodata" offset="none"/>
      <hash value="16#4b1afd922d1241c304d116368dc62e4a45073e231135fd0d2f182879ced11d4b#"/>
     </memory>
     <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress="
     16#0021_0000#" writable="true">
      <file filename="vt_data" offset="none"/>
      <hash value="16#e522c0bf5552ca79bf7c140ea79b4b4c757fa9274e30685707f1384453f55e55#"/>
     </memory>
     <memory executable="false" logical="bss" size="16#2000#" type="subject_binary" virtualAddress="
     16#0020_a000#" writable="true">
      <fill pattern="16#00#"/>
     </memory>
     <memory executable="false" logical="stack" size="16#4000#" type="subject_binary" virtualAddress=
     "16#1000#" writable="true">
      <fill pattern="16#00#"/>
     </memory>
    </provides>
   </component>
   <component name="isolation_tests_monitor" profile="native">
    <requires>
     <vcpu>
      <registers>
       <gpr>
        <rip>16#0020_0000#</rip>
       </gpr>
      </registers>
     </vcpu>
     <memory>
      <memory executable="false" logical="result_state" size="16#1000#" virtualAddress="16#0100_0000#
     " writable="true"/>
     </memory>
     <events>
      <source>
       <event id="1" logical="resume_tests"/>
      </source>
      <target>
       <event logical="trap_to_monitor"/>
      </target>
     </events>
    </requires>
    <provides>
     <memory executable="true" logical="text" size="16#1000#" type="subject_binary" virtualAddress="
     16#0020_0000#" writable="false">
      <file filename="isolation_tests_monitor_text" offset="none"/>
      <hash value="16#c23ef1624f62adf30405412be62beb5f4c161f861e8dd506ccd120c8296f3240#"/>
     </memory>
     <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary" virtualAddress
     ="16#0020_1000#" writable="false">
      <file filename="isolation_tests_monitor_rodata" offset="none"/>
      <hash value="16#dd605fb441991ff03476a906d81d12420eeeb511a501874bdcda0a761a311657#"/>
     </memory>
     <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress="
     16#0020_2000#" writable="true">
      <file filename="isolation_tests_monitor_data" offset="none"/>
```

```
      <hash value="16#ad7facb2586fc6e966c004d7d1d16b024f5805ff7cb47c7a85dabd8b48892ca7#"/>
     </memory>
     <memory executable="false" logical="stack" size="16#4000#" type="subject_binary" virtualAddress=
     "16#1000#" writable="true">
      <fill pattern="16#00#"/>
     </memory>
    </provides>
   </component>
   <component name="linux" profile="linux">
    <requires>
     <memory>
      <memory executable="true" logical="lowmem" size="16#0008_0000#" virtualAddress="16#0002_0000#"
     writable="true"/>
      <memory executable="true" logical="ram" size="16#1000_0000#" virtualAddress="16#0100_0000#"
     writable="true"/>
     </memory>
    </requires>
    <provides>
     <memory executable="true" logical="binary" size="16#005a_8000#" type="subject_binary"
     virtualAddress="16#0040_0000#" writable="true">
      <file filename="bzImage" offset="none"/>
     </memory>
     <memory executable="false" logical="modules_initramfs" size="16#0001_5000#" type="subject_initrd
     " virtualAddress="16#90e0_0000#" writable="false">
      <file filename="modules_initramfs.cpio.gz" offset="none"/>
     </memory>
    </provides>
   </component>
   <component name="isolation_tests" profile="native">
    <config>
     <!--
      Components may declare their own configuration values in the config
      section. Just like global system config values, these can also be
      used in `<if>` expressions and XML attribute value expansion.
     -->
       <integer name="log_entry_max" value="128"/>
       <integer name="log_buffer_size" value="65535"/>
      </config>
      <depends>
       <library ref="libmudebuglog"/>
      </depends>
      <requires>
       <vcpu>
        <msrs>
         <msr end="16#0174#" mode="r" start="16#0174#"/>
        </msrs>
        <registers>
         <gpr>
          <rip>16#0020_0000#</rip>
         </gpr>
        </registers>
       </vcpu>
       <memory>
        <memory executable="false" logical="read_only" size="16#1000#" virtualAddress="16#1000_0000#"
      writable="false"/>
        <memory executable="false" logical="result_state" size="16#1000#" virtualAddress="16#0100
     _0000#" writable="true"/>
       </memory>
       <events>
        <target>
         <event logical="resume_tests"/>
        </target>
       </events>
      </requires>
      <provides>
       <memory executable="true" logical="text" size="16#7000#" type="subject_binary" virtualAddress=
     "16#0020_0000#" writable="false">
        <file filename="isolation_tests_text" offset="none"/>
        <hash value="16#3c90a6ca9e1e6f8e3c9138ace1e474efe523532cead4c06615262b1fa5b11ce4#"/>
       </memory>
       <memory executable="false" logical="rodata" size="16#2000#" type="subject_binary"
     virtualAddress="16#0021_8000#" writable="false">
        <file filename="isolation_tests_rodata" offset="none"/>
        <hash value="16#8d2c91c6b63eb8a5bcba54e839b115759669302baf58962ae29583305653c975#"/>
       </memory>
       <memory executable="false" logical="data" size="16#b000#" type="subject_binary" virtualAddress
     ="16#0021_a000#" writable="true">
        <file filename="isolation_tests_data" offset="none"/>
        <hash value="16#731b8abc3033235953e353ac5e0d68f1611e9739edb0e1ef274a0403ce883bef#"/>
       </memory>
       <memory executable="false" logical="bss" size="16#0001_1000#" type="subject_binary"
     virtualAddress="16#0020_7000#" writable="true">
        <fill pattern="16#00#"/>
       </memory>
       <memory executable="false" logical="stack" size="16#4000#" type="subject_binary"
     virtualAddress="16#1000#" writable="true">
        <fill pattern="16#00#"/>
```

```
      </memory>
     </provides>
    </component>
    <component name="dbgserver" profile="native">
     <config>
      <boolean name="default_channel_enabled_state" value="true"/>
      <boolean name="sink_pcspkr" value="false"/>
      <boolean name="sink_shmem" value="false"/>
      <boolean name="sink_serial" value="true"/>
      <boolean name="sink_xhcidbg" value="false"/>
      <boolean name="hsuart_enabled" value="false"/>
      <string name="debugconsole_port_start" value="16#60b0#"/>
      <string name="debugconsole_port_end" value="16#60b7#"/>
      <string name="logchannel_size" value="16#0002_0000#"/>
       <!-- CSV list of channel names that are enabled irrespective of the default. -->
      <string name="enabled_channels_override" value=""/>
     </config>
     <requires>
      <vcpu>
       <registers>
        <gpr>
         <rip>16#0020_0000#</rip>
        </gpr>
       </registers>
      </vcpu>
      <memory>
       <memory executable="false" logical="crash_audit" size="16#1000#" virtualAddress="16#1000_0000
#" writable="false"/>
      </memory>
      <channels>
       <array elementSize="16#0002_0000#" logical="log_channels" virtualAddressBase="16#a000_0000#">
        <reader logical="log_channel1"/>
        <reader logical="log_channel2"/>
        <reader logical="log_channel3"/>
        <reader logical="log_channel4"/>
        <reader logical="log_channel5"/>
        <reader logical="log_channel_example"/>
        <reader logical="log_channel_6"/>
        <reader logical="log_channel7"/>
        <reader logical="log_channel8"/>
       </array>
      </channels>
      <devices>
       <device logical="debugconsole">
        <ioPort end="16#60b7#" logical="port" start="16#60b0#"/>
       </device>
      </devices>
      <events>
       <source>
        <event id="30" logical="shutdown">
         <system_poweroff/>
        </event>
        <event id="31" logical="reboot">
         <system_reboot/>
        </event>
       </source>
      </events>
     </requires>
     <provides>
      <memory executable="true" logical="text" size="16#a000#" type="subject_binary" virtualAddress=
"16#0020_0000#" writable="false">
       <file filename="dbgserver_text" offset="none"/>
       <hash value="16#60c26b68a7e0192e970189a64191630690458a901d81660e6b71a38dab547a5b#"/>
      </memory>
      <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
virtualAddress="16#0021_8000#" writable="false">
       <file filename="dbgserver_rodata" offset="none"/>
       <hash value="16#f5ad233b4e8da945aa5fc73c256f2a5bfb904c1b6b6d4b6dd277052cd0513c62#"/>
      </memory>
      <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress
="16#0021_9000#" writable="true">
       <file filename="dbgserver_data" offset="none"/>
       <hash value="16#0f34e786f747c7d50777202a76bf0015016d64ca7dc2583a68aa301e32d0b632#"/>
      </memory>
      <memory executable="false" logical="bss" size="16#e000#" type="subject_binary" virtualAddress=
"16#0020_a000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
      <memory executable="false" logical="stack" size="16#4000#" type="subject_binary"
virtualAddress="16#1000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
     </provides>
    </component>
    <component name="ahci_drv" profile="native">
     <depends>
      <library ref="libmudebuglog"/>
```

```
1381      </depends>
          <requires>
1383       <vcpu>
            <registers>
1385         <gpr>
              <rip>16#0020_0000#</rip>
1387         </gpr>
            </registers>
1389       </vcpu>
           <memory>
1391        <!-- for 32 ports 16#c000# bytes are needed for descriptor tables + 16K for device init -->
            <memory executable="false" logical="dma_region" size="16#0001_0000#" virtualAddress="16#
        a000_0000#" writable="true"/>
1393        <array elementSize="16#0100_0000#" executable="false" logical="blockdev_shm"
        virtualAddressBase="16#a100_0000#" writable="true">
             <memory logical="blockdev_shm1"/>
1395         <memory logical="blockdev_shm2"/>
            </array>
1397       </memory>
           <channels>
1399        <array elementSize="16#0000_8000#" logical="blockdev_request" vectorBase="50"
        virtualAddressBase="16#0200_0000#">
             <reader logical="blockdev_request1"/>
1401         <reader logical="blockdev_request2"/>
            </array>
1403        <array elementSize="16#0000_4000#" eventBase="40" logical="blockdev_response"
        virtualAddressBase="16#0300_0000#">
             <writer logical="blockdev_response1"/>
1405         <writer logical="blockdev_response2"/>
            </array>
1407       </channels>
           <devices>
1409        <device logical="ahci_controller">
             <irq logical="irq" vector="48"/>
1411         <memory executable="false" logical="ahci_registers" size="16#1000#" virtualAddress="16#
        e000_0000#" writable="true"/>
             <memory executable="false" logical="mmconf" size="16#1000#" virtualAddress="16#f800_8000#"
        writable="true"/>
1413        </device>
           </devices>
1415      </requires>
          <provides>
1417       <memory executable="true" logical="text" size="16#9000#" type="subject_binary" virtualAddress=
        "16#0020_0000#" writable="false">
            <file filename="ahci_drv_text" offset="none"/>
1419        <hash value="16#cb5fe9ea6b738af3b90a1ad0dd189d54da430e86922bf306a7ecb7139c07026d#"/>
           </memory>
1421       <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
        virtualAddress="16#0020_c000#" writable="false">
            <file filename="ahci_drv_rodata" offset="none"/>
1423        <hash value="16#3001ba235caf63cd06d9bedfabe0b80add4be287cc55e1275bdcbbcc3f5edbf8#"/>
           </memory>
1425       <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress
        ="16#0020_d000#" writable="true">
            <file filename="ahci_drv_data" offset="none"/>
1427        <hash value="16#1cb2148aef42e097dad59630d8940dce44f8b351f88fcfac9ded03d0a2a831b1#"/>
           </memory>
1429       <memory executable="false" logical="bss" size="16#3000#" type="subject_binary" virtualAddress=
        "16#0020_9000#" writable="true">
            <fill pattern="16#00#"/>
1431       </memory>
           <memory executable="false" logical="stack" size="16#4000#" type="subject_binary"
        virtualAddress="16#1000#" writable="true">
1433        <fill pattern="16#00#"/>
           </memory>
1435      </provides>
         </component>
1437     <component name="dm" profile="native">
          <depends>
1439       <library ref="libmudebuglog"/>
          </depends>
1441      <requires>
           <vcpu>
1443        <registers>
             <gpr>
1445          <rip>16#0020_0000#</rip>
             </gpr>
1447        </registers>
           </vcpu>
1449       <channels>
            <reader logical="request" size="16#1000#" virtualAddress="16#0010_0000#"/>
1451        <writer event="9" logical="response" size="16#1000#" virtualAddress="16#0010_1000#"/>
           </channels>
1453      </requires>
          <provides>
1455       <memory executable="true" logical="text" size="16#4000#" type="subject_binary" virtualAddress=
        "16#0020_0000#" writable="false">
```

```
            <file filename="dm_text" offset="none"/>
1457        <hash value="16#4c4abfbdaa02fbdda0a93d9379ad2b910671da7b5808aca012664f3d6e3e06ac#"/>
           </memory>
1459       <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
       virtualAddress="16#0020_5000#" writable="false">
            <file filename="dm_rodata" offset="none"/>
1461        <hash value="16#16618d141be7aa0818730c944df3146b185b467f39c1a2163074c4af1d3e8894#"/>
           </memory>
1463       <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress
       ="16#0020_6000#" writable="true">
            <file filename="dm_data" offset="none"/>
1465        <hash value="16#9348d0aec58fa2e80f74bf3a52440ec842aa7e6bf948bcd98d40d1c30dc218ec#"/>
           </memory>
1467       <memory executable="false" logical="bss" size="16#1000#" type="subject_binary" virtualAddress=
       "16#0020_4000#" writable="true">
            <fill pattern="16#00#"/>
1469       </memory>
           <memory executable="false" logical="stack" size="16#4000#" type="subject_binary"
       virtualAddress="16#1000#" writable="true">
1471        <fill pattern="16#00#"/>
           </memory>
1473      </provides>
         </component>
1475     <component name="sm" profile="native">
          <config>
1477       <boolean name="debug_cpuid" value="false"/>
           <boolean name="debug_cr" value="false"/>
1479       <boolean name="debug_ioport" value="false"/>
           <boolean name="debug_ept" value="false"/>
1481       <boolean name="debug_rdmsr" value="false"/>
           <boolean name="debug_wrmsr" value="false"/>
1483       <boolean name="pciconf_emulation_enabled" value="true"/>
          </config>
1485      <depends>
           <library ref="libmutime"/>
1487       <library ref="libmudebuglog"/>
           <library ref="libmudm"/>
1489       <library ref="muinit"/>
          </depends>
1491      <requires>
           <vcpu>
1493        <registers>
             <gpr>
1495          <rip>16#0020_0000#</rip>
             </gpr>
1497        </registers>
           </vcpu>
1499       <events>
            <source>
1501         <event id="4" logical="resume_subject"/>
            </source>
1503        <target>
             <event logical="handle_subject_trap"/>
1505        </target>
           </events>
1507      </requires>
          <provides>
1509       <memory executable="true" logical="text" size="16#6000#" type="subject_binary" virtualAddress=
       "16#0020_0000#" writable="false">
            <file filename="sm_text" offset="none"/>
1511        <hash value="16#b5ae0fc0b027b1f322f86f4687843a625627300ae8f287a3d986e3675359f367#"/>
           </memory>
1513       <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
       virtualAddress="16#0020_8000#" writable="false">
            <file filename="sm_rodata" offset="none"/>
1515        <hash value="16#4fa186779d49d11d2a6b08ae9f22f873d49c4e65fa4954c1a679db350194213a#"/>
           </memory>
1517       <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress
       ="16#0020_9000#" writable="true">
            <file filename="sm_data" offset="none"/>
1519        <hash value="16#b12fca5364fb3bb09206fd164c3102d102293f0cf69efb7996620f73b8e702ba#"/>
           </memory>
1521       <memory executable="false" logical="bss" size="16#2000#" type="subject_binary" virtualAddress=
       "16#0020_6000#" writable="true">
            <fill pattern="16#00#"/>
1523       </memory>
           <memory executable="false" logical="stack" size="16#4000#" type="subject_binary"
       virtualAddress="16#1000#" writable="true">
1525        <fill pattern="16#00#"/>
           </memory>
1527      </provides>
         </component>
1529     <component name="idle" profile="native">
          <requires>
1531       <vcpu>
            <registers>
1533         <gpr>
```

```xml
         <rip>16#0020_0000#</rip>
        </gpr>
       </registers>
      </vcpu>
     </requires>
     <provides>
      <memory executable="true" logical="text" size="16#1000#" type="subject_binary" virtualAddress=
    "16#0020_0000#" writable="false">
       <file filename="idle_text" offset="none"/>
       <hash value="16#324dc6f52b0fdfeef0a0cdbf632725689f1f3f12857a57ad80b240a1e8b21f9b#"/>
      </memory>
      <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
    virtualAddress="16#0020_1000#" writable="false">
       <file filename="idle_rodata" offset="none"/>
       <hash value="16#db97c62b590d580647fe04fcc6c8a962697fa51f0a7ab475a16967e29cbb4cb9#"/>
      </memory>
      <memory executable="false" logical="stack" size="16#4000#" type="subject_binary"
    virtualAddress="16#1000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
     </provides>
    </component>
    <component name="sl" profile="native">
     <depends>
      <library ref="libmuinit"/>
     </depends>
     <requires>
      <vcpu>
       <registers>
        <gpr>
         <rip>16#0020_0000#</rip>
        </gpr>
       </registers>
      </vcpu>
      <events>
       <source>
        <event id="0" logical="start"/>
       </source>
       <target>
        <event logical="handle_reset"/>
       </target>
      </events>
     </requires>
     <provides>
      <memory executable="true" logical="text" size="16#3000#" type="subject_binary" virtualAddress=
    "16#0020_0000#" writable="false">
       <file filename="sl_text" offset="none"/>
       <hash value="16#8c26954ad22a1373e19faae880be9b66726fd4996ac928fff72abd4ff841138f#"/>
      </memory>
      <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
    virtualAddress="16#0020_3000#" writable="false">
       <file filename="sl_rodata" offset="none"/>
       <hash value="16#680bc1a5bea402fb9232ce73620ece595c32696b86c6b9c4f3eb159cbd270e31#"/>
      </memory>
      <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress
    ="16#0020_4000#" writable="true">
       <file filename="sl_data" offset="none"/>
       <hash value="none"/>
      </memory>
      <memory executable="false" logical="stack" size="16#4000#" type="subject_binary"
    virtualAddress="16#1000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
     </provides>
    </component>
    <component name="example" profile="native">
     <config>
      <boolean name="ahci_drv_enabled" value="false"/>
      <boolean name="print_serial" value="false"/>
      <boolean name="print_vcpu_speed" value="true"/>
      <integer name="serial" value="123456789"/>
      <string name="greeter" value="Subject running"/>
     </config>
     <depends>
      <library ref="libmudebuglog"/>
      <library ref="muinit"/>
     </depends>
     <requires>
      <vcpu>
       <registers>
        <gpr>
         <rip>16#0020_0000#</rip>
        </gpr>
       </registers>
      </vcpu>
      <memory>
      </memory>
```

```
      <channels>
       <reader logical="example_request" size="16#1000#" virtualAddress="16#000A_0000#"/>
       <writer event="1" logical="example_response" size="16#1000#" virtualAddress="16#000B_0000#"/>
      </channels>
      <events>
       <source>
        <event id="2" logical="yield"/>
        <event id="3" logical="timer"/>
       </source>
       <target>
        <event logical="inject_timer">
         <inject_interrupt vector="37"/>
        </event>
       </target>
      </events>
     </requires>
     <provides>
      <memory executable="true" logical="text" size="16#3000#" type="subject_binary" virtualAddress=
    "16#0020_0000#" writable="false">
       <file filename="example_text" offset="none"/>
       <hash value="16#53f0e114bf17530a875da0ca5db136a44a94918ff10bcc16d66189841b37c04f#"/>
      </memory>
      <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
    virtualAddress="16#0020_5000#" writable="false">
       <file filename="example_rodata" offset="none"/>
       <hash value="16#4ed69a004cb8f0b5c10ccd1cc622ae409ea86ecd1de9039ec7c741e113f3497d#"/>
      </memory>
      <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress
    ="16#0020_6000#" writable="true">
       <file filename="example_data" offset="none"/>
       <hash value="16#09889515e4a06764057f9cb9a5801fa8a8063442d4708e24a07cd348a834e76b#"/>
      </memory>
      <memory executable="false" logical="bss" size="16#2000#" type="subject_binary" virtualAddress=
    "16#0020_3000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
      <memory executable="false" logical="stack" size="16#4000#" type="subject_binary"
    virtualAddress="16#1000#" writable="true">
       <fill pattern="16#00#"/>
      </memory>
     </provides>
    </component>
    <component name="controller" profile="native">
     <depends>
      <library ref="libmudebuglog"/>
     </depends>
     <requires>
      <vcpu>
       <registers>
        <gpr>
         <rip>16#0020_0000#</rip>
        </gpr>
       </registers>
      </vcpu>
      <memory>
       <array elementSize="16#1000#" executable="false" logical="control" virtualAddressBase="16#
    a000_0000#" writable="true">
        <memory logical="control_1"/>
        <memory logical="control_2"/>
        <memory logical="control_3"/>
        <memory logical="control_4"/>
        <memory logical="control_5"/>
       </array>
       <array elementSize="16#1000#" executable="false" logical="status" virtualAddressBase="16#
    a100_0000#" writable="false">
        <memory logical="status_1"/>
        <memory logical="status_2"/>
        <memory logical="status_3"/>
        <memory logical="status_4"/>
        <memory logical="status_5"/>
       </array>
       <memory executable="false" logical="slot_control_1" size="16#1000#" virtualAddress="16#9000
    _0000#" writable="false"/>
      </memory>
      <events>
       <source>
        <event id="10" logical="reset_slot_1_sm"/>
        <event id="11" logical="reset_slot_1_linux"/>
       </source>
      </events>
     </requires>
     <provides>
      <memory executable="true" logical="text" size="16#2000#" type="subject_binary" virtualAddress=
    "16#0020_0000#" writable="false">
       <file filename="controller_text" offset="none"/>
       <hash value="16#322941a56d080a742d523c5895fededca7156ea8c9a1071ee0bc2ec4f799f19b#"/>
      </memory>
```

```
    <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
  virtualAddress="16#0020_3000#" writable="false">
     <file filename="controller_rodata" offset="none"/>
     <hash value="16#df41e836ab51453074f093d2824347d90d272ec8abaf4197c5ddb7636fd83320#"/>
    </memory>
    <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress
  ="16#0020_4000#" writable="true">
     <file filename="controller_data" offset="none"/>
     <hash value="16#ad7facb2586fc6e966c004d7d1d16b024f5805ff7cb47c7a85dabd8b48892ca7#"/>
    </memory>
    <memory executable="false" logical="bss" size="16#1000#" type="subject_binary" virtualAddress=
  "16#0020_2000#" writable="true">
     <fill pattern="16#00#"/>
    </memory>
    <memory executable="false" logical="stack" size="16#4000#" type="subject_binary"
  virtualAddress="16#1000#" writable="true">
     <fill pattern="16#00#"/>
    </memory>
   </provides>
  </component>
  <component name="time" profile="native">
   <depends>
    <library ref="libmudebuglog"/>
    <library ref="libmucontrol"/>
   </depends>
   <requires>
    <vcpu>
     <vmx>
      <controls>
       <proc>
        <RDTSCExiting>0<!--
         This is an example of a component that customizes the vCPU
         settings. In this case, direct access to the Time-Stamp Counter
         (TSC) is enabled. The settings made here are merged with the
         (default) values defined by the component profile during policy
         expansion by the Mucfgexpand tool.
         --></RDTSCExiting>
       </proc>
      </controls>
     </vmx>
     <registers>
      <gpr>
       <rip>16#0020_0000#</rip>
      </gpr>
     </registers>
    </vcpu>
    <channels>
     <array elementSize="16#1000#" logical="export_channels" virtualAddressBase="16#000f_ffd0_0000
  #">
      <writer logical="time_export1"/>
     </array>
    </channels>
    <devices>
     <device logical="cmos_rtc">
      <ioPort end="16#0071#" logical="ports" start="16#0070#"/>
     </device>
    </devices>
   </requires>
   <provides>
    <memory executable="true" logical="text" size="16#2000#" type="subject_binary" virtualAddress=
  "16#0020_0000#" writable="false">
     <file filename="time_text" offset="none"/>
     <hash value="16#fdf60d1ec7dd9bd0363469636a3058f5d58a13a5f592d05d7e6aa3f1c4b1f9a6#"/>
    </memory>
    <memory executable="false" logical="rodata" size="16#1000#" type="subject_binary"
  virtualAddress="16#0020_3000#" writable="false">
     <file filename="time_rodata" offset="none"/>
     <hash value="16#b0fd849e1cb2a24c297924bebe30b0d11e47258ded43364b726431e322a233b6#"/>
    </memory>
    <memory executable="false" logical="data" size="16#1000#" type="subject_binary" virtualAddress
  ="16#0020_4000#" writable="true">
     <file filename="time_data" offset="none"/>
     <hash value="16#ad7facb2586fc6e966c004d7d1d16b024f5805ff7cb47c7a85dabd8b48892ca7#"/>
    </memory>
    <memory executable="false" logical="bss" size="16#1000#" type="subject_binary" virtualAddress=
  "16#0020_2000#" writable="true">
     <fill pattern="16#00#"/>
    </memory>
    <memory executable="false" logical="stack" size="16#4000#" type="subject_binary"
  virtualAddress="16#1000#" writable="true">
     <fill pattern="16#00#"/>
    </memory>
   </provides>
  </component>
 </components>
 <subjects>
  <!--
```

```
   The 'subjects' element holds a list of subjects.
  -->
  <subject name="vt">
   <!--
    A subject is an instance of a component, i.e. an active component in the
    system policy that may be scheduled. Its specification references a
    component and maps all requested logical resources to physical resources
    provided by the system. Additional resources to the ones requested by
    the component can be specified here. This enables specialization of the
    base component specification.
   -->
   <events>
    <!--
     The subject 'events' element specifies all events originating from or
     directed at this subject. The physical attribute is the name of a event
     defined in the global events section.
    -->
    <source>
     <!--
      The event 'source' element specifies events that are allowed to be
      triggered by the associated subject.

      Source events are divided into two groups: 'vmx_exit' and 'vmcall'. For
      event group 'vmx_exit' the id attribute specifies the trap number while
      in the 'vmcall' group it designates the hypercall number.

      The 'vmx_exit' group is translated to a lookup table for handling VMX
      exit traps as defined by Intel SDM Vol. 3D, "Appendix C VMX Basic Exit
      Reasons". The 'vmcall' group on the other hand is translated into a
      lookup table to handle hypercalls.
     -->
     <group name="vmx_exit">
      <default physical="system_panic">
       <!--
        The 'default' element entry can be used to specify an event which should
        be added for all event ids that have not been explicitly specified.
       -->
       <system_panic/>
      </default>
     </group>
    </source>
   </events>
   <component ref="vt">
    <!--
     The 'component' reference element specifies which component this subject
     instantiates. All logical resources required by the component must be
     mapped to physical resources of the appropriate type. Validators make
     sure that all requirements are satisfied and that no mapping has been
     omitted.
    -->
    <map logical="NIC Linux" physical="virtual_console_1"/>
    <map logical="Storage Linux" physical="virtual_console_2"/>
    <map logical="input_events" physical="input_events"/>
    <map logical="input_device_1" physical="virtual_input_1"/>
    <map logical="input_device_2" physical="virtual_input_2"/>
    <map logical="debuglog" physical="debuglog_subject1"/>
    <map logical="vga" physical="vga">
     <map logical="buffer" physical="buffer"/>
     <map logical="ports" physical="ports">
      <!--
       The 'map' element maps a physical resource provided by the system with a
       resource requested by the referenced component.

       This element allows recursion to map child resources as well (e.g.
       device memory, I/O ports etc).
      -->
     </map>
    </map>
    <map logical="slot_control_1" physical="slot_control_1"/>
    <map logical="shutdown" physical="system_poweroff"/>
    <map logical="reboot" physical="system_reboot"/>
   </component>
  </subject>
  <subject name="nic_sm">
   <memory>
    <memory executable="false" logical="status_linux" physical="status_linux_1" virtualAddress="
  16#0200_0000#" writable="false"/>
   </memory>
   <events>
    <source>
     <group name="vmx_exit">
      <default physical="system_panic">
       <system_panic/>
      </default>
     </group>
     <group name="vmcall">
      <event id="0" logical="serial_irq4" physical="serial_irq4_linux_1">
```

```
          <!--
           A source `event` entry specifies a source event node, i.e. it registers
           a handler for the given event `id`. These IDs, depending on the event
           group, are either hypercall numbers or VMX basic exit reasons.

           It is possible to assign event actions to event source entries.
           Currently supported source event actions are `system_reboot` and
           `system_poweroff`, which both have the kernel itself as endpoint.
          -->
         </event>
         <event id="1" logical="reset_linux" physical="reset_linux_1"/>
         <event id="2" logical="load_linux" physical="load_linux_1"/>
        </group>
       </source>
       <target>
        <!--
         The event `target` element specifies events that the subject is an
         *endpoint* of.
        -->
        <event logical="resume_after_load" physical="start_linux_1">
         <!--
          The `event` element in the target section specifies one event endpoint
          by referencing a physical event and assigning a logical name to it.
         -->
        </event>
        <event id="63" logical="reset" physical="reset_sm_1">
         <reset/>
        </event>
       </target>
      </events>
      <monitor>
       <!--
        The monitor abstraction enables subjects to request access to certain
        data of another subject specified by name. Possible child elements are:

        - State

        - Timed\_Events

        - Interrupts

        - Loader

        See the Muen Component Specification document for details about these
        subject monitor interfaces.
       -->
       <state logical="monitor_state" subject="nic_linux" virtualAddress="16#001e_0000#" writable="
     true"/>
       <loader logical="reload" subject="nic_sm" virtualAddress="16#0000#">
        <!--
         The `loader` mechanism effectively puts the loaded subject denoted by
         the `subject` attribute under loader control, as it is not able to start
         without the help of the loader.

         In more detail, the `loader` monitor element instructs the expander tool
         to map all memory regions of the referenced subject into the address
         space of the monitor subject, using the specified `virtualAddress` as
         offset in the address space of the loader.

         If a memory region of the loaded subject is writable and file-backed,
         the region is replaced with an empty region and linked via the `hashRef`
         mechanism to the original region which is mapped into the loader.

         The state of the loaded subject is then invalidated by clearing the
         `CR4.VMXE` bit in the initial subject CR4 register value. If such a
         subject is scheduled by the kernel, a VMX exit *VM-entry failure due to
         invalid guest state* (33) occurs. See Intel SDM Vol. 3C, "23.7 Enabling
         and Entering VMX Operation" and Intel SDM Vol. 3C, "23.8 Restrictions on
         VMX Operation" for more details. This trap is linked to the loader via
         normal VMX event handling. After handover, the loader initializes the
         memory regions replaced by the expander with the designated content.

         All information required to *load* the loaded subject is provided to the
         loader subject via its own sinfo API. Memory regions prefixed with
         `monitor_sinfo_` provide access to the sinfo regions of the loaded
         subjects. Regions prefixed with `monitor_state_` specify memory regions
         containing the subject register state of the loaded subject.

         The difference between the `monitor_sinfo_` memory region address in the
         loader and the address of the `sinfo` memory region in the target sinfo
         information denotes the `virtualAddress` offset attribute of the
         `loader` element in the policy. This information combined is enough to
         fully construct the initial state of the loaded subject, or to reset a
         subject to its initial state on demand.

         The loader may also optionally check the hashes of the restored regions,
         as this information is provided via the sinfo mechanism as well.
```

```
1941        -->
           </loader>
1943      </monitor>
          <component ref="sm">
1945       <map logical="time_info" physical="time_info"/>
           <map logical="debuglog" physical="debuglog_subject2"/>
1947       <map logical="dm_pciconf_req" physical="nic_dm_request"/>
           <map logical="dm_pciconf_res" physical="nic_dm_response"/>
1949       <map logical="resume_subject" physical="resume_linux_1"/>
           <map logical="handle_subject_trap" physical="trap_to_sm_1"/>
1951       <map logical="status" physical="status_sm_1"/>
           <map logical="control" physical="control_sm_1"/>
1953      </component>
         </subject>
1955     <subject name="storage_sm">
          <events>
1957       <source>
            <group name="vmcall">
1959         <event id="0" logical="serial_irq4" physical="serial_irq4_linux_2"/>
             <event id="1" logical="reset_linux" physical="reset_linux_2"/>
1961        </group>
            <group name="vmx_exit">
1963         <default physical="system_panic">
              <system_panic/>
1965         </default>
            </group>
1967       </source>
          </events>
1969      <monitor>
           <state logical="monitor_state" subject="storage_linux" virtualAddress="16#001e_0000#" writable
         ="true"/>
1971       <loader logical="reload" subject="storage_sm" virtualAddress="16#0000#"/>
          </monitor>
1973      <component ref="sm">
           <map logical="time_info" physical="time_info"/>
1975       <map logical="debuglog" physical="debuglog_subject3"/>
           <map logical="dm_pciconf_req" physical="storage_dm_request"/>
1977       <map logical="dm_pciconf_res" physical="storage_dm_response"/>
           <map logical="resume_subject" physical="resume_linux_2"/>
1979       <map logical="handle_subject_trap" physical="trap_to_sm_2"/>
           <map logical="status" physical="status_sm_2"/>
1981       <map logical="control" physical="control_sm_2"/>
          </component>
1983     </subject>
         <subject name="time">
1985      <events>
           <source>
1987        <group name="vmx_exit">
             <default physical="system_panic">
1989          <system_panic/>
             </default>
1991        </group>
           </source>
1993      </events>
          <component ref="time">
1995       <map logical="time_export1" physical="time_info"/>
           <map logical="debuglog" physical="debuglog_subject4"/>
1997       <map logical="cmos_rtc" physical="cmos_rtc">
            <map logical="ports" physical="ports"/>
1999       </map>
           <map logical="status" physical="status_time"/>
2001       <map logical="control" physical="control_time"/>
          </component>
2003     </subject>
         <subject name="nic_sl">
2005      <events>
           <source>
2007        <group name="vmx_exit">
             <default physical="system_panic">
2009          <system_panic/>
             </default>
2011        </group>
           </source>
2013      </events>
          <monitor>
2015       <loader logical="monitor_loader_nic_linux" subject="nic_linux" virtualAddress="16#0001
         _0000_0000#"/>
          </monitor>
2017      <component ref="sl">
           <map logical="start" physical="start_linux_1"/>
2019       <map logical="handle_reset" physical="load_linux_1"/>
           <map logical="status" physical="status_linux_1"/>
2021       <map logical="control" physical="control_linux_1"/>
          </component>
2023     </subject>
         <subject name="ps2">
2025      <events>
```

```
          <source>
2027       <group name="vmx_exit">
            <default physical="system_panic">
2029         <system_panic/>
            </default>
2031       </group>
          </source>
2033     </events>
         <component ref="ps2_drv">
2035      <map logical="input_events" physical="input_events"/>
          <map logical="debuglog" physical="debuglog_subject5"/>
2037      <map logical="ps2" physical="ps2">
           <map logical="kbd_irq" physical="kbd_irq"/>
2039       <map logical="mouse_irq" physical="mouse_irq"/>
           <map logical="port_60" physical="port_60"/>
2041       <map logical="port_64" physical="port_64"/>
          </map>
2043     </component>
        </subject>
2045    <subject name="example">
         <events>
2047      <source>
           <group name="vmx_exit">
2049        <default physical="system_panic">
             <system_panic/>
2051        </default>
           </group>
2053      </source>
         </events>
2055     <monitor>
          <state logical="monitor_state" subject="storage_linux" virtualAddress="16#001e_0000#" writable
        ="true"/>
2057      <loader logical="reload" subject="example" virtualAddress="16#0000#"/>
         </monitor>
2059     <component ref="example">
          <map logical="example_request" physical="example_request"/>
2061      <map logical="example_response" physical="example_response"/>
          <map logical="debuglog" physical="debuglog_example"/>
2063      <map logical="yield" physical="example_yield"/>
          <map logical="timer" physical="example_self"/>
2065      <map logical="inject_timer" physical="example_self"/>
          <map logical="control" physical="control_example"/>
2067      <map logical="status" physical="status_example"/>
         </component>
2069    </subject>
        <subject name="controller">
2071     <events>
          <source>
2073       <group name="vmx_exit">
            <default physical="system_panic">
2075         <system_panic/>
            </default>
2077       </group>
          </source>
2079     </events>
         <component ref="controller">
2081      <map logical="debuglog" physical="debuglog_controller"/>
          <map logical="control_1" physical="control_time"/>
2083      <map logical="control_2" physical="control_sm_1"/>
          <map logical="control_3" physical="control_sm_2"/>
2085      <map logical="control_4" physical="control_example"/>
          <map logical="control_5" physical="control_linux_1"/>
2087      <map logical="status_1" physical="status_time"/>
          <map logical="status_2" physical="status_sm_1"/>
2089      <map logical="status_3" physical="status_sm_2"/>
          <map logical="status_4" physical="status_example"/>
2091      <map logical="status_5" physical="status_linux_1"/>
          <map logical="slot_control_1" physical="slot_control_1"/>
2093      <map logical="reset_slot_1_sm" physical="reset_sm_1"/>
          <map logical="reset_slot_1_linux" physical="reset_slot_1"/>
2095     </component>
        </subject>
2097    <subject name="nic_dm">
         <devices>
2099      <!--
           List of device references. Used to grant a subject access to hardware
2101       devices and their resources.
          -->
2103      <device logical="nic" physical="ethernet_controller_1">
           <!--
2105        The `device` element allows a subject access to devices referenced via
            the `physical` attribute.
2107
            For PCI devices only a single virtual bus is provided (bus 0). The `pci`
2109        element may be used to place the device at a specific location (BDF). If
            no other logical device resources of the device are specified, then the
2111        expander tool will map all physical devices resources into the subject.
```

```
    When logical device resources are explicitly specified, then only access
    to those are actually granted. The physical attribute must be either a
    reference to an existing physical device, device alias or device class.
    Validators check that this is the case.
   -->
   <pci bus="16#00#" device="16#01#" function="0"/>
   <memory executable="false" logical="mmconf" physical="mmconf" writable="true">
    <!--
     The device 'memory' element maps the device memory region referenced via
     the 'physical' attribute into the subject address space at address
     'virtualAddress'. The 'executable', 'writable' attributes define the
     access permissions for the subject.
    -->
   </memory>
  </device>
 </devices>
 <events>
  <source>
   <group name="vmx_exit">
    <default physical="system_panic">
     <system_panic/>
    </default>
   </group>
  </source>
 </events>
 <component ref="dm">
  <map logical="debuglog" physical="debuglog_subject6"/>
  <map logical="request" physical="nic_dm_request"/>
  <map logical="response" physical="nic_dm_response"/>
 </component>
</subject>
<subject name="storage_dm">
 <events>
  <source>
   <group name="vmx_exit">
    <default physical="system_panic">
     <system_panic/>
    </default>
   </group>
  </source>
 </events>
 <component ref="dm">
  <map logical="debuglog" physical="debuglog_subject7"/>
  <map logical="request" physical="storage_dm_request"/>
  <map logical="response" physical="storage_dm_response"/>
 </component>
</subject>
<subject name="dbgserver">
 <events>
  <source>
   <group name="vmx_exit">
    <default physical="system_panic">
     <system_panic/>
    </default>
   </group>
  </source>
 </events>
 <component ref="dbgserver">
  <map logical="log_channel1" physical="debuglog_subject1"/>
  <map logical="log_channel2" physical="debuglog_subject2"/>
  <map logical="log_channel3" physical="debuglog_subject3"/>
  <map logical="log_channel4" physical="debuglog_subject4"/>
  <map logical="log_channel5" physical="debuglog_subject5"/>
  <map logical="log_channel_example" physical="debuglog_example"/>
  <map logical="log_channel_6" physical="debuglog_controller"/>
  <map logical="log_channel7" physical="debuglog_subject6"/>
  <map logical="log_channel8" physical="debuglog_subject7"/>
  <map logical="crash_audit" physical="crash_audit"/>
  <map logical="debugconsole" physical="serial_device_1">
   <map logical="port" physical="ioport1"/>
  </map>
  <map logical="reboot" physical="system_reboot"/>
  <map logical="shutdown" physical="system_poweroff"/>
 </component>
</subject>
<subject name="nic_linux">
 <bootparams>console=hvc console=ttyS0 hostname=nic_linux</bootparams>
 <memory>
  <memory executable="false" logical="initramfs" physical="initramfs" virtualAddress="16#9000
_0000#" writable="false"/>
 </memory>
 <devices>
  <device logical="eth0" physical="nic_1">
   <pci bus="16#00#" device="16#01#" function="0"/>
  </device>
  <device logical="aditional_nics" physical="additional_nics"/>
 </devices>
```

```
     <events>
2199  <source>
       <group name="vmx_exit">
2201    <default physical="trap_to_sm_1"/>
       </group>
2203   <group name="vmcall">
        <event id="30" logical="reboot" physical="reboot_linux_1"/>
2205    <event id="31" logical="timer" physical="timer_linux_1"/>
       </group>
2207  </source>
      <target>
2209   <event logical="resume_after_trap" physical="resume_linux_1"/>
       <event id="63" logical="reset" physical="reset_linux_1">
2211    <reset/>
       </event>
2213   <event id="62" logical="reset_from_vt" physical="reset_slot_1">
        <reset/>
2215   </event>
       <event id="61" logical="reboot" physical="reboot_linux_1">
2217    <reset/>
       </event>
2219   <event logical="serial_irq4" physical="serial_irq4_linux_1">
        <inject_interrupt vector="52">
2221     <!--
          Instructs the SK to inject a guest interrupt with given vector on event
2223      occurrence.
         -->
2225    </inject_interrupt>
       </event>
2227   <event logical="timer" physical="timer_linux_1">
        <inject_interrupt vector="236"/>
2229   </event>
      </target>
2231 </events>
     <channels>
2233  <!--
       The 'channel' section of a subject declares references to communication
2235   channels. The referenced channels become accessible to the requesting
       subject either as reader or writer endpoint.
2237  -->
      <reader logical="virtual_input" physical="virtual_input_1" vector="100" virtualAddress="
    16#3000#">
2239   <!--
        A channel 'reader' element references a global communication channel as
2241    reader endpoint, i.e. the channel is mapped read-only into the subject
        address space.
2243   -->
      </reader>
2245  <writer event="1" logical="virtual_console" physical="virtual_console_1" virtualAddress="
    16#4000#">
       <!--
2247    A channel 'writer' element references a global communication channel as
        writer endpoint, i.e. the channel is mapped with write permissions into
2249    the subject address space.
       -->
2251  </writer>
      <reader logical="testchannel_2" physical="testchannel_2" virtualAddress="16#000e_00f0_0000#"/>
2253  <writer logical="testchannel_1" physical="testchannel_1" virtualAddress="16#000e_00f0_1000#"/>
      <reader logical="testchannel_4" physical="testchannel_4" virtualAddress="16#000e_00f0_2000#"/>
2255  <writer logical="testchannel_3" physical="testchannel_3" virtualAddress="16#000e_0100_2000#"/>
     </channels>
2257 <component ref="linux">
      <map logical="lowmem" physical="nic_linux|lowmem"/>
2259  <map logical="ram" physical="nic_linux|ram"/>
     </component>
2261 </subject>
    <subject name="storage_linux">
2263 <bootparams>console=hvc console=ttyS0 hostname=storage_linux</bootparams>
     <memory>
2265  <memory executable="false" logical="initramfs" physical="initramfs" virtualAddress="16#9000
    _0000#" writable="false"/>
     </memory>
2267 <devices>
      <device logical="xhci" physical="usb_controller_1"/>
2269 </devices>
     <events>
2271  <source>
       <group name="vmx_exit">
2273    <default physical="trap_to_sm_2"/>
       </group>
2275   <group name="vmcall">
        <event id="31" logical="timer" physical="timer_linux_2"/>
2277   </group>
      </source>
2279  <target>
       <event logical="resume_after_trap" physical="resume_linux_2"/>
2281   <event logical="from_example_subject" physical="example_yield"/>
```

```
        <event id="63" logical="reset" physical="reset_linux_2">
         <reset/>
        </event>
        <event logical="serial_irq4" physical="serial_irq4_linux_2">
         <inject_interrupt vector="52"/>
        </event>
        <event logical="timer" physical="timer_linux_2">
         <inject_interrupt vector="236"/>
        </event>
       </target>
      </events>
      <channels>
       <reader logical="virtual_input" physical="virtual_input_2" vector="49" virtualAddress="
     16#3000#"/>
       <writer event="1" logical="virtual_console" physical="virtual_console_2" virtualAddress="
     16#4000#"/>
       <reader logical="example_response" physical="example_response" virtualAddress="16#000
     e_0210_0000#"/>
       <writer event="2" logical="example_request" physical="example_request" virtualAddress="16#000
     e_0210_1000#"/>
       <reader logical="testchannel_1" physical="testchannel_1" virtualAddress="16#000e_00f0_0000#"/>
       <writer logical="testchannel_2" physical="testchannel_2" virtualAddress="16#000e_00f0_1000#"/>
       <reader logical="testchannel_3" physical="testchannel_3" virtualAddress="16#000e_00f0_2000#"/>
       <writer logical="testchannel_4" physical="testchannel_4" virtualAddress="16#000e_0100_2000#"/>
      </channels>
      <component ref="linux">
       <map logical="lowmem" physical="storage_linux|lowmem"/>
       <map logical="ram" physical="storage_linux|ram"/>
      </component>
     </subject>
    </subjects>
    <scheduling tickRate="100000">
     <!--
      The Muen SK implements a fixed, cyclic scheduler. The `scheduling`
      element is used to specify such a static plan by means of a major frame.
      A major frame consist of an arbitrary number of minor frames. Minor
      frames in turn specify a duration in number of ticks a subject is
      scheduled before preemption in terms of the tick rate.

      The tickRate attribute has the unit Hertz (Hz) and specifies the number
      of clock ticks per second. The ticks attribute of minor frames is
      expressed in terms of this tick rate. As an example: if we want to
      declare the minor frame duration in terms of microseconds (\(10^{-6}\))
      then a tick rate of 1000000 must be used.

      The duration of a major frame must be the same on each CPU, meaning the
      sum of all minor frame ticks for any given CPU must be identical.
      However, different major frames can have arbitrary length.

      The Tau0 subject designates to the kernel which major frame is the
      currently active one. At the end of each major frame, the kernel
      determines the active major frame and switches to that scheduling plan
      for the duration of the major frame.

      All subjects which can hand over execution to another subject via a
      switch event form a so called scheduling group. Membership to a
      scheduling group is determined by the specified switch events and how
      they link subjects together. Minor frames designate the subject that is
      to be executed for the given amount of ticks. The subject name
      identifies the *initial* subject of a minor frame but any member of the
      scheduling group of the given subject may be executed during that minor
      frame.
     -->
     <majorFrame>
      <!--
       A major frame consists of a sequence of minor frames for a given CPU.
       When the end of a major frame is reached, all CPUs synchronize and the
       scheduler starts over from the beginning using the first minor frame
       again. This means that major frames are repeated in a cyclic fashion
       until a different major frame is designated via the Tau0 interface.
      -->
      <cpu id="0">
       <!--
        The `cpu` element is used to specify major frames for each CPU of the
        system.
       -->
       <minorFrame subject="nic_linux" ticks="4">
        <!--
         A minor frame specifies the number of scheduling ticks a subject is
         allowed to run on the CPU specified by the parent `cpu` element.
        -->
       </minorFrame>
       <minorFrame subject="ps2" ticks="1"/>
       <minorFrame subject="nic_linux" ticks="4"/>
       <minorFrame subject="controller" ticks="1"/>
       <minorFrame subject="nic_linux" ticks="4"/>
       <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
```

```
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
```

```xml
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        <minorFrame subject="nic_linux" ticks="4"/>
        <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
       </cpu>
       <cpu id="1">
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="time" ticks="1"/>
        <minorFrame subject="vt" ticks="1"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="1"/>
        <minorFrame subject="tau0" ticks="1"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="1"/>
        <minorFrame subject="vt" ticks="1"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="1"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="1"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
```

```
      <minorFrame subject="example" ticks="3"/>
      <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
      <minorFrame subject="example" ticks="3"/>
      <minorFrame subject="dbgserver" ticks="2"/>
      <minorFrame subject="example" ticks="3"/>
      <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
      <minorFrame subject="example" ticks="3"/>
      <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
      <minorFrame subject="example" ticks="3"/>
      <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
      <minorFrame subject="example" ticks="3"/>
      <minorFrame subject="dbgserver" ticks="2"/>
      <minorFrame subject="example" ticks="3"/>
      <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
      <minorFrame subject="example" ticks="3"/>
      <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
      <minorFrame subject="example" ticks="3"/>
      <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
     </cpu>
    </majorFrame>
    <majorFrame>
     <cpu id="0">
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="ps2" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="controller" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
      <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
      <minorFrame subject="nic_linux" ticks="4"/>
```

```
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
         <minorFrame subject="nic_linux" ticks="4"/>
         <minorFrame subject="mugenschedcfg_auto_idle_0" ticks="1"/>
        </cpu>
        <cpu id="1">
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="dbgserver" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="time" ticks="1"/>
         <minorFrame subject="vt" ticks="1"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="vt" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="vt" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="dbgserver" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="vt" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="vt" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="vt" ticks="1"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="1"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="dbgserver" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="dbgserver" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="dbgserver" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="dbgserver" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="dbgserver" ticks="2"/>
         <minorFrame subject="example" ticks="3"/>
         <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="1"/>
```

```
        <minorFrame subject="vt" ticks="1"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="vt" ticks="1"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="1"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="dbgserver" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
        <minorFrame subject="example" ticks="3"/>
        <minorFrame subject="mugenschedcfg_auto_idle_1" ticks="2"/>
      </cpu>
    </majorFrame>
  </scheduling>
</system>
```

Listing 8.1: Demo System (VT-d)

Chapter 9

Bibliography

[1] Adrian-Ken Rueegsegger and Reto Buerki. *Muen Component Specification*. 2021.

[2] Adrian-Ken Rueegsegger and Reto Buerki. *Muen Separation Kernel*. 2021.