LynxSecure 6.3.0 Advanced Configuration Guide

LynxSecure Release 6.3.0-rev16326

Product names, screen captures, and other related information listed in LynxSecure® product documentation are copyright materials or trademarks, as recorded, of the respective manufacturers and are included for attribution purposes.

Copyright © 2014-2018 Lynx Software Technologies, Inc. All rights reserved.
Copyright © 2004-2014 LynuxWorks, Inc. All rights reserved.

U.S. Patents 9,390,267; 8,745,745; 9,218,489; 9,129,123; 8,782,365; 9,208,030; 9,213,840.

Printed in the United States of America.

All rights reserved. No part of *LynxSecure® 6.3.0 Advanced Configuration Guide* may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photographic, magnetic, or otherwise, without the prior written permission of Lynx Software Technologies, Inc.

Lynx Software Technologies, Inc. makes no representations, express or implied, with respect to this documentation or the software it describes, including (with no limitation) any implied warranties of utility or fitness for any particular purpose; all such warranties are expressly disclaimed. Neither Lynx Software Technologies, Inc., nor its distributors, nor its dealers shall be liable for any indirect, incidental, or consequential damages under any circumstances.

(The exclusion of implied warranties may not apply in all cases under some statutes, and thus the above exclusion may not apply. This warranty provides the purchaser with specific legal rights. There may be other purchaser rights which vary from state to state within the United States of America.)

Contents

PREFACE ..... **xi**
About this Guide xi
Intended Audience xi
For More Information xi
Typographical Conventions xii
Special Notes xiii
Technical Support xiii
Lynx Software Technologies, Inc. U.S. Headquarters xiii
Lynx Software Technologies, Inc. Europe xiii

CHAPTER 1 INTRODUCTION TO CONFIGURING LYNXSECURE **1**

CHAPTER 2 CONFIGURATION VECTOR ORGANIZATION **3**
Boilerplate 3
Environment 3
System Resources 4
Scheduling Policies 4
Exported Resources and Partitions 6
Platform Features 6
Memory Flows and Memory Regions 7
Guest and Host Addresses 10
Device Flows and External Devices 10
Virtual Device Flows and Virtual Devices 11
Message Buffer Flows 12
Audit Flows 12
Absolute Clock Flows 12
Built-In Test Subject Specifics 13
Paravirtualized Linux Specifics 14
Fully Virtualized Subject Specifics 15
RMRRs 19
Optional Configuration Items 20

CHAPTER 3 CONFIGURING LYNXSECURE FEATURES **21**
Audit 21
Errors in Rules Specifications 22
Audit Filter: Empty 22
Audit Filter: Match All 22
Audit Filter: Basic Case 1 22
Audit Filter: Basic Case 2 23
Audit Filter: Basic Case 3 23
Audit Filter: Override 1 23
Audit Filter: Override 2 23
Audit Filter: Two Non-Overlapping Rules 23
Audit Filter: Override 3 23

Audit Filter: Override 4 24
Audit Filter: Override 5 24
Audit Filter: Override 6 24
Audit Filter: Override 7 24
Audit Filter: Override 8 24
Audit Filter: Override 9 24
Audit Filter: Multiple Overrides 1 25
Audit Filter: Multiple Overrides 2 25
Subject Access to Audit 25
Example Audit Configuration 25
Memory Sanitization 26
Requirements 26
Building a Sanitization Image 26
Configuration Example 27
In-focus Scheduling 27
Using the USB Debug Cable/Device with LynxSecure 28
How to Identify the Correct USB Connection/Orientation for the Debug Device/Cable 28
How to Identify the USB Debug Port on a Target 28
How to Identify the USB Debug Port with LynxSecure 29
Using Identity-Mapping to enable DMA on Systems with No I/O MMU 29
The Host Physical Memory Layout and Customizing the Load Addresses 30
Enabling Additional Modules 32
Booting SKH Runtime Package Using the Trusted Initialization 33
Enhancing SRPs with Digital Signature and Verification 34
Overview 34
Signing the SRP Files 35
OpenSSL Commands to Create Test Keys and Certificates 35
Creating the Hash Value 36
Setting up the TPM on the Target machine 36
Preventing Non-Error Messages from LynxSecure 37
Other Features 37

CHAPTER 4 XML REFERENCE 39
Human-readable Configuration Vector (HCV) 39
<hcv> — Root Element 39
Configured Security Policy 40
<lsenvironment> — System Parameters 40
LynxSecure architecture 43
Value 43
Value 43
<rif> — Options to RIF 43
<skh> — Hypervisor Options 44
<diagoutput> — Diagnostic Output 44
Diagnostic Output Settings 45
<serial> — Serial Port 45
<systemresources> — System Resources 47
<mainmemory> — Main Memory 47
<memblock> — Memory Block 48
<processor> — Processor 48
<bridgeinfo> — PCI Bridges 49
<sibitschedpolicy> — SBIT/IBIT Scheduling Policy 50
<majorframe> — Major Frame 51
<dynamicsched> — Dynamic Schedule 51
<thread> — Thread 52

Dynamic Scheduling Algorithm 53
<schedulingpolicy> — Scheduling Policy 54
<maintschedpolicy> — Maintenance Scheduling Policy 55
<partition> — Partition 55
<partitionflow> — Partition-to-Partition Flow 56
<platformfeature> — Platform Feature 57
Supported Platform Features 57
<memoryregion> — Memory Region 58
Supported Memory Region Types 59
<alignment> — Alignment 65
Memory Region Alignment 66
<starthpa> — Starting Host Physical Address 66
<guestimage> — Guest Image 67
<externaldevice> — External Device 68
System Bus Type 70
Device Interrupt Emulation Modes 70
<ioports> — I/O Ports 71
<iomem> — Memory-Mapped I/O Regions 72
<virtualdevice> — Virtual Device 73
A Virtual Device Type 74
<capflow> — Bidirectional Flow Capability 74
<memreg> — Memory Region for Virtual Device 75
<interface> — Virtual Device Interface 76
<pci> — PCI Device Emulation 76
<capflow> — Unidirectional Flow Capability 77
<msgbuf> — Message Buffer for Virtual Device 77
<ioports> — I/O Port Range 78
<iomem> — Memory-mapped I/O 79
<messagebuffer> — Message Buffer 79
<absoluteclock> — Absolute Clock 80
<audit> — Audit 81
<fullbufferaction> — Buffer Overflow Action 81
Supported Full Buffer Actions 82
<auditrule> — Audit Rule 82
Supported Audit Actions 83
<subject> — Subject 83
Supported Subject Types 89
Instruction Set Architecture 89
Supported Subject Roles 89
Subject Failure Actions 90
Initial Subject States 90
Supported WBINVD Emulation Methods 90
Device Resource Allocation Policies 91
<platformfeatureflow> — Plaftorm Feature Flow 91
<memoryflow> — Memory Flow 92
<externaldeviceflow> — External Device Flow 94
<virtualdeviceflow> — Virtual Device Flow 95
<messagebufferflow> — Message Buffer Flow 96
<absoluteclockflow> — Absolute Clock Flow 97
<auditflow> — Audit Flow 98
<subjectflow> — Subject Flow 99
<injectintperm> — Interrupt Injection Permission 100
<virtualprocessor> — Virtual Processor Flow 100

APPENDIX A GLOSSARY 103
Acronyms 106

APPENDIX B LEGACY TRANSITION GUIDANCE 109
Automatic Upgrade 109

List of Figures

3-1. LynxSecure SRP Module Placement in Host RAM 31

List of Tables

2-1. CBIT/SIBIT Command Line Arguments 14
2-2. PV Linux Command Line Arguments 14
2-3. Command Line Options in a Fully Virtualized Subject 16
3-1. Modules available in LynxSecure 32
3-2. PCR Assignments 33
4-1. Attributes for <hcv> 39
4-2. Configured Security Policy 40
4-3. Attributes for <lsenvironment> 40
4-4. LynxSecure architecture 43
4-5. Value 43
4-6. Value 43
4-7. Attributes for <rif> 43
4-8. Attributes for <skh> 44
4-9. Attributes for <diagoutput> 45
4-10. Diagnostic Output Settings 45
4-11. Attributes for <serial> 46
4-12. Attributes for <mainmemory> 47
4-13. Attributes for <memblock> 48
4-14. Attributes for <processor> 49
4-15. Attributes for <bridgeinfo> 49
4-16. Attributes for <sibitschedpolicy> 50
4-17. Attributes for <majorframe> 51
4-18. Attributes for <thread> 52
4-19. Dynamic Scheduling Algorithm 53
4-20. Attributes for <schedulingpolicy> 54
4-21. Attributes for <maintschedpolicy> 55
4-22. Attributes for <partition> 55
4-23. Attributes for <partitionflow> 56
4-24. Attributes for <platformfeature> 57
4-25. Supported Platform Features 57
4-26. Attributes for <memoryregion> 58
4-27. Supported Memory Region Types 60
4-28. Memory Region Alignment 66
4-29. Attributes for <guestimage> 67
4-30. Attributes for <externaldevice> 69
4-31. System Bus Type 70
4-32. Device Interrupt Emulation Modes 70
4-33. Attributes for <ioports> 71
4-34. Attributes for <iomem> 72
4-35. Attributes for <virtualdevice> 73
4-36. A Virtual Device Type 74
4-37. Attributes for <memreg> 75
4-38. Attributes for <interface> 76
4-39. Attributes for <pci> 76
4-40. Attributes for <msgbuf> 78

4-41. Attributes for <ioports> 78
4-42. Attributes for <iomem> 79
4-43. Attributes for <messagebuffer> 80
4-44. Attributes for <absoluteclock> 80
4-45. Attributes for <audit> 81
4-46. Attributes for <fullbufferaction> 81
4-47. Supported Full Buffer Actions 82
4-48. Attributes for <auditrule> 82
4-49. Supported Audit Actions 83
4-50. Attributes for <subject> 83
4-51. Supported Subject Types 89
4-52. Instruction Set Architecture 89
4-53. Supported Subject Roles 89
4-54. Subject Failure Actions 90
4-55. Initial Subject States 90
4-56. Supported WBINVD Emulation Methods 90
4-57. Device Resource Allocation Policies 91
4-58. Attributes for <platformfeatureflow> 91
4-59. Attributes for <memoryflow> 92
4-60. Attributes for <externaldeviceflow> 94
4-61. Attributes for <virtualdeviceflow> 95
4-62. Attributes for <messagebufferflow> 97
4-63. Attributes for <absoluteclockflow> 97
4-64. Attributes for <auditflow> 98
4-65. Attributes for <subjectflow> 99
4-66. Attributes for <injectintperm> 100
4-67. Attributes for <virtualprocessor> 101

Preface

About this Guide

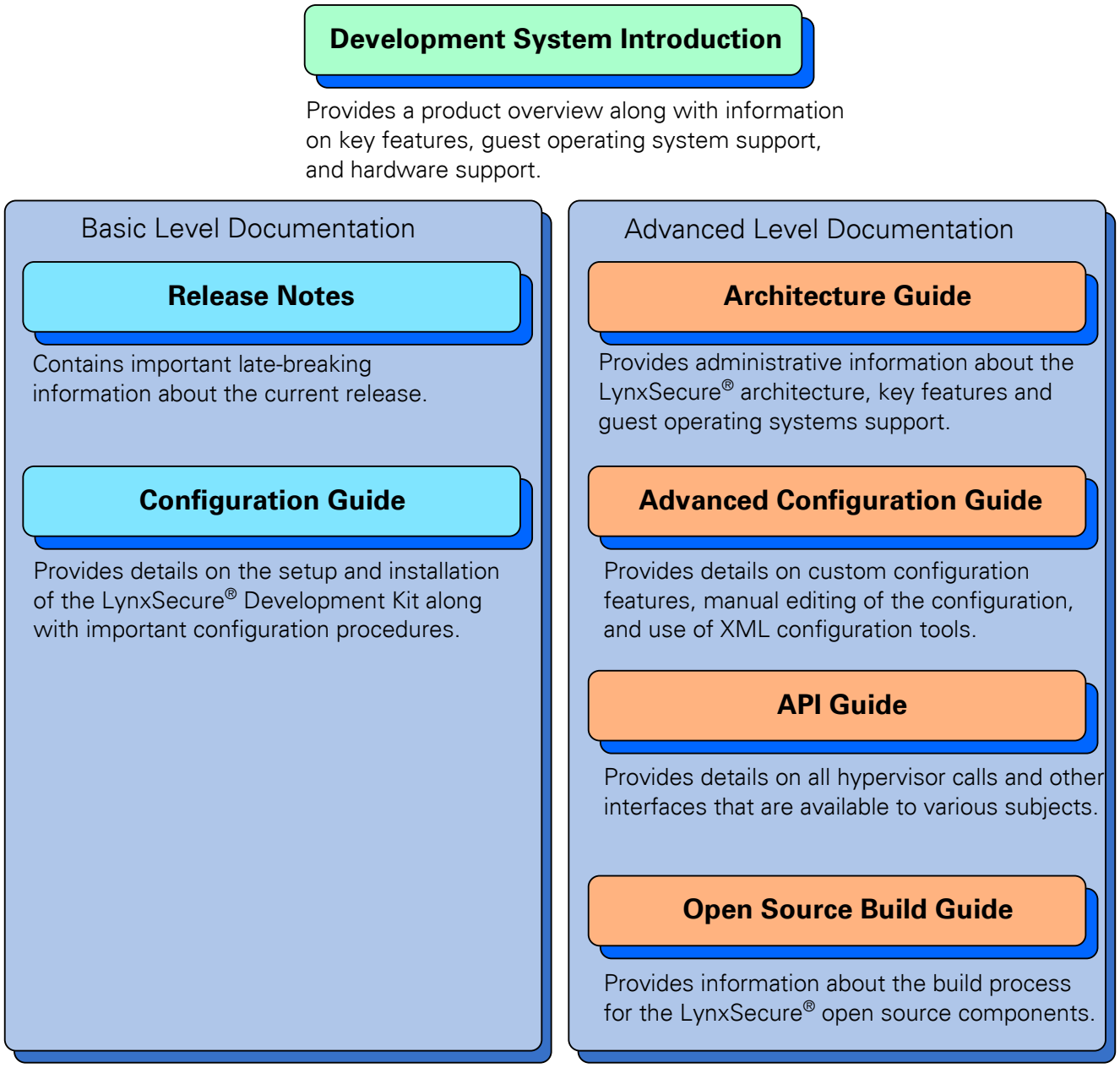
This guide, *LynxSecure® Advanced Configuration Guide* provides details on custom configuration features and provides the option to manually edit configuration settings using the XML configuration tool.

Intended Audience

The information in this guide is designed and written for system integrators and software developers who will build applications on top of LynxSecure and the available subjects. A basic understanding of Linux®/Unix® and familiarity with fairly complex configuration procedures are recommended.

For More Information

For more information on the features of LynxSecure, refer to the following printed and online documentation.



Typographical Conventions

The typefaces used in this manual, summarized below, emphasize important concepts. All references to file names and commands are case sensitive and should be typed accurately.

Font and Description	Examples
Times New Roman 10 pt. - Used for body text; *italicized* for emphasis, new terms, and book titles.	Refer to the *LynxSecure User's Guide*

Courier New 9 pt. - Used for environment variables, file names, functions, methods, options, parameter names, path names, commands, and computer data. Commands that need to be	`ls -l`
	`myprog.c`
	`/dev/null`
	`login: myname`

Font and Description	Examples
highlighted within body text, or commands that must be typed as-is by the user are **bolded**.	`# cd /usr/home`
Courier New Italic 9 pt. - Used for text that represents a variable, such as a file name or a value that must be entered by the user.	`cat filename` `mv file1 file2`

Courier New 7 pt. - Used for blocks of text that appear on the display screen after entering instructions or commands.	`Kernel: target1.srp`
	`> Loading: target1.srp`
	`> .....`
	`> Booting`

Univers 45 Light Bold 8 pt. - keyboard options, button names, and menu sequences.	Enter, Ctrl-C

Special Notes

The following notations highlight any key points and cautionary notes that may appear in this manual.

NOTE: These callouts note important or useful points in the text.

CAUTION! Used for situations that present minor hazards that may interfere with or threaten equipment/performance.

Technical Support

Lynx Software Technologies, Inc. Technical Support is available Monday through Friday (excluding holidays) between 8:00 AM and 5:00 PM Pacific Time (U.S. Headquarters) or between 9:00 AM and 6:00 PM Central European Time (Europe).

The Lynx Software Technologies, Inc. World Wide Web home page [http://www.lynx.com] provides additional information about our products.

Lynx Software Technologies, Inc. U.S. Headquarters

Internet: <support@lynx.com>
Phone: (408) 979-3940
Fax: (408) 979-3920

Lynx Software Technologies, Inc. Europe

Internet: <tech_europe@lynx.com>
Phone: (+33) 1 30 85 06 00
Fax: (+33) 1 30 85 06 06

CHAPTER 1 *Introduction to Configuring LynxSecure*

This guide describes the configuration details for LynxSecure® and its subjects. It is written for system integrators who will use LynxSecure in conjunction with LynxSecure subjects.

Configuration of LynxSecure involves the following key aspects:

1. The XML configuration vector is an XML file input to the configuration tool. The XML configuration vector may also be referred to as the HCV (Human-readable Config Vector) or just "the XML file".
2. System parameters input and edited in the given XML configuration vector.
3. GuestOS'es (also known as "subjects") and their interfaces to LynxSecure and the resources they will access, specified in the XML configuration vector.
4. The configuration tool named `mkcv`. (`mkcv` stands for "make configuration vector").

   The Configuration Tool generates a binary image that is booted on the target system. That binary image is referred to as an SRP (Separation Kernel Runtime Package). For a high level description of the SRP, its components, how to install and boot an SRP and other details, please refer to *LynxSecure 6.3.0 Getting Started and Configuration Guide*.
5. The output files of the configuration tool (`.srp`, `.txt`).
6. The Autoconfig Tool can automate most of the tasks listed above. Refer to Chapter 4, "*Introduction to the Autoconfig Tool*" in *LynxSecure 6.3.0 Getting Started and Configuration Guide* for further information on this tool.

The chapters of this guide are organized as follows:

- Chapter 1 provides the brief introduction to LynxSecure configuration and an overview of the individual chapters.
- Chapter 2 describes in detail the Configuration Vector organization.
- Chapter 3 provides details on how to configure LynxSecure features
- Chapter 4 provides the XML reference describing the XML elements, attributes and their constraints.

CHAPTER 2 *Configuration Vector Organization*

The primary input to the configuration tool (`mkcv`) is an XML file, referred to as the configuration vector. Required inputs to the XML file itself are the path names to the GuestOS'es (also referred to as subjects) that the configuration tool will package into the SRP. Sample XML configuration vectors illustrated within this guide show these path specifiers, in addition to the other XML parameters.

If requested by a command line option (`-t`) the configuration tool will also produce a text based output file to show how the configuration tool has allocated resources with the given input XML file. For example, if one constructs an SRP named `server_build.srp`, the configuration tool will generate a text file, named `server_build.txt`.

At the high level, within a given XML file, the parameters defining a given system consist of the following:

- System resources: the number of CPUs and available memory blocks.
- Exported resources: external devices, virtual devices and other resources.
- Subjects: defines a Guest OS and its interface to LynxSecure®.
- Flows: associations of resources with subjects.
- Scheduling Policies: assign subject virtual processors to physical CPUs for the specified time slice.

The following sections illustrate a ground-up approach to creating an XML configuration vector.

Boilerplate

The head and tail of the XML configuration vector requires boilerplate XML similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<hcv hcvschemaversion="HCV Schema Version 0023">
  <!-- The rest of the XML configuration vector resides within the hcv element-->
</hcv>
```

Environment

The `environment` XML element is mandatory:

```
<lsenvironment ibitpath="../ibit.bin" tickspersec="1000" iommu="true">
  <diagoutput vga="false">
    <serial port="03f8" baudrate="115200" clockrate="1843200" lcr="8n1"/>
  </diagoutput>
</lsenvironment>
```

The `environment` element defines the following resources, denoted by XML attributes:

- `ibitpath`: the path to the Initiated Built-In Test binary. This code and data image is used for automatically generated IBIT subjects. In the example, a relative path is used, but an absolute path may also be used.
- `tickspersec`: the number of System Clock Tick interrupts per second. LynxSecure scheduling of subjects is based on this value.
- `iommu`: the on/off switch for the I/O MMU. By default, the I/O MMU is enabled.

Fully virtualized subjects with directly assigned physical devices must run with `iommu="true"` for normal function (this is the default). As an exception, in certain configurations one may configure a single FV subject with directly assigned physical devices without IOMMU; see section “The Host Physical Memory Layout and Customizing the Load Addresses” (page 30).

The `environment` element has a `diagoutput` sub-element. The example shows typical values for the legacy COM1 serial port at address `0x3f8`. If the diagnostic output is enabled, LynxSecure will display limited messages, which can be useful when configuring and bringing up a system for the first time. Diagnostic output can also be directed to the VGA console or to a USB debug port.

System Resources

Each HCV requires a `systemresources` element:

```
<systemresources>
  <mainmemory hostmemname="host_main_memory">
    <memblock memblocksize="00000000000A0000" memblockstart="0000000000000000"/>
    <memblock memblocksize="00000000CAD00000" memblockstart="0000000000100000"/>
    <memblock memblocksize="000000002C000000" memblockstart="0000000100000000"/>
  </mainmemory>
  <processor processorname="phys_core_0"/>
  <processor processorname="phys_core_1"/>
  <bridgeinfo pcibridges="6" hostbridges="2" hbresources="13"/>
</systemresources>
```

In the example above, the optional `mainmemory` element defines the RAM ranges available to software in the system. Memory regions that don't have their start address explicitly specified in the HCV are allocated from this memory. If this element is not specified, LynxSecure determines the available RAM automatically during startup.

The `systemresources` also specifies the number of physical CPUs used by this configuration. In this case, as the XML file was created for a dual-processor system, there are two CPUs specified. CPUs may be specified up to the physical limit of the given system; however, each CPU specified must have some subjects associated with it (illustrated later).

`bridgeinfo` is a optional entry which assists in optimizing RO page size assignment. This entry is needed for `mkcv` to calculate the size of the RO page for each subject, so that it can emit a build-time error in case the size is insufficient.

Scheduling Policies

HCV contains the following scheduling policies:

- The S/IBIT scheduling policy: for the Startup/Initiated Built-In Test subjects (optional)
- The normal scheduling policy: for the subjects (there may be more than one such policy)
- The maintenance mode scheduling policy: for subjects executing during maintenance mode (optional)

The primary scheduling policy is the first normal policy, specified with the `schedulingpolicy` XML element:

```
<schedulingpolicy policyname="schd_normal_operation_policy">
  <majorframe processorname="phys_core_0">
    <minorframe timeslice="5" vpname="cbit0_virtproc0"/>
    <minorframe timeslice="100" vpname="fv1_virtproc0"/>
  </majorframe>
  <majorframe processorname="phys_core_1">
    <minorframe timeslice="5" vpname="cbit1_virtproc1"/>
    <minorframe timeslice="50" vpname="pvlinux1_virtproc1"/>
    <minorframe timeslice="50" vpname="pvsubject_virtproc1"/>
  </majorframe>
</schedulingpolicy>
```

In the above example, there are two major frames, one for each physical CPU. Each CPU specified in the system resources requires a unique major frame in each of the scheduling policies. A major frame is a collection of minor frames associated with a unique CPU.

Within each major frame, there are a number of minor frames specified. The attribute named "`vpname`" specifies which one of the subjects' VCPUs is uniquely associated with the minor frame. A minor frame is associated with a unique subject's virtual processor. It specifies how many system clock ticks transpire before the next minor frame within that major frame is scheduled by LynxSecure.

The scheduling of minor frames within a major frame is cyclic. Example, when the normal scheduling policy activates, the CBIT subject will run on "`phys_core_0`" CPU for 5 ticks of its minor frame, then the fully virtualized subject will run within its minor frame for 100 ticks, then the CBIT subject will run again for 5 ticks. On "`phys_core_1`" CPU, a different CBIT subject runs for 5 ticks, the paravirtualized Linux® subject runs for 50 ticks, another paravirtualized subject runs for 50 ticks, then the CBIT subject runs again.

The "size" of the major frames is the sum of the sizes (timeslices) of its minor frames. All major frames within a given scheduling policy must be the same size. In the example above, in the normal scheduling policy, the two minor frames on "`phys_core_0`" are of size 5+100=105, and the size of the three minor frames on "`phys_core_1`" are 5+50+50=105, so each major frame is the same size.

A given scheduling policy covers a set of CPUs. At any given time within the system, only one scheduling policy is active. LynxSecure performs transitions between scheduling policies.

At system start up, the S/IBIT policy will run, and perform system checks. Provided those checks pass, LynxSecure then performs a change of scheduling policy to the normal scheduling policy. The scheduling policy remains the normal scheduling policy until either an authorized subject (such as CBIT, Continuous Built-In Test) asks LynxSecure to change to the SIBIT policy, or LynxSecure makes a transition to Maintenance mode, or an authorized subject changes to another normal scheduling policy.

The specification of the S/IBIT policy is similar to that of the normal scheduling policy, and subject to the same basic constraints, however, as in the example below, the `role` of subject must be `IBIT`. In the example below, the `vpname` specifies the name of virtual CPU that belongs to an IBIT subject:

```
<sibitschedpolicy policyname="schd_bit_policy">
  <majorframe processorname="phys_core_0">
    <minorframe timeslice="10" vpname="ibit0_virtproc0"/>
  </majorframe>
  <majorframe processorname="phys_core_1">
    <minorframe timeslice="10" vpname="ibit1_virtproc1"/>
  </majorframe>
</sibitschedpolicy>
```

The S/IBIT policy specification is optional. If present, IBIT subjects must be explicitly specified in the HCV as well. If missing, LynxSecure automatically generates both the IBIT subjects and the S/IBIT scheduling policy. An explicit specification may still be required if there is a need to customize the IBIT subjects. Further examples in this document will illustrate how to specify a BIT subject, if it is necessary.

When LynxSecure makes a transition to maintenance mode, this implies a change of scheduling policy to the maintenance mode scheduling policy. The subjects that run within maintenance mode can be normal subjects (non-BIT subjects) or the CBIT subjects. In the example below, the subjects are the same as those in the normal scheduling policy:

```
<maintschedpolicy policyname="schd_maintenance_policy">
  <majorframe processorname="phys_core_0">
    <minorframe timeslice="5" vpname="cbit0_virtproc0"/>
    <minorframe timeslice="100" vpname="fv1_virtproc0"/>
  </majorframe>
  <majorframe processorname="phys_core_1">
    <minorframe timeslice="5" vpname="cbit1_virtproc1"/>
    <minorframe timeslice="50" vpname="pvlinux1_virtproc1"/>
    <minorframe timeslice="50" vpname="pvsubject_virtproc1"/>
  </majorframe>
</maintschedpolicy>
```

A LynxSecure XML configuration vector may contain an unlimited number of normal scheduling policies. Only one scheduling policy is active at a time. An authorized subject can make a hypercall to change the active scheduling policy.

There is no limit to the number of subjects that the configuration can support and each major frame can have an unlimited number minor frames (and hence subjects) associated with it.

If the HCV contains more than one scheduling policy, the first scheduling policy in the HCV will be the default scheduling policy. The default scheduling policy is executed unless an authorized subject hypercall changes the active scheduling policy.

Additional constraints on scheduling policies are as follows:

- All scheduling policies must have the same number of major frames, one per physical processor.
- The minimum length of a minor frame is 1 tick.
- The minimum length of a major frame (all minor frames combined) is 4 ticks.
- All minor frames of a particular virtual processor must be scheduled on the same physical CPU. This includes not being on different CPUs in different policies.
- A subject may not schedule more than one of its virtual processors on the same physical CPU.

Exported Resources and Partitions

The `partition` XML element contains the exported resources associated with the subjects. It typically consists of platform features, memory regions, external devices, virtual devices and the subjects themselves.

The exported resources are grouped into the partitions. In the simplest configuration, the assignment of resources to partitions does not matter, so the example below assigns all resources to a single partition.

```
<partition pname="default_partition">
  <!--platform features,memory regions, external devices, virtual devices, subjects -->
</partition>
```

Within the exported resources element, one first defines platform features, memory regions, external devices and virtual devices. Then, during the addition of the subjects, one creates associations between the subjects and other exported resources using flows.

For example, in constructing a paravirtualized subject, the `subject` element is as follows:

```
<subject sname="pvsubject" type="PARAVIRT" isa="32BIT" startaddr="00001020">
  <!--memory flows, device flows, flows to other resources -->
</subject>
```

`sname` is the unique name for the subject. The mode is set to a 32-bit paravirtualized subject, and the start address of this subject is 0x00001020, a virtual address.

Platform Features

Platform features are parts of the CPU or the motherboard that are not devices per se, but represent a function that can be assigned to subjects individually. Each platform feature must be first declared with a `platformfeature` element, and then assigned to the subject using the `platformfeatureflow` element. Unlike other exported resources, which may be named arbitrarily, a platform feature's name uniquely identifies it and must be selected from the list of supported feature names. Refer to Table 4-25 (page 57) for a list of supported platform features.

In this example, the subject is assigned the PAT (Page Attribute Table) feature:

```
<partition pname="default_partition">
  <platformfeature featurename="PAT"/>
  ...
  <subject sname="pvsubject" type="PARAVIRT" isa="32BIT" startaddr="00001020">
    <platformfeatureflow featurename="PAT" readperm="ALLOW" writeperm="ALLOW"/>
    ...
  </subject>
  ...
</partition>
```

Memory Flows and Memory Regions

Within the `subject` element, several memory flows are defined. The key aspect of the memory flow is the memory region with which it associates. Each flow must associate with a memory region. Each memory region may have one or more flows to it. Later, there will be examples of multiple flows to the same region (for shared memory).

In this example, the `memoryflow` associates with the memory region labeled "`pvsubject_init_image`":

```
<partition pname="default_partition">
  ...
  <memoryregion memregionname="pvsubject_init_image" memorytype="PROGRAM" size="09C00000"/>
  ...
  <subject sname="pvsubject" type="PARAVIRT" isa="32BIT" startaddr="00001020">
    <memoryflow memregname="pvsubject_init_image" gva="00000000" readperm="ALLOW" writeperm="ALLOW"/>
  </subject>
  ...
</partition>
```

The `memoryregion` element specifies the following information:

- The size of the memory region is 0x09C00000.
- The memory region is aligned to 4MB boundary.
- The type of the memory region is `PROGRAM`, the generic memory type, appropriate for general use by subjects (including use for shared memory).

The `memoryflow` elements provides the following information:

- The subject is granted both read and write access to the memory region.
- The `memoryflow` specifies a subject-specific (or subject-private) virtual address to use with a given memory region. This memory region is mapped at guest virtual address 0x00000000.

Here and below, the `gva` (Guest Virtual Address) and `gpa` (Guest Physical Address) attributes of memory flows are optional. If set to `AUTO`, LynxSecure picks the corresponding guest address automatically. If the attribute is not present, LynxSecure may add a mapping for certain memory region types automatically, following the same logic as in the case when certain memory region is not defined in the HCV at all (see below). Users should use the value `AUTO` for the `gva` attribute and omit the `gpa` attribute (since it defaults to `AUTO`) in most cases and only specify an explicit value if automatic allocation is not desired.

NOTE: The Guest Virtual Addresses are only used for the initial mapping in PV subjects. FV subjects don't create the initial mapping, so the `gva` attribute is ignored in memory flows, and shouldn't be specified for FV subjects.

At every subject start or restart, before LynxSecure starts executing subject code, it first copies the subject's boot image (the contents of the memory region with the type `BOOT`) to its runtime location in RAM. The runtime location of the boot image in RAM is determined by the subject's `startaddr` attribute: whichever mapping it points to is the boot image copy target. If there is no such mapping, but the `ramsize` attribute is specified for the subject, and the subject is a PV subject, subject RAM is created at `startaddr`, rounded down to a page size, and automatically becomes the boot image copy target.

NOTE: In some older releases of LynxSecure, the boot image copy target was determined by the first memory flow in the subject. This is no longer the case; the first memory flow in a subject has no special properties.

A subject is allowed to have multiple flows to `BOOT` memory regions and those flows may be read-only or read/write. Read/write flows let a subject modify the boot image for itself or for another subject. However, only one of the `BOOT`

memory regions that the subject has flows to is used for booting that subject. It is determined as follows: the BOOT memory region used for booting a particular subject must be read-only in that subject or owned by that subject (which can be configured using the owner attribute of the memory flow). There must be exactly one BOOT memory region satisfying those criteria for each subject.

The image source file name is specified by the guestimage element:

```
<guestimage memregionname="pvsubject_init_image_boot" memorytype="BOOT" path="net.img"/>
```

The guestimage element above specifies the following:

- It is a memory region with the unique name "pvsubject_init_image_boot"
- It uses the special memory type BOOT, only used with guestimage elements. That designates the memory region as the boot image for the subject it is assigned to.
- The contents of the region are the contents of the file "net.img"; typically, the file is a kernel or other executable code and data image.

Optionally, a subject (para-virtualized or fully virtualized) may be given flows to page table memory pools. If these flows are not specified in the Configuration Vector, the corresponding pools and flows to them are generated automatically. In this example, the page table pools are specified explicitly. Since the subject is a 32-bit PV subject, it only needs two pools: PDT and PTE. PDT is the 2nd level page table, PTE is the first level page table. In Intel® x86, the page tables reside in memory. The memory flows to the memory region "pvsubject_PDT" and "pvsubject_PTE" are defined as follows:

```
<memoryflow memregname="pvsubject_PDT" gva="D0000000" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="pvsubject_PTE" gva="D0800000" readperm="ALLOW" writeperm="ALLOW"/>
```

The memoryregion elements named "pvsubject_PDT" and "pvsubject_PTE" are defined as follows:

```
<memoryregion memregionname="pvsubject_PDT" memorytype="PDT" size="400000"/>
<memoryregion memregionname="pvsubject_PTE" memorytype="PTE" size="400000"/>
```

Together, the flows to those regions define the page table memory resources for the paravirtualized subject. They form the following association:

- The memoryflow to those memoryregion elements are read/write flows. However, LynxSecure allows writing to page table memory regions only via dedicated hypercalls. A subject cannot map these regions read-write and modify them directly.
- The memoryflow to the memoryregion of type PDT associates the virtual address of 0xd0000000 with the physical memory of size 0x00400000 (4MB).
- The memoryflow to the memory region of type PTE associates the virtual address of 0xd0800000 with the physical memory of size 0x00400000 (4MB).
- The actual location of the memoryregion's physical memory is anonymous; it is selected at runtime.
- LynxSecure allocates the memory for 32-bit paravirtualized subjects within the low 4GB of physical memory. For 64-bit paravirtualized subjects, special alignment values need to be used to request allocation from low 4GB. See Table 4-28 (page 66) for more details.

The next memory flows of the subject are for the two special memory regions used to interface with LynxSecure. These interfaces are the Read-Only page and the Argument page. The flows are as follows:

```
<memoryflow memregname="pvsubject_LSK_RO_PAGE" gva="F0008000" readperm="ALLOW"/>
<memoryflow memregname="pvsubject_LSK_ARG_PAGE" gva="F0018000" readperm="ALLOW"/>
```

Their associated memoryregion elements are as follows:

```
<memoryregion memregionname="pvsubject_LSK_RO_PAGE" memorytype="ROPAGE" size="010000"/>
<memoryregion memregionname="pvsubject_LSK_ARG_PAGE" memorytype="ARGPAGE" size="1000"/>
```

The association between these memoryregion elements and memoryflow elements is as follows:

- The RO-page, uniquely labeled for this subject as "pvsubject_LSK_RO_PAGE" starts at the virtual address 0xF0008000, with the read permission. The RO-page is of size 0x10000 (64KB).

- The ARG-page, uniquely labeled for this subject as "pvsubject_LSK_ARG_PAGE" starts at the virtual address 0xF0018000, with the read permission. The ARG-page is of size 0x1000 (4KB).

Each subject must have an RO-page and an ARG-page, however, their specification in the HCV is optional. If either is not explicitly specified, it is generated automatically.

LynxSecure uses the RO-page to pass read-only configuration information and the ARG-page to pass the startup parameters to a given subject. The Chapter 3, "*Read-Only and Argument Pages*" in *LynxSecure 6.3.0 API Guide* provides further data on the RO-page and the ARG-page.

The next memory flow is an optional BIOS flow which allows the subject to inspect the BIOS of the host system:

```
<memoryflow memregname="BIOS" gva="F0020000" readperm="ALLOW"/>
```

The memoryregion named "BIOS" is defined as follows:

```
<memoryregion memregionname="BIOS" memorytype="BIOS" size="020000">
  <starthpa>000E0000</starthpa>
</memoryregion>
```

The association between the PV subject and the BIOS region is as follows:

- The BIOS flow is a flow to a memoryregion of type "BIOS", associating the virtual address of 0xF0020000 with the BIOS physical address of 0xE0000, for 0x20000 bytes. The BIOS type should only be used for regions overlapping the host firmware memory and memory that is not considered available RAM (reserved memory).
- The host BIOS is typically split into two regions: the BIOS proper and the Video BIOS. A subject with access to the VGA or graphics resources may be granted a flow to the Video BIOS region.

The next memory flow of this subject is to a special BOOTSTRAP region. The BOOTSTRAP memory flow is optional; if not explicitly specified, it is generated automatically. The contents of this region are set automatically; it can't be used for any other purpose. This region contains bootstrap code that allows the subject's memory initialization to be performed in the context of the subject itself. Please refer to section "Subject Initialization" in *LynxSecure 6.3.0 API Guide* for more details on the BOOTSTRAP memory region.

The BOOTSTRAP flow is defined as follows:

```
<memoryflow memregname="BOOTSTRAP" gva="F1000000" readperm="ALLOW"/>
```

These flows associate with the following memoryregion elements:

```
<memoryregion memregionname="BOOTSTRAP" memorytype="BOOTSTRAP" size="1000"/>
```

The final memory flow within this subject is a flow to a shared memory region:

```
<memoryflow memregname="SHAREDMEM" readperm="ALLOW" writeperm="ALLOW"/>
```

It associates with the memoryregion named "SHAREDMEM" which is of type "SHM" (shared memory):

```
<memoryregion memregionname="SHAREDMEM" memorytype="SHM" size="100000" fill="00"/>
```

The "SHAREDMEM" memoryregion above also makes use of the fill attribute, so that the memory region will be initialized with the value specified by the fill attribute during LynxSecure startup and prior to subject execution. In our case, the memoryregion is initialized with zeros.

The remainder of the flows in this subject are as follows:

```
<externaldeviceflow externaldevname="NET1" readperm="ALLOW" writeperm="ALLOW"/>
<externaldeviceflow externaldevname="SERIAL1" readperm="ALLOW" writeperm="ALLOW"/>
<absoluteclockflow absclockname="Master_timepiece" readabsclockperm="ALLOW" writeabsclockperm="DENY"/>
<auditflow auditname="audit_master_2000" readperm="ALLOW" writeperm="ALLOW"/>
<virtualprocessor vpname="pvsubject_virtproc1"/>
```

The device flows are covered in the next section. The absolute clock and auditflow elements are covered in subsequent sections. They are optional flows.

The virtualprocessor element is mandatory:

```
<virtualprocessor vpname="pvsubject_virtproc1"/>
```

Subjects may have more than one `virtualprocessor` element, one for each VCPU in the subject. Each VCPU's name must be unique. Virtual processor names are used in scheduling policies, which determine when a subject's virtual processors execute on the physical CPUs. For example:

```
<schedulingpolicy policyname="schd_normal_operation_policy">
  ...
  <majorframe processorname="phys_core_1">
    <minorframe timeslice="5" vpname="cbit1_virtproc1"/>
    <minorframe timeslice="50" vpname="pvlinux1_virtproc1"/>
    <minorframe timeslice="50" vpname="pvsubject_virtproc1"/>
  </majorframe>
</schedulingpolicy>
```

Guest and Host Addresses

Just like the host physical memory addresses of the subject's `memoryregion` elements may be either explicitly specified in the configuration or chosen automatically by LynxSecure, guest addresses of `memoryflow`'s and other objects in a subject's guest address space may be either specified explicitly or allocated automatically by setting the attribute value to `AUTO`. There are constraints on guest addresses that depend on the subject type.

The `gva` attribute should only be used with paravirtualized subjects. It specifies the guest *virtual* address of the initial mapping of a memory region, that is its address in the initial layout of the guest virtual address space. The subject can later modify the initial layout: map regions at different addresses, unmap them, etc. The subject may also create a completely new page table with arbitrary mappings. Therefore, it is possible for a paravirtualized subject to start with some regions not mapped and map them during the subject's startup. If the attribute is not specified for a memory flow, the memory region is not mapped in the initial virtual address space. If the attribute has the value `AUTO`, the guest virtual address is picked automatically by LynxSecure.

For both para- and fully virtualized subjects, the `gpa` attribute specifies the guest *physical* address of a memory region, that is its address in the subject's guest physical address space. Unlike the guest virtual address space layout, the guest physical address space layout is mostly static during the subject's lifetime. If the memory flow has no `gpa` attribute or it has the value `AUTO`, the guest physical address is picked automatically by LynxSecure. If the attribute has the value `NONE`, the memory region is not present in the guest physical address space of the subject. In that case, the subject can't map it to its virtual address space and therefore can't access it directly.

32-bit subjects are expected to use guest addresses in the range of 0 to 2^{32}-1, and 64-bit subjects use 0 to 2^{64}-1.

Guest address ranges for memory regions cannot overlap. Unless automatic allocation is requested, guest addresses may need to be adjusted in case the memory flow configuration changes for a given subject.

For example, looking at the previously described subject's flows, and their guest virtual addresses, if the RO-page size increased from 32KB to 64KB, then the virtual addresses would require adjustment from the original:

```
<memoryflow memregname="pvsubject_LSK_RO_PAGE" gva="F0008000" readperm="ALLOW"/>
<memoryflow memregname="pvsubject_LSK_ARG_PAGE" gva="F0010000" readperm="ALLOW"/>
<memoryflow memregname="BIOS" gva="F0020000" readperm="ALLOW"/>
```

To this:

```
<memoryflow memregname="pvsubject_LSK_RO_PAGE" gva="F0008000" readperm="ALLOW"/>
<!-- bumped the ARG-page virt addr up by 32KB: -->
<memoryflow memregname="pvsubject_LSK_ARG_PAGE" gva="F0018000" readperm="ALLOW"/>
<!-- the BIOS virt-addr is OK, as the size of the ARG-page is only 4KB, so no overlap -->
<memoryflow memregname="BIOS" gva="F0020000" readperm="ALLOW"/>
```

Device Flows and External Devices

In the example below, there are two device flows in this subject, one to an E100 Ethernet device, and one to the serial device:

```
<externaldeviceflow externaldevname="NET1" readperm="ALLOW" writeperm="ALLOW"/>
<externaldeviceflow externaldevname="SERIAL1" readperm="ALLOW" writeperm="ALLOW"/>
```

The E100 Ethernet device `externaldevice` element is specified as follows:

```
<externaldevice externaldevname="NET1" devid="80861229" bustype="PCI">
  <!-- externaldevice specific info -->
</externaldevice>
```

The association between the `externaldeviceflow` and the external device above is as follows:

- The `externaldevname` "NET1" specified in the flow associates the flow to the `externaldevice` with the unique identifier "NET1".
- The `externaldevice` has a "`devid`" which is either a PCI ID or a PNP ID. In this case, it is the PCI vendor/device ID. Vendor is Intel (0x8086), device is E100 (0x1229).
- The `bustype` set to "PCI" indicates the device is a PCI device. Had the `bustype` been set to "legacy" (or omitted), the device would be an ISA legacy device.

In case there are multiple devices of the same vendor ID and device ID, one may use the `busnum`, `devicenum` and `funcnum` attributes to distinguish between the devices.

In general, external devices cannot be shared by subjects and must be assigned with both read and write permissions.

There is an exception for pseudodevices. If the platform uses Devicetree to describe external devices, it may contain device nodes that only contain information and do not correspond to any actual device. These are still external devices from the LynxSecure point of view. These device nodes may be assigned to multiple subjects and they may be assigned read-only. The exact criteria are:

Devices that don't have any of the following resources (pseudodevices) may be shared by multiple subjects and may be assigned to a subject read-only. Devices that do have any of the following resources cannot be shared by multiple subjects and must be assigned to a subject with both read and write permissions:

- I/O memory
- I/O ports
- Interrupts
- PCI configuration space (that is, any PCI device)

Inside the `externaldevice` element are the device-specific properties:

```
<externaldevice externaldevname="NET1" devid="80861229" bustype="PCI">
  <deviceio devionamesuffix="_DEV_IO" barnum="1"/>
  <devicememory devmemnamesuffix="_DEV_MEM0" barnum="0"/>
  <devicememory devmemnamesuffix="_DEV_MEM1" barnum="2"/>
</externaldevice>
```

These specific properties are derived from the knowledge of the specific vendor and device. The above properties specify that for the E100 Ethernet device, there is one PCI I/O BAR and two PCI memory BARs. Those properties, including the number and type of PCI BARs can be derived from several sources, including a PCI probe of the target hardware, and device reference manuals. One method includes booting an OS natively on the target, then running a utility to display the characteristics of the system's PCI devices.

NOTE: Specifying BARs is optional for PCI devices; if unspecified, LynxSecure detects them automatically. However, the I/O regions of *legacy* devices must always be specified. Also, the IRQ should not be specified for PCI devices (it is determined automatically), but must be specified for legacy devices (if the device does generate interrupts).

Virtual Device Flows and Virtual Devices

Similarly to physical devices, `virtualdevice` and `virtualdeviceflow` elements define the emulated (virtual) devices. Virtual device descriptions should be created using the Hardware Discovery Linux as described in the *LynxSecure 6.3.0 Getting Started and Configuration Guide*.

Message Buffer Flows

Message buffers are uni-directional data transfer channels that are used to transmit small fixed-size packets of data via a hypercall to LynxSecure. For example, message buffers are used for communication between the IBIT subjects:

```
<messagebuffer msgbufname="IBIT_master_to_slave1_msg_buffer"/>
<messagebuffer msgbufname="IBIT_slave1_to_master_msg_buffer"/>
```

All subjects, not just the BIT subjects, can use message buffers. The `buffersize` must be kept fixed at 64. Each buffer is of size 4096 bytes, and can store up to 64 messages. Like the other elements within the XML file, each message buffer is given a unique name.

It is within the flows to the message buffers, that the direction of the data transfer is specified. Example, the `IBIT0` subject is given a `messagebufferflow` named "`IBIT_master_to_slave`" with write permissions:

```
<messagebufferflow msgbufname="IBIT_master_to_slave1_msg_buffer" writeperm="ALLOW"/>
```

And the IBIT1 subject is given a `messagebufferflow` of the same name (so it associates with the same `messagebuffer`), with read permissions and an IRQ specified:

```
<messagebufferflow msgbufname="IBIT_master_to_slave1_msg_buffer" irq="128" readperm="ALLOW"/>
```

Those two flows specify the directionality of the data flow, and the interrupt request number (`irq`) used to inform the receiver of the message.

The IRQ is injected to the receiver subject as a synthetic interrupt.

Audit Flows

In this example, the subject has an `auditflow` that allows it to read and write audit records by making hypercalls to LynxSecure:

```
<auditflow auditname="audit_master_2000" readperm="ALLOW" writeperm="ALLOW"/>
```

The CBIT subjects also have an `auditflow`, they are authorized to make hypercalls to write audit records:

```
<auditflow auditname="audit_master_2000" readperm="DENY" writeperm="ALLOW"/>
```

The `auditname` attribute refers to the `audit` element:

```
<audit auditname="audit_master_2000" maxentries="2048">
  <fullbufferaction action="DROP"/>
</audit>
```

See section “Audit” (page 21) in Chapter 3, “*Configuring LynxSecure Features*” (page 21) for more details.

Absolute Clock Flows

The example subject is given a flow to read, but not write, the absolute clock via hypercall:

```
<absoluteclockflow absclockname="Master_timepiece" readabsclockperm="ALLOW" writeabsclockperm="DENY"/>
```

Other subjects may be given additional authorities for accessing the absolute clock. For example, the flow below allows for both reading and writing the absolute clock, in addition to getting and setting the calibration value:

```
<absoluteclockflow absclockname="Master_timepiece"
    readabsclockperm="ALLOW" writeabsclockperm="ALLOW"
    readcalibrationperm="ALLOW" writecalibrationperm="ALLOW"/>
```

LynxSecure supports a single absolute clock, so a single `absoluteclock` element should be used in the XML document. The `absclockname` attribute in `absoluteclock` element provides a unique identifier for the absolute clock, which is then accessible by a subject using the `absclockname` attribute in `absoluteclockflow` element with a name that matches the name in `absclockname` attribute in `absoluteclock` element. A subject can be given discrete

permissions to read the clock (`readabsclockperm`), update the clock (`writeabsclockperm`), read the calibration (`readcalibrationperm`), and set the calibration (`writecalibrationperm`). These actions are performed via hypercalls.

Built-In Test Subject Specifics

The Built-In Test subjects include SIBIT (Startup/Initiated BIT) subjects and CBIT (Continuous BIT) subjects. The CBIT subject may run within the normal and maintenance mode scheduling policies. CBIT subjects are optional; one can create an SRP without CBIT subjects. The CBIT subjects are kernels running diagnostic tests, and can determine when to run an initiated BIT test. When a CBIT subject detects an anomalous condition in the system, a scheduling policy change is induced from the current scheduling policy to the SIBIT scheduling policy. The SIBIT scheduling policy may in turn induce transition to the Maintenance mode policy. Within the Maintenance mode, authorized subjects may make hypercalls to return execution to the normal scheduling policy.

SIBIT subjects are mandatory, however, they do not need to be explicitly defined in the HCV. If missing, LynxSecure generates them automatically. CBIT subjects, however, need to be defined explicitly if they are required.

In case there are multiple processors in the system, each of them must run an instance of a SIBIT or CBIT subject, as applicable. One of SIBIT subjects must be designated as the "master". The master SIBIT subject is connected with each other SIBIT subject with a pair of message buffers, one for reading and one for writing. An example SIBIT subject configuration for a 2-processor system is shown below:

```
<sibitschedpolicy policyname="sibitschedpolicy">
  <majorframe processorname="phys_core_0">
    <minorframe timeslice="4" vpname="ibit0_vcpu_0"/>
  </majorframe>
  <majorframe processorname="phys_core_1">
    <minorframe timeslice="4" vpname="ibit1_vcpu_0"/>
  </majorframe>
</sibitschedpolicy>
...
<messagebuffer msgbufname="ibit0_to_ibit1"/>
<messagebuffer msgbufname="ibit1_to_ibit0"/>
...
<subject sname="ibit0" type="PARAVIRT" isa="32BIT" role="IBIT" startaddr="C0000000"
    initparams="verbosity=false"
    on_fatalfault="maintenance" on_watchdog="maintenance"
    watchdog_timeout="1000" ramsize="0000000000100000">
  <messagebufferflow msgbufname="ibit0_to_ibit1" writeperm="ALLOW"/>
  <messagebufferflow msgbufname="ibit1_to_ibit0" irq="AUTO" readperm="ALLOW"/>
  <absoluteclockflow absclockname="absclock" readabsclockperm="ALLOW"/>
  <auditflow auditname="audit" readperm="DENY" writeperm="ALLOW"/>
  <virtualprocessor vpname="ibit0_vcpu_0"/>
</subject>
<subject sname="ibit1" type="PARAVIRT" isa="32BIT" role="IBIT" startaddr="C0000000"
    initparams="verbosity=false"
    on_fatalfault="maintenance" on_watchdog="maintenance"
    watchdog_timeout="1000" ramsize="0000000000100000">
  <messagebufferflow msgbufname="ibit1_to_ibit0" writeperm="ALLOW"/>
  <messagebufferflow msgbufname="ibit0_to_ibit1" irq="AUTO" readperm="ALLOW"/>
  <absoluteclockflow absclockname="absclock" readabsclockperm="ALLOW"/>
  <auditflow auditname="audit" readperm="DENY" writeperm="ALLOW"/>
  <virtualprocessor vpname="ibit1_vcpu_0"/>
</subject>
```

In the example SIBIT subjects above, all the optional configuration items are omitted. Note that the initial `BOOT` image is created automatically using the `ibitpath` attribute of the `lsenvironment` element as the image path; there is no need to explicitly define a `guestimage` for this purpose.

The CBIT subjects run independently of each other. No message buffer connections between them are required.

Each SIBIT or CBIT subject requires a memory flow to the BRMR - the BIT Results Memory Region. This is a special region used to store the results of the tests performed by the CBIT and SIBIT subjects. The specification of this flow is optional; if missing, it is automatically generated based on the subject's `role` attribute. An explicit specification is only necessary if the same region needs to be assigned to some other subject for introspection.

An example CBIT subject element is shown below.

```
<subject sname="cbit0" type="PARAVIRT" isa="32BIT" role="CBIT" startaddr="C0000000"
    initparams="verbosity=false"
    on_fatalfault="maintenance" on_watchdog="maintenance"
    watchdog_timeout="1000" ramsize="0000000000100000" maintperm="ALLOW">
  <memoryflow memregname="ibit0_BRMR" gva="AUTO" readperm="ALLOW" writeperm="ALLOW"/>
  <absoluteclockflow absclockname="absclock" readabsclockperm="ALLOW"/>
  <auditflow auditname="audit" readperm="DENY" writeperm="ALLOW"/>
  <virtualprocessor vpname="cbit0_vcpu_0"/>
</subject>
```

Both SIBIT and CBIT subjects handle the options described in Table 2-1 (page 14).

Table 2-1: CBIT/SIBIT Command Line Arguments

Option	Description
`serial`	Serial port configuration for debug output. Can be specified as `none` to disable the output to the serial port. Otherwise, a string in the following format: `[compat:]device[,baudrate[,clockrate[,lcr]]]`, where: • *`compat`* is a device compatibility string, as specified by the devicetree specification (by default, determined by the actual device's compatibility string if applicable, and on the x86 platform, "ns16550a" as a fallback); • *`device`* is the HCV name for a device or a hexadecimal address of the device's registers; • *`baudrate`* is the selected baud rate (by default, 115200); • *`clockrate`* is the device's clock rate (if not specified by the physical device; on the x86 platform, defaults to 1843200); • *`lcr`* is the desired line control settings - data bits, parity, stop bits (by default, `8n1`: 8 data bits, no parity, 1 stop bit). If this option is not specified, the serial output is directed to the first HCV flow from this subject to a serial device.
`vga`	Enable debug output to VGA if set to "true" (default: false)
`vgacolor`	Color attribute for VGA output (default: 11, "light cyan over black")

Paravirtualized Linux Specifics

LynxSecure supports paravirtualized 64-bit Linux subjects:

```
<subject sname="pvlinux1" type="PARAVIRT" isa="64BIT" startaddr="FFFFFFFF80000000"
    initparams="clocksource=tsc tsc=reliable disableapic net.ifnames=0 acpi=off" irqbase="48">
```

NOTE: PV Linux kernel requires the IRQ base of 48 specified in the `subject` element. This is different from most other subjects.

The command line options handled by PV subjects are regular Linux command line options with the addition of several options summarized in Table 2-2 (page 14)

Table 2-2: PV Linux Command Line Arguments

Option	Description
`lsk_ramdisk_region`	The name of the memory region containing the RAM disk image.

The first 40MB of the main program memory region of a PV Linux subject need to be mapped as specified below:

```
<memoryflow memregname="pvlinux1_init_image" gva="FFFFFFFF80000000" guestsize="02800000"
    readperm="ALLOW" writeperm="ALLOW"/>
```

NOTE: PV Linux kernel doesn't support RAM allocation using the `ramsize` attribute. Don't use this attribute with PV Linux subjects!

PV Linux page tables do not need to be mapped to the guest virtual address space initially. The PV kernel maps them during the subject startup. Note that these regions' specification is optional.

```
<memoryflow memregname="pvlinux1_PML4" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="pvlinux1_PDPT" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="pvlinux1_PDT" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="pvlinux1_PTE" readperm="ALLOW" writeperm="ALLOW"/>
```

Then, there are (optional) specifications for the special memory regions that need to be mapped initially. Such memory regions may be mapped in the 0xFFFFFFFFF0000000..0xFFFFFFFFF8000000 range:

```
<memoryflow memregname="pvlinux1_LSK_RO_PAGE" gva="FFFFFFFFF0000000" readperm="ALLOW"/>
<memoryflow memregname="pvlinux1_LSK_ARG_PAGE" gva="FFFFFFFFF0010000" readperm="ALLOW"/>
```

A read-only flow to `VIDEO_BIOS` memory region with read/write permission may be required in the PV Linux subject if the subject is going to use the real mode BIOS to configure the graphics. This region also does not need to be mapped initially; the subject will map it during startup. The `VIDEO_BIOS` region is defined as follows:

```
<memoryregion memregionname="VIDEO_BIOS" memorytype="BIOS" size="0000000000020000">
  <starthpa>00000000000C0000</starthpa>
</memoryregion>
```

Similarly, using real mode BIOS calls for graphics configuration requires a flow named `LOWMEM` which associates with a physical `memoryregion` that spans from physical address 0 to 640K:

```
<memoryflow memregname="LOWMEM" readperm="ALLOW" writeperm="ALLOW"/>
```

Linux needs a flow to the RAM disk (`initrd`) which contains the filesystem it will execute its `init` from.

```
<memoryflow memregname="pvlinux1_RAMDISK" readperm="ALLOW"/>
```

For the initial PV Linux boot image, the storage location of the kernel prior to its movement to its runtime location, must also use a virtual address range that does not overlap other virtual address ranges:

```
<memoryflow memregname="pvlinux1_init_image_boot" gva="FFFFFFFF10000000" readperm="ALLOW">
```

PV Linux subjects require a RAM region below the host physical address of 4GB to perform DMA from/to physical devices.

```
<memoryflow memregname="pvlinux1_lowmem" readperm="ALLOW" writeperm="ALLOW"/>
```

To ensure that the flow associates with a physical memory region below 4GB, the region must be specified with a special alignment value such as `4MbyteLow`:

```
<memoryregion memregionname="pvlinux1_lowmem" memorytype="PROGRAM" size="08000000">
  <alignment>4MByteLow</alignment>
</memoryregion>
```

Fully Virtualized Subject Specifics

In this section, a subject running fully virtualized Windows® will be described. However, the guidance below is also valid for other fully virtualized subjects as well.

The fully virtualized subject has the mode of `FULLVIRT`:

```
<subject sname="fv1" type="FULLVIRT" initparams="CD:1 DISK:2">
```

LynxSecure supports both explicit and automatic assignment of guest addresses to memory flows. For automatic assignment, the `gpa` attribute of a memory flow is set to `AUTO` or simply omitted. The automatic allocation algorithm is intentionally simple and has some limitations discussed below. Our example configuration has all guest addresses assigned explicitly for illustrative purposes. We'll look at what would change if automatic allocation was desired a little later.

The `startaddr` subject attribute should be omitted or set to `AUTO`.

The `initparams` attribute specifies the options passed to the virtual BIOS. The options supported by a fully virtualized subject's BIOS are listed in Table 2-3 (page 16). The BIOS options values are separated from the option names by a colon.

Table 2-3: Command Line Options in a Fully Virtualized Subject

Option	Description
`BOOTMENUTIMEOUT`	The period, for which the boot device selection menu is displayed, in seconds.
`CD`	The priority for booting from a CD-ROM, from 1 (highest) to 4 (lowest).
`DISK`	The priority for booting from a hard drive.
`KDI`	The priority for booting from an in-memory kernel image.
`BEV`	The priority for booting from other bootable devices (such as PXE).
`KDISRC`	The original address of an in-memory kernel image.
`KDIDST`	The destination address of an in-memory kernel image.
`KDISIZE`	The size of an in-memory kernel image.
`KDICMDLN`	The address of the command line to pass to the in-memory kernel image.
`JUMP`	The execution start address. Instead of the normal boot procedure, the virtual BIOS will jump to the specified address.
`JUMPREALMODE`	Perform the jump after switching to Real mode. Unless this option is set, the `JUMP` option transfers control to the specified address in Protected mode.
`OROMSRC`	The original address of an Option ROM (also known as the Expansion ROM) image to be deployed. This option allows one to include an Option ROM (firmware) in addition to those provided by PCI devices.
`OROMSIZE`	The size of an Option ROM to be deployed.
`OROMDEPLOY`	Force the deployment of the Option ROMs. Some Option ROMs on the x86 architecture depend on the availability of the so called "unreal mode". By default, the Virtual BIOS only deploys Option ROMs if the virtual environment supports this mode.
`PATABDE`	Force probing the legacy ("PATA") IDE devices. In particular, this option is needed to boot from an emulated legacy IDE device. By default, only PCI IDE devices are probed.
`PRINT_BOOT_IMAGE_INFO`	When this option is set to "true", FVS will print the name, GPA, size, and crc32 of discovered boot images. This option is useful during the configuration of segmented boot to verify the integrity of pre-loaded images. Note that specifying this option will increase boot time.

The flow named `fv1_FW` maps the RAM region used by the fully virtualized subject firmware into the subject's address space. It should not have its `gpa` modified:

```
<memoryflow memregname="fv1_FW" gpa="FFC00000" readperm="ALLOW" writeperm="ALLOW"/>
```

The above flow associates with the 2MB memory region below, which should not be modified either:

```
<memoryregion memregionname="fv1_FW" memorytype="PROGRAM" size="00200000"/>
```

In this example configuration, the subject is intended to run Windows and is only accessible via the Remote Desktop. The fake VGA buffer replaces the real VGA buffer to make sure that software that relies on the VGA buffer address range to be writable still works:

```
<memoryflow memregname="FAKEVGABUF" gpa="000A0000" readperm="ALLOW" writeperm="ALLOW"/>
```

The FAKEVGABUF flow associates Windows' guest physical addresses from 0xA0000 to 0xC0000 with the memory region below:

```
<memoryregion memregionname="FAKEVGABUF" memorytype="BIOS" size="020000"/>
```

NOTE: If a graphics controller is assigned to a fully virtualized subject, the subject needs a flow to the real video buffer (at the host physical address 0xA0000) and the real video BIOS (at the host physical address 0xC0000).

LynxSecure provides two alternative ways of assigning RAM to a fully virtualized subject: the simple (implicit) one using the ramsize subject attribute, and the complex (explicit) one, where all memory regions and flows are explicitly present in the configuration file. With the ramsize attribute, LynxSecure creates all subject's RAM memory regions and flows to them automatically. It would have the Windows subject element defined as follows:

```
<subject sname="fv1" type="FULLVIRT" initparams="CD:1 DISK:2" ramsize="20000000">
```

Note that the ramsize attribute is intended for simple configurations only. It assumes that all special (non-RAM) mappings in the subject address space are either below the guest physical address of 1MB or not far below the guest physical address of 4GB. If the assumption is wrong, the method's behavior is unspecified. Because the corresponding exported resource is created implicitly, another limitation of the ramsize attribute is that there is no way to reference that exported resource elsewhere in the HCV. It prevents, for example, creating audit rules for subject RAM.

The more flexible and complex way of assigning RAM to a fully virtualized subject is to define all RAM regions and flows explicitly. This way is used in the current example for illustrative purposes. RAM regions must have the memory type PROGRAM. The following memory flows describe the subject's regular RAM regions and shouldn't be modified, except for the last one:

```
<memoryflow memregname="fv1_RAM1" gpa="00000000" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_RAM2" gpa="000C0000" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_RAM3" gpa="00200000" readperm="ALLOW" writeperm="ALLOW"/>
```

The fv1_RAM1 flow associates Windows' virtual address from 0 to 0xA0000, with the memory region below:

```
<memoryregion memregionname="fv1_RAM1" memorytype="PROGRAM" size="000A0000"/>
```

The fv1_RAM2 flow associates Windows' virtual address from 0xC0000 to 0x200000, with the memory region below:

```
<memoryregion memregionname="fv1_RAM2" memorytype="PROGRAM" size="00040000"/>
```

The fv1_RAM3 flow associates Windows' virtual address from 0x200000 to 0x20000000 (512MB), with the memory region below:

```
<memoryregion memregionname="fv1_RAM3" memorytype="PROGRAM" size="1FE00000"/>
```

The fv1_RAM4 memory region's size may be modified to adjust the RAM size of the fully virtualized subject based on available system memory and should be in multiples of 2MB (0x200000). The value in the example above gives the subject 512MB of RAM.

You can see that regions fv1_RAM2 and fv1_RAM3 are mapped next to each other and form a contiguous range of PROGRAM memory in the subject. The only reason this range is split into two mappings is the alignment: the large region fv1_RAM3 should be aligned at 2MB both in host physical address space and guest physical address space. This is beneficial for performance and memory consumption in the page table pools (see below).

The following flows assign page table pools, subject memory regions containing the page table that maps the fully virtualized subject's physical memory to host physical memory. These flows and memory regions normally shouldn't require modification, but if the subject is assigned extraordinary amounts of RAM, the size of these regions may need to be increased. Keeping large memory regions assigned to a subject aligned at 2MB both in host and in guest physical address space helps reduce memory consumption in these pools and improves performance. These flows are optional; if missing, LynxSecure generates them automatically with some default sizes.

```
<memoryflow memregname="fv1_PML4" gpa="FF700000" readperm="ALLOW" writeperm="ALLOW"/>
```

```
<memoryflow memregname="fv1_PDPT" gpa="FF708000" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_PDT" gpa="FF800000" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_PTE" gpa="FFA00000" readperm="ALLOW" writeperm="ALLOW"/>
```

The following flow is only necessary for fully virtualized subjects if the target system doesn't support Extended Page Tables (EPT):

```
<memoryflow memregname="fv1_shadowpt" readperm="ALLOW" writeperm="ALLOW"/>
```

The size of its associated memory region can be modified:

```
<memoryregion memregionname="fv1_shadowpt" memorytype="SHADOW" size="100000"/>
```

In the above memory region of type `SHADOW`, the default size of the region is `0x100000`, but can be expanded to allow for larger memory overheads in the fully virtualized subject. The size should be at least 128KB per one virtual processor in the subject. This region contains shadow page tables, which may be thought of as a form of a software cache for paging related actions of the fully virtualized subject. Again, if the target system's CPU supports EPT, the flow to a SHADOW memory region is not necessary. This flow is also optional; if missing, LynxSecure generates one automatically with some default size.

The fully virtualized subject also uses the RO and ARG pages. These flows are optional; if missing, LynxSecure generates them automatically with some default sizes:

```
<memoryflow memregname="fv1_LSK_RO_PAGE" gpa="FF801000" readperm="ALLOW"/>
<memoryflow memregname="fv1_LSK_ARG_PAGE" gpa="FF811000" readperm="ALLOW"/>
```

Fully virtualized subjects require a flow to a memory region that holds the subject's initial boot image and a separate flow to the BIOS image. The initial boot image of the fully virtualized subject contains the virtual firmware and other subject support code. The BIOS binary contains BIOS software, responsible for the subject startup and booting the Guest OS. It is separate from the firmware image in order to allow its user customization. All fully virtualized subjects may share these two guest images:

```
<guestimage memregionname="fullvirt_init_image_boot" memorytype="BOOT" path="../fvs.bin"/>
<guestimage memregionname="bios_bin" memorytype="PROGRAM" path="../bios.bin"/>
...
<memoryflow memregname="fullvirt_init_image_boot" gpa="FF930000" readperm="ALLOW"/>
<memoryflow memregname="bios_bin" gpa="FF910000" readperm="ALLOW"/>
```

Here is how the fully virtualized subject's memory flow configuration would look with automatic guest address allocation:

```
<memoryflow memregname="fv1_FW" gpa="FFC00000" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="FAKEVGABUF" gpa="000A0000" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_RAM1" gpa="00000000" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_RAM2" gpa="000C0000" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_RAM3" gpa="AUTO" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_PML4" gpa="AUTO" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_PDPT" gpa="AUTO" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_PDT" gpa="AUTO" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_PTE" gpa="AUTO" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_shadowpt" readperm="ALLOW" writeperm="ALLOW"/>
<memoryflow memregname="fv1_LSK_RO_PAGE" gpa="AUTO" readperm="ALLOW"/>
<memoryflow memregname="fv1_LSK_ARG_PAGE" gpa="AUTO" readperm="ALLOW"/>
<memoryflow memregname="fullvirt_init_image_boot" gpa="AUTO" readperm="ALLOW"/>
<memoryflow memregname="bios_bin" gpa="AUTO" readperm="ALLOW"/>
```

Note that the guest address is still explicitly specified for memory flows `fv1_RAM1` and `fv1_RAM2`. This ensures the proper RAM layout in the low addresses and the proper 2MB alignment for `fv1_RAM3`. Any additional RAM flows may have their guest address auto-allocated.

RAM flows (those mapping memory regions of the type `PROGRAM`) and non-RAM flows are allocated differently. Non-RAM flows are the auxiliary regions required to run a fully virtualized subject, virtual device BARs, etc. Normally, the total size of these regions is relatively small. LynxSecure allocates non-RAM flows first, just below the guest physical address of 4GB. If non-RAM flows require more room than there is available below 4GB, LynxSecure panics with an error message during startup. This error can be resolved by explicitly assigning non-RAM regions guest addresses above 4GB in the guest physical address space.

RAM flows are allocated after the non-RAM flows are allocated. They start at the bottom of the subject's address space and occupy address space left available after the non-RAM flow allocation. If there is not enough room for all RAM

flows below 4GB, the remaining RAM flows are allocated above the 4GB mark. Note that if exceedingly large RAM mappings are used, they may not fit below 4GB; this may result in insufficient RAM allocated below 4GB and cause guest software running in the subject to fail to start. To avoid this problem when using auto-allocated RAM mappings, make sure there is an auto-allocated RAM mapping with the size of 3GB or less. Reduce that size, if the problem still occurs.

Finally, here's the specification for the example subject in case the `ramsize` attribute is used and all the optional configuration items are omitted. The ellipsis stands for flows to resources other than memory flows, which are not important in this example:

```
<subject sname="fv1" type="FULLVIRT" initparams="CD:1 DISK:2" ramsize="20000000">
  <memoryflow memregname="FAKEVGABUF" gpa="000A0000" readperm="ALLOW" writeperm="ALLOW"/>
  <memoryflow memregname="fullvirt_init_image_boot" gpa="AUTO" readperm="ALLOW"/>
  <memoryflow memregname="bios_bin" gpa="AUTO" readperm="ALLOW"/>
  ...
</subject>
```

RMRRs

Certain devices, such as chipset-integrated graphics cards and some USB controllers, require a special memory region called an RMRR (Reserved Memory Region Reporting) to be assigned to the subject and mapped at a guest physical address matching its host physical address. Any subject that has a flow to a device requiring an RMRR must also have a flow to that RMRR memory region. LynxSecure determines the RMRR locations and creates the necessary flows automatically. RMRR regions are only accessible to device DMA, but not the CPU.

NOTE: In fully virtualized subjects, the user should avoid mapping other regions such that they overlap with an RMRR (RMRR is the only type of a memory region that is allowed to overlap with another memory region in guest physical address space). Such a configuration prevents device DMA from working properly in the overlapping range. LynxSecure prints a warning during startup if it detects an overlap.

The user must be aware of the following security implications of RMRRs: when two devices associated with the same RMRR are assigned to two different subjects, this creates an implicit information flow between those two subjects through the RMRR region. Also, if one of those two devices is not assigned to any subject, there is still an implicit information flow between that device and the assigned device's owner subject through the RMRR region. Moreover, there is an implicit information flow between any unassigned device with any RMRR to that subject; that happens because all unassigned devices with RMRRs are placed in one I/O MMU domain by LynxSecure. If these flows are unwanted, the user should assign all devices that share an RMRR to the same subject. If these RMRR-related issues are detected, LynxSecure prints a warning to the diagnostic console during the boot time.

If a static configuration is desired, the user may declare the RMRRs and their corresponding flows in the HCV explicitly. In the PV Linux example, the RMRR memory region is defined explicitly as follows:

```
<memoryregion memorytype="RMRR" size="00A00000" memregionname="RMRR0">
  <starthpa>CF600000</starthpa>
</memoryregion>
```

A flow to this memory region should be created in the subject that owns the device using that RMRR:

```
<memoryflow memregname="RMRR0" gpa="XXXXXXXX" readperm="ALLOW" writeperm="ALLOW"/>
```

The RMRR memory region flow must specify the `gpa` attribute. Its value must be the same as the `starthpa` defined for the RMRR memory region; this maps the RMRR at GPA=HPA as it needs to be.

Not all devices require RMRRs. However, to determine whether or not the devices on a given target board have RMRRs, one needs to use a tool which can parse the target system's ACPI tables. In the case of a native Linux OS, `acpidump` tool can be used to dump the contents of the ACPI tables. Tools that can parse ACPI tables (such as IASL from Intel) can then be used to disassemble the tables into a human-readable format.

Optional Configuration Items

As mentioned in multiple places throughout this chapter, some configuration items in the HCV are optional and can be omitted or, in case of attributes, set to `AUTO`. In most of these cases, LynxSecure generates the corresponding object or picks the corresponding attribute value automatically. Although this greatly simplifies the configuration file, it also has some drawbacks.

First, the algorithm used by LynxSecure to pick the corresponding value may have its limitations; in most cases these algorithms were designed to work for typical configurations and weren't intended for extreme cases.

For example, LynxSecure uses its best judgment to pick memory region sizes, such as the sizes for page table pools, which could turn out to be insufficient in extreme cases. This is important since resource allocation in LynxSecure is static by design. If a region turns out to be too small during runtime, there is no way to make it bigger during runtime as well; the only solution is to define the corresponding configuration item explicitly with a manually picked value.

Second, some configurations may require explicit references to certain configuration items, which is impossible if these items are generated implicitly. For example, the system integrator may want one subject to be able to access another subject's RAM. In such a situation, the other subject's RAM cannot be defined using the `ramsize` attribute because that would make it impossible to create a memory flow to it in another subject. Instead, RAM memory regions and flows will have to be specified explicitly.

The Autoconfig tool (described in the *LynxSecure 6.3.0 Getting Started and Configuration Guide*) omits optional configuration items in the generated HCV by default. However, it can be forced to generate them using the `--explicit` option.

The following is a list of the "automatically allocated/determined/created if missing" HCV configuration items:

- SIBIT subjects and the SIBIT schedule
- PCI device I/O regions and RMRRs
- Host physical address and alignment for memory regions that don't have their start address specified
- Memory region size for memory regions of the following types: BOOTSTRAP, PTE, PDT, PDPT, PML4, ARGPAGE, ROPAGE, SHADOW, BITRESULTS, RMRR
- Subject RAM memory regions and flows (if the `ramsize` attribute is specified)
- Subject non-RAM memory regions and flows for memory regions of the following types: BOOTSTRAP, PTE, PDT, PDPT, PML4, ARGPAGE, ROPAGE, SHADOW, BITRESULTS
- Subject flows to PCI device RMRRs
- Guest physical addresses for memory flows (if the `gpa` attribute is omitted or set to `AUTO`)
- The initial guest virtual addresses for memory flows in PV subjects (the `gva` attribute must be set to `AUTO`; omitting this attribute results in the region not mapped at any address in the subject)
- Host and guest IRQs for various configuration objects
- Virtual device I/O regions, flows and guest addresses
- Virtual device guest PCI addresses
- The subject execution start address for FV subjects

CHAPTER 3 Configuring LynxSecure Features

Audit

The Audit feature has these basic auditing capabilities for LSK and trusted subjects:

- LynxSecure® runtime generation of audit events that occur during the course of subject execution;
- Temporary storage of audit events generated by the LynxSecure runtime and trusted subjects;
- Configurable audit events filter;
- Permissions for a single subject to retrieve audit events for permanent storage (see section "RETRIEVE_AUDIT_RECORD_V - Read a Record from the Audit Buffer" in *LynxSecure 6.3.0 API Guide*);
- Permissions for a trusted subject to send audit events to the LynxSecure runtime (see section "STORE_AUDIT_RECORD_V - Store Audit Record" in *LynxSecure 6.3.0 API Guide*;
- Configurable action to take when the temporary audit buffer in the LynxSecure runtime is full.

LynxSecure supports a single audit buffer, so the number of `audit` elements must be one.

Depending on the audit configuration, the size of the dynamic memory allocation heap (specified as a memory region with `SKH_HEAP` memory type) may need to be increased so that the audit filter rules bitmap, the audit buffer, and other small data structures can be allocated during LynxSecure initialization. If there is insufficient heap size specified in the HCV, LynxSecure will fail to initialize due to a "bootmem exhausted" error. Should this occur, the heap size can be increased, for example in half megabyte chunks, until the bootmem error is cleared. The size of the audit filter rules bitmap is not the same for every configuration of LynxSecure as the size of the bitmap is dependent on the number of exported resources.

The size of the audit buffer is configurable using the `maxentries` attribute in the HCV, and the value must be a power of two. The audit buffer is allocated from the LynxSecure runtime's memory space. If the audit buffer becomes full, the action to be taken is configured using `fullbufferaction`. Possible settings include halting the entire system (`SHUTDOWN`), switching to a different scheduling policy (`RESCHED`), dropping some of the audit information and only retaining a count of the dropped audit event types (`DROP`) in an overflow buffer, or transitioning the system to maintenance mode (`MAINTMODE`).

To prevent the audit buffer from becoming full of audit events not deemed important for a particular environment, `auditrule` elements can define as many audit filters as desired. If left empty, all audit events will be recorded. The `filteraction` specifies the filter action to be performed if the audit event matches the filter. The value of `filteraction` attribute can be to drop the event (`DISCARD`), halt the system (`SHUTDOWN`), store the event (`STOREONLY`), and transition the entire system into maintenance mode (`MAINTMODE`). The `STOREONLY` action should only be used to override an earlier action as described below.

A filter match is determined by searching for commonality in the audit event with the `evtype`, `initresource`, and `recipresource` fields. All populated fields in the filter must match for the audit event to match the audit filter (i.e., it is an "AND" operation). All audit events will have an audit event type, but not all audit events will include the initiating resource or recipient resource fields. The enumerated type of audit event can be filtered by `evtype`.

The audit filter in the XML Configuration Vector is governed by the following rules:

- The sequence of filters is important. A matching filter rule later in the sequence overrides a matching filter rule earlier in the sequence.
- The `evtype` and `filteraction` fields must always have a value.
- A "*" symbol represents a match for all possible values, including blanks, for that field.
- A blank in a field (`initresource` or `recipresource`) means that a match only occurs on an audit event that has a blank in that field.
- An audit event of a given type is specified as requiring either a blank or a populated value for the `initresource` field in the filter rule. The filter rule must abide by this restriction, which is per event type.
- An audit event of a given type is specified as requiring either a blank or a populated value for the `recipresource` field in the filter rule. The filter rule must abide by this restriction, which is per event type.

The following text documents possible combinations of filter behaviors using examples.

Errors in Rules Specifications

The examples below illustrate the rules which are improperly specified and are rejected by the configuration tool.

The following rules will be rejected due to event type specification being mandatory:

```
<auditrule evtype="" initresource="" recipresource="" filteraction="DISCARD"/>
<auditrule evtype="" initresource="SUB1" recipresource="" filteraction="DISCARD"/>
<auditrule evtype="" initresource="" recipresource="SUB2" filteraction="DISCARD"/>
<auditrule evtype="" initresource="SUB1" recipresource="SUB2" filteraction="DISCARD"/>
```

The following rules will be rejected due to filter action being mandatory:

```
<auditrule evtype="EVENTA" initresource="SUB1" recipresource="SUB2" filteraction=""/>
```

The following rule will be rejected if event A requires blank initiator resource specification:

```
<auditrule evtype="EVENTA" initresource="SUB1" recipresource="" filteraction="DISCARD"/>
```

Similarly, the following rule will be rejected if event A requires blank recipient resource specification:

```
<auditrule evtype="EVENTA" initresource="" recipresource="SUB2" filteraction="DISCARD"/>
```

The following rule will be rejected if event B requires non-blank recipient resource:

```
<auditrule evtype="EVENTB" initresource="SUB1" recipresource="" filteraction="DISCARD"/>
```

The following rule will be rejected if event C requires non-blank initiator resource:

```
<auditrule evtype="EVENTC" initresource="" recipresource="SUB2" filteraction="DISCARD"/>
```

The following rules will be rejected because the second rule is a duplicate of the first one:

```
<auditrule evtype="EVENTB" initresource="SUB1" recipresource="SUB2" filteraction="DISCARD"/>
<auditrule evtype="EVENTB" initresource="SUB1" recipresource="SUB2" filteraction="DISCARD"/>
```

Audit Filter: Empty

If no filter rules are specified in the HCV, no audit event ever matches. All audit events are stored in the audit buffer.

Audit Filter: Match All

The following rule matches and discards all possible audit events, including audit events with either blanks or values for `initresource`, `recipresource`.

```
<auditrule evtype="*" initresource="*" recipresource="*" filteraction="DISCARD">
```

Audit Filter: Basic Case 1

The following rule demonstrates usage of "*".

```
<auditrule evtype="*" initresource="SUB1" recipresource="" filteraction="DISCARD">
```

- Matches: event EVENTA, initiator SUB1, recipient blank, action DISCARD
- Matches: event EVENTB, initiator SUB1, recipient blank, action DISCARD
- Does not match: event EVENTA, initiator SUB2, recipient blank
- Does not match: event EVENTC, initiator SUB1, recipient SUB2

Audit Filter: Basic Case 2

The following rule demonstrates usage of a blank field.

```
<auditrule evtype="EVENTA" initresource="SUB1" recipresource="" filteraction="DISCARD">
```

- Matches: event EVENTA, initiator SUB1, recipient blank, action DISCARD
- Does not match: event EVENTA, initiator SUB2, recipient blank
- Does not match: event EVENTB, initiator SUB1, recipient SUB2

NOTE: If event A requires non-blank recipient resource, the config tool will reject this configuration as an invalid filter.

Audit Filter: Basic Case 3

The following rule demonstrates that "*" matches a blank and a populated value does not match a blank.

```
<auditrule evtype="*" initresource="SUB1" recipresource="*" filteraction="DISCARD">
```

- Matches: event EVENTA, initiator SUB1, recipient blank, action DISCARD
- Matches: event EVENTB, initiator SUB1, recipient SUB2, action DISCARD
- Does not match: event EVENTC, initiator blank, recipient SUB2

Audit Filter: Override 1

The latter rule is less restrictive, so it will override the first rule. Consequentially, first rule will never be used.

```
<auditrule evtype="EVENTA" initresource="SUB1" recipresource="SUB2" filteraction="DISCARD">
<auditrule evtype="*" initresource="SUB1" recipresource="SUB2" filteraction="MAINTMODE">
```

Audit Filter: Override 2

The latter rule is more restrictive.

```
<auditrule evtype="*" initresource="SUB1" recipresource="SUB2" filteraction="MAINTMODE">
<auditrule evtype="EVENTA" initresource="SUB1" recipresource="SUB2" filteraction="DISCARD">
```

Audit Filter: Two Non-Overlapping Rules

Two rules that don't overlap for audit events that have a blank field. This is not an override.

```
<auditrule evtype="*" initresource="SUB1" recipresource="SUB2" filteraction="MAINTMODE">
<auditrule evtype="EVENTA" initresource="SUB1" recipresource="" filteraction="DISCARD">
```

- Matches: event EVENTA, initiator SUB1, recipient blank, action DISCARD

Audit Filter: Override 3

The latter rule is less restrictive. Use of * and blank. The first rule will always be overridden.

```
<auditrule evtype="EVENTA" initresource="SUB1" recipresource="" filteraction="DISCARD">
<auditrule evtype="*" initresource="SUB1" recipresource="" filteraction="MAINTMODE">
```

- Matches: event EVENTA, initiator SUB1, recipient blank, action MAINTMODE
- Does not match: event EVENTB, initiator SUB1, recipient SUB2

Audit Filter: Override 4

The latter rule is less restrictive. The first rule will never match anything, since * includes matching on blank values (override a blank with a *).

```
<auditrule evtype="EVENTA" initresource="SUB1" recipresource="" filteraction="DISCARD">
<auditrule evtype="*" initresource="SUB1" recipresource="*" filteraction="MAINTMODE">
```

- Matches: event EVENTA, initiator SUB1, recipient blank, action MAINTMODE

Audit Filter: Override 5

The latter rule is more restrictive. Latter rule overrides a * with a value and * with a blank.

```
<auditrule evtype="*" initresource="SUB1" recipresource="*" filteraction="MAINTMODE">
<auditrule evtype="EVENTA" initresource="SUB1" recipresource="" filteraction="DISCARD">
```

- Matches: event EVENTA, initiator SUB1, recipient blank, action DISCARD

Audit Filter: Override 6

The latter rule is more restrictive. Later rule overrides 2 asterisks with values.

```
<auditrule evtype="*" initresource="SUB1" recipresource="*" filteraction="MAINTMODE">
<auditrule evtype="EVENTA" initresource="SUB1" recipresource="SUB2" filteraction="DISCARD">
```

- Matches: event EVENTA, initiator SUB1, recipient SUB2, action DISCARD
- Matches: event EVENTA, initiator SUB1, recipient SUB3, action MAINTMODE
- Matches: event EVENTB, initiator SUB1, recipient blank, action MAINTMODE

Audit Filter: Override 7

The latter rule is more restrictive. Latter rule overrides * with blank, while having * in the evtype.

```
<auditrule evtype="*" initresource="SUB1" recipresource="*" filteraction="DISCARD">
<auditrule evtype="*" initresource="SUB1" recipresource="" filteraction="MAINTMODE">
```

- Matches: event EVENTA, initiator SUB1, recipient SUB1, action DISCARD
- Matches: event EVENTB, initiator SUB1, recipient blank, action MAINTMODE

Audit Filter: Override 8

The latter rule is less restrictive. Latter rule overrides blank with *, while having * in the evtype. First rule never matches any event.

```
<auditrule evtype="*" initresource="SUB1" recipresource="" filteraction="DISCARD">
<auditrule evtype="*" initresource="SUB1" recipresource="*" filteraction="MAINTMODE">
```

- Matches: event EVENTA, initiator SUB1, recipient blank, action MAINTMODE

Audit Filter: Override 9

The latter rule is more restrictive. It only uses asterisks and blanks.

```
<auditrule evtype="*" initresource="*" recipresource="*" filteraction="DISCARD">
<auditrule evtype="*" initresource="" recipresource="" filteraction="MAINTMODE">
```

- Matches: event EVENTA, initiator SUB1, recipient SUB2, action DISCARD
- Matches: event EVENTB, initiator blank, recipient SUB2, action DISCARD
- Matches: event EVENTC, initiator SUB1, recipient blank, action DISCARD
- Matches: event EVENTD, initiator blank, recipient blank, action MAINTMODE

Audit Filter: Multiple Overrides 1

Rule 2 is more restrictive than rule 1. Rule 3 is less restrictive than rule 2. Rule 2 never matches anything.

```
<auditrule evtype="*" initresource="*" recipresource="*" filteraction="DISCARD">
<auditrule evtype="EVENTA" initresource="" recipresource="" filteraction="RESCHED">
<auditrule evtype="*" initresource="" recipresource="" filteraction="MAINTMODE">
```

- Matches: event EVENTA, initiator blank, recipient blank, action MAINTMODE
- Matches: event EVENTB, initiator SUB1, recipient blank, action DISCARD

Audit Filter: Multiple Overrides 2

Rules 2 and 3 override rule 1. Rules 2 and 3 do not overlap.

```
<auditrule evtype="*" initresource="" recipresource="*" filteraction="STOREONLY">
<auditrule evtype="EVENTA" initresource="" recipresource="" filteraction="RESCHED">
<auditrule evtype="*" initresource="" recipresource="SUB2" filteraction="MAINTMODE">
```

- Matches: event EVENTA, initiator blank, recipient blank, action RESCHED
- Matches: event EVENTB, initiator blank, recipient SUB2, action MAINTMODE
- Matches: event EVENTB, initiator blank, recipient SUB3, action STOREONLY

Subject Access to Audit

Subjects can interact with the LynxSecure audit buffer by defining `auditflow` between the subject and the named `audit`. A subject (there can be multiple) that can send audit events to the buffer should have `writeperm`. A subject (there can only be one) that can read from the buffer should have `readperm`. Once an audit event is read, it is deleted from the LynxSecure audit buffer. This subject can be notified of new audit events with a synthetic interrupt specified in vector. Note that x86 hardware and the guest operating systems have reserved interrupt vectors for pre-defined uses, so care should be taken in assigning this number for the particular target platform and target guest operating systems. Or, the developer can ensure that the interrupt service routine in the guest operating system is aware of the overloaded interrupt vector by the developer and behave accordingly.

Example Audit Configuration

An example audit XML configuration is shown below. The ellipses (...) denote where many additional XML elements would normally be located — only the relevant portions are shown for this example; this is not a valid LynxSecure XML document.

```
<hcv ...>
  ...
  <partition pname="default_partition">
  ...
    <audit auditname="audit_buffer" maxentries="1024">
      <fullbufferaction action="DROP"/>
      <auditfilter filteraction="SHUTDOWN" evtype="GET_SCHED_DENIED" initresource="SecretSubject"/>
    </audit>
    ...
    <subject sname="SecretSubject" type="PARAVIRT" isa="32BIT" startaddr="C0000000">
      ...
```

```
      <auditflow auditname="audit buffer" writeperm="ALLOW"/>
      ...
    </subject>
    <subject sname="TopSecretSubject" type="PARAVIRT" isa="32BIT" startaddr="A0000000">
      ...
      <auditflow auditname="audit buffer" readperm="ALLOW" writeperm="ALLOW"/>
      ...
      </subject>
    ...
    </partition>
</hcv>
```

In this example, the audit buffer is in the `TopSecretPartition`. The name of the audit buffer is "`audit buffer`". The audit buffer can hold 1024 entries, and when it becomes full the action is "`DROP`". There is one filter, which will match the audit event if the initiating subject for audit event `GET_SCHED_POLICY` is the SecretSubject, and the response to be taken is `HALT`. The `TopSecretSubject` has the ability to send audit events to the LynxSecure runtime audit buffer, and to read (and clear) audit events from the LynxSecure runtime audit buffer. The `SecretSubject` can only send audit events to the LynxSecure runtime audit buffer.

Memory Sanitization

Subject sanitization is an advanced configuration option. Subject sanitization is performed at a per-subject level of granularity. Each subject can have their own sanitization behaviors. A description is provided here for basic configuration guidance.

Requirements

Enabling subject sanitization requires the LSA component and the LynxSecure ODE. A subject with sanitization enabled requires two additional memory flows: a read-only one to a `guestimage` of the type `SANSRC`, containing the code that performs memory sanitization, and a read/write flow to a memory region of the type `SANDST`, which is a designated RAM region where the `SANSRC` image contents are copied to and executed. If a subject doesn't have both of these flows, Memory Sanitization is not enabled for that subject.

Building a Sanitization Image

For each subject that is to have sanitization, the following steps should be performed:

1. Determine the `mode` of subject. The sanitization image code runs in the same mode as the subject itself. For example let's assume it is a fully virtualized subject.
2. The code contained in the sanitization image (`SANSRC`) must be linked at the address that the `SANDST` region is mapped at in the subject. Therefore, that address must be picked manually such that the flow doesn't overlap with any other flows with a non-automatic address in the subject. This address will be used in the steps below. The address for the `SANSRC` flow, on the other hand, may be automatically allocated (`gpa="AUTO"`).
3. Build the sanitization image, linking it to the address selected at the previous step. Multiple subjects may share a sanitization image so long as the required address space aperture is available in all of them.

Custom LSAs may be implemented to perform the sanitization. An example sanitization LSA is included in LynxSecure in the `$ENV_PREFIX/examples/lsa` directory. The sanitization logic is implemented in the `payload.c` file. That example LSA performs the following actions:

- For each memory region of the subject of type `PROGRAM`, and of read/write permission, zero that region.
- For each memory region of the subject that has a `fill` value specified, fill that region with the fill value specified via the XML file.
- When done with such actions, make a hypercall to restart the subject.

To build the Sanitization image, perform the following steps:

- Go to the LSA ODE directory (`$ENV_PREFIX/examples/lsa`) or copy the sources from it to a new directory where the sanitization code will be developed.
- Modify the LSA code to perform memory sanitization according to your requirements.
- Change the start address in the `Makefile` to match the available aperture for the `SANDST` memory flow.

  For FV subjects the start addresses must be between 0 and 4GB. For PV subjects, the start addresses must be between 0 and 2GB.
- Run `make all` to rebuild the sanitization binary.

NOTE: The default sanitization LSA has a limitation that in paravirtualized mode it will only zero the memory regions that are mapped in the original guest virtual address space. Partially mapped memory regions will only have the mapped part zeroed.

Configuration Example

An example of an LSA subject, configured for an additional sanitization image:

```
<memoryregion memregionname="AUX_OS_RUNTIME" memorytype="SANDST" size="00100000"/>
...
<guestimage memregionname="AUX_OS_LOAD_IMG" memorytype="SANSRC" path="lsasan64.bin"/>
...
<subject sname="lsa1" ...>
  <memoryflow memregname="AUX_OS_LOAD_IMG" gva="AUTO" readperm="ALLOW"/>
  <memoryflow memregname="AUX_OS_RUNTIME" gva="30000000" readperm="ALLOW" writeperm="ALLOW"/>
  ...
```

In-focus Scheduling

The in-focus scheduling feature is an advanced feature which can be used to increase the performance of the in-focus[1] fully virtualized subject in case several subjects share one physical processor. It works by changing the schedule to one where the in-focus subject has the biggest time-slice of the subjects scheduled on that physical processor.

To enable this feature in an HCV, where the KVM feature is enabled, but the in-focus scheduling feature is not enabled, make the following changes to the HCV:

1. The VDS subject must be able to set the scheduling policies. Add the `writeschedpolicyperm="ALLOW"` attribute to the VDS subject definition.
2. The first normal scheduling policy will be used when switched to the VDS console. Define additional scheduling policies as necessary with preferential scheduling for the in-focus subject. These will be numbered 2, 3, 4, and so on.
3. In the VDS subject description, locate the `virtualdeviceflow`'s of `VIRTKBDMOUSE` devices. Each such device has two interfaces; the server interface is assigned to the VDS and the client interface is assigned to the client subject. Find the client (non-VDS) subject where an interface of the VIRTKBDMOUSE device is assigned. Determine the index of the schedule that gives preference to that subject. Modify the `virtualdeviceflow` in the VDS as follows: set the `config` attribute to `schedpolicy=N`, where N is the index of the schedule to enable when the corresponding subject is in-focus. For example, if the client interface of `VIRTKBDMOUSE0` is assigned to subject X and schedule number 2 is the one that gives a bigger timeslice to subject X, modify the VDS `virtualdeviceflow` as follows:

   ```
   <virtualdeviceflow config="schedpolicy=2" vdevname="VIRTKBDMOUSE0"/>
   ```

The following is an example of HCV scheduling policies for two Virtual KVM client subjects, where the in-focus subject gets the 90-tick timeslice and the out-of-focus subject gets the 10-tick timeslice:

[1]The in-focus subject is the subject currently switched to with the Virtual KVM console.

```
<schedulingpolicy policyname="defaultschedulingpolicy">
  <majorframe processorname="phys_core_0">
    <minorframe timeslice="200" vpname="vds0_vcpu_0"/>
  </majorframe>
  <majorframe processorname="phys_core_1">
    <minorframe timeslice="100" vpname="fv1_vcpu_0"/>
    <minorframe timeslice="100" vpname="fv2_vcpu_0"/>
  </majorframe>
</schedulingpolicy>
<schedulingpolicy policyname="fv1_infocus">
  <majorframe processorname="phys_core_0">
    <minorframe timeslice="100" vpname="vds0_vcpu_0"/>
  </majorframe>
  <majorframe processorname="phys_core_1">
    <minorframe timeslice="90" vpname="fv1_vcpu_0"/>
    <minorframe timeslice="10" vpname="fv2_vcpu_0"/>
  </majorframe>
</schedulingpolicy>
<schedulingpolicy policyname="fv2_infocus">
  <majorframe processorname="phys_core_0">
    <minorframe timeslice="100" vpname="vds0_vcpu_0"/>
  </majorframe>
  <majorframe processorname="phys_core_1">
    <minorframe timeslice="10" vpname="fv1_vcpu_0"/>
    <minorframe timeslice="90" vpname="fv2_vcpu_0"/>
  </majorframe>
</schedulingpolicy>
```

Using the USB Debug Cable/Device with LynxSecure

The USB debug cable can be used with LynxSecure to observe the debug output; the supported/tested device is "Ajays USB 2.0 debug device/cable". In order to use the debug cable with LynxSecure:

- USB legacy support should be disabled in BIOS
- USB host controller should not be assigned to any of the subjects.
- USB debug cable should be connected to a USB debug port during LynxSecure system startup.
- Specify `usb="true"` option in the XML. For example:

  ```
  <diagoutput vga="true" usb="true">
  ```

- If the USB debug device is unplugged and plugged-in again, LynxSecure will not be able to output to the device. LynxSecure should be rebooted in this scenario.

NOTE: Some motherboards may not wire the debug port to a physical USB port; in this case, the USB debug cable/device cannot be used.

How to Identify the Correct USB Connection/Orientation for the Debug Device/Cable

From the Ajays USB 2.0 debug device's appearance, it's hard to distinguish which end to connect to the host and which end to the target.

To confirm proper orientation, connect one end of the device to the host. If oriented and connected properly, the Windows®/Linux® host should NOT detect it. If it is detected, connect the opposite end of the debug cable to the host. Connect the open end to the target. When the target is powered-on, the Windows/Linux host should find the USB debug cable attached.

How to Identify the USB Debug Port on a Target

Not all USB ports on the target are USB debug ports. Only one of them is a USB debug port.

Refer to Step 1 in the following link [http://blogs.msdn.com/b/usbcoreblog/archive/2010/10/25/setting-up-kernel-debugging-with-usb-2-0.aspx] on how to identify the USB debug port using Windows.

How to Identify the USB Debug Port with LynxSecure

Connect the debug cable as mentioned above, boot LynxSecure and check for the output on the host. If you see LynxSecure output (see below for an example), that is the debug-port on that target. Otherwise, connect the debug cable to a different port on the target and boot LynxSecure and check for the output on the host. Repeat this step until you see the output. A rule of thumb is that the device should be powered by the target.

LynxSecure sample output on the host (the debug cable appears as `/dev/ttyUSBx` on the host when the target is powered on):

```
[root@star62]# cat /dev/ttyUSB0

       Starting up the other CPUs...
            CPUs online: #0 #1
        Initializing Scheduler...
        Initializing the VCPU module...
        Initializing Device Configuration Virtualization...
        Initializing Subject Resources...
LS<3> DCV: device AUDIO1 (01:00.1) doesn't support generic software reset
        Initializing I/O MMU...
            Remapping unit #0: DMA (3/256 domains, 7/20 devices), Interrupts
            Remapping unit #1: DMA (2/256 domains, 1/1 devices), Interrupts
        Initializing Interrupt Routing...
        Initializing Hypercalls...
        Heap memory used by LynxSecure: 9705248 (0x941720) bytes
        Enabling I/O MMU...
        Launching Subjects
```

Using Identity-Mapping to enable DMA on Systems with No I/O MMU

If the target platform does not have an I/O MMU, such as the VT-d technology on the x86 platform, or it is disabled, there are limitations for physical devices that utilize DMA to transfer data to and from system memory. If such a device is assigned to a subject, its DMA transfers may not work and may corrupt random memory. The reason is that the guest physical address (GPA) for RAM in a subject differs from its address in the host physical address (HPA) space. Guest software running in the subject programs DMA-capable devices using guest physical addresses. In the absence of an I/O MMU, those guest addresses are not translated to the host ones in bus transactions issued by DMA-capable devices. As a result, those devices do not work as expected.

There is a way to circumvent this limitation: make the subject use an identity map between the host physical and the guest physical address spaces. In such a configuration, DMA address translation is not required, because host physical addresses match the corresponding guest physical addresses.

In case the target system has no I/O MMU or it is disabled in the configuration, LynxSecure automatically forces PV subjects with assigned physical devices to use identity-mapped GPA space. Normally, it is possible with any number of PV subjects, as long as PV software running in those subjects is prepared to deal with sparse RAM in its GPA space and the target system has enough RAM to accommodate them. However, if those PV subjects are 32-bit, they will compete for RAM below the 4GB HPA mark, which limits the number of such PV subjects.

For FV subjects, LynxSecure doesn't enable identity-mapping automatically. If one is desired, it must be configured explicitly in the HCV by setting the subject's RAM regions to use identical HPA and GPA. In the HCV, it is done by setting the `starthpa` (page 66) element of the RAM memory region to the same value as the `gpa` attribute of that region's memory flow in the subject. Such a configuration can also be generated automatically using the Autoconfig Tool with the `identitymem` subject option.

On the x86 platform, only one FV subject may be configured as identity-mapped, because guest software expects certain GPA ranges to be occupied by RAM (for example, the range starting at the GPA of 1MB and up), but only one subject can have that host memory range assigned to it. This limitation is less severe on the ARM platform, because RAM location in the physical address space is more flexible on that platform and guest software is more likely to be able to deal with an arbitrary RAM layout.

CAUTION! While identity-mapping may allow a subject to use direct assignment of DMA-capable devices, the hypervisor cannot guarantee subject isolation in the absence of an I/O MMU. A malicious or misbehaving subject may program the DMA engine of a device to read from or write to the physical memory that belongs to a different subject. In the absence of an I/O MMU, DMA-capable devices should only be assigned to trusted subjects.

The range occupied by the RIF and SKH binaries must be excluded from the identity-mapped subject's RAM, sometimes creating a "hole" in the subject's RAM (depending on the amount of subject's RAM and the RIF and SKH load address).

By default, LynxSecure RIF and SKH binaries occupy the physical address range just above the address of 1MB. However, on the x86 platform, the GuestOS running in the identity-mapped subject usually needs that range of its RAM to be available and would fail to start if it isn't available. To enable the identity-mapped configuration, RIF and SKH must be moved to higher load addresses. Addresses above the identity-mapped RAM range for the FV subject should be chosen in order not to create a hole in the subject's RAM. See section “The Host Physical Memory Layout and Customizing the Load Addresses” (page 30) for how to configure RIF and SKH load addresses.

Note that on the x86 platform, RIF and SKH must be located in the bottom 2GB in the host physical address space. Therefore, it makes sense to load RIF at address 2GB-16MB (7F000000 hexadecimal) and the SKH 3MB above that (at 7F300000 hexadecimal) in case an identity-mapped FV subject is present in the configuration, assuming the target system has RAM at those addresses.

The Host Physical Memory Layout and Customizing the Load Addresses

The host physical memory layout is created by LynxSecure RIF at runtime using the following rules:

- Memory regions that have their start HPA specified in the HCV, occupy the corresponding address ranges.
- I/O Memory regions of hardware devices and other regions whose location is fixed and determined by the platform rather than the HCV, occupy the corresponding address ranges.
- RIF and SKH binaries are placed at the HPA called "the load address". If desired, the load address may be configured in the HCV independently for RIF and SKH. The RIF and the SKH along with their data form a single contiguous area in RAM; the range between the end of the RIF binary and the start of the SKH binary is used for the RIF dynamic memory allocation heap. It is not available for allocation of arbitrary location memory regions.
- If there is any overlap between the above "fixed location" memory regions, LynxSecure stops with an error.
- Memory regions without a configured start address ("arbitrary location" regions) are assumed to be RAM and are allocated from the remaining available RAM. RIF reports the amount of host RAM required by the arbitrary location regions and the amount available for their allocation to the diagnostic console during LynxSecure startup.
- The arbitrary location memory regions may have limitations on their location, such as alignment and the "low" memory requirement. The low memory is memory in the bottom 4GB of the host physical address space, which corresponds to a 32-bit physical address width. For example, all memory regions belonging to identity-mapped 32-bit PV subjects[2] are restricted to low memory, in order to allow correct operation of 32-bit guest software that doesn't support physical addresses wider than 32 bits. This makes low memory a precious resource in a configuration where a lot of memory regions have this limitation and effectively limits the number of identity-mapped 32-bit PV subjects sharing a single system.
- Arbitrary location RAM regions belonging to FV subjects are considered to be "choppable". It means that if the region cannot be allocated in a single physically contiguous range (in the host physical address space), it may

[2]Identity-mapped PV subjects are used on systems with no I/O MMU; all RAM in such a subject is mapped such that its guest physical address is equal to its host physical address.

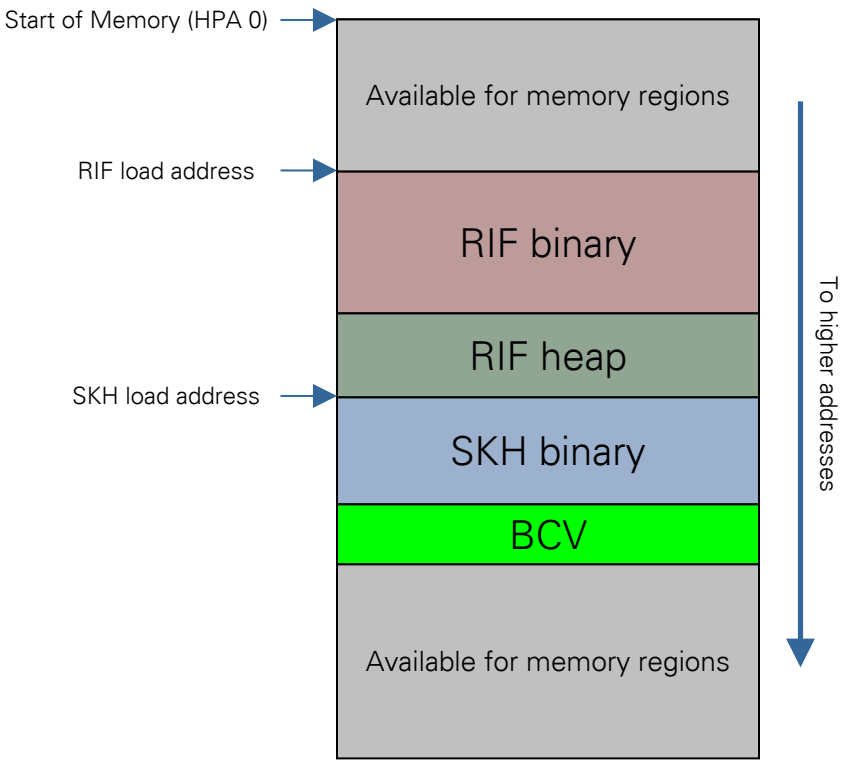
be split ("chopped") into two or more physically contiguous ranges, each of which will be a separate memory region. These memory regions combined will be physically contiguous in the guest physical address space, but not in the host physical address space. This doesn't affect the correct operation of DMA-capable devices as long as the platform contains an I/O MMU and it is enabled.

- If arbitrary location memory region allocation fails, LynxSecure stops with an error.
- Any RAM outside of the RIF/SKH address range and of the memory regions remains unused. SKH uses the static memory layout created by RIF and does not utilize the unused RAM dynamically at runtime. RIF reports the amount of unused RAM to the diagnostic console during LynxSecure startup; that information may be used to avoid wasting the target system's RAM.

As mentioned above, RIF and SKH load addresses may be customized in the HCV. When changing those load addresses, keep in mind the following:

- Because the range between the end of the RIF binary and the start of the SKH binary is used for the RIF dynamic memory allocation heap, the SKH binary should be located within several megabytes above the start of the RIF binary. For example, on the x86 platform, RIF is located at the HPA of 1MB and SKH is located at the HPA of 4MB by default.
- On the x86 64-bit platform, RIF and SKH cannot be located higher than 2GB (80000000 hexadecimal) in the host physical address space. This is due to the code model limitations imposed by the C compiler.
- The RIF binary, the RIF heap, the SKH binary, and the BCV (Binary Configuration Vector, which is the binary representation of the HCV) form a contiguous range that must be available in the target system's physical memory when the LynxSecure SRP image is loaded and started. The exact addresses and sizes of those SRP modules may be found out from the SRP image using the `vsrp -r filename.srp` command. However, a simple rule of thumb that is good in most cases is that at least 16MB of RAM must be available starting at the RIF load address.

Figure 3-1: LynxSecure SRP Module Placement in Host RAM



The figure above shows the relative locations of LynxSecure SRP modules in the target system's physical memory after the SRP starts execution. Memory above and below the shown range may contain arbitrary memory regions. However, if any configured or automatically discovered memory region with a fixed start address overlaps with that range, LynxSecure will fail to start.

Sometimes it may be desired to set some region of RAM aside, so that neither LynxSecure itself nor the subjects use that RAM. It can be achieved by creating a memory region with the type `RESERVED` and explicitly specified start HPA and size in the HCV and not assigning it to any subject. However, that memory region must not overlap with LynxSecure SRP modules in the figure above; if it does, the load addresses of RIF and SKH must be changed.

To change the load addresses for RIF and SKH, use the `loadaddr` attribute of the section "<rif> — Options to RIF" (page 43) and section "<skh> — Hypervisor Options" (page 44).

NOTE: The Autoconfig Tool can select the loading addresses for RIF and SKH automatically using the `identitymem` option.

Enabling Additional Modules

Additional modules may be added to the core hypervisor functionality that is not included into the stock binary. Table 3-1 (page 32) lists the currently available modules. To enable them, list the modules' name(s), space-separated, in the `modules` attribute of section "<skh> — Hypervisor Options" (page 44) or section "<rif> — Options to RIF" (page 43), for example:

```
<lsenvironment …>
    <skh modules="skdb hvcall_subject_log startup_timing"/>
    <rif modules="startup_timing"/>
     …
</lsenvironment>
```

Table 3-1: Modules available in LynxSecure

Module name	Description
`hvcall_subject_log`	Enables the `HVCALL_SUBJECT_LOG` hypercall. This hypercall allows subjects to output messages to the hypervisor's diagnostic console. It is not included in the stock SKH binary because it may be used to violate the hypervisor timing constraints.
`skdb`	Enables the Separation Kernel Debugger (SKDB) and the hypercalls associated with it (`HVCALL_SKDB_ENTER` and `HVCALL_SKDB_EXECUTE`). The SKDB has limited functionality in the current LynxSecure release and should be considered experimental.
`paniccall`	Makes LynxSecure transfer control to a configured location upon encountering most fatal errors. Very early failures and hangs may not get detected. See the `paniccall` description in section "Passing Options to RIF from a Custom Bootstrap Loader" in *LynxSecure 6.3.0 Getting Started and Configuration Guide* for further details.
`dump_bcv`	Prints the contents of the Binary Configuration Vector (BCV) received by the hypervisor, after it may have been augmented by RIF, to the configured consoles.
`dump_bsp_skh`	Prints the contents of the BSP_SKH structure that conveys certain hardware information collected by RIF, to the configured consoles.
`dump_ropage`	Prints the contents of RO pages for all subjects before launching them, to the configured consoles.
`startup_timing`	Enables printing of timestamps during startup. These timestamps are stored into an internal datastructure and printed to the configured console right before control is handed off to the subjects.

Additionally, you may have received modules to allow certain devices to operate under the hypervisor.

Booting SKH Runtime Package Using the Trusted Initialization

A Trusted Platform Module (TPM) is a hardware device specified by the Trusted Computing Group in the Trusted Platform Module Specifications Rev 1.2.

Trusted Initialization is one of the many aspects of trusted computing for which the TPM was designed. The Trusted Initialization feature makes use of a TPM's Platform Configuration Registers (PCRs) to store measurements of each software component executed in the initialization of the LynxSecure Security Function (LSF). Measurement refers to a cryptographic digest (hash) of software, firmware, and/or data. The algorithm used for computing measurements is the SHA-1 message digest algorithm.

To make use of the Trusted Initialization feature, TPM must be enabled in the BIOS.

During the initialization of the LSF, each initialization stage will compute a measurement of the subsequent stage prior to transferring control. Specific measurements are assigned to specific PCR numbers. This assignment provides the verifying party with some information about what piece(s) of software may have been compromised. The LynxSecure tool `rmltool` will supply reference values for PCRs.

Table 3-2: PCR Assignments

PCR Index	Measurement(s) Extended for	Measured by
5	Bootloader (2nd stage)	Bootloader (1st stage)
8	RIF	Bootloader (2nd stage)
9	SKH	RIF
11	Configuration Data (e.g. BCV, MRH)	RIF
12…15	Subject Software Modules	RIF

The verifying party will need to have the expected reference values for the PCRs listed in Table 3-2 (page 33) The `rmltool` utility provides these reference values for the System Integrator's convenience.

To create an SRP with Trusted Initialization feature, please follow the steps shown below.

The step below creates the SRP on a drive and the system can be booted from the USB flash drive. This assumes the USB flash drive is `/dev/sdb`. To find the actual device node on a Linux system, insert the USB flash drive and run `dmesg` to see what device node was created.

1. Ensure that the current user can access the device for writing. This may require logging in as `root`, or adding the current user to the appropriate groups (e.g. `disk`).
2. Set up the LynxSecure cross development environment by sourcing the `SETUP.bash` script. Then run the `install-srp` script with the `-t` to enable trusted initialization, for example:

```
$ install-srp -d /dev/sdc1 -f sample.srp -t
SHA-1 digests of the installed bootloaders:

  1st stage: 0da73c30b26054a4b4d8af60555f61c5a4acd8fb    (extended by BIOS)
  2nd stage: 53f3584e64ea4dead91ecf40ebfc44952d8cf4b8    (extended by 1st stage into PCR 5)
```

NOTE: The hash digests printed by the `install-srp` script can be supplied to the `rmltool`. However, the BIOS extends the digest of the first stage bootloader into some PCR, and the first stage bootloader may not be the only digest that is extended into that PCR.

Similarly, the digest of the second stage bootloader is extended into PCR 5. However, BIOS may have extended other digests into that PCR prior to handing off the control to the LynxSecure bootloader.

If the base values in the PCRs are known, `rmltool` can be used to calculate the expected PCR values. For example, if the first stage bootloader digest is extended in PCR 4, and the base values for PCRs 4 and 5 are, respectively, `3a3f780f11a4b49969fcaa80cd6e3957c33b2275` and `0e4b9e4a25b6a21485a1ffb7d3b80b500909c5a8`, the above output from the `install-srp` can be supplied to `rmltool` as follows:

```
$ rmltool -r 4 -i 3a3f780f11a4b49969fcaa80cd6e3957c33b2275 -d 0da73c30b26054a4b4d8af60555f61c5a4acd8fb \
    -r 5 -i 0e4b9e4a25b6a21485a1ffb7d3b80b500909c5a8 -d 53f3584e64ea4dead91ecf40ebfc44952d8cf4b8
PCR[04]: BD4F2EABEDA17CD8312466CBACB2DF223C131FDA
PCR[05]: 7B0B093988F4C98B0139C38E9C6856267CB0D5A3
```

3. Insert the USB flash drive in the target system. Power up the target system and get into BIOS setup window and select the boot device menu. USB flash drive should be selected as the first one in the Boot priority. Make sure the TPM option under Security menu is enabled. Save the changes and exit the BIOS screen. The target system will boot from the USB flash drive.

The `rmltool` provides expected values for the PCRs which can be compared with the PCR values that are output during the bootup of the SRP on the target system. To obtain the `rmltool` utility provided reference values, please follow the steps shown below.

```
$ rmltool -x sample.srp
PCR[08]: 9C96CE3AE65A5738B4D4B882E5DB37614271CDCC
PCR[09]: EE8406A14DCE9021711F8627960A190C2E229806
PCR[11]: D6DBE0B13CEA5F23F12B2F2932E8F817BA77420F
```

Enhancing SRPs with Digital Signature and Verification

Overview

In this version of LynxSecure, the `mkcv` utility has been enhanced to allow SRPs to be signed and the bootloader has been enhanced to allow the SRPs to be validated.

Prior to LynxSecure 5.1, a SHA256 Hash generated over the SRP protects the SRP's integrity; by signing the Hash (encrypting it with the private half of an RSA key pair) and verifying (decrypting) the signature by the LSK bootloader (using the public key portion of the RSA key pair), not only can the integrity of the SRP be guaranteed, but the originator of the SRP can be verified. Both `mkcv` and LynxSecure bootloader use OpenSSL cryptographic library for digital signature signing and verification. The bootloader uses a mini version of cryptographic library ported from OpenSSL 0.9.8j distribution.

There are different levels of encryption/signing available:

- Not signed; hash is SHA256 - in this level, there is no encryption or decryption of the Hash value.
- Private Key/Public Key signing/verification of the hash value
- Private Key/Certificate signing/verification of the hash value with the private key. The bootloader extracts the public key from the certificate.
- Private Key/Certificate/Chain of Trust - signing/verification of the hash Value with the Private key. The bootloader extracts the public key from the certificate after validating the certificate against the Chain of Trust.

- Private Key/Certificate/Chain of Trust with verification against value saved in the TPM - signing/verification of the hash Value with the Private key. The bootloader extracts the public key from the certificate after validating the certificate against the chain of trust with the SHA256 hash of the trusted chain being verified against the value saved in the TPM's NV-RAM index `0x20007801`.

Signing the SRP Files

The first level does not require encryption or decryption.

The second level allows the hash to be encrypted with a private key (i.e., signed) and then decrypted with a public key (i.e., verified) which is attached to the SRP. To provide this, use the `-K` and `-P` flags to `mkcv` to specify the private and public key files, respectively.

```
mkcv -K private-key-filename -P public-key-filename xml-filename
```

The third level uses a signing certificate which contains the public key associated with the specified private key. To provide this, use the `-K` and `-C` flags to specify the private key and certificate files, respectively.

```
mkcv -K private-key-filename -C certificate filename xml-filename
```

The fourth level provides verification of the certificate against a chain of trust. To provide this, use the `-K`, `-C`, `-U` and `-T` flags to specify the private key, certificate and chain of trust files, respectively.

```
mkcv -K private-key-filename -C certificate-filename \
    -U untrusted-chain-filename -T trusted-chain-filename xml-filename
```

The fifth level uses the same `mkcv` command as the fourth level. The difference is the addition of storing a SHA256 hash value into the target system's TPM which is used during the boot process to verify the Chain of Trust.

OpenSSL Commands to Create Test Keys and Certificates

The commands described in this section can be used to generate a sample signing certificate with a chain of trust. For details, see the documentation at http://openssl.org.

1. Prepare a `gencert.cfg` configuration file with the following content:

   ```
   [ v3_ca ]
   subjectKeyIdentifier=hash
   authorityKeyIdentifier=keyid:always,issuer
   basicConstraints = CA:true
   ```

2. Generate root (trusted) CA certificate: create RSA public/private keys, create a certificate request with these keys and self-sign this certificate request.

   ```
   openssl genrsa -out trusted.key 2048
   openssl req -new -key trusted.key -out trusted.csr
   openssl x509 -req -days 3650 -in trusted.csr -signkey trusted.key -set_serial 01 \
       -extfile gencert.cfg -extensions v3_ca -out trusted.crt
   ```

   If necessary, multiple certificates generated in the same manner can be concatenated to form a *trusted chain*.

3. Optionally, generate intermediate (untrusted) CA certificate: create RSA keys, create a certificate request and sign it using one of the trusted certificates (or, if generating multiple intermediate certificates, using one of the previously generated certificates).

   ```
   openssl genrsa -out untrusted.key 2048
   openssl req -new -key untrusted.key -out untrusted.csr
   openssl x509 -req -days 3650 -in untrusted.csr -CA trusted.crt -CAkey trusted.key -set_serial 01 \
       -extfile gencert.cfg -extensions v3_ca -out untrusted.crt
   ```

   Again, multiple certificates may be concatenated together to form an *untrusted chain*.

4. Create an application (signing) certificate, signing it with either intermediate or root CA certificate:

   ```
   openssl genrsa -out app.key 2048
   openssl req -new -key app.key -out app.csr
   openssl x509 -req -days 3650 -in app.csr -CA untrusted.crt -CAkey untrusted.key -set_serial 01 -out app.crt
   ```

5. Optionally, if using the "level 2" signed SRPs, extract the public key from the application certificate:

```
openssl rsa -in app.key -pubout -out app.pubkey
```

NOTE: The commands above will prompt the user for additional information about the issuer of the certificates. Ensure that the certificates for the CAs (certificate authorities) do not have the same subject information as the application/ server certificates.

With the example certificates generated as described above, use the `mkcv` utility to generate signed SRPs as follows:

```
mkcv -K app.key -P app.pubkey sample.xml                                  # Level 2
mkcv -K app.key -C app.crt sample.xml                                     # Level 3
mkcv -K app.key -C app.crt -U untrusted.crt -T trusted.crt sample.xml     # Levels 4/5
```

Creating the Hash Value

A new utility (`hasher`) can be used to compute the SHA256 hash of the file contents, writing the binary hash value to a new file - *`original-filename`*`.hash`.

To make the hash value that is needed to initialize the TPM, run the *`trusted.crt`* from the example above through `hasher` to create the hash file.

```
hasher trusted.crt
```

Setting up the TPM on the Target machine

NOTE: It is assumed that the target is booted with a TPM tools supported native Guest OS. The example below describes running under Linux (Red Hat® Enterprise Linux® 5.2) but may work with other versions of Linux.

You will need the following utilities installed on your target machine:

- `trousers` management tools (http://trousers.sourceforge.net)
- `tboot` tools(http://sourceforge.net/project/tboot)

TPM must be enabled in the BIOS. If it is not available on any of the setup screens (most likely the advanced screen, depending on the BIOS), then the machine does not have a TPM.

1. Get the TPM modules running

```
[root@xxx]# modprobe tpm_bios
[root@xxx]# modprobe tpm
[root@xxx]# modprobe tpm_tis force=1 interrupts=0
[root@xxx]# /usr/local/sbin/tcsd start
```

2. Verify that the modules are working and all is well.

```
[root@xxx]# tpm_version
TPM 1.2 Version Info:
Chip Version:        1.2.5.0
Spec Level:          2
Errata Revision:     2
TPM Vendor ID:       INTC
Vendor Specific data: 00050000 00010457
TPM Version:         01010000
Manufacturer Info:   494e5443
```

3. Take ownership of the TPM

```
[root@xxx]# tpm_takeownership
Enter owner password:
Confirm password:
Enter SRK password:
Confirm password:
```

You must specify passwords for the TPM and the Storage Root Key (SRK). These are entered only once during the whole set up process, so be sure to remember or record the passwords you assign. Use the following recommended names:

*machinename*_tpm123
*machinename*_srk123

NOTE: If error occurs in reference to EK, then do `sudo tpm_createek` and repeat step 3.

4. Create the LSK bootloader specific index (`0x20007801`) in the TPM NV-RAM

```
[root@xxx]$ ./tpmnv_defindex -i 0x20007801 -pv 2 -r 0x1f -s 0x20 -p owner passwd
Successfully defined index 0x20007801 as permission 0x2, data size is 32
```

5. Verify that the Index exists in TPM NV-RAM

```
[root@xxx]$ ./tpmnv_getcap
```

The response data is:

```
20 00 78 01 50 00 00 01 50 00 00 02 10 00 00 01
4 indices have been defined
list of indices for defined NV storage areas:
0x20007801 0x50000001 0x50000002 0x10000001
```

6. Write the data (`.hash` file) to the TPM NV-RAM

```
[root@xxx]$ ./lcp_writepol -i 0x20007801 -f trusted.cert.hash -p owner passwd
Successfully write policy into index 0x20007801.
```

7. Verify the data

```
[root@xxx]$ ./lcp_readpol -i 0x20007801 -f readback.hash
No size has been specified. Will read all index data.
begin to call the tss Tspi_NV_ReadValue
Successfully read value from index: 0x20007801.
[root@xxx]$ diff trusted.crt.hash readback.hash
```

Make sure both hash files are not empty.

Preventing Non-Error Messages from LynxSecure

To prevent any output from LynxSecure (except error and panic messages), the following steps need to be performed:

1. Disable all output consoles in the `diagoutput` (page 44) element in the HCV.
2. Add `quiet` option to the command line passed to the SRP by the bootloader, as described in section "Passing Options to RIF from a Custom Bootstrap Loader" in *LynxSecure 6.3.0 Getting Started and Configuration Guide*.

Other Features

Configuration of other LynxSecure features is described in the *LynxSecure 6.3.0 Architecture Guide*.

Also, Chapter 5, "*Hypercalls Alphabetic Listing*" in *LynxSecure 6.3.0 API Guide* lists the XML elements and attributes required in order for a subject to be able to perform certain hypercalls.

CHAPTER 4 XML Reference

The following tables describe the XML elements and attributes as well as their constraints.

Human-readable Configuration Vector (HCV)

<hcv> — Root Element

Description

This is the root element of a Human-readable Configuration Vector (HCV).

Attributes

Table 4-1: Attributes for <hcv>

Name	Description	Additional Constraints
`hcvschemaversion` *(hexadecimal)*	Identifies the syntax and semantics of a provided XML configuration vector as complying with this specification. A configuration tool built to this specification will reject any provided XML configuration vector that does not include the correct version information.	
`policy` *(refer to Table 4-2 (page 40))* *(optional, default: `S2R`)*	Specifies access policy for the configuration.	
`uuid` *(UUID)* *(optional, default: empty)*	A Universally Unique Identifier (UUID) that provides an unambiguous identifier for the configuration vector.	If not provided in the XML configuration vector, the configuration tool will automatically generate a value.

Child Elements

The following sequence of child elements is expected:

- `lsenvironment` *(required, single instance)*
- `systemresources` *(required, single instance)*
- `sibitschedpolicy` *(optional, single instance)*
- `schedulingpolicy` *(required)*
- `maintschedpolicy` *(optional, single instance)*
- `partition` *(required)*

Schema Type

`hcvType`

Location

- `/hcv`

Configured Security Policy

Table 4-2: Configured Security Policy

Policy	Description
`S2R`	This policy, also known as the *least privilege* policy, defines specific access rights as separate flows from the subject to every resource that the subject has access to. *Additional constraints:* • The assignment of exported resources (including subjects) to partitions is purely informational in this policy. It does not affect any access the subject may have to any resource.
`P2P`	This policy assigns all the resources to *partitions* and the access policy is defined by the partition-to-partition flows. *Additional constraints:* • This policy results in a coarse-grained specification of access rights. For example, all actions affecting subject's state (starting, stopping, suspending, injecting a synthetic interrupt, giving a timeslice in flexible scheduling) are included in the *write* permission on the partition-to-partition flow.
`P2P+S2R`	This policy combines the subject-to-resource and partition-to-partition policies. For access to be granted to a subject to access a resource, both partition-to-partition and subject-to-resource flows must exist. The partition-to-partition flow must allow the access. The subject-to-resource flow must either allow the access, or omit the corresponding permission.

<lsenvironment> — System Parameters

Description

A single sub-element that defines various system- wide parameters necessary to initialize the Separation Kernel/ Hypervisor (SKH) on the target platform.

Attributes

Table 4-3: Attributes for <lsenvironment>

Name	Description	Additional Constraints
`arch` *(refer to Table 4-4 (page 43))* *(optional, default: `x86_64`)*	The CPU architecture for which the SRP is built. This determines the LynxSecure hypervisor executable architecture, not the subject architecture, although the hypervisor architecture may impose limits on supported subject architectures.	
`ibitpath` *(string)*	The pathname of the Initiated Built-In Test object code module to be included in the	This may be an absolute or relative (to the configuration file's directory)

Name	Description	Additional Constraints
(optional, default: `ibit.bin`)	SRP. This guest image is used as the boot image for IBIT subjects by default; it can be overridden by explicitly specifying a memory flow to a different boot image in the IBIT subject description.	pathname valid on the platform on which the configuration tool is running. The IBIT image should have a Multiboot Specification-compliant header describing its load address and entry point. If the image has no Multiboot header, LynxSecure assumes the entry point is at the address 0xC0000000. The IBIT image must be that of a 32-bit paravirtualized subject. The IBIT image entry point must be within its first 4KB.
`dtpath` *(string)* *(optional, default: empty)*	The pathname of the flattened device tree binary to be included in the SRP. This structure describes the target board hardware and is usually provided by the board manufacturer.	This may be an absolute or relative (to the configuration file's directory) pathname valid on the platform on which the configuration tool is running. This attribute is ignored on the x86 platform.
`tickspersec` *(integer; >0)* *(optional, default: `1000`)*	The number of System Clock Ticks (SCTs) per second.	
`iommu` *(boolean)* *(optional, default: `true`)*	Whether or not the I/O MMU is active at runtime.	
`smt` *(boolean)* *(optional, default: `false`)*	Whether to use Symmetrical Multi-Threading (Hyper-Threading) or not; in other words, whether to use more than one hardware thread per CPU core. Note that LynxSecure represents all Virtual CPUs as individual processor cores in the subject. It means that guest software will not be able to detect that two or more VCPUs occupy hardware threads of the same physical core and will treat them as individual cores.	Making hardware threads of the same physical core run different subjects at the same time results in an uncontrollable division of the core's processing power between the subjects and a potential covert channel. Therefore enabling SMT is not recommended for security-oriented environments. Because the way LynxSecure associates configured processors with physical processors in the absence of a Hardware ID is undefined, it is recommended that Hardware IDs are specified for configured processors in the HCV when SMT is enabled. That provides precise control over which hardware thread executes which subject.
`smidisable` *(boolean)* *(optional, default: `false`)*	If set to true and LynxSecure recognizes the system's chipset, it will attempt to disable System Management Interrupts (SMIs). This could be done to guarantee adherence to scheduling requirements since SMI handlers may execute for an arbitrary period of time. However, disabling SMIs is likely to disrupt system power management. It may also affect other platform features configured in the system BIOS, such as emulating the legacy keyboard with a USB keyboard. If this option is set to false (or left unspecified), LynxSecure will keep SMIs in the state configured by the BIOS.	This attribute is ignored on platforms other than the x86.

Name	Description	Additional Constraints
`ibrs` *(refer to Table 4-5 (page 43))* *(optional, default: `subject`)*	The level of Indirect Branch Restricted Speculation support. IBRS provides hardware mitigation against Spectre variant 2 attacks. Because this feature comes with a significant performance penalty on some processors, LynxSecure normally mitigates the vulnerability using software techniques such as retpolines. Note that IBRS requires hardware support and is disabled in case the hardware doesn't support it. Some processors require a microcode update in order to support IBRS.	This attribute is ignored on platforms other than the x86.
`l1tf` *(refer to Table 4-6 (page 43))* *(optional, default: `conditional`)*	The mitigation level for the L1TF vulnerability of the x86 CPUs. The mitigation involves flushing the level 1 data cache before executing subject code. It comes with a significant performance penalty, so the user may choose the desired balance between security and performance using this setting. LynxSecure defaults to a mitigation level that provides moderate security with a less severe performance penalty than the maximum mitigation level. This mitigation's performance impact may be reduced with a CPU microcode update. If the CPU isn't vulnerable to the L1TF attack, this attribute is ignored and the mitigation is disabled.	This attribute is ignored on platforms other than the x86. This mitigation does NOT protect from L1TF attacks on sibling Hyper-Threading threads at any mitigation level. Users should consider not using Hyper-Threading on systems with vulnerable CPUs.

Child Elements

The following sequence of child elements is expected:

- `rif` *(optional, single instance)*
- `skh` *(optional, single instance)*
- `diagoutput` *(optional, single instance)*

Schema Type

`lsenvironmentType`

Location

- `/hcv/lsenvironment`

LynxSecure architecture

Table 4-4: LynxSecure architecture

Value	Description
`x86_64`	x86 64-bit (Intel® 64).
`aarch64`	ARM 64-bit (AArch64).
`ppc`	PowerPC 64-bit.

Value

Table 4-5: Value

Value	Description
`disabled`	IBRS is disabled and not available to subjects. This also disables IBPB when switching between subjects. This is the only supported behavior if the hardware doesn't support IBRS.
`subject`	IBRS is not utilized by the hypervisor, but is available to subjects. Context switch between subjects includes an IBPB barrier. This is the default behavior if the hardware supports IBRS.
`full`	IBRS is on in hypervisor mode and available to guests. Context switch between subjects includes an IBPB barrier.

Value

Table 4-6: Value

Value	Description
`none`	The mitigation is disabled.
`conditional`	The L1 cache is flushed before executing subject code if the cache may contain data belonging to a non-current subject. This is the default behavior.
`always`	The L1 cache is always flushed before executing subject code. This is the maximum mitigation level.

<rif> — Options to RIF

Description

This element specifies the load address of the Runtime Initialization Function (RIF) in the host physical memory.

Attributes

Table 4-7: Attributes for <rif>

Name	Description	Additional Constraints
`modules` *(string)* *(optional, default: empty)*	Optional modules to be linked into the binary, space separated.	
`loadaddr` *(hexadecimal, page multiple)*	Load address for the binary.	

Name	Description	Additional Constraints
(optional, default: empty)		
`builtinargs` *(string)* *(optional, default: empty)*	This string will be used as boot parameters by RIF.	

Child Elements

None.

Schema Type

`rifBinaryType`

Location

- `/hcv/lsenvironment/rif`

<skh> — Hypervisor Options

Description

This element specifies the modules to link into the hypervisor and its load address in the host physical memory.

Attributes

Table 4-8: Attributes for <skh>

Name	Description	Additional Constraints
`modules` *(string)* *(optional, default: empty)*	Optional modules to be linked into the binary, space separated.	
`loadaddr` *(hexadecimal, page multiple)* *(optional, default: empty)*	Load address for the binary.	

Child Elements

None.

Schema Type

`createBinaryType`

Location

- `/hcv/lsenvironment/skh`

<diagoutput> — Diagnostic Output

Description

This element specifies methods the SKH can use to output diagnostic text.

If the User does not specify the diagoutput element then SKH will not produce any output.

Attributes

Table 4-9: Attributes for <diagoutput>

Name	Description	Additional Constraints
`vga` *(boolean)* *(optional, default: `false`)*	Boolean value indicates whether or not to output to the VGA console.	
`usb` *(boolean)* *(optional, default: `false`)*	Boolean value indicates whether or not to output to the USB debug port.	
`runtimediag` *(refer to Table 4-10 (page 45))* *(optional, default: `none`)*	This attribute controls diagnostic output from LynxSecure.	

Child Elements

The following sequence of child elements is expected:

- `serial` *(optional, single instance)*

Schema Type

`diagoutputType`

Location

- `/hcv/lsenvironment/diagoutput`

Diagnostic Output Settings

Table 4-10: Diagnostic Output Settings

Value	Description
`none`	No diagnostic output.
`fatalonly`	Print diagnostic information when a subject experiences a fatal fault only.
`all`	All available diagnostic output (including that of the subject fatal faults).

<serial> — Serial Port

Description

This element specifies the hardware configuration of the serial port used by SKH diagnostic output.

If the element is not specified, the default is no serial output.

Additional Constraints

If the serial port diagnostic output is enabled, avoid assigning the same serial port device to a subject. Besides garbled output, it may result in subject slowdown or freezing. LynxSecure does not prevent or warn about such an assignment.

Attributes

Table 4-11: Attributes for <serial>

Name	Description	Additional Constraints
`externaldevname` *(string ID reference)*	The name of the external device that will be used for the hypervisor's diagnostic output	The device may be used concurrently by the hypervisor and by a single subject, with the following restrictions: • The hypervisor operates the serial device in polling mode; therefore any interrupt conditions caused by the hypervisor output will appear as spurious interrupts from the device in the subject. The hypervisor only prints serious errors to the serial port after the initial start up. • The hypervisor does not reconfigure the serial device when it needs to produce any output; therefore, the subject must preserve the configuration of the serial device such as the baudrate or the line control settings.
`baudrate` *(integer; >0)* *(optional, default: `115200`)*	Specifies the baud rate for communication on the serial port.	(`clockrate` / `baudrate`) is the value used to set the divisor latch.
`clockrate` *(integer; >0)* *(optional, default: empty)*	Specifies the clock rate for the serial port. If not specified, a platform-specific default will be used (115200 on the x86 architecture; determined by the devicetree on the other architectures).	
`lcr` *(string)* *(optional, default: `8n1`)*	Specifies the line control settings. Default value is 8 data bits, 1 stop bit, no parity. The format of the string is the number of data bits, followed by a character indicating the parity (`n` for no parity, `o`/`e` for odd/even parity, `m`/`s` for one/zero parity), followed by the number of stop bits.	Not all possible combinations may be supported by the serial device on the target platform.

Child Elements

None.

Schema Type

`serialType`

Location

- `/hcv/lsenvironment/diagoutput/serial`

<systemresources> — System Resources

Description

A single sub-element that acts as a container for system resource definitions. System resources include target platform hardware capabilities that are used internally by the SKH or exported to subjects as virtualized resources.

Attributes

None.

Child Elements

The following sequence of child elements is expected:

- `mainmemory` *(optional, single instance)*
- `processor` *(1…64 instances)*
- `bridgeinfo` *(optional, single instance)*

Schema Type

`systemresourcesType`

Location

- `/hcv/systemresources`

<mainmemory> — Main Memory

Description

The physical address space on a typical platform is not contiguous, but contains "holes" that cannot be used. The purpose of the `mainmemory` is to identify addressable host memory such that the configuration tool can:

- Assign a memory region to a valid physical block if the configuration vector does not explicitly specify any host physical address for the region.
- Validate that a memory region fits within a valid physical memory block if the configuration vector explicitly specifies a host physical address for a defined memory region.

This element is optional. If missing, the available memory is detected automatically during LynxSecure startup.

Attributes

Table 4-12: Attributes for <mainmemory>

Name	Description	Additional Constraints
`hostmemname` *(string ID, up to 63 chars)*	The name of the system resource.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.

Child Elements

The following sequence of child elements is expected:

- `memblock` *(1…16 instances)*

Schema Type

`hostmemoryType`

Location

- `/hcv/systemresources/mainmemory`

<memblock> — Memory Block

Description

Defines an available memory block in host physical address space.

Attributes

Table 4-13: Attributes for <memblock>

Name	Description	Additional Constraints
`memblockstart` *(hexadecimal, page multiple)*	The start address of the memory block.	
`memblocksize` *(hexadecimal, page multiple)*	The size of the memory block in bytes.	

Child Elements

None.

Schema Type

None (defined inside an element).

Location

- `/hcv/systemresources/mainmemory/memblock`

<processor> — Processor

Description

The processor is used to define a processing element capable of providing a concurrent and independent thread of execution on the target platform. This may be a single core processor (CPU) or a core on a multi-core processor (a "logical processor" in Intel terminology). LynxSecure matches processors defined in the configuration vector to specific physical processors using constraints specified in the vector (e.g. the hardware ID) or, if no constraints, picking available CPUs in an unspecified order; all physical processors are assumed to be of equivalent capability and power. It may be desirable to assign subjects to specific processors based on the CPU's attributes such as shared caches. If that is the case processor hardware IDs must be assigned manually based on the physical processor topology.

Each identified processor is assigned at any point in time to a virtual processor in accordance with the active scheduling policy.

Attributes

Table 4-14: Attributes for <processor>

Name	Description	Additional Constraints
`processorname` *(string ID, up to 63 chars)*	The name of the system resource.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.
`hwid` *(hexadecimal)* *(optional, default: empty)*	The hardware ID of the processor. On the x86 platform, this is the CPU APIC ID. If specified, LynxSecure will select the physical CPU with the specified hardware ID to represent this configured processor. If not specified, LynxSecure will use other criteria to match this configured processor to a physical CPU.	The hardware ID may not be specified for the first configured processor. That processor is always the boot CPU (the CPU that executed LynxSecure boot sequence).

Child Elements

None.

Schema Type

`processorType`

Location

- `/hcv/systemresources/processor`

<bridgeinfo> — PCI Bridges

Description

This element specifies the number of PCI/host bridges and the number of resources reported by host bridges on the target system, to be used by the configuration tool. This information is needed to estimate the size of the read-only page.

Additional Constraints

If not specified or set to 0, the configuration tool will underestimate the amount of memory needed for the read-only page and may not detect if RO page size is not of a sufficient size. Other than that, this element is optional and is not translated into the BCV.

Attributes

Table 4-15: Attributes for <bridgeinfo>

Name	Description	Additional Constraints
`pcibridges` *(integer; >=0)*	Number of PCI bridges.	
`hostbridges` *(integer; >=0)*	Number of host bridges (reported by ACPI).	
`hbresources` *(integer; >=0)*	Number of resources reported by host bridges (via ACPI _CRS method).	

Child Elements

None.

Schema Type

`bridgeinfoType`

Location

- `/hcv/systemresources/bridgeinfo`

<sibitschedpolicy> — SBIT/IBIT Scheduling Policy

Description

The designated SBIT and IBIT scheduling policy.

See `schedulingpolicy` (page 54) description for details.

Additional Constraints

This scheduling policy specification is optional. If missing, LynxSecure automatically generates necessary IBIT subjects and the SBIT/IBIT scheduling policy. Note that any IBIT subjects explicitly present in the HCV are ignored in that case.

As a consequence, the SBIT/IBIT scheduling policy and IBIT subjects are always present in running LynxSecure instances. It is not possible to create a running instance with no IBIT subjects or scheduling policy.

Attributes

Table 4-16: Attributes for <sibitschedpolicy>

Name	Description	Additional Constraints
`policyname` *(string ID, up to 63 chars)*	The name of the policy.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.

Child Elements

The following sequence of child elements is expected:

- `majorframe` *(required)*

Schema Type

`schedulingpolicyType`

Location

- `/hcv/sibitschedpolicy`

<majorframe> — Major Frame

Description

Defines the major frame for a physical processor.

Additional Constraints

Up to 32 major frames can be specified per scheduling policy (one per processor core).

The duration of a major frame must be a minimum of 4 clock ticks, computed as the sum of the durations of all minor frames that comprise the major frame.

All major frames within the same set must be of identical duration.

Attributes

Table 4-17: Attributes for <majorframe>

Name	Description	Additional Constraints
`processorname` *(string ID reference)*	The name of the processor system resource.	A processor resource of this name must be defined as part of the `systemresources` (page 47) element.

Child Elements

The following sequence of child elements is expected:

- `dynamicsched` *(optional, single instance)*

Schema Type

`majorframeType`

Location

- `/hcv/sibitschedpolicy/majorframe`
- `/hcv/schedulingpolicy/majorframe`
- `/hcv/maintschedpolicy/majorframe`

<dynamicsched> — Dynamic Schedule

Description

Defines the schedule used in the dynamically scheduled minor frames of a major frame. The schedule lists the set of threads to run and their scheduling parameters. This dynamic schedule's runtime scope is the entire major frame, not individual minor frames where it is used. Scheduling parameters, such as `period`, include ticks consumed by intervening fixed schedule minor frames.

Additional Constraints

The same virtual processor may participate in the dynamic schedule of multiple major frames with different parameters (as long as it's the same physical processor). However, the same virtual processor may not participate in both fixed and dynamic schedules globally.

If a major frame contains no dynamic schedule, all of its minor frames must be fixed schedule.

Attributes

None.

Child Elements

The following sequence of child elements is expected:

- `thread` *(required)*

Schema Type

`dynamicschedType`

Location

- `/hcv/sibitschedpolicy/majorframe/dynamicsched`
- `/hcv/schedulingpolicy/majorframe/dynamicsched`
- `/hcv/maintschedpolicy/majorframe/dynamicsched`

<thread> — Thread

Description

Defines a thread participating in a dynamic schedule.

Attributes

Table 4-18: Attributes for <thread>

Name	Description	Additional Constraints
`vpname` *(string ID reference)*	The name of the virtual processor to run on the physical processor whenever this thread is scheduled to execute.	A virtual processor with this name must be defined as part of a subject.
`algorithm` *(refer to Table 4-19 (page 53))*	The scheduling algorithm to use.	
`priority` *(Dynamic Scheduling Priority)*	The scheduling priority of the thread. Higher values mean higher priority.	
`period` *(integer; >0)* *(optional, default: empty)*	The scheduling period of the thread. This number is interpreted according to the scheduling algorithm. Normally, it is the interval between thread capacity replenishing, in clock ticks. If this parameter is not applicable with the selected algorithm, it is ignored. If this parameter is unspecified, its default value depends on the thread's scheduling algorithm.	The scheduling algorithm may restrict this parameter's values.
`capacity` *(integer; >0)*	The initial scheduling capacity of the thread, in clock ticks. This number is	The scheduling algorithm may restrict this parameter's values.

Name	Description	Additional Constraints
(optional, default: empty)	interpreted according to the scheduling algorithm. If this parameter is not applicable with the selected algorithm, it is ignored. If this parameter is unspecified, its default value depends on the thread's scheduling algorithm.	

Child Elements

None.

Schema Type

`threadType`

Location

- `/hcv/sibitschedpolicy/majorframe/dynamicsched/thread`
- `/hcv/schedulingpolicy/majorframe/dynamicsched/thread`
- `/hcv/maintschedpolicy/majorframe/dynamicsched/thread`

Dynamic Scheduling Algorithm

Table 4-19: Dynamic Scheduling Algorithm

Name	Description
`deferrable-server`	Deferrable Server The thread is characterized by two scheduling parameters, period and (initial) capacity, both specified in clock ticks. The capacity determines how much time within each period the thread is allowed to run. Each period, the thread's runtime capacity starts at the configured initial capacity value. As the thread runs, its capacity gradually diminishes until it reaches 0. Threads with 0 capacity are taken off the CPU and preempted by other threads. The capacity is replenished to its initial value at even intervals specified by the period parameter. If a thread's period is unspecified, it is set to the containing major frame length. If a thread's initial capacity is unspecified, it is set to the thread's period. If a Deferrable Server thread goes idle (indicating that it has no work to do), its capacity is preserved. If that thread receives an asynchronous request (work to do) before the next replenish period, it becomes runnable and may resume running within its remaining capacity as long as its has sufficient priority. A thread indicates that it is going idle by executing the platform's "wait for an interrupt in a low power state" CPU instruction (`HLT` on x86 and `WFI` on Arm). Asynchronous requests are normally signaled by interrupts, but may also be other events, such as subject state management operations. Note that threads may become idle or preempt other threads at any time, not just at system clock tick boundaries. For that reason, thread runtime capacity is tracked in high resolution time units rather than system clock ticks. *Additional constraints:* • The period and capacity parameter values must not exceed 100 seconds, measured in system clock ticks. For example, if `tickspersec` value is 1000, these parameters cannot be greater than 100000.

Name	Description
`polling-server`	Polling Server This algorithm is the same as the Deferrable Server in all respects, except for the following: Whenever a Polling Server thread goes idle (indicating that it has no work to do), its runtime capacity is set to 0 until the next period and the thread is taken off the CPU. If any asynchronous requests signaling that there is work to do arrive before the replenish time, they must wait until the thread's capacity is replenished at the period boundary. A Polling Server thread is woken up (stops being idle if it is idle) at each capacity replenish. Thus, the thread may poll for work without using interrupts.

<schedulingpolicy> — Scheduling Policy

Description

A scheduling policy assigns virtual processors to physical processor cores. A scheduling policy consists of a set of major frames of identical duration, with one major frame per physical processor core. The major frame for each processor core is then sub-divided into a variable number of minor frames. A virtual processor is assigned to a physical processor core for the duration of the minor frame. The number of minor frames can be different for each major frame, but the total duration of all minor frames within a major frame must be the same for all major frames in a given scheduling policy.

Multiple scheduling policies can be defined in a single configuration vector. A given virtual processor must be assigned to the same physical processor in all configured scheduling policies (i.e., a virtual processor cannot migrate from one physical processor to a different physical processor if the scheduling policy changes); however, a given virtual processor is not required to appear in each scheduling policy. If the virtual processor does not appear in a scheduling policy, the subject that owns that virtual processor is suspended while that scheduling policy is active.

Attributes

Table 4-20: Attributes for <schedulingpolicy>

Name	Description	Additional Constraints
`policyname` *(string ID, up to 63 chars)*	The name of the policy.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.

Child Elements

The following sequence of child elements is expected:

- `majorframe` *(required)*

Schema Type

`schedulingpolicyType`

Location

- `/hcv/schedulingpolicy`

<maintschedpolicy> — Maintenance Scheduling Policy

Description

The designated maintenance mode scheduling policy. The SKH switches to this scheduling policy when it transitions to maintenance mode. Note: this document does not provide guidance on how the maintenance mode scheduling policy should be constructed.

See `schedulingpolicy` (page 54) description for details.

Additional Constraints

If this element is omitted, the system will halt if the system detects a loss of secure state.

Attributes

Table 4-21: Attributes for <maintschedpolicy>

Name	Description	Additional Constraints
`policyname` *(string ID, up to 63 chars)*	The name of the policy.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.

Child Elements

The following sequence of child elements is expected:

- `majorframe` *(required)*

Schema Type

`schedulingpolicyType`

Location

- `/hcv/maintschedpolicy`

<partition> — Partition

Description

Partition is a container for exported resources; thus, at least one partition is required.

Attributes

Table 4-22: Attributes for <partition>

Name	Description	Additional Constraints
`pname` *(string ID, up to 63 chars)*	A unique name assigned to the partition	

Child Elements

The following sequence of child elements is expected:

- `partitionflow` *(optional)*
- `platformfeature` *(optional)*
- `memoryregion` *(optional)*
- `guestimage` *(optional)*
- `externaldevice` *(optional)*
- `virtualdevice` *(optional)*
- `messagebuffer` *(optional)*
- `absoluteclock` *(optional, single instance)*
- `audit` *(optional, single instance)*
- `subject` *(optional)*

Schema Type

`partitionType`

Location

- `/hcv/partition`

<partitionflow> — Partition-to-Partition Flow

Description

Access permissions for a given subject to various resources can be specified via specific permissions on the subject-to-resource flow. In case a flow does not specify the permissions, the appropriate flow from the subject's partition to the resource's partition is consulted. The requested access is granted only if such partition-to-partition flow exists and allows the appropriate read/write access.

Additional Constraints

This element must be specified inside the subject's (not resource's) partition.

Attributes

Table 4-23: Attributes for <partitionflow>

Name	Description	Additional Constraints
`pname` *(string ID reference)*	This attribute specifies the name of the partition container for the resource being accessed.	The named partition must be defined as a `partition`
`readperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission for the read access to the resources in the specified partition.	
`writeperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission for the write access to the resources in the specified partition.	

Child Elements

None.

Schema Type

`partitionflowType`

Location

- `/hcv/partition/partitionflow`

<platformfeature> — Platform Feature

Description

A feature of the platform identified by its name. A platform feature is a functional part of the CPU or the motherboard designated as an exported resource so that it could be assigned to subjects on an individual basis. Unlike other exported resources, which may be given arbitrary names, platform feature names identify the feature and must be selected from one of the supported values.

Attributes

Table 4-24: Attributes for <platformfeature>

Name	Description	Additional Constraints
`featurename` *(refer to Table 4-25 (page 57))*	The unique name identifying the feature.	Feature names can only be the ones supported by LynxSecure. See Table 4-25 (page 57).

Child Elements

None.

Schema Type

`platformfeatureType`

Location

- `/hcv/partition/platformfeature`

Supported Platform Features

Table 4-25: Supported Platform Features

Platform Feature	Description
`CPU_RNG`	CPU built-in hardware Random Number Generator. *Additional constraints:* • Only meaningful if the CPU has a built-in hardware RNG, ignored otherwise. Note that on some platforms, the hardware RNG may get temporarily depleted if a subject asks for a new random value too often, returning a "try again later" status to the user. That state may be used for communication between the users of this feature. Thus, assigning this feature to more than one subject creates an information flow channel between them that may need to be analyzed from the security perspective.
`RDT_MONITOR`	Intel's RDT Monitoring feature access.

Platform Feature	Description
	Additional constraints: • Only meaningful if the platform supportes Intel's RDT Monitoring technology, ignored otherwise. This feature is only limited for one subject.

<memoryregion> — Memory Region

Description

A memory region is an area of memory that may be accessed by a single subject or shared by multiple subjects. Each memory region has a type that is used internally by the SKH to set up a memory environment for subjects. The type of the memory region may also constrain the information flows that are permitted.

Attributes

Table 4-26: Attributes for <memoryregion>

Name	Description	Additional Constraints
`memregionname` *(string ID, up to 63 chars)*	The name of the memory region exported resource.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.
`memorytype` *(refer to Table 4-27 (page 60))* *(optional, default: `PROGRAM`)*	The type of memory region. Used internally by the SKH to set up a suitable memory environment for subjects. See Table 4-27 (page 60) for a complete list of memory types and their constraints.	
`size` *(hexadecimal, page multiple, or AUTO)* *(optional, default: `AUTO`)*	The size of the memory region in bytes.	LynxSecure can automatically determine the size of certain auxiliary memory regions, so this attribute is optional. However, if the start HPA of a memory region is explicitly specified, then its size must be explicitly specified as well. Memory regions defined with the `guestimage` element are an exception; their size, if unspecified, is set to the file size.
`noclobber` *(boolean)* *(optional, default: empty)*	Prevent RIF from using the HPA associated with this memory region during initialization.	During early initialization, LynxSecure utilizes host memory to unpack the SRP. To protect a given memory region from being used during this process, enable this attribute. The size and HPA of the memory region must also be specified for this option to take effect. For RESERVED memory regions with a defined HPA and size, this option is enabled by default.
`fill` *(hexadecimal, 2 digits)* *(optional, default: empty)*	If specified, LynxSecure will initialize the region, setting each byte to the specified value. This only happens once at LynxSecure startup. If specified for a `guestimage`, only the part of the memory region between the end of the file image and the end of the memory region is initialized this way (the rest is initialized using the file image).	

Name	Description	Additional Constraints
caching *(one of: DEFAULT/ UNCACHEABLE/ WRITE-COMBINING/ WRITE-THROUGH/ WRITE-PROTECTED/ WRITEBACK)* *(optional, default: DEFAULT)*	The caching type to use for all mappings of this memory region. If unspecified, or set to DEFAULT, the memory will be mapped with the base caching type specified for the memory region by the host firmware (or uncacheable, if the owner subject is not allowed to use memory caches by its own caching attribute). The guest internal mapping mechanisms may override/complement the base caching type according to the host platform caching type merge rules. If this attribute is set to any value other than DEFAULT, the behavior depends on the platform. On the x86, the specified caching type will be in effect in all subjects mapping this region regardless of which caching type is specified for the memory region by the host firmware or any guest mapping mechanisms. On ARM, the specified caching type will be merged with the caching type specified by the guest internal mapping mechanisms according to the ARM platform caching type merge rules.	Use this attribute with care, because specifying a wrong memory type for a memory region may cause hardware faults. For example, device memory shouldn't be mapped cacheable.

Child Elements

One of the following child elements is expected:

- alignment
- starthpa

Schema Type

memoryregionType

Location

- /hcv/partition/memoryregion

Supported Memory Region Types

In the Table 4-27 (page 60), a readable flow is a flow which has a readperm="ALLOW" attribute. Similarly, a writable flow is a flow which has a writeperm="ALLOW" attribute. Unless specified otherwise, subject can directly read from a memory region to which it has a readable flow and can directly write to a memory region to which it has a writable flow.

Table 4-27: Supported Memory Region Types

Memory Region Type	Description
PROGRAM	A PROGRAM memory region is a general-purpose memory region that contains code and data. PROGRAM memory regions provide working memory for a subject, and can also be shared among subjects. In fully virtualized subjects, PROGRAM memory regions that are mapped both readable and writable and owned by the same subject are treated as RAM. Virtual firmware reports these regions as available RAM to other software running in the subject; their contents thus may be overwritten. Use other memory types such as SHM to create reserved memory regions. Also, if a PROGRAM memory region flow owner attribute is false, it is treated as reserved memory in that subject. Unlike the other memory region types, PROGRAM memory regions that don't have a starthpa attribute in the HCV may be automatically split into smaller pieces by LynxSecure in order to allocate them within the host memory constraints. *Additional constraints:* • Size: >= 4Kbyte • A PROGRAM memory region may be initialized with binary data (for example, by executable code and data) by being declared using the guestimage element.
BOOT	A BOOT memory region contains an image with the subject's custom startup code, which is executed every time the subject starts or restarts. At that time, the BOOT image is first automatically copied to another location in the subject and then executed at the new location. The destination of the copy is normally a readable and writable PROGRAM memory region referred to as the "startup RAM". Subject execution then continues in the startup RAM. In a fully virtualized subject, the boot image must always contain the virtual firmware image and is copied to the virtual firmware RAM just below guest physical address 4GB. In a para-virtualized subject, the boot image is copied to the memory region pointed to by the subject start address (specified in the startaddr attribute). The start address may point anywhere inside the destination memory region. If the subject also has a ramsize attribute, the destination RAM memory region is automatically created at the subject start address rounded down to a page boundary (4KB) and is automatically designated as the boot image copy target. Note that the copy destination address is the start of the destination memory region, not the address specified by the startaddr attribute. This way, the subject initialization entry point may be located anywhere in the boot image copy, not just at its start. Typically, BOOT memory regions are declared using the guestimage element. The flows to these memory regions are normally read-only (non-writable); however, a BOOT region could be made writable for a subject if that subject is tasked with modifying the startup code for itself or another subject. If using writable BOOT images, the following must be taken into account: a subject will not boot from a writable BOOT region unless it is also the owner of that region. *Additional constraints:* • Size: >= 4Kbyte • Any subject may only have exactly one bootable BOOT region (that is, one that is read-only or owned by that subject). • BOOT memory regions may be shared between subjects. • A bootable BOOT memory region should be mapped into its subject's address space (i.e. have a gpa or gva attribute); if it isn't, however, LynxSecure will automatically pick an address and create a mapping for this region.
BOOTSTRAP	This memory region contains code that runs in the context of a subject and copies the contents of the subject's read-only BOOT memory region to the startup RAM location.

Memory Region Type	Description
	This happens each time the subject is started or restarted. The bootstrap code then continues execution at the subject's configured start address, which normally points to the subject's initialization entry point in the startup RAM region. The size of this memory region may be set to AUTO, and it will be picked automatically by LynxSecure. The BOOTSTRAP memory region and flow specifications are optional in the HCV. If missing, LynxSecure automatically creates a unique BOOTSTRAP memory region and a memory flow to it for each subject. *Additional constraints:* • Typically, one system-wide BOOTSTRAP memory region is defined. All regions of this type are equivalent. The BOOTSTRAP memory region's contents are automatically initialized by LynxSecure. • Size: 4Kbyte or AUTO • Each subject may only have exactly one BOOTSTRAP memory region flow. • For the bootstrap mechanism to work, the three memory regions involved in the subject's initialization (BOOT, BOOTSTRAP and the initial PROGRAM region) should be mapped into the subject's guest address space (that is, have a gpa or gva attribute). However, LynxSecure will automatically pick addresses and map both BOOTSTRAP and BOOT, as long as the subject has a memory flow to the BOOT region.
SANSRC	The SANSRC memory region serves as an alternate boot image source for the subject. This region is optional, if no SANSRC memory region is defined for a subject, memory sanitization is not performed during that subject's restart.
SANDST	The SANDST memory region serves as an alternate boot image destination for the subject. This region is optional, if no SANSRC memory region is defined for a subject, memory sanitization is not performed during that subject's restart.
ARGPAGE	An ARGPAGE memory region contains initialization arguments passed to a subject. Every subject's ARGPAGE memory region is initialized by LynxSecure with the arguments defined in the initparams attribute for the subject in the HCV. The size of this memory region may be set to AUTO, and it will be picked automatically by LynxSecure. The ARGPAGE memory region and flow specifications are optional in the HCV. If missing, LynxSecure automatically creates a unique ARGPAGE memory region and a memory flow to it for each subject. *Additional constraints:* • Size: 4Kbyte or AUTO • Every subject must have exactly one associated ARGPAGE memory region. The memory flow from the owner subject to this region must be read-only. • ARGPAGE memory regions may not be shared between subjects. • Memory regions of this type may not use the fill attribute, nor can they be specified using the guestimage elements.
PML4	A PML4 memory region contains the Page Map Level 4 Tables for a subject. This memory region is required for all subjects except 32-bit paravirtualized subjects. Direct writes are not permitted for this memory region even if a subject has a writable flow. The subject must use hypercalls to modify this region. The size of this memory region may be set to AUTO, and LynxSecure will pick some default size for it. If that size is insufficient, the subject may fail to create page tables during the boot time or, if it creates new page tables at runtime, during the runtime. The solution is to specify a sufficient size explicitly in the HCV.

Memory Region Type	Description
	The PML4 memory region and flow specifications are optional in the HCV. If missing, LynxSecure automatically creates a unique PML4 memory region and a memory flow to it for each subject. *Additional constraints:* • Size: >= 4Kbyte or AUTO • Every subject must have exactly one associated PML4 memory region. The memory flow from the owner subject to this region must be read/write. • Every PML4 memory region is initialized and updated by LynxSecure on behalf of the subject which has a writable flow to that region. • Memory regions of this type may not use the fill attribute, nor can they be specified using the guestimage elements.
PDPT	A PDPT memory region contains the Page Directory Pointer Tables for a subject. This memory region is required for all subjects except 32-bit paravirtualized subjects. Direct writes are not permitted for this memory region even if a subject has a writable flow. The subject must use hypercalls to modify this region. The size of this memory region may be set to AUTO, and LynxSecure will pick some default size for it. If that size is insufficient, the subject may fail to create page tables during the boot time or, if it creates new page tables at runtime, during the runtime. The solution is to specify a sufficient size explicitly in the HCV. The PDPT memory region and flow specifications are optional in the HCV. If missing, LynxSecure automatically creates a unique PDPT memory region and a memory flow to it for each subject. *Additional constraints:* • Size: >= 4Kbyte or AUTO • Every subject must have exactly one associated PDPT memory region. The memory flow from the owner subject to this region must be read/write. • Every PDPT memory region is initialized and updated by LynxSecure on behalf of the subject which has a writable flow to that region. • Memory regions of this type may not use the fill attribute, nor can they be specified using the guestimage elements.
PDT	A PDT memory region contains the Page Directory Tables for a subject. Direct writes are not permitted for this memory region even if a subject has a writable flow. The subject must use hypercalls to modify this region. The size of this memory region may be set to AUTO, and LynxSecure will pick some default size for it. If that size is insufficient, the subject may fail to create page tables during the boot time or, if it creates new page tables at runtime, during the runtime. The solution is to specify a sufficient size explicitly in the HCV. The PDT memory region and flow specifications are optional in the HCV. If missing, LynxSecure automatically creates a unique PDT memory region and a memory flow to it for each subject. *Additional constraints:* • Size: >= 4Kbyte or AUTO • Every subject must have exactly one associated PDT memory region. The memory flow from the owner subject to this region must be read/write. • Every PDT memory region is initialized and updated by LynxSecure on behalf of the subject which has a writable flow to that region. • Memory regions of this type may not use the fill attribute, nor can they be specified using the guestimage elements.

Memory Region Type	Description
PTE	A PTE memory region contains the Page Table Entries for a subject. Direct writes are not permitted for this memory region even if a subject has a writable flow. The subject must use hypercalls to modify this region. The size of this memory region may be set to AUTO, and LynxSecure will pick some default size for it. If that size is insufficient, the subject may fail to create page tables during the boot time or, if it creates new page tables at runtime, during the runtime. The solution is to specify a sufficient size explicitly in the HCV. The PTE memory region and flow specifications are optional in the HCV. If missing, LynxSecure automatically creates a unique PTE memory region and a memory flow to it for each subject. *Additional constraints:* • Size: >= 4Kbyte or AUTO • Every subject must have exactly one associated PTE memory region. The memory flow from the owner subject to this region must be read/write. • Every PTE memory region is initialized and updated by LynxSecure on behalf of the subject which has a writable flow to that region. • Memory regions of this type may not use the fill attribute, nor can they be specified using the guestimage elements.
ROPAGE	An ROPAGE memory region contains subject configuration information for its owner subject. The region's contents are filled in automatically by LynxSecure. The size of this memory region may be set to AUTO, and it will be picked automatically. The ROPAGE memory region and flow specifications are optional in the HCV. If missing, LynxSecure automatically creates a unique ROPAGE memory region and a memory flow to it for each subject. *Additional constraints:* • The amount of information to be put in this region depends on the subject's configuration. LynxSecure panics if this memory region size is insufficient for any subject. The recommended size, if specified, is 64KB or above. • Every subject must have exactly one associated ROPAGE memory region. Any memory flow to this region must be read-only. • Memory regions of this type may not use the fill attribute, nor can they be specified using the guestimage elements.
ECM	An ECM memory region is used in fully virtualized subjects for communication between the full virtualization subsystem running in the subject context and the Event Capture Mechanism module in the hypervisor. It's normally inaccessible to guest code. The size of this memory region may be set to AUTO, and it will be picked automatically. The ECM memory region and flow specifications are optional in the HCV. If missing, LynxSecure automatically creates a unique ECM memory region and a memory flow to it for each FV subject. *Additional constraints:* • LynxSecure panics if this memory region's size is insufficient. It is recommended to leave it as AUTO. • Every FV subject has exactly one associated ECM memory region. Any memory flow to this region must be read/write. • Memory regions of this type may not use the fill attribute, nor can they be specified using the guestimage elements.
RESERVED	A RESERVED memory region is reported as reserved memory to the GuestOS. Such memory regions may be used to map special firmware regions in host physical memory

Memory Region Type	Description
	to subject address space. This includes the System BIOS, VGA BIOS, and possibly other firmware regions depending on the target hardware. *Additional constraints:* • Size: >= 4Kbyte • Unlike most other memory region types, a RESERVED memory region may overlap device I/O memory.
CMA	This is a subtype of the RESERVED memory region type intended for use with subjects running Linux. If the platform uses Devicetree, the memory region is reported as the default memory pool for the Contiguous Memory Allocator in the Devicetree.
DMA	This is a subtype of the RESERVED memory region type intended for use with subjects running Linux. If the platform uses Devicetree, the memory region is reported as the default memory pool for the consistent DMA allocator in the Devicetree.
SHADOW	A SHADOW memory region is a read-only region that contains special memory reserved for a fully virtualized subject. Whether or not a flow to this region is necessary depends on hardware capabilities of the target system. The size of this memory region may be set to AUTO, and it will be picked automatically by LynxSecure. The SHADOW memory region and flow specifications are optional in the HCV. If missing (and necessary), LynxSecure automatically creates a unique SHADOW memory region and a memory flow to it for each subject. *Additional constraints:* • At most one SHADOW memory region must be defined per fully virtualized subject. Any memory flow to this region must be read-write. The SHADOW memory region may not be mapped to subject address space. • Memory regions of this type may not use the fill attribute, nor can they be specified using the guestimage elements.
BITRESULTS	A BITRESULTS memory region contains the results from the built-in test. An authorized subject can read this memory region to get these results. The format of the memory region is described in the api.h header file. The size of this memory region may be set to AUTO, and it will be picked automatically by LynxSecure. The BITRESULTS memory region and flow specifications are optional in the HCV. If missing, LynxSecure automatically creates a unique BITRESULTS memory region and a memory flow to it for each IBIT and CBIT subject. Note that automatic creation of this region prevents specifying a memory flow to it from another authorized subject and, therefore, should only be used if such a flow is unnecessary.
VDEVIO	The VDEVIO memory region is a region that supports memory based virtual devices. Such regions may either be specified explicitly in the HCV, or they can be implicitly created using memreg (page 75) elements in virtual device definitions.
SHM	Same as PROGRAM memory type, used to designate memory regions intended for sharing between subjects.
RMRR	An RMRR memory region corresponds to a Reserved Memory Region lying in host physical memory. RMRRs are associated with devices that need a range of host physical memory to be identity-mapped in the DMA redirection tables for the device. These regions are typically used by USB and on-board graphics devices. RMRRs are not visible to the CPU; they are only accessible by device DMA. On each system, RMRR locations are determined by the host BIOS and reported to software such as LynxSecure via ACPI tables. RMRR specification in the HCV is optional. If not specified, LynxSecure automatically creates the RMRR memory regions and any necessary memory flows to subjects, which

Memory Region Type	Description
	have the devices with associated RMRRs assigned to them. If RMRRs are explicitly specified in the HCV, LynxSecure verifies that the configuration information for those RMRRs matches that of the host BIOS when LynxSecure is starting up. RMRRs may overlap with other memory region types, and RMRR mappings (memory flows) may overlap with mappings of other memory region types; in this case, the RMRR takes precedence over the other regions for DMA transfers to the overlapping address range. The size of this memory region may be set to `AUTO`, and it will be determined automatically. *Additional constraints:* • Memory flows to `RMRR` regions must permit both reading and writing the region. • Memory regions of this type may not use the `fill` attribute, nor can they be specified using the `guestimage` elements.
`SKH_HEAP`	An `SKH_HEAP` memory region defines the size and optionally the location of the memory region used as the dynamic memory ("heap") by the hypervisor. If not defined, the SKH heap will be created (with the default size) at runtime. *Additional constraints:* • Only a single region of this type may be defined. • Memory regions of this type may not use the `fill` attribute, nor can they be specified using the `guestimage` elements. • For the PowerPC target architecture, the size of this region must be a power of 2 and it must be naturally aligned (that is, aligned at its size).

<alignment> — Alignment

Description

Specifies the desired alignment for the start host physical address of the memory region, which is allocated automatically. LynxSecure treats this attribute as a hint: it may use a smaller alignment for the memory region in order to allocate them within the host memory constraints. If this happens, a warning is printed during LynxSecure startup.

Content

Refer to Table 4-28 (page 66).

Attributes

None.

Child Elements

None.

Schema Type

`alignmentType`

Location

- `/hcv/partition/memoryregion/alignment`

- /hcv/partition/guestimage/alignment

Memory Region Alignment

Table 4-28: Memory Region Alignment

Alignment	Description
AUTO	Automatically selected
auto	Automatically selected
4KByte	Aligned to small page boundary (4KByte)
4KByteLow	Aligned to small page boundary (4KByte); physical address below 4GByte.
64KByte	Aligned to 64KB.
4MByte	Aligned to large non-PAE page boundary (4MByte)
4MByteLow	Aligned to large non-PAE page boundary (4MByte); physical address below 4GByte.
2MByte	Aligned to large PAE page boundary (2MByte)
2MByteLow	Aligned to large PAE page boundary (2MByte); physical address below 4GByte.
Natural	Aligned to the memory region's size, which must be explicitly specified and a power of 2.

<starthpa> — Starting Host Physical Address

Description

This element forces a memory region to be allocated at a fixed physical address in host memory.

Content

Memory address, in hexadecimal.

Attributes

None.

Child Elements

None.

Schema Type

memaddrType

Location

- /hcv/partition/memoryregion/starthpa
- /hcv/partition/guestimage/starthpa

<guestimage> — Guest Image

Description

A region of memory allocated from host physical memory that has been pre-initialized with code and/or data from a file.

Attributes

Table 4-29: Attributes for <guestimage>

Name	Description	Additional Constraints
`memregionname` *(string ID, up to 63 chars)*	The name of the memory region exported resource.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.
`memorytype` *(refer to Table 4-27 (page 60))* *(optional, default: `PROGRAM`)*	The type of memory region. Used internally by the SKH to set up a suitable memory environment for subjects. See Table 4-27 (page 60) for a complete list of memory types and their constraints.	
`size` *(hexadecimal, page multiple, or AUTO)* *(optional, default: `AUTO`)*	The size of the memory region in bytes.	LynxSecure can automatically determine the size of certain auxiliary memory regions, so this attribute is optional. However, if the start HPA of a memory region is explicitly specified, then its size must be explicitly specified as well. Memory regions defined with the `guestimage` element are an exception; their size, if unspecified, is set to the file size.
`noclobber` *(boolean)* *(optional, default: empty)*	Prevent RIF from using the HPA associated with this memory region during initialization.	During early initialization, LynxSecure utilizes host memory to unpack the SRP. To protect a given memory region from being used during this process, enable this attribute. The size and HPA of the memory region must also be specified for this option to take effect. For RESERVED memory regions with a defined HPA and size, this option is enabled by default.
`fill` *(hexadecimal, 2 digits)* *(optional, default: empty)*	If specified, LynxSecure will initialize the region, setting each byte to the specified value. This only happens once at LynxSecure startup. If specified for a `guestimage`, only the part of the memory region between the end of the file image and the end of the memory region is initialized this way (the rest is initialized using the file image).	
`caching` *(one of: `DEFAULT`/ `UNCACHEABLE`/ `WRITE-COMBINING`/ `WRITE-THROUGH`/ `WRITE-PROTECTED`/ `WRITEBACK`)* *(optional, default: `DEFAULT`)*	The caching type to use for all mappings of this memory region. If unspecified, or set to `DEFAULT`, the memory will be mapped with the base caching type specified for the memory region by the host firmware (or uncacheable, if the owner subject is not allowed to use memory caches by its own `caching` attribute). The guest internal mapping mechanisms may override/complement the base caching type	Use this attribute with care, because specifying a wrong memory type for a memory region may cause hardware faults. For example, device memory shouldn't be mapped cacheable.

Name	Description	Additional Constraints
	according to the host platform caching type merge rules. If this attribute is set to any value other than `DEFAULT`, the behavior depends on the platform. On the x86, the specified caching type will be in effect in all subjects mapping this region regardless of which caching type is specified for the memory region by the host firmware or any guest mapping mechanisms. On ARM, the specified caching type will be merged with the caching type specified by the guest internal mapping mechanisms according to the ARM platform caching type merge rules.	
`path` *(string)* *(optional, default: empty)*	Path to the image used to initialize the memory region. If this attribute is not specified, the memory region contents is the text contained in this `guestimage` element.	This may be an absolute or relative (to the configuration file's directory) pathname valid on the platform on which the configuration tool is running.
`pcrnum` *(integer, 12…15)* *(optional, default: empty)*	When present this attribute determines which Platform Configuration Register (PCR) the associated guest image gets extended into.	

Child Elements

One of the following child elements is expected:

- `alignment`
- `starthpa`

Schema Type

`guestimageType`

Location

- `/hcv/partition/guestimage`

<externaldevice> — External Device

Description

An external device is a device (controller) attached to a bus that is not virtualized by the SKH. A device is uniquely assigned to one subject. A subject granted access to the device uses standard I/O instructions or memory-mapped I/O to send requests to the device.

Each device should be unambiguously identified. The Device ID (also known as the PCI ID for a PCI device) may be used if it is the only device of that type on the target platform. If there are multiple devices of the same type (e.g., a separate network interface card for each classification level), each device should be further identified by its bridge number and Bus/Device/Function (BDF) number (also known as the PCI address for a PCI device). If a device on the target platform cannot be unambiguously identified as just described, it will be associated with an `externaldevice` element containing a matching `devid` attribute in an indeterminate manner.

The configuration includes a bridge number that identifies non-PCI mainboard devices (value 0) and PCI devices attached via a Host-PCI bridge (value 1).

In addition to identifying the device, associated I/O ports, memory-mapped I/O regions, and interrupt request numbers may be defined for each device.

Some devices are host controller devices. In the case where a host controller device is present in the target system, the host controller can be assigned to a given subject, but the devices subordinate to that host controller must then all be assigned to that same subject.

For example, if there are two SATA controllers, SATA0 and SATA1, and SATA0 supports 2 SATA ports, and SATA1 supports 4 SATA ports, then both ports under SATA0 must be assigned to the same subject, and all 4 ports under SATA1 must be assigned to a single subject - which could be the same subject as SATA0, or another subject.

LynxSecure limits each external device to an information flow with a single subject (no simultaneous or sequential access).

Attributes

Table 4-30: Attributes for <externaldevice>

Name	Description	Additional Constraints
`externaldevname` *(string ID, up to 63 chars)*	The name of the external device exported resource.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.
`devid` *(hexadecimal)* *(optional, default: `00000000`)*	A 32-bit device identifier constructed from a 16-bit Vendor ID and 16-bit Device ID. Also known as the PCI ID for a PCI or PCI Express devices.	If more than one device has the same `devid` value, the Configuration Tool generates a warning if those devices cannot be distinguished using a combination of `bustype`, `busnum`, `devnum`, and `funcnum`.
`bustype` *(refer to Table 4-31 (page 70))* *(optional, default: `legacy`)*	The system bus type that the device is connected to.	
`path` *(string)* *(optional, default: empty)*	Path to the device. If the platform uses Devicetree, this is either the full absolute Devicetree path of the device, or just its name (which can be ambiguous, unlike the full path) or its Devicetree alias. In case an ambiguous name is used, LynxSecure will match it to one of the devices with that name, but which one is going to be picked is unspecified.	
`busnum` *(integer, 0…255 or AUTO)* *(optional, default: empty)*	Bus number relative to the bridge to which the device is attached. The first part of the BDF value.	
`devnum` *(integer, 0…31 or AUTO)* *(optional, default: empty)*	Device number relative to the bus to which the device is attached. The second part of the BDF value.	
`funcnum` *(integer, 0…7 or AUTO)* *(optional, default: empty)*	Function number relative to the device. The third part of the BDF value.	
`irq` *(integer or AUTO)* *(optional, default: empty)*	The IRQ for the device. If not specified, the system automatically selects a suitable value for the device.	This attribute is only supported for the `x86_64` target architecture.

Name	Description	Additional Constraints
`resetonrestart` *(boolean)* *(optional, default: `true`)*	Whether or not this device is reset when the owner subject is restarted. Only devices that support generic device reset can be reset this way. If the device doesn't support generic reset, this setting is ignored.	Normally, all devices should be reset on a subject restart to ensure proper operation. However, some devices require a device-specific initialization sequence and won't work after a generic reset until a device-specific driver is loaded (for example, graphics cards). If that presents a problem, use this attribute to work around.
`irqpolling` *(refer to Table 4-32 (page 70))* *(optional, default: `AUTO`)*	Determines whether the device interrupt can be emulated by polling the device status.	This attribute is only meaningful on PCI devices; it is ignored on legacy devices.
`mandatory` *(boolean)* *(optional, default: `true`)*	Whether LynxSecure panics if this device is not present when LynxSecure boots.	

Child Elements

The following child elements may appear in any order:

- `ioports`
- `iomem`

Schema Type

`externaldeviceType`

Location

- `/hcv/partition/externaldevice`

System Bus Type

Table 4-31: System Bus Type

Bus Type	Description
`PCI`	The PCI bus.
`legacy`	A legacy system bus (ISA, Low Pin Count or other non-enumerable bus).

Device Interrupt Emulation Modes

Table 4-32: Device Interrupt Emulation Modes

Mode	Description
`AUTO`	In configurations where multiple devices sharing the same IRQ line are assigned to different subjects, the hypervisor selects a subject which will use the real IRQ based on the number of devices with that IRQ assigned to each subject. The rest of the subjects emulate legacy interrupt through some other mechanisms such as polling. *Additional constraints:* • In certain scenarios, such automatic selection may result in a suboptimal configuration, e.g. when one subject owns fewer devices but those devices generate a lot of interrupts.

Mode	Description
NONE	This setting will force the hypervisor to use the hardware interrupt rather than polling in the subject owning this device.
FORCE	For testing purposes, force emulation of the device interrupt through polling even if it is not required due to device assignments.

<ioports> — I/O Ports

Description

The deviceioType is used to specify the I/O ports for an external device. I/O ports are assigned from a 64K (16-bit) I/O address space. A PCI or PCI Express device may have a number of Base Address Registers (BARs), which can be I/O BARs or memory BARs. I/O BARs point into the 16-bit I/O address space. Each PCI device supports up to 13 BARs (up to 6 of them I/O BARs) and, for each I/O BAR, the device responds to a range of I/O ports starting at the port address contained in that BAR. Each ioports sub-element of externaldevice defines the I/O ports associated with a single BAR. For PCI device BARs, LynxSecure detects their start address and size automatically at runtime. Even the BAR specification itself is optional.

Platform devices do not have BARs, but may have associated I/O ports, which are described in a similar manner. The barnum attribute should not be used with platform devices. The start address and the size of these resources must be explicitly configured.

Additional Constraints

This element is only supported for the x86_64 target architecture.

Attributes

Table 4-33: Attributes for <ioports>

Name	Description	Additional Constraints
devioname *(string ID, up to 63 chars)* *(optional, default: empty)*	If provided, defines an export resource for the I/O port region used by a device.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.
barnum *(integer, 0…5)* *(optional, default: empty)*	The Base Address Register (BAR) number. The regular PCI BARs correspond to numbers from 0 to 5.	Each ioports and iomem sub-element for a given external device must have a unique value of this attribute.
addr *(hexadecimal)* *(optional, default: empty)*	The offset into the 64K I/O port address space at which this BAR points. If not specified, the value is determined automatically by LynxSecure.	Requires explicit specification for non-PCI devices.
size *(hexadecimal)* *(optional, default: empty)*	The number of ports in the I/O port range. If not specified, the value is determined automatically by LynxSecure.	Requires explicit specification for non-PCI devices.

Child Elements

None.

Schema Type

deviceioType

Location

- `/hcv/partition/externaldevice/ioports`

<iomem> — Memory-Mapped I/O Regions

Description

The `devicememoryType` is used to specify the memory-mapped I/O regions for an external device. A PCI or PCI Express device may have a number of Base Address Registers (BARs), which can be I/O BARs or memory BARs. Memory BARs point to the physical address space. Each device supports up to 13 BARs and, for each memory BAR, the device responds to accesses to a memory-mapped I/O region starting at the address contained in that BAR. Each `iomem` sub-element of `externaldevice` defines the memory region associated with a single BAR. For PCI device BARs, LynxSecure detects their start address and size automatically at runtime. Even the BAR specification itself is optional.

Legacy devices can't have memory-mapped I/O regions.

Additional Constraints

This element is only supported for the `x86_64` target architecture.

Attributes

Table 4-34: Attributes for <iomem>

Name	Description	Additional Constraints
`devmemname` *(string ID, up to 63 chars)* *(optional, default: empty)*	If provided, defines an exported resource for a memory-mapped I/O region of the device.	The fully qualified name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.
`barnum` *(integer, 0...12)* *(optional, default: empty)*	The PCI Base Address Register (BAR) number. The regular PCI BARs correspond to numbers from 0 to 5, and the Expansion ROM BAR is number 6. SR-IOV BARs are numbered 7 to 12.	This attribute should only be used with PCI devices. Each `ioports` and `iomem` sub-element for a given external device must have a unique value of this attribute.

Child Elements

None.

Schema Type

`devicememoryType`

Location

- `/hcv/partition/externaldevice/iomem`

<virtualdevice> — Virtual Device

Description

A virtual device is a device which is not present in the hardware, but is emulated by software.

Attributes

Table 4-35: Attributes for <virtualdevice>

Name	Description	Additional Constraints
`vdevname` *(string ID, up to 48 chars)*	The name of the virtual device exported resource.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.
`devtype` *(refer to Table 4-36 (page 74))* *(optional, default: `OTHER`)*	The device type of the virtual device. This tells LynxSecure what type of service this virtual device provides in the owner subject. A device type may be general (for example, "AUDIO" or "USB") and specialized (for example, "AUDIO-HDA" or "USB-EHCI"). If the `virtualdevice` element has no child elements, LynxSecure will determine the virtual device details using a built-in template based on the value of this attribute. If this attribute specifies a general device type, LynxSecure picks one of the supported specialized device types automatically. However, if there are child elements, LynxSecure ignores any specialization of the device type and determines the device details based on those elements instead.	

Child Elements

The following sequence of child elements is expected:

- `capflow` *(optional, single instance)*
- `interface` *(0...2 instances)*

Schema Type

`virtualdeviceType`

Location

- `/hcv/partition/virtualdevice`

A Virtual Device Type

Table 4-36: A Virtual Device Type

Device Type	Description
USB	USB host controller: default
USB-UHCI	USB host controller: UHCI
USB-OHCI	USB host controller: OHCI
USB-EHCI	USB host controller: EHCI
NET	Network Interface Card (NIC): default
NET-NE2000	Network Interface Card (NIC): NE2000
NET-E1000	Network Interface Card (NIC): E1000
GRAPHICS	Graphics controller: default Virtual KVM graphics
UART	Universal Asynchronous Receiver/Transmitter: default
AUDIO	Audio controller: default
AUDIO-AC97	Audio controller: AC-97
AUDIO-HDA	Audio controller: High Definition Audio
KBDMOUSE	Keyboard and Mouse: PC-compatible
STORAGE	Storage controller: default
IDE	Storage controller: Integrated Drive Electronics
IDE-ICH7	Storage controller: Integrated Drive Electronics (ICH7)
IDE-ICH9	Storage controller: Integrated Drive Electronics (ICH9)
IDE-LYNX	Storage controller: Integrated Drive Electronics (Lynx)
IDE-LEGACY	Storage controller: Integrated Drive Electronics (Legacy)
AHCI	Storage controller: Advanced Host Controller Interface
MONITOR	LynxSecure VDS subject monitor device
LOGGER	LynxSecure VDS subject logger device
FIFO	LynxSecure inter-subject FIFO device
SHMEM	LynxSecure generic shared memory device
LPC	PCI-to-LPC bridge
LPC-ICH9	ICH9 PCI-to-LPC bridge
INTERRUPT	A virtual interrupt controller
INTERRUPT-GICV2	A virtual Generic Interrupt Controller version 2 (ARM)
TIMER	A virtual timer device
TIMER-ARMV8	A virtual ARMv8 generic timer device
OTHER	Any other device type

<capflow> — Bidirectional Flow Capability

Description

This element defines a resource that is created with read/write flows from both subjects that have flows to this device's interfaces.

Additional Constraints

There can only be one of these elements per virtual device.

Attributes

None.

Child Elements

One of the following child elements is expected:

- `memreg`

Schema Type

`vdevbiflowType`

Location

- `/hcv/partition/virtualdevice/capflow`

<memreg> — Memory Region for Virtual Device

Description

This element creates a memory region which is used for device emulation. Depending on the element's parent (bidirectional or unidirectional flow) and the interface number assigned to the subject, the implicit memory flow created for the subject will have either read-write or read-only permissions.

Attributes

Table 4-37: Attributes for <memreg>

Name	Description	Additional Constraints
`size` *(hexadecimal, page multiple)*	Size of the memory region.	
`alignment` *(refer to Table 4-28 (page 66))* *(optional, default: `AUTO`)*	Alignment of the memory region.	

Child Elements

None.

Schema Type

`vdevmemregType`

Location

- `/hcv/partition/virtualdevice/capflow/memreg`

<interface> — Virtual Device Interface

Description

This element describes one of the virtual device's interfaces. Virtual device may have 1 or 2 interfaces; where there are two interfaces, by convention, the first one (interface #0) is usually the "client" interface assigned to the client subject, and the second one (interface #1) is the "server" interface assigned to the server subject.

Attributes

Table 4-38: Attributes for <interface>

Name	Description	Additional Constraints
`notify` *(boolean)* *(optional, default: `false`)*	Whether the interface may generate a peer notification interrupt in the subject that it is assigned to.	

Child Elements

The following child elements may appear in any order:

- `pci`
- `capflow`
- `ioports`
- `iomem`

Schema Type

`vdevinterfaceType`

Location

- `/hcv/partition/virtualdevice/interface`

<pci> — PCI Device Emulation

Description

This element describes the PCI interface provided by this virtual device interface.

Attributes

Table 4-39: Attributes for <pci>

Name	Description	Additional Constraints
`devid` *(hexadecimal, 32-bit)*	PCI vendor and device IDs that this interface will report.	
`class` *(hexadecimal, 24-bit)* *(optional, default: `FFFFFF`)*	PCI class reported by the virtual device.	
`command` *(hexadecimal, 8-bit)*	Base value for PCI command register.	

Name	Description	Additional Constraints
(optional, default: `00`)		
`status` *(hexadecimal, 8-bit)* *(optional, default: `00`)*	Base value for PCI status register.	

Child Elements

None.

Schema Type

`vdevpciType`

Location

- `/hcv/partition/virtualdevice/interface/pci`

<capflow> — Unidirectional Flow Capability

Description

This element defines a resource with write-only (if applicable) or read-write flow from the first interface of the virtual device, and read-only flow from the second interface of the virtual device.

Additional Constraints

There can only be one of these elements per interface.

Attributes

None.

Child Elements

The following child elements may appear in any order:

- `msgbuf`

Schema Type

`vdevuniflowType`

Location

- `/hcv/partition/virtualdevice/interface/capflow`

<msgbuf> — Message Buffer for Virtual Device

Description

This element creates a message buffer used by device emulation.

Attributes

Table 4-40: Attributes for <msgbuf>

Name	Description	Additional Constraints
`irq` *(integer)* *(optional, default: empty)*	If present, the `irq` attribute specifies the IRQ to send to the receiving subject when a message arrives into the message buffer.	

Child Elements

None.

Schema Type

None (defined inside an element).

Location

- `/hcv/partition/virtualdevice/interface/capflow/msgbuf`

<ioports> — I/O Port Range

Description

This element declares a range of I/O ports for accessing the device. For virtual PCI devices, it creates a PCI BAR (Base Address Register). When a fully virtualized subject guest code accesses this range, the access is intercepted and directed to the virtual device driver. In para-virtualized subjects, these ranges are ignored.

Attributes

Table 4-41: Attributes for <ioports>

Name	Description	Additional Constraints
`addr` *(hexadecimal, 16-bit, or AUTO)* *(optional, default: `AUTO`)*	First I/O port of the range. If set to `AUTO`, the port value is automatically allocated from the available subject I/O port space.	
`size` *(hexadecimal, 16-bit)*	Size of the I/O port range.	
`barnum` *(integer, 0…5)* *(optional, default: empty)*	The index of the BAR in a virtual PCI device. If not specified, BAR indices will be assigned incrementally starting from 0 or from the previous explicitly specified index.	This attribute should only be used with virtual PCI devices. Each `ioports` and `iomem` sub-element for a given device must have a unique value of this attribute.

Child Elements

None.

Schema Type

`ioportsType`

Location

- `/hcv/partition/virtualdevice/interface/ioports`

<iomem> — Memory-mapped I/O

Description

This element specifies range of memory for accessing the device. For virtual PCI devices, it creates a PCI BAR (Base Address Register). When a fully virtualized subject guest code accesses this range, the access is intercepted and directed to the virtual device driver. In para-virtualized subjects, these ranges are ignored.

Attributes

Table 4-42: Attributes for <iomem>

Name	Description	Additional Constraints
`addr` *(hexadecimal, page multiple, or AUTO)* *(optional, default: `AUTO`)*	Starting guest physical address of the memory-mapped registers region, or `AUTO` for automatic allocation.	
`size` *(hexadecimal, page multiple)*	Size of the memory-mapped registers region. Must be a power of 2.	
`barnum` *(integer, 0…12)* *(optional, default: empty)*	The index of the BAR in a virtual PCI device. If not specified, BAR indices will be assigned incrementally starting from 0 or from the previous explicitly specified index.	This attribute should only be used with virtual PCI devices. Each `ioports` and `iomem` sub-element for a given device must have a unique value of this attribute.

Child Elements

None.

Schema Type

`iomemoryType`

Location

- `/hcv/partition/virtualdevice/interface/iomem`

<messagebuffer> — Message Buffer

Description

A message buffer provides a unidirectional FIFO queue of short messages between subjects. Each instance of a message buffer is restricted to one writer and one reader. The reader receives an interrupt whenever a message is queued (defined in the corresponding `messagebufferflow` (page 96)).

Attributes

Table 4-43: Attributes for <messagebuffer>

Name	Description	Additional Constraints
`msgbufname` *(string ID, up to 63 chars)*	The name of the message buffer exported resource.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.

Child Elements

None.

Schema Type

`messagebufferType`

Location

- `/hcv/partition/messagebuffer`

<absoluteclock> — Absolute Clock

Description

An abstract resource that provides calibrated time-of-day to authorized subjects.

Attributes

Table 4-44: Attributes for <absoluteclock>

Name	Description	Additional Constraints
`absclockname` *(string ID, up to 63 chars)*	The name of the absolute clock exported resource.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.

Child Elements

None.

Schema Type

`absoluteclockType`

Location

- `/hcv/partition/absoluteclock`

<audit> — Audit

Description

An abstract resource that collects SKH audit records and makes them available to authorized subjects for retrieval.

Attributes

Table 4-45: Attributes for <audit>

Name	Description	Additional Constraints
`auditname` *(string ID, up to 63 chars)*	The name of the audit buffer exported resource.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.
`maxentries` *(integer; >0)* *(optional, default: `1024`)*	The maximum number of audit events that the audit buffer can hold.	Must be a power of 2.

Child Elements

The following sequence of child elements is expected:

- `fullbufferaction` *(required, single instance)*
- `auditrule` *(optional)*

Schema Type

`auditType`

Location

- `/hcv/partition/audit`

<fullbufferaction> — Buffer Overflow Action

Description

This element defines the action to take if the audit buffer is full.

Attributes

Table 4-46: Attributes for <fullbufferaction>

Name	Description	Additional Constraints
`action` *(refer to Table 4-47 (page 82))*	Defines the action to take upon overflow.	
`parameter` *(string ID reference)* *(optional, default: empty)*	For `RESCHED` action, this attribute specifies the name of the scheduling policy to switch to. Ignored for other actions.	

Child Elements

None.

Schema Type

None (defined inside an element).

Location

- `/hcv/partition/audit/fullbufferaction`

Supported Full Buffer Actions

Table 4-47: Supported Full Buffer Actions

Full Buffer Action	Description
`MAINTMODE`	Transition to the maintenance mode scheduling policy.
`SHUTDOWN`	Halt the SKH, including all subjects. This is an immediate halt, not a clean shutdown, so any information cached by subjects may be lost.
`DROP`	Audit event information is dropped, but a count of the dropped event for each audit type is held in an overflow buffer.
`RESCHED`	Switch to a different scheduling policy. The name of the scheduling policy is specified in the `parameter` attribute.

<auditrule> — Audit Rule

Description

The specification of the audit buffer may include a sequence of zero or more audit rule elements. When an audit record is generated, the SKH checks each audit rule in turn until it finds a match. It then takes the action defined by the matching rule. Each audit rule contains several optional filterable parameters. Each specified parameter must be matched for the rule to match. The rules are searched in sequential order. If more than on match is found, the last match is used to determine the filter action. If no match is found, the event is stored.

Attributes

Table 4-48: Attributes for <auditrule>

Name	Description	Additional Constraints
`filteraction` *(refer to Table 4-49 (page 83))*	The action to take when an audit record is matched.	
`evtype` *(string)*	The name of the event to match. A value of "*" matches on any event.	If a specific event name is configured, it must be a valid event name.
`initresource` *(string)* *(optional, default: empty)*	The name of the initiating subject to match. A value of "*" matches any subject. If the attribute is omitted, an initiating subject must be absent in the matching event.	An initiating subject with the provided name must exist.
`recipresource` *(string)* *(optional, default: empty)*	The name of the recipient resource to match. A value of "*" matches any resource. If the attribute is omitted, a recipient resource must be absent in the matching event.	A recipient resource with the provided name must exist.

Child Elements

None.

Schema Type

`auditruleType`

Location

- `/hcv/partition/audit/auditrule`

Supported Audit Actions

Table 4-49: Supported Audit Actions

Action	Description
`DISCARD`	Discard the audit record (do not store in the audit buffer).
`MAINTMODE`	Transition to the maintenance mode scheduling policy.
`SHUTDOWN`	Halt the SKH, including all subjects. This is an immediate halt, not a clean shutdown, so any information cached by subjects may be lost.
`STOREONLY`	Store the event as if no matching rule was found. A more specific rule with this action may be used to override a previously matched less specific rule that contains wildcards.

<subject> — Subject

Description

A subject is an executable entity (like a GuestOS) with access to a set of virtual resources (including virtual processors), and has access to other exported resources via configured information flows.

Attributes

Table 4-50: Attributes for <subject>

Name	Description	Additional Constraints
`sname` *(string ID, up to 48 chars)*	The name of the subject exported resource.	The name must not be shared with any other system resource name, exported resource name, or scheduling policy name within the same configuration vector.
`type` *(refer to Table 4-51 (page 89))*	The type of the subject.	
`isa` *(refer to Table 4-52 (page 89))* *(optional, default: `default`)*	The initial CPU Instruction Set Architecture in the subject. Whether subject code may switch to a different ISA at runtime depends on the architecture; please refer to the architecture documentation. Ignored for fully virtualized subjects on the x86 platform, which always start in 32-bit mode.	32-bit LynxSecure doesn't support 64-bit subjects.
`role` *(refer to Table 4-53 (page 89))* *(optional, default: `OTHER`)*	The role of the subject.	

Name	Description	Additional Constraints
`initparams` *(string)* *(optional, default: empty)*	A text string used to initialize the subject. Allows a single `BOOT` memory region to be used by multiple subjects with different results. The `initparams` for a subject are stored in the `ARGPAGE` memory region for which that subject has read permission.	The format of the `initparams` string is interpreted in a subject-specific fashion. The length of the string is limited to 4KB. While the `initparams` are optional, for many subjects they are required. E.g., some subjects may require initparams to set up their virtual networking.
`startaddr` *(hexadecimal or AUTO)* *(optional, default: `AUTO`)*	The guest address at which the subject begins execution. The address is a guest virtual address for paravirtualized subjects, and a guest physical address for fully virtualized subjects. For fully virtualized subjects, this attribute should be omitted or specified as `AUTO`. In that case, it is automatically set to the guest address of the virtual firmware.	PV subjects must have this attribute set to an explicit, non-`AUTO` value. If the `ramsize` attribute is also specified, the start address determines where the automatically created RAM starts in the subject.
`watchdog_timeout` *(integer; >0)* *(optional, default: empty)*	If specified, set the period for subject's watchdog to be strobed. The period is specified as a number of system clock ticks of subject's runtime.	If subject has different timeslices configured for its VCPUs, the runtime of its bootstrap VCPU is counted towards watchdog expiration. Configuration with such unbalanced VCPU timeslices is not recommended. Subject watchdog may not be used with real-time subjects.
`on_watchdog` *(refer to Table 4-54 (page 90))* *(optional, default: `none`)*	This attribute controls the action taken when a subject encounters watchdog timeout.	System state transitions, as requested by `maintenance`, `sys_restart` and `sys_shutdown` are subject to permission checks for system state transition validity. Subject's permissions to perform such transitions are not checked, however (as those permissions are applicable to the corresponding hypercalls).
`on_fatalfault` *(refer to Table 4-54 (page 90))* *(optional, default: `restart`)*	This attribute controls the action taken when a subject encounters a fatal fault, such as a triple fault on the x86 architecture, or a slave error on the ARM architecture.	Same constraints as described for `on_watchdog` attribute.
`restartsysperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission to restart the SKH and all subjects. Subjects are not cleanly shut down, so any information cached by subjects may be lost.	
`haltsysperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission to halt the SKH and all subjects. Subjects are not cleanly shut down, so any information cached by subjects may be lost.	
`skdbperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission to enter SKDB or execute SKDB commands via hypercalls. Note that SKDB entry pauses execution of all currently running subjects, including those executed on other CPUs and allows access to all system resources - effectively bypassing flow controls in the configuration vector.	Only allowed in the configuration with SKDB module enabled; ignored otherwise.

Name	Description	Additional Constraints
`readschedpolicyperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission to read the SKH scheduling policy id.	
`writeschedpolicyperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission to switch the SKH to a new scheduling policy.	
`maintperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission to switch the SKH to maintenance mode.	
`bitperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission to initiate built-in test.	
`flexreturnperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission to execute "Yield Donated Time" hypercall.	
`guesthypercallperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: `DENY`)*	Permission for guest code running in the subject to execute hypercalls.	This permission only applies to fully virtualized subjects; para-virtualized subjects are always permitted to issue hypercalls. This does not affect subject's ability to execute hypercalls while in "event capture" mode - that is, during device/BIOS emulation.
`mirrorhostbridgeid` *(boolean)* *(optional, default: `true`)*	Emulate the PCI host bridge with vendor and device IDs the same as present in hardware.	
`irqbase` *(integer, 0…255 or AUTO)* *(optional, default: `AUTO`)*	On the x86 platform, the interrupt vector generated by IRQ #0; this is the base for conversion between IRQs and interrupt vectors in a PV subject. Not used on the ARM platform.	Used only on the x86 platform and only with paravirtualized subjects. One the ARM platform, shouldn't be specified for any subject. Shouldn't normally be specified, because LynxSecure can find it out from the PV GuestOS image. If specified, PV Linux kernel must use the value 48. Other PV subjects must use the value 32. IRQ numbers generated by all devices assigned to the subject must be less than 256 minus the IRQ base of the subject.
`minorframeirq` *(integer or AUTO)* *(optional, default: empty)*	Enable the "minor frame" synthetic interrupt, delivered to the subject at the start of each of its minor scheduling frames, with the specified IRQ. This interrupt can be used in conjunction with the `timeroverride` attribute.	
`timeroverride` *(boolean)* *(optional, default: `false`)*	If set to `true`, disables the System Clock Tick (SCT) interrupt in a PV subject.	This attribute is only meaningful for PV subjects on the x86 platform. Other platforms don't have the SCT interrupt.
`caching` *(boolean)* *(optional, default: `true`)*	If set to false, all CPU data and instruction caches are disabled in the subject.	
`cacheflushafter` *(boolean)* *(optional, default: `false`)*	If set to true, CPU data and instruction caches are flushed when switching from this subject's minor frame to another	

Name	Description	Additional Constraints
	subject's minor frame on the same physical CPU.	
`apic` *(boolean)* *(optional, default: `true`)*	This attribute defines whether local APIC and I/O APIC emulation is provided for the subject (`true`) or not (`false`). If local APIC and I/O APIC are not enabled, the subject can only access the legacy PIC (8259A). By default, local and I/O APICs are enabled.	This attribute is only used for fully virtualized subjects on the x86 platform. It is ignored in all other cases.
`initialstate` *(refer to Table 4-55 (page 90))* *(optional, default: `RUNNING`)*	The initial state of the subject.	
`ramsize` *(hexadecimal, page multiple)* *(optional, default: empty)*	The size of a subject's RAM in bytes. This is a simplified syntax, alternative to explicitly configuring the subject's RAM layout using PROGRAM memory regions. When this attribute is used, the subject's RAM is still composed of PROGRAM memory regions, but those regions and the corresponding memory flows are generated automatically by LynxSecure. For a fully virtualized subject, its RAM is created according to the prototype hardware platform's typical RAM layout. For a para-virtualized subject, a single contiguous RAM region is created and mapped at the subject start address (the value of the `startaddr` attribute) rounded down to a page boundary (4KB).	This attribute is intended for simple guest physical address space configurations only. For fully virtualized subjects, custom non-RAM mappings in the subject address space should be located either not far below the guest physical address of 4GB, or above 4GB. For para-virtualized subjects, there shouldn't be any other memory regions mapped at where the RAM is going to be created. If the assumptions are wrong, the attribute's behavior is unspecified and the subject may fail to start or may malfunction after start. It is recommended to use automatic guest address allocation for custom mappings when using the `ramsize` attribute. The automatically created RAM is not guaranteed to be contiguous in the host physical memory even if it is contiguous in the guest address space. However, individual `PROGRAM` memory regions that compose the subject RAM are contiguous in the host address space. This is important when doing device DMA, except in fully virtualized subjects with an active I/O MMU. In the latter case, RAM only needs to be contiguous in the guest physical address space. A subject's layout of its RAM may be obtained in its read-only page. Note that `PROGRAM` memory regions that don't have a fixed start host physical address in the HCV may be automatically split by LynxSecure into multiple smaller regions.
`wbinvd` *(refer to Table 4-56 (page 90))* *(optional, default: `PERFORMANCE`)*	The emulation method for the WBINVD (Write Back and Invalidate Cache) CPU instruction. The WBINVD instruction flushes the entire data and instruction cache of the CPU it is executed on. If that CPU shares a cache with another CPU that runs another subject, this may interfere with that subject's operation despite the absence of an explicit information flow between the two subjects. Another	

Name	Description	Additional Constraints
	problem with this instruction is that it may take a long time to complete and is not interruptible during that time. This may interfere with LynxSecure subject scheduling or worsen the real-time interrupt response of the subjects. There are three possible ways that LynxSecure can handle this CPU instruction when guest code attempts to execute it: • No emulation: guest code is allowed to execute the WBINVD instruction directly. This handling is insecure, as described above. • Fast secure emulation: the WBINVD instruction is ignored. This handling is not applicable to all subjects. In particular, if physical devices assigned to the subject may perform Direct Memory Accesses (DMA) that don't snoop the CPU cache, guest code may want to use the WBINVD instruction to synchronize memory contents and the CPU cache contents. Skipping this instruction could result in an incorrect guest code behavior. • Slow secure emulation: only the data belonging to the subject that issued the WBINVD instruction is flushed from the CPU cache. Due to hardware limitations, this operation may take an extremely long time. The time it takes is directly proportional to the amount of RAM and other memory regions assigned to the subject. This emulation is only supported for fully virtualized subjects. LynxSecure provides the following options for configuring WBINVD behavior on a per-subject basis, called emulation methods. Each emulation method provides a different trade-off between security and performance. Note that if the hardware platform doesn't support WBINVD instruction emulation, the behavior is always "no emulation" regardless of this setting.	
`realtime` *(boolean)* *(optional, default: `false`)*	Improve the real-time response of the subject. For real-time subjects, LynxSecure programs the physical interrupt controller to honor interrupt priorities specified by the subject for its interrupts in its virtual interrupt controller. This prevents physical CPU interruptions from low priority interrupts when a higher	This attribute imposes some limitations on the subject. For example, a real-time subject's virtual CPUs can't share a physical CPU with virtual CPUs of other subjects. On some platforms with multiple physical interrupt controllers, real-time subjects cannot have their external device

Name	Description	Additional Constraints
	priority interrupt is being serviced in the real-time subject. LynxSecure also reduces its schedule-driving clock interrupt priority on a physical CPU occupied by a real-time subject to the lowest priority. However, some other internal LynxSecure interrupts may still be serviced at the highest interrupt priority.	interrupts come from more than one physical interrupt controller.
`userhypercalls` *(boolean)* *(optional, default: `false`)*	If set to true, hypercalls are allowed from user-level code.	Fully-virtualized subjects still require the `guesthypercallperm` permission to use this feature. Currently, this feature is only supported on the x86 platform. Guest code issuing a hypercall must have its current privilege level (CPL) less than or equal to its I/O privilege level (IOPL); otherwise, the hypercall returns a Permission Denied error code.
`devres` *(refer to Table 4-57 (page 91))* *(optional, default: `auto`)*	If set to `host`, allocate device resources at their host locations, if possible. This includes peripheral memory ranges, I/O ports and interrupts of all physical and some virtual devices assigned to this subject. The virtual device must be of the same type as a physical device present in the host system. For example, if the virtual device is an ARM Generic Interrupt Controller, but the host system has no interrupt controllers of that type, this option has no effect for that virtual device. This option allows for running GuestOS images in a LynxSecure subject, which don't use automatic configuration, but have the resource locations hardcoded for the host.	
`breakonstart` *(boolean)* *(optional, default: `false`)*	If set to true, a breakpoint is added at the subject start address for its boot VCPU. If external debugging is enabled, this results in entry to Debug state as soon as the subject starts. If there is no external debugger, the breakpoint is ignored. The breakpoint remains enabled afterwards, unless the external debugger clears it.	This option is only supported on the AArch64 platform and is ignored on the x86 platform.
`devicereset` *(boolean)* *(optional, default: `true`)*	Determines whether the subject is allowed to reset physical devices using a generic platform-specific mechanism (such as the FLR reset for the PCI bus).	
`pcie` *(boolean)* *(optional, default: `true`)*	PCI Express extended configuration space. This feature is required by some PCI Express devices. This feature requires a portion of the guest physical address space to be allocated for exclusive use by this feature, and therefore is optional. The location and the size of the exclusive use portion is determined automatically.	

Child Elements

The following sequence of child elements is expected:

- `platformfeatureflow` *(optional)*
- `memoryflow` *(optional)*
- `externaldeviceflow` *(optional)*
- `virtualdeviceflow` *(optional)*
- `messagebufferflow` *(optional)*
- `absoluteclockflow` *(optional, single instance)*
- `auditflow` *(optional, single instance)*
- `subjectflow` *(optional)*
- `virtualprocessor` *(required)*

Schema Type

`subjectType`

Location

- `/hcv/partition/subject`

Supported Subject Types

Table 4-51: Supported Subject Types

Type	Description
`PARAVIRT`	Paravirtualized subject.
`FULLVIRT`	Fully virtualized subject.

Instruction Set Architecture

Table 4-52: Instruction Set Architecture

Type	Description
`default`	Use the same instruction set as the LynxSecure hypervisor (defined by the `arch` attribute).
`32BIT`	32-bit. For the x86 platform, this means the i386 (IA-32) instruction set. For the ARM platform, this means the AArch32 instruction set.
`64BIT`	64-bit. For the x86 platform, this means the (Intel 64) instruction set. For the ARM platform, this means the AArch64 instruction set.

Supported Subject Roles

Table 4-53: Supported Subject Roles

Role	Description
`IBIT`	Startup/Initiated Built-In Test subject.
`CBIT`	CBIT subject.
`VDS`	A Virtual Device Server subject.

Role	Description
LSA	A subject running a LynxSecure Application or other software with minimal virtual memory management.
OTHER	A generic subject with no special privileges. LynxSecure makes no assumptions about the subject. The subject may run a full-featured Guest Operating System.

Subject Failure Actions

Table 4-54: Subject Failure Actions

Action	Description
none	No action is taken.
restart	Restart the subject. *Additional constraints:* • If memory sanitization is enabled for the subject, and the subject fails while executing in the sanitization context, the restart action will be replaced by a halt action. Restarting the subject would switch it back to normal mode, which would defeat the purpose of sanitization.
halt	Stop the subject.
maintenance	Switch SKH to Maintenance/Insecure state.
sys_restart	Restart the target.
sys_shutdown	Shut down the target.
panic	Report SKH panic.

Initial Subject States

Table 4-55: Initial Subject States

State	Description
RUNNING	The subject is initially running.
STOPPED	The subject is initially stopped.

Supported WBINVD Emulation Methods

Table 4-56: Supported WBINVD Emulation Methods

Emulation Method	Description
PERFORMANCE	The default behavior. Use the fast secure emulation, if it is correct for the subject, otherwise use no emulation (insecure).
NATIVE	Force insecure emulation (no emulation, guest code may execute the WBINVD instruction directly).
SECURE_FAST_DMA	Force secure emulation with fast DMA, but possibly slow WBINVD. Whenever guest code issues a WBINVD instruction, it is emulated in a way that doesn't interfere with other subjects. Physical devices are allowed to use non-snooping DMA. LynxSecure automatically selects either the fast secure or the slow secure emulation for WBINVD based on the subject's configuration. Thus, this emulation method may result in extremely slow performance of the WBINVD instruction. The slow secure emulation is selected if the subject is fully virtualized and is assigned physical PCIe

Emulation Method	Description
	devices capable of non-snooping DMA. This emulation method cannot be used in para-virtualized subjects that have assigned physical PCIe devices capable of non-snooping DMA. Because the WBINVD instruction is non-interruptible, this emulation method should be avoided for subjects that require real-time response. Alternatively, guest code running in the subject should avoid using the WBINVD instruction.
`SECURE_FAST_CACHE_FLUSH`	Force secure emulation with fast WBINVD, but possibly slower DMA (snooping instead of non-snooping). Whenever guest code issues a WBINVD instruction, it is emulated in a way that doesn't interfere with other subjects. Physical devices assigned to this subject are disallowed to use non-snooping DMA, making the fast secure WBINVD emulation correct for this subject.

Device Resource Allocation Policies

Table 4-57: Device Resource Allocation Policies

State	Description
`pool`	Allocate from the available resource pool without regard to the host resource location.
`host`	Attempt to allocate at the host location first, if that is impossible for any reason, fall back to allocation from the available resource pool.
`auto`	Choose the behavior more suitable for the platform. The x86 platform defaults to `pool` and the ARM platform defaults to `host`.

<platformfeatureflow> — Plaftorm Feature Flow

Description

A platform feature flow defines a subject-exported resource authorization for a subject to access a platform feature. Multiple subjects can have authorized flows to the same platform feature.

Attributes

Table 4-58: Attributes for <platformfeatureflow>

Name	Description	Additional Constraints
`featurename` *(string ID reference)*	The name of the feature.	The named feature must be defined as a `platformfeature` exported resource.
`readperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: empty)*	Permission to read the feature.	Must be the same as `writeperm`.
`writeperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: empty)*	Permission to write the feature.	Must be the same as `readperm`.

Child Elements

None.

Schema Type

`platformfeatureflowType`

Location

- `/hcv/partition/subject/platformfeatureflow`

<memoryflow> — Memory Flow

Description

A memory flow defines a subject-exported resource authorization for a subject to access a memory region. Multiple subjects can have authorized flows to the same memory region, with some exceptions.

Attributes

Table 4-59: Attributes for <memoryflow>

Name	Description	Additional Constraints
`memregname` *(string ID reference)*	The name of the exported resource that acts as the object of the flow.	The named memory region must be defined as a `memoryregion` or `guestimage` exported resource. Each memory flow for a given subject must have a different target memory region.
`gpa` *(hexadecimal, page multiple, or AUTO, or NONE)* *(optional, default: `auto`)*	This is the guest physical address of the memory region in the guest physical address space of the subject. If this attribute is not specified, it defaults to `AUTO`. If specified as `AUTO`, the address is allocated automatically from the available guest address space. If specified as `NONE`, the memory region is not mapped into the subject's GPA space, implying that the subject has other means of information flow to this memory region than by directly accessing it in memory.	The automatic allocation is always below the 4GB boundary. Note that this reduces the address space below 4GB available for the subject's RAM.
`gva` *(hexadecimal, page multiple, or AUTO, or NONE)* *(optional, default: `none`)*	This is the virtual address of the memory region in the guest virtual address space of a paravirtualized subject when the subject starts. If not specified, it defaults to `NONE`. If specified as `AUTO`, the address is allocated automatically from the available guest address space. If specified as `NONE` or not specified, the memory region is not mapped into the subject's initial GVA space, but may still be mapped to any virtual address later by modifying the page tables. The PV header in the subject's boot image may request that specific memory types are automatically mapped in the GVA space, overriding this value.	Only used with PV subjects, ignored for FV subjects.
`guestsize` *(hexadecimal, page multiple)* *(optional, default: empty)*	The length of the mapping defined by the `gpa` attribute (in FV subjects) or by the `gva` attribute (in PV subjects). If the attribute is omitted, the mapping is the same length as the memory region.	

Name	Description	Additional Constraints
`readperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: empty)*	Permission to read from the memory region.	May be restricted by the memory region type.
`writeperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: empty)*	Permission to write to the memory region.	May be restricted by the memory region type. If set to `ALLOW`, then `readperm` must also be set to `ALLOW`.
`owner` *(boolean)* *(optional, default: empty)*	Whether this flow makes the subject the owner of the memory region. If multiple "owner" flows to the same region from different subjects exist, the owner is determined by the first such flow. The owner of a memory region is the subject for which that memory region has a special function based on its type: for example, a PROGRAM memory region is the subject's RAM, an ROPAGE memory region is its Read-Only page, etc. A non-owner subject with a flow to a memory region can read or write its contents (if permitted by the flow), but the region doesn't have any special function based on its type for that subject. An exception to this are memory regions of the type BOOTSTRAP, which have a special function in a subject regardless if it's their owner or not. Another exception are BOOT memory regions, which have a special function in a subject not only if the subject owns them, but also if they are simply read-only (this lets multiple subjects boot from the same BOOT memory region). If the attribute is not specified, it defaults to true except when the memory flow references a memory region that disregards the ownership (BOOTSTRAP, SANSRC), that is designed to be shared between multiple subjects (SHM, VDEVIO, BITRESULTS) or special memory regions (RESERVED, BOOT). It also defaults to false for the flows to the read-only guest images (as such regions typically represent some additional boot images in a subject).	

Child Elements

None.

Schema Type

`memoryflowType`

Location

- `/hcv/partition/subject/memoryflow`

<externaldeviceflow> — External Device Flow

Description

An `externaldeviceflow` consists of a reference to an external device and permissions for the information flow between the subject and that device. Most devices should have both read and write permissions. A read-only permission is only supported for "pseudodevices", that is devices with no resources: no device memory, no I/O ports, no PCI configuration space registers and no interrupts. This type of devices occurs on platforms using Devicetree for configuration.

If the flow is to a device which is not present (or not active) at runtime, LynxSecure will halt with an error message indicating a missing device.

Only "pseudodevices" may be shared between multiple subjects. All other devices may only be assigned to a single subject.

Attributes

Table 4-60: Attributes for <externaldeviceflow>

Name	Description	Additional Constraints
`externaldevname` *(string ID reference)*	The name of the external device to create a flow to.	The name must refer to an external device.
`readperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: empty)*	Must be set to `ALLOW` to allow access to device.	External devices must have read/write or no-access flows; read-only or write-only flows are not supported.
`writeperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: empty)*	Must be set to `ALLOW` to allow access to device.	External devices must have read/write or no-access flows; read-only or write-only flows are not supported.
`vf` *(non-negative integer, or `AUTO`, or `ALL`, >or `NONE`)* *(optional, default: `none`)*	For SR-IOV devices, selects the Virtual Function (VF) that this flow applies to. If this attribute is absent or `none`, the Physical Function (PF) is selected. If this attribute is `auto`, the first available VF is selected at runtime. If this attribute is `all`, PF and all its VFs are assigned to the subject.	The VF of the specified number must be supported by the device. AUTO selects one VF automatically.
`config` *(string)* *(optional, default: empty)*	The configuration string for the device.	The string's format is specific to the device's type; this value is parsed by the device owner subject. Typically, it is a sequence of *`name`*=*`value`* pairs separated by spaces.

Child Elements

None.

Schema Type

`externaldeviceflowType`

Location

- `/hcv/partition/subject/externaldeviceflow`

<virtualdeviceflow> — Virtual Device Flow

Description

Create a flow to one of the interfaces of a virtual device.

Attributes

Table 4-61: Attributes for <virtualdeviceflow>

Name	Description	Additional Constraints
`vdevname` *(string ID reference)*	Virtual device to create a flow to.	The name must refer to a defined virtual device.
`interface` *(0 or 1)* *(optional, default: empty)*	The interface number of the virtual device to be used for this flow. Virtual devices that connect two subjects have two interfaces, interface #0 and interface #1. If not specified, LynxSecure will pick its default value as follows: if one of the interfaces in this virtual device is already occupied, use the other one; otherwise, if the flow is owned by a subject with the role `VDS`, then use interface #1, otherwise use interface #0.	Must not exceed the number of available interfaces defined for a virtual device.
`gpa` *(hexadecimal, page multiple, or AUTO)* *(optional, default: empty)*	Specifies the guest physical address where the memory region used by virtual device is mapped. If this attribute is not specified, it defaults to `AUTO`.	
`busnum` *(integer, 0…255 or AUTO)* *(optional, default: empty)*	The PCI bus number of the virtual device in the guest PCI address space.	Only applicable to PCI virtual device interfaces. In a flow, the `busnum`, `devnum` and `funcnum` attributes must be either all set to fixed numeric values or all unspecified/AUTO. LynxSecure supports PCI virtual devices on bus 0 only. Consequently, this attribute's value must be either 0 or unspecified/AUTO.
`devnum` *(integer, 0…31 or AUTO)* *(optional, default: empty)*	The PCI device number of the virtual device in the guest PCI address space.	Only applicable to PCI virtual device interfaces. In a flow, the `busnum`, `devnum` and `funcnum` attributes must be either all set to fixed numeric values or all unspecified/AUTO.
`funcnum` *(integer, 0…7 or AUTO)* *(optional, default: empty)*	The PCI function number of the virtual device in the guest PCI address space.	Only applicable to PCI virtual device interfaces. In a flow, the `busnum`, `devnum` and `funcnum` attributes must be either all set to fixed numeric values or all unspecified/AUTO.
`irq` *(integer or AUTO)*	The IRQ that this virtual device will use to report an interrupt to subject software.	The value specified by this attribute must be different from the one specified by the

Name	Description	Additional Constraints
(optional, default: empty)	For FV subjects, this value is used by device emulation software running in the subject transparently to guest software. The device IRQ visible to FV subject guest software is separated from this value by another layer of virtualization and may differ. If not specified or set to `AUTO`, the device emulation may allocate the IRQ automatically.	`notifyirq` attribute. The latter refers to the IRQ used for inter-subject communication by device emulation software. Only virtual devices may share the same value of this attribute in the same subject.
`notifyirq` *(integer or AUTO)* *(optional, default: `AUTO`)*	The IRQ (interrupt request number) used in this subject to report virtual device notifications from the peer subject. For FV subjects, this value is used by device emulation software running in the subject transparently to guest software.	The value specified by this attribute must be different from the one specified by the `irq` attribute.
`config` *(string)* *(optional, default: empty)*	The configuration string for the virtual device.	The string's format is specific to the virtual device; this value is parsed by the virtual device's implementation. Typically, it is a sequence of *`name`*=*`value`* pairs separated by spaces.
`readperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: empty)*	Must be set to `ALLOW` to allow access to device.	Virtual devices must have read/write flows; read-only, write-only or no-access flows are not supported.
`writeperm` *(one of: `ALLOW`/`DENY`)* *(optional, default: empty)*	Must be set to `ALLOW` to allow access to device.	Virtual devices must have read/write flows; read-only, write-only or no-access flows are not supported.

Child Elements

None.

Schema Type

`virtualdeviceflowType`

Location

- `/hcv/partition/subject/virtualdeviceflow`

<messagebufferflow> — Message Buffer Flow

Description

There are only two kinds of message buffer flows, one for message channel senders, and one for message channel receivers.

Each message channel must consist of exactly one unique sender subject, and one unique receiver subject. The flow to the receiver subject may specify a (synthetic) interrupt request number, generated to the receiver subject when the sender makes the message send hypercall.

Attributes

Table 4-62: Attributes for <messagebufferflow>

Name	Description	Additional Constraints
`msgbufname` *(string ID reference)*	Specifies the unique message buffer.	
`irq` *(integer or AUTO)* *(optional, default: empty)*	The synthetic interrupt generated by LynxSecure to the unique receiver subject associated with the message channel.	Caveat: choose the irq carefully. The irq should be chosen so that it does not collide with irqs already defined within a given subject. Example: if one specifies a irq of 0, that will collide with the subject's timer interrupt.
`readperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Set to `ALLOW` for receiver.	
`writeperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Set to `ALLOW` for sender.	

Child Elements

None.

Schema Type

`messagebufferflowType`

Location

- `/hcv/partition/subject/messagebufferflow`

<absoluteclockflow> — Absolute Clock Flow

Description

An absolute clock flow defines a subject-exported resource authorization for a subject to access the absolute clock.

Attributes

Table 4-63: Attributes for <absoluteclockflow>

Name	Description	Additional Constraints
`absclockname` *(string ID reference)*	The name of the exported resource that acts as the object of the flow.	The named absolute clock must be defined as an `absoluteclock` exported resource.
`readabsclockperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Permission to read the absolute time.	
`writeabsclockperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Permission to write the absolute time.	
`readcalibrationperm`	Permission to read the calibration value.	

Name	Description	Additional Constraints
(one of: ALLOW/DENY) *(optional, default: empty)*		
writecalibrationperm *(one of: ALLOW/DENY)* *(optional, default: empty)*	Permission to write the calibration value.	

Child Elements

None.

Schema Type

absoluteclockflowType

Location

- /hcv/partition/subject/absoluteclockflow

<auditflow> — Audit Flow

Description

An audit flow defines a subject-exported resource authorization for a subject to access an audit buffer. For a subject with read permission, an interrupt request number (irq) may be specified to alert the subject that an audit record has been queued to a previously empty buffer. If not specified, the reader subject must periodically poll the audit buffer to check for queued audit records.

Note that the audit buffer is intended to hold audit records generated by the SKH and by subjects trusted with respect to the SKH. It is not intended to provide a general- purpose audit capability for arbitrary subjects.

Attributes

Table 4-64: Attributes for <auditflow>

Name	Description	Additional Constraints
auditname *(string ID reference)*	The name of the exported resource that acts as the object of the flow.	The named audit buffer must be defined as an audit exported resource. Each audit flow for a given subject must have a different target audit buffer.
irq *(integer or AUTO)* *(optional, default: empty)*	If present, specifies the IRQ delivered to the subject when an audit event can be retrieved.	Must only be specified for a subject with read permission.
readperm *(one of: ALLOW/DENY)* *(optional, default: empty)*	Permission to read from the audit buffer.	Only one subject is permitted to have read access to the audit buffer.
writeperm *(one of: ALLOW/DENY)* *(optional, default: empty)*	Permission to write to the audit buffer.	

Child Elements

None.

Schema Type

`auditflowType`

Location

- `/hcv/partition/subject/auditflow`

<subjectflow> — Subject Flow

Description

The subject flow defines an inter-subject permission. While some inter-subject communications make use of shared memory mechanisms or message buffers, there are several operations that a subject can directly perform on another subject.

Attributes

Table 4-65: Attributes for <subjectflow>

Name	Description	Additional Constraints
`sname` *(string ID reference)*	The name of the exported resource that acts as the object of the flow.	The named subject must be defined as an `subject` exported resource. Each subject flow for a given subject must have a different target subject. A subject cannot have a flow to itself.
`startperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Permission to start the target subject.	
`suspendperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Permission to suspend the target subject.	
`resumeperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Permission to resume the target subject after being suspended.	
`stopperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Permission to stop (halt) the target subject.	
`restartperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Permission to restart the target subject.	
`statusperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Permission to get the status of the target subject.	
`flexdonateperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Permission to donate schedule time to another subject.	The target subject must be scheduled to run on the same processor core as this subject.
`injectanyirqperm` *(one of: `ALLOW/DENY`)* *(optional, default: empty)*	Permission to inject any interrupt to another subject. This provides the default setting that can be overridden by the children `injectintperm` elements.	

Child Elements

The following sequence of child elements is expected:

- `injectintperm` *(optional)*

Schema Type

`subjectflowType`

Location

- `/hcv/partition/subject/subjectflow`

<injectintperm> — Interrupt Injection Permission

Description

Permission to inject the identified interrupts into the target subject. Each interrupt request number is associated with a discrete permission.

Attributes

Table 4-66: Attributes for <injectintperm>

Name	Description	Additional Constraints
`irq` *(integer)*	Interrupt request number.	
`perm` *(one of: `ALLOW`/`DENY`)* *(optional, default: empty)*	Must be set to `ALLOW` to permit injection.	

Child Elements

None.

Schema Type

None (defined inside an element).

Location

- `/hcv/partition/subject/subjectflow/injectintperm`

<virtualprocessor> — Virtual Processor Flow

Description

In LynxSecure 3.1 each subject was associated with one unique virtual processor. LynxSecure 4.0 introduced Symmetric Multi-Processor (SMP) Subjects, so an SMP capable subject can have multiple virtual processor flows.

The virtual processors are allocated time on physical processors in accordance with the active scheduling policy.

Each subjectflow must have at least one virtualprocessor flow.

Attributes

Table 4-67: Attributes for <virtualprocessor>

Name	Description	Additional Constraints
`vpname` *(string ID, up to 63 chars)*	The name of the exported resource that acts as the object of the flow.	The name must not be shared with any other system resource name, exported resource name, scheduling policy name within the same configuration vector.
`affinity` *(string ID reference)* *(optional, default: empty)*	If present, binds the specified virtual processor to a certain physical CPU.	
`cachecapacity` *(hexadecimal)* *(optional, default: empty)*	If present, specifies the last level cache (LLC) capacity bitmask of the current virtual CPU. All virtual CPUs without an explicit cache capacity specification will automatically share the capacity unused by virtual CPUs that do specify the capacity. If the unused capacity is zero, LynxSecure startup will fail with an error.	

Child Elements

None.

Schema Type

`virtualprocessorType`

Location

- `/hcv/partition/subject/virtualprocessor`

APPENDIX A *Glossary*

Absolute Time
The number of seconds since the epoch (Midnight, 1 January 1970 Universal Coordinated Time). Also referred to as Wall Time.

Authorized Subject
has the ability to invoke privileged functions in the SKH Runtime Software. For example, an authorized subject can be given permission to request the SKH to startup or shutdown another subject. An authorized subject should be trusted by the system integrator to make proper use of its privileges, but is not trusted as part of SKH itself.

Autoconfig Tool
A LynxSecure® command line utility to obtain target system hardware data and generate an HCV based on that data and the desired configuration passed as arguments.

Binary Configuration Vector
A binary image of the unique configuration for a given SRP. Whenever the *Configuration Tool* is run, it produces an SRP which contains a Binary Configuration Vector.

Block Device Emulation (BDE)
Provides virtual IDE/AHCI controller interfaces to a fully virtualized subject.

Configuration Tool
A command line utility that converts a *Human Readable Configuration Vector* into the *Binary Configuration Vector*. It also enforces correct syntax and semantics in the data formats. Not to be confused with the *Autoconfig Tool*.

Configuration Vector
LynxSecure is configured using an offline configuration tool. The configuration vector is instantiated in multiple formats when in various stages of the system lifecycle: Human-Readable Configuration Vector (HCV), Binary Configuration Vector (BCV), and Runtime Configuration Vector (RCV).

Exported Resource
An exported resource is a resource made available by the Separation Kernel-Hypervisor (SKH) to one or more subjects.

Full Virtualization
A mechanism by which an operating system, designed to run directly on a computing platform, can execute as a subject guest operating system without modification (i.e., without knowledge that it has been virtualized). Some implementations of full virtualization still use paravirtualized device drivers; but the operating system kernel itself remains unmodified.

Guest Operating System (GuestOS)
An operating system executing in a virtualized context, rather than directly on the hardware with direct access to the hardware and privileged execution operations. When a GuestOS runs in LynxSecure it is also referred to as a Subject.
See Also Subject.

Human Readable Configuration Vector
An Extensible Markup Language (XML) file conformant to the XML Schema which defines LynxSecure configuration data.

Hypercall
An explicit software trap from subject code to the Hypervisor is a hypercall. A hypercall is similar to a system call on a non-virtual operating system environment. A hypercall allows indirect access to shared resources provided by the hypervisor.

Hypervisor or Virtual Machine Monitor (VMM)
For LynxSecure, software that runs directly on the hardware to provide virtualization for execution of multiple operating systems. In LynxSecure separation kernel and hypervisor are used interchangeably since both are realized at the same architectural layer.

Image
A contiguous collection of data and executable bits which makes up the LynxSecure on an executable hardware platform. The image is first deployed or downloaded as a file or a collection of files. When the hardware is powered on, the image is loaded from a storage medium, verified, and executed.

Interrupt
A system event, typically asynchronous, generated by a system device. An interrupt is generally used to indicate readiness of resources in an asynchronous manner. Interrupts usually have a well-defined source, path, and destination. Interrupts are usually "vectored," meaning that the control of some components of the source, path, and destination of a given interrupt are parameterized elements of an interrupt driven system.

Major Frame
A major frame consists of one or more minor frames. A major frame consists of a fixed amount of System Clock Tick (SCT) intervals. The sum of all SCT intervals assigned to each minor frame within a major frame equals the fixed amount of SCT intervals assigned to the major frame. A major frame is processor specific. Though two major frames may have identical minor frames, each major frame is distinct in that it is assigned to a unique processor and scheduling policy. In other words, a major frame is uniquely specified by its encompassing scheduling policy and assigned processor.

Memory region
A Memory region allows LynxSecure Security Kernel (LSK) to describe special handling required during the system boot up and during the working of the system.

Minor Frame
A minor frame consists of a subject id and a fixed number of SCT intervals. A minor frame binds a subject to a processor for a fixed amount of SCT intervals specified by the minor frame. Each minor frame belongs to a major frame.

Native
This is an operating system execution mode where the operating system has direct access to the hardware and privileged instructions without intermediary layers (such as a hypervisor or separation kernel). This is the typical way operating systems are deployed (no virtualization).

Paravirtualization
An approach of virtualization technology in which a guest OS is modified and recompiled prior to installation inside a virtual machine. A paravirtualized guest OS may run near native mode speed on the target hardware. This is also known as co-operative virtualization.

Read-Only Page (RO Page)
A per-subject memory structure describing the subject's configuration. Each subject can only access its own RO page, and the access is read-only. Despite the name, the structure is typically longer than one memory page. The RO page structure is defined in the `api.h` header file.

Reboot/Restart

Restarting a subject or the entire LynxSecure system under software control, without removing the power or (directly) triggering a reset line. It usually, though not always, refers to an orderly shutdown and restarting of the machine.

Resource

Resources are the totality of all hardware, firmware and software, and data that are executed, utilized, created, or protected by LynxSecure runtime software.

RMRR

An RMRR memory region corresponds to a Reserved Memory Region lying in host physical memory. RMRRs are associated with devices that need a range of host physical memory to be identity-mapped in the DMA redirection tables for the device. These regions are typically used by USB and on-board graphics devices. RMRRs are not visible to the CPU; they are only accessible by device DMA. RMRRs may overlap with other memory region types, and RMRR mappings (memory flows) may overlap with mappings of other memory region types; in this case, the RMRR takes precedence over the other regions for DMA transfers to the overlapping address range. RMRR locations are determined by the host BIOS and reported to other software via ACPI tables.

Scheduling Policy

A configuration encompassing a description of the subjects, and time frames involved and of how much time each subject executes upon all utilized processor cores on the target platform. A scheduling policy contains one major frame per processor core. A scheduling policy is the highest level of abstraction regarding scheduling. A scheduling policy consists of one or more major frames. Each major frame within a distinct scheduling policy is processor specific, meaning that each major frame is assigned to one and only one distinct processor. Only one scheduling policy is active at a time. A configuration vector can contain multiple scheduling policies.

Separation Kernel

A software layer responsible for providing secure, non-bypassable isolation and separation of platform resources amongst entities wishing to utilize the platform resources. In LynxSecure, separation kernel and hypervisor are used interchangeably since both are realized by the same architectural component.

Separation Kernel / Hypervisor

A portion of the LynxSecure system responsible for providing separation kernel and virtualization (hypervisor-VMM). While providing different functionality, the terms are often combined due to their close intertwining into overlapping software layers within the overall architecture. Includes hardware and/or firmware and/or software mechanisms whose primary function is to establish, isolate and separate multiple *subjects* and control information flow between the subjects and exported resources allocated to those subjects.

SKH Runtime Package

A data format stored as a binary image which contains the runtime software and configuration to be loaded into memory by the *SKH* bootloader.

Subject

A subject is an active entity that causes operations to be performed, and executes within the context of the LynxSecure *Separation Kernel / Hypervisor*. Since the only information flows which are allowed are between subjects and exported resources, a subject is also considered a resource so that subject to subject information flows are possible.

Synthetic Interrupt

Interrupts that are either generated internally by SKH and injected into a subject or initiated by a subject with right permissions and injected into another subject. The subject initiates the interrupt using a hypercall.

System Clock Tick

An event that causes each scheduler to be run in a near synchronous manner. SCTs occur at a periodic rate. The time between two SCTs is called a SCT interval. A SCT is an interrupt that is generated by a hardware timer. Also known as System Tick.

System Integrator
Develops code and configurations specific to a program or application to meet the needs of an end user.

Target Board
The realized instance of a circuit board which executes the software.

Target Platform
The realized instance of a hardware platform which executes the LynxSecure software composed of a target board and many other hardware elements.

Virtual KMA Switch
Provides support for switching the keyboard, mouse, audio and USB devices control between the fully virtualized subjects using key combinations.

Virtual KVM Switch
Provides support for switching the keyboard, mouse, video, audio and USB devices control between the fully virtualized subjects using key combinations.

Virtual Machine (VM)
The software that creates a virtualized environment upon which guest software executes. The following phrases are synonymous: "the Linux® and Windows® Guests," "the Linux and Windows guest OSs," and "the Linux and Windows virtual machines."

Virtual Network
An emulation of a network topology and node interfaces through software and wholly contained within the LynxSecure hardware platform itself. It uses well-known network protocols rather than custom network protocols. The virtual network can interface with an external network through software configuration and a physical NIC.

Virtualization
An added level of indirection and abstraction that hides physical characteristics of a resource from the way in which subjects use it. For LynxSecure, this allows multiple subjects (software) to utilize resources (hardware) where normally only a single subject could interface with the resource. This enables multiple operating systems to concurrently utilize a single hardware platform.

Wall Time
See *Absolute Time*.

Acronyms

Acronyms	Description
ACPI	Advanced Control and Power Interface
AHCI	Advanced Host Controller Interface
APIC	Advanced Programmable Interrupt Controller
BAR	Base Address Register
BCV	Binary Configuration Vector
BDE	Block Device Emulation
BIOS	Basic Input/Output System
BIT	Built-In Test
CBIT	Continuous Built-In Test
CD	Compact Disc
CDK	Cross-Development Kit
CPU	Central Processing Unit

Acronyms	Description
DMA	Direct Memory Access
FV	Fully Virtualized
HC	Host Controller
HCV	Human Readable Configuration Vector
HPA	Host Physical Address
I/O	Input/Output
IBIT	Initiated by an Event Built-in Test
I/O MMU	Input Output Memory Management Unit
IP	Internet Protocol
IRQ	Interrupt Request
ISA	Industry Standard Architecture
KMA	Virtual Keyboard Mouse Audio
KVM	Keyboard Video Mouse (usually, virtual)
LSA	LynxSecure Application
LSK	LynxSecure Separation Kernel
MAC	Media Access Control
NIC	Network Interface Card
OS	Operating System
PCI	Peripheral Component Interconnect
PNP	Plug-and-Play
PTE	Page Table Entry
PV	Para-Virtualized
PXE	Preboot Execution Environment
RAM	Random Access Memory
RIF	Runtime Init Function
RO	Read-Only
RMRR	Reserved Memory Range Region
SATA	Serial Advanced Technology Attachment
SBIT	Startup Initiated Built-In Test
SKDB	Separation Kernel Debugger
SKH	Separation Kernel Hypervisor
SMP	Symmetric Multiprocessing
SMT	Symmetric Multithreading
SRP	SKH Runtime Package
SW	Software
USB	Universal Serial Bus
VDS	Virtual Device Server
VGA	Video Graphics Array
VM	Virtual Machine
VT	Virtualization Technology
VT-d	Intel®'s VT for devices (includes I/O MMU)

Acronyms	Description
XML	Extensible Markup Language

APPENDIX B *Legacy Transition Guidance*

This Appendix is intended for users who need to upgrade a configuration XML (HCV) from previous versions of LynxSecure® to the current version.

Automatic Upgrade

The LynxSecure CDK contains a utility, `upgrade-hcv.sh`, for upgrading HCV configuration files from previous versions to the current version. This script automatically detects the schema version used by the HCV.

The `upgrade-hcv.sh` tool is run with the configuration file names to be upgraded passed as arguments. If the upgrade succeeds, the original file is saved with a `.orig` extension appended.

The `upgrade-hcv.sh` tool uses the `xsltproc` utility as its backend. This utility must be installed on the development host.

NOTE: For LynxSecure release 5.3, HCVs from earlier releases cannot be fully automatically upgraded due to significant changes in how virtual devices are specified. The `upgrade-hcv.sh` tool does upgrade the syntax to a certain extent, but isn't expected to produce a fully operational HCV. These HCVs should be re-created using the LynxSecure 5.3 Autoconfig tool.