# Xen Hyperlaunch Device Tree Bindings

The Xen hyperlaunch device tree adopts the dom0less device tree structure and extends it to meet the requirements for the hyperlaunch capability. The primary difference is the introduction of the `hypervisor` node that is under the `/chosen` node. The move to a dedicated node was driven by:

1. Reduces the need to walk over nodes that are not of interest, e.g. only nodes of interest should be in `/chosen/hypervisor`
2. Allows for the domain construction information to easily be sanitized by simple removing the `/chosen/hypervisor` node.

## Example Configuration

Below are two example device tree definitions for the hypervisor node. The first is an example of a multiboot-based configuration for x86 and the second is a module-based configuration for Arm.

## Multiboot x86 Configuration:

```
hypervisor {
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = “hypervisor,xen”

    // Configuration container
    config {
        compatible = "xen,config";

        module {
            compatible = "module,microcode", "multiboot,module";
            mb-index = <1>;
        };

        module {
            compatible = "module,xsm-policy", "multiboot,module";
            mb-index = <2>;
        };
    };

    // Boot Domain definition
    domain {
        compatible = "xen,domain";

        domid = <0x7FF5>;

        // FUNCTION_NONE            (0)
        // FUNCTION_BOOT            (1 << 0)
        // FUNCTION_CRASH           (1 << 1)
        // FUNCTION_CONSOLE         (1 << 2)
        // FUNCTION_XENSTORE        (1 << 30)
        // FUNCTION_LEGACY_DOM0     (1 << 31)
        functions = <0x00000001>;

        memory = <0x0 0x20000>;
        cpus = <1>;
        module {
            compatible = "module,kernel", "multiboot,module";
            mb-index = <3>;
        };

        module {
            compatible = "module,ramdisk", "multiboot,module";
            mb-index = <4>;
        };
        module {
            compatible = "module,config", "multiboot,module";
            mb-index = <5>;
        };

    // Classic Dom0 definition
    domain {
        compatible = "xen,domain";
```

```
        domid = <0>;

        // PERMISSION_NONE          (0)
        // PERMISSION_CONTROL       (1 << 0)
        // PERMISSION_HARDWARE      (1 << 1)
        permissions = <3>;

        // FUNCTION_NONE            (0)
        // FUNCTION_BOOT            (1 << 0)
        // FUNCTION_CRASH           (1 << 1)
        // FUNCTION_CONSOLE         (1 << 2)
        // FUNCTION_XENSTORE        (1 << 30)
        // FUNCTION_LEGACY_DOM0     (1 << 31)
        functions = <0xC0000006>;

        // MODE_PARAVIRTUALIZED     (1 << 0) /* PV | PVH/HVM */
        // MODE_ENABLE_DEVICE_MODEL (1 << 1) /* HVM | PVH */
        // MODE_LONG                (1 << 2) /* 64 BIT | 32 BIT */
        mode = <5>; /* 64 BIT, PV */

        // UUID
        domain-uuid = [B3 FB 98 FB 8F 9F 67 A3];

        cpus = <1>;
        memory = <0x0 0x20000>;
        security-id = “dom0_t;

        module {
            compatible = "module,kernel", "multiboot,module";
            mb-index = <6>;
            bootargs = "console=hvc0";
        };
        module {
            compatible = "module,ramdisk", "multiboot,module";
            mb-index = <7>;
        };
};
```

The multiboot modules supplied when using the above config would be, in order:

- (the above config, compiled)
- CPU microcode
- XSM policy
- kernel for boot domain
- ramdisk for boot domain
- boot domain configuration file
- kernel for the classic dom0 domain
- ramdisk for the classic dom0 domain

## Module Arm Configuration:

```
hypervisor {
    compatible = “hypervisor,xen”

    // Configuration container
    config {
        compatible = "xen,config";

        module {
            compatible = "module,microcode”;
            module-addr = <0x0000ff00 0x80>;
        };

        module {
            compatible = "module,xsm-policy";
            module-addr = <0x0000ff00 0x80>;

        };
    };

    // Boot Domain definition
    domain {
        compatible = "xen,domain";

        domid = <0x7FF5>;

        // FUNCTION_NONE            (0)
        // FUNCTION_BOOT            (1 << 0)
        // FUNCTION_CRASH           (1 << 1)
        // FUNCTION_CONSOLE         (1 << 2)
        // FUNCTION_XENSTORE        (1 << 30)
        // FUNCTION_LEGACY_DOM0     (1 << 31)
        functions = <0x00000001>;

        memory = <0x0 0x20000>;
        cpus = <1>;
        module {
            compatible = "module,kernel";
            module-addr = <0x0000ff00 0x80>;
        };

        module {
            compatible = "module,ramdisk";
            module-addr = <0x0000ff00 0x80>;
        };
        module {
            compatible = "module,config";
            module-addr = <0x0000ff00 0x80>;
        };

    // Classic Dom0 definition
    domain@0 {
        compatible = "xen,domain";

        domid = <0>;
```

```
        // PERMISSION_NONE          (0)
        // PERMISSION_CONTROL       (1 << 0)
        // PERMISSION_HARDWARE      (1 << 1)
        permissions = <3>;

        // FUNCTION_NONE            (0)
        // FUNCTION_BOOT            (1 << 0)
        // FUNCTION_CRASH           (1 << 1)
        // FUNCTION_CONSOLE         (1 << 2)
        // FUNCTION_XENSTORE        (1 << 30)
        // FUNCTION_LEGACY_DOM0     (1 << 31)
        functions = <0xC0000006>;

        // MODE_PARAVIRTUALIZED     (1 << 0) /* PV | PVH/HVM */
        // MODE_ENABLE_DEVICE_MODEL (1 << 1) /* HVM | PVH */
        // MODE_LONG                (1 << 2) /* 64 BIT | 32 BIT */
        mode = <5>; /* 64 BIT, PV */

        // UUID
        domain-uuid = [B3 FB 98 FB 8F 9F 67 A3];

        cpus = <1>;
        memory = <0x0 0x20000>;
        security-id = “dom0_t”;

        module {
            compatible = "module,kernel";
            module-addr = <0x0000ff00 0x80>;
            bootargs = "console=hvc0";
        };
        module {
            compatible = "module,ramdisk";
            module-addr = <0x0000ff00 0x80>;
        };
};
```

The modules that would be supplied when using the above config would be:

- (the above config, compiled into hardware tree)
- CPU microcode
- XSM policy
- kernel for boot domain
- ramdisk for boot domain
- boot domain configuration file
- kernel for the classic dom0 domain
- ramdisk for the classic dom0 domain

The hypervisor device tree would be compiled into the hardware device tree and provided to Xen using the standard method currently in use. The remaining modules would need to be loaded in the respective addresses specified in the *module-addr* property.

# The Hypervisor node

The hypervisor node is a top level container for the domains that will be built by hypervisor on start up. On the `hypervisor` node the `compatible` property is used to identify the type of hypervisor node present..

**compatible**

Identifies the type of node. Required.

# The Config node

A config node is for detailing any modules that are of interest to Xen itself. For example this would be where Xen would be informed of microcode or XSM policy locations. If the modules are multiboot modules and are able to be located by index within the module chain, the `mb-index` property should be used to specify the index in the multiboot module chain.. If the module will be located by physical memory address, then the `module-addr` property should be used to identify the location and size of the module.

**compatible**

Identifies the type of node. Required.

# The Domain node

A domain node is for describing the construction of a domain. It may provide a domid property which will be used as the requested domain id for the domain with a value of “0” signifying to use the next available domain id, which is the default behavior if omitted. A domain configuration is not able to request a domid of “0”. After that a domain node may have any of the following parameters,

**compatible**

Identifies the type of node. Required.

**domid**

Identifies the domid requested to assign to the domain. Required.

**permissions**

This sets what Discretionary Access Control permissions a domain is assigned. Optional, default is none.

**functions**

This identifies what system functions a domain will fulfill. Optional, the default is none.

> ***Note***
>
> The *functions* bits that have been selected to indicate `FUNCTION_XENSTORE` and `FUNCTION_LEGACY_DOM0` are the last two bits (30, 31) such that should these features ever be fully retired, the flags may be dropped without leaving a gap in the flag set.

**mode**

The mode the domain will be executed under. Required.

**domain-uuid**

A globally unique identifier for the domain. Optional, the default is NULL.

**cpus**

The number of vCPUs to be assigned to the domain. Optional, the default is “1”.

**memory**

The amount of memory to assign to the domain, in KBs. Required.

**security-id**

The security identity to be assigned to the domain when XSM is the access control mechanism being used. Optional, the default is “domu_t”.

# The Module node

This node describes a boot module loaded by the boot loader. The required compatible property follows the format: module,<type> where type can be “kernel”, “ramdisk”, “device-tree”, “microcode”, “xsm-policy” or “config”. In the case the module is a multiboot module, the additional property string “multiboot,module” may be present. One of two properties is required and identifies how to locate the module. They are the mb-index, used for multiboot modules, and the module-addr for memory address based location.

**compatible**

This identifies what the module is and thus what the hypervisor should use the module for during domain construction. Required.

**mb-index**

This identifies the index for this module in the multiboot module chain. Required for multiboot environments.

**module-addr**

This identifies where in memory this module is located. Required for non-multiboot environments.

**bootargs**

This is used to provide the boot params to kernel modules.

> ***Note***
>
> The bootargs property is intended for situations where the same kernel multiboot module is used for more than one domain.