Introduction to Microsemi Ethernet Switch API

User Guide

UG1070
Rev. 4.3.0
2017-12-15

Confidential

Table of Contents

1. Introduction

MSCC-ENT offers a single API to access the functionallity of ethernet switches and PHY's. Currently this API offers two different `C` interfaces, the `Unified-API` (which has existed for a long time), and the `MESA` layer. Both layers are equally supported, and customer can choose to use the layer which fits best into their design.

The difference between the `Unified-API` and `MESA`, is that public header files part of the `Unified-API` is heavily annotated with pre-processor statements (`#ifdef`, `#else` and `#endif`). These pre-processor statements is controlled by compiler defines, that is specifying a single chip (along with a list of features) supported by the binary library. These pre-processor statements is causing both the application programming interface (API) and application binary interface (ABI) of the library to differ for each supported chip and the optional features.

The `MESA` interface on the other hand is offering a stable API/ABI across all targets (and options) supported by a given release. Instead of using pre-processor statements the `MESA` interface is using capabilities to annotate what features is available in a given instance of the library. The `MESA` interface is thereby allowing customers to only build a single application, and link (load- or compile-time) it with the `MESA` library supporting a given chip.

Currently the `MESA` interface, is implemented as a thin layer on-top of the `Unified-API` library. The `MESA` layer is only converting between the stable API/ABI offered by the `MESA` headers, to the chip specific API/ABI offered by a given compilation of the `Unified-API`.

This means that the `Unified-API` and the `MESA` interfaces is offering the same level of abstraction. Documentation written for the `Unified-API` will also be useful if using the `MESA` API.

1.1. Scope

This document is intended as an introduction document of the `MESA` interface, not a reference or description of all the API's included as part of the `MESA` interface.

The document will describe the general terms/facilities that is available when the `MESA` interface is used.

Beside, from the core `MESA` library, the document also describe the `mesa_demo` application, which can be used to experiment with the API, using one of the reference boards.

1.2. Audience

The intended audience for this document is software developers who need to understand the facilities offered by the `MESA` interface. The document is expected to be useful in the following scenario's:

- Customers who is new to the API SW offered by MSCC-ENT, and wants to evaluate it for specific product/purpose.

- Customers who is starting up a new project based on the `MESA` interface.
- Customers who is considering chancing from the `Unified-API` to the `MESA` interface.

1.3. Prerequisites

This document assumes the reader possesses the following skills/resources:

1. Fluent in C and to some extent Makefiles.
2. Root access to a recent Linux development environment, and experienced in working with a Linux shell.
   a. Building new images from the sources (including boot-loader, BSP and application) requires a 64-bit Linux machine with at least 8GB of RAM, 50GB of disk space and 4 CPU cores. This document uses Ubuntu 16.04LTS as reference.
   b. Access to a TFTP server that can be used for SW upgrades.
   c. RS232 terminal to access the target.
3. MSCC API source package.
4. MSCC reference board (with WebStaX/SMBStaX/IStaX/CEServices 4.0 or newer).

2. Architecture

The `MESA` library is a driver which can help the application configuring the switch core and/or read status. The driver contains no threads, it implements no protocols and have no awareness of network frames. Without an application, the driver can not do anything.

The driver is OS agnostic, and requires that the application provide function pointers for accessing registers, locking, tracing etc.

2.1. The big picture

The MESA API is just a single component in a much larger system. This section will try to illustrate how/where the API fits in to the system architecture. This section will assume that the target system is running Linux.

2.1.1. Main Application

The application in the system which is linking in the MEBA/MESA libraries is referred to as the main application. This application is exclusively owning the API, and can use it to build the needed functionality. The main application will often expose a number of machine and/or human targeted interfaces that the outside world can use to apply configuration or query status. In turnkey systems delivered by MSCC, the WebStaX application is typically the main application, while in API projects, the customer either have an existing application or will design an application which owns the API. The MESA project does include a small and simple sample main-application which shows how to instantiate the API, and to setup simple stuff, this is the mini-application. The mini application is covered in more details later. See [ug1068] for more details.

2.1.2. MEBA

The MSCC Ethernet Board API (MEBA) is created to provide an abstract interface to all the board facilities. MSCC provides a MEBA library for each of the reference systems, and the main application should use it to handle the board specific details. If customers are doing a custom board, then they should also do an implementation of the MEBA library which matches their board. See [ug1069] for more details.

2.1.3. MESA

The MSCC Ethernet Switch API (MESA) is the component covered in this document.

2.1.4. Third-party Application

A system typically contains many applications running on the same CPU (either the integrated CPU or an external CPU), this is well supported. But there is only a single application which can instantiate the API, the main application. All other application is being referred to as third-party applications. Third-party applications can communicate with the Linux kernel, with other application, but if they need resources from MESA/MEBA, then it must go through the main application. In the WebStaX system, the WebStaX application is offering a JSON-IPC connection to third-party application which wants to add certain functionality.

2.1.5. Linux Kernel

The Linux kernel which can be found in the BSP, is offering kernel drivers for a number of the CPU peripherals (assuming the integrated CPU is used). The following subsections is elaborating a bit more on these facilities. See [ug1068] for more details.

2.1.5.1. I2C/SPI/UIO

When the MEBA/MESA libraries are instantiated, then function pointers to access register in the various devices must be provided. This is how these libraries can be OS agnostic.

To reach the register in the various peripherals, the switch-core is offering I2C and SPI controllers which is connected to the peripherals by the board. To reach the registers in the SwitchCore it self a UIO driver is exposed.

These buses/devices is typically exposed as character devices in the `/dev/` file system by the kernel. They can either be memory mapped, or the application can issues read/write commands to control the bus.

The main application can then use these devices, and provide function pointers for reading/writing the needed registers.

2.1.5.2. NIC(s)

The Switch Core have the concept of a CPU port, which allows frames to be moved between the CPU and the Switch core. Frames going in/out if the CPU port (typically) have an interface-frame-header which carry information on why the packet is being copied/moved to the CPU, or how they should be injected in the Switch Core.

The MSCC BSP will expose the CPU port as a normal NIC interface. The NIC driver will expose the frames as-is without doing any processing. This means that they will include the IFH, which carry important information. The application running in user-space can then open a raw socket and receive the frames, process then, (set up the MESA API if needed), and inject frames as a response.

2.1.5.3. NAND/NOR Drivers

The kernel also provide NAND/NOR drivers, which is exposed as block devices. A file system is typically created on-top of the block devices, which then can be used from user-space applications.

3. MESA source introduction

The key difference between the Unified-API and the MESA library, is that the MESA library is API/ABI identical for a given release. This means that a single application can be build once, and linked with the API matching the target.

Most of the API definitions in Unified-API and MESA, are very similar. In most cases the differences follows the rules explained in this section.

3.1. Co-Existing with Unified-API

The MESA library and the Unified-API are mutually exclusive from an application point of view. The application must choose if it want to use the one or the other, and only include header files from the desired layer.

3.2. Capabilities

The Unified-API is heavily annotated with `ifdef` which specifies what facilities is available in a given configuration. The MESA library does not use these pre-processor statements as they generally breaks API/ABI compatibility between different targets. Instead of using `ifdef`, the MESA layer uses capabilities.

A capability is an enumerated value, representing a given feature (see `mesa_cap_t`). Along with the list of defined capabilities, the MESA library implements a function to query the status of a given capability:

uint32_t mesa_capability(mesa_inst_t inst, int cap);

mscc/ethernet/switch/api/capability.h

```
typedef enum {
    MESA_CAP_MISC_GPIO_CNT = 0,  /**< Number of GPIOs */
    MESA_CAP_MISC_SGPIO_CNT,     /**< Number of SGPIO groups */
    MESA_CAP_MISC_PORT_GPIO,     /**< Port GPIO */
    MESA_CAP_MISC_INTERRUPTS,    /**< Interrupts */
    MESA_CAP_MISC_IRQ_CONTROL,   /**< IRQ control */
    MESA_CAP_MISC_FAN,           /**< Fan control */

    // Port
    MESA_CAP_PORT_CNT = 100,     /**< Maximum number of ports */

    // Many more
} mesa_cap_t;

uint32_t mesa_capability(mesa_inst_t inst, int cap);
```

The capabilities is used to signals booleans (if something is there or not) or to specify number (how many instances does the chip support).

The following part of this section explains how the capability system is ued to achieve the stable API/ABI.

3.2.1. Availability of functions

There are many examples of complete function groups, which only exists in certain targets. Either it is because new features are added in the chips, or because a given component is completely redesigned in such way that it is not possible to backwards compatible.

An example of this can be found in `vtss_afi_api.h`. A small simplified snippet of that file which illustrates how the pre-processors is used currently is shown below:

vtss_afi_api.h

```c
typedef u32 vtss_afi_id_t;
#if defined(VTSS_AFI_V1)
typedef struct {
    u32 fps;
    u32 actual_fps;
} vtss_afi_frm_dscr_t;
vtss_rc vtss_afi_alloc(const vtss_inst_t inst, vtss_afi_frm_dscr_t *const dscr,
                       vtss_afi_id_t *const id);
vtss_rc vtss_afi_free(const vtss_inst_t inst, vtss_afi_id_t id);
#endif /* defined(VTSS_AFI_V1) */

#if defined(VTSS_AFI_V2)
typedef u32 vtss_afi_fastid_t;

typedef struct {
    vtss_port_no_t port_no;
    vtss_prio_t prio;
    u32 frm_cnt;
} vtss_afi_fast_inj_alloc_cfg_t;

vtss_rc vtss_afi_fast_inj_alloc(const vtss_inst_t inst,
                                const vtss_afi_fast_inj_alloc_cfg_t *const cfg,
                                vtss_afi_fastid_t *const fastid);

vtss_rc vtss_afi_fast_inj_free(const vtss_inst_t inst, vtss_afi_fastid_t fastid);
#endif  // VTSS_AFI_V2
```

vtss_afi_api.h

```c
typedef uint32_t mesa_afi_id_t CAP(AFI_V1);
typedef struct {
    uint32_t fps;
} mesa_afi_frm_dscr_t CAP(AFI_V1);
typedef struct {
    uint32_t fps;
    u32 actual_fps;
} mesa_afi_frm_dscr_actual_t CAP(AFI_V1);

mesa_rc mesa_afi_alloc(const mesa_inst_t          inst,
                       const mesa_afi_frm_dscr_t  *const dscr,
                       mesa_afi_frm_dscr_actual_t *const actual,
                       mesa_afi_id_t              *const id) CAP(AFI_V1);

mesa_rc mesa_afi_free(const mesa_inst_t inst,
                      mesa_afi_id_t     id) CAP(AFI_V1);

typedef uint32_t mesa_afi_fastid_t CAP(AFI_V2);
typedef struct {
    mesa_port_no_t port_no;
    mesa_prio_t prio;
    uint32_t frm_cnt;
} mesa_afi_fast_inj_alloc_cfg_t CAP(AFI_V2);

mesa_rc mesa_afi_fast_inj_alloc(
        const mesa_inst_t                    inst,
        const mesa_afi_fast_inj_alloc_cfg_t *const cfg,
        mesa_afi_fastid_t                   *const fastid) CAP(AFI_V2);

mesa_rc mesa_afi_fast_inj_free(const mesa_inst_t       inst,
                               mesa_afi_fastid_t fastid) CAP(AFI_V2);
```

Instead of having either `VTSS_AFI_V1` or `VTSS_AFI_V2` the header files now contains the prototypes of both. The source code is annotated with `CAP(XXX)` which is a macro expanding to nothing. The purpose of this is to document what capability the application should use to check if a given group of functions are supported or not.

An example of this is show here:

Application code with ifdef's

```c
#if defined(VTSS_AFI_V1)
    vtss_afi_alloc(inst, ...);
#elif defined(VTSS_AFI_V2)
    vtss_afi_v2_fast_inj_free(inst, ...);
#endif
```

Application code without ifdef's

```c
if (mesa_ent_api_capability(inst, MESA_CAP_AFI_V1)) {
    mesa_afi_alloc(inst, ...);
} else if (mesa_ent_api_capability(inst, MESA_CAP_AFI_V2)) {
    mesa_afi_fast_inj_alloc(inst, ...);
}
```

If a function is called, even though it is not implemented (according to the capability system), then it must return `MESA_RC_NOT_IMPLEMENTED`.

3.2.2. Availability of `struct` members

In the Unified API the pre-processor is often used to express small variations of the different targets supported by the API. This is an efficient way of expressing these differences, but the downside is that the application needs to have the same level of `ifdef` abstractions and thereby causing the API and APPL to be very closely bound together.

Following is an example of this pattern:

vtss_afi_api.h

```c
typedef struct {
  vtss_vcap_vr_t dscp;
#if defined(VTSS_FEATURE_ECE_KEY_IP_PROTO)
  vtss_vcap_bit_t fragment;
  vtss_vcap_u8_t proto;
#endif
#if defined(VTSS_FEATURE_ECE_KEY_SIP)
  vtss_vcap_ip_t sip;
#endif
#if defined(VTSS_FEATURE_ECE_KEY_DIP)
  vtss_vcap_ip_t dip;
#endif
#if defined(VTSS_FEATURE_ECE_KEY_SPORT)
  vtss_vcap_vr_t sport;
#endif
#if defined(VTSS_FEATURE_ECE_KEY_DPORT)
  vtss_vcap_vr_t dport;
#endif
} vtss_ece_frame_ipv4_t;
```

In reality, most targets actually supports most the keys. The `ifdef` will allow saving a few bytes, but a detailed analysis of this specific case showed that the `dip` field was the only difference when considering the current set of targets.

In the cases where we find that the actual difference in sizes, can be neglected, then we simply let the MESA header files include the super set of all members.

This is illustrated below:

mscc/ethernet/switch/api/evc.h

```c
typedef struct {
  mesa_vcap_vr_t  dscp;
  mesa_vcap_bit_t fragment;
  mesa_vcap_u8_t  proto;
  mesa_vcap_ip_t  sip;
  mesa_vcap_ip_t  dip CAP(EVC_ECE_DIP);
  mesa_vcap_vr_t  sport;
  mesa_vcap_vr_t  dport;
} mesa_ece_frame_ipv4_t CAP(EVC_ECE_CNT);
```

The `CAP(x)` macro is again used to annotate what capability is used to signal the availability of the different members.

If this approach (pulling in the super set of members) will cause a significant overhead, then the structure needs to be split into smaller structures. The majority of the structures was designed in such a way that all members could be included without causing significant overhead.

The few cases where a super set of all flags, is posing too big overhead, needed to be handled differently. This is covered in the next section.

3.2.2.1. Structure splitting

Just adding the super set of all attributes is not always the right thing to do, especially if that is causing large memory overhead, or if the extra attributes is very special/ exotic and only useful in rare cases.

An example of this can be found in the MAC table, snippet is included below:

vtss_l2_api.current.h

```c
typedef struct {
  vtss_vid_mac_t vid_mac;
  BOOL destination[VTSS_PORT_ARRAY_SIZE];
  BOOL copy_to_cpu;
  BOOL locked;
  BOOL aged;
#if defined(VTSS_FEATURE_MAC_CPU_QUEUE)
  vtss_packet_rx_queue_t cpu_queue;
#endif

#if defined(VTSS_FEATURE_VSTAX_V2)
  struct {
    BOOL enable;
    BOOL remote_entry;
    vtss_vstax_upsid_t upsid;
    vtss_vstax_upspn_t upspn;
  } vstax2;
#endif

#if defined(VTSS_FEATURE_SEAMLESS_REDUNDANCY)
  struct {
    BOOL enable;
    vtss_sr_stream_id_t id;
  } sr;
#endif
} vtss_mac_table_entry_t;

vtss_rc vtss_mac_table_add(const vtss_inst_t inst,
                           const vtss_mac_table_entry_t *const entry);

vtss_rc vtss_mac_table_get(const vtss_inst_t inst,
                           const vtss_vid_mac_t *const vid_mac,
                           vtss_mac_table_entry_t *const entry);
```

The `destination` will be translated to a `mesa_port_mask_t` as illustrated earlier. The `cpu_queue` will just be included always with a `MCAP` attribute, but the `vstax2` and `sr` are quite special and does consume some space - we do not want to included them every representation of a mac-table in user space.

mscc/ethernet/switch/api/l2.suggested.h

```c
typedef struct {
  mesa_vid_mac_t vid_mac;
  mesa_port_mask_t destination;
  BOOL copy_to_cpu;
  BOOL locked;
  BOOL aged;
  mesa_packet_rx_queue_t cpu_queue MCAP(MAC_CPU_QUEUE);
} mesa_mac_table_entry_t;

struct {
    BOOL enable;
    BOOL remote_entry;
    mesa_vstax_upsid_t upsid;
    mesa_vstax_upspn_t upspn;
} mesa_mac_table_entry_vstax2_t;

struct {
    BOOL enable;
    mesa_sr_stream_id_t id;
} mesa_mac_table_entry_sr_t;

mesa_rc mesa_mac_table_add(const mesa_inst_t inst,
                           const mesa_mac_table_entry_t *const entry);

mesa_rc mesa_mac_table_get(const mesa_inst_t inst,
                           const mesa_vid_mac_t *const vid_mac,
                           mesa_mac_table_entry_t *const entry);

mesa_rc mesa_mac_table_adv_add(const mesa_inst_t inst,
                               const mesa_mac_table_entry_t *const entry,
                               const mesa_mac_table_entry_vstax2_t *const vstax,
                               const mesa_mac_table_entry_sr_t *cosnt sr);

mesa_rc mesa_mac_table_adv_get(const mesa_inst_t inst,
                               const mesa_vid_mac_t *const vid_mac,
                               mesa_mac_table_entry_t *const entry,
                               mesa_mac_table_entry_vstax2_t *const vstax,
                               mesa_mac_table_entry_sr_t *cosnt sr);
```

The `vstax2` and `sr` structures has been removed from the `mesa_mac_table_entry_t`, and the `mesa_mac_table_entry_t` has thereby been unified for all target platforms. Users that needs to control the `vstax2` and/or `sr` fields will have to use the `mesa_mac_table_adv_add` method instead of the `mesa_mac_table_add` method.

3.2.3. Fixed length arrays

Defines known at compile time can be used to specify arrays, but if the length is a runtime parameter, which is the case with return values from `mesa_capability`, then the storage for the array needs to be allocated dynamic.

The Unified-API generally do not allocate any memory after it has been instantiated, and the same is true for the MESA library. It is always the responsibility of the application to allocate/handle-error/free any needed memory resources. The MESA library does not change that.

3.2.3.1. Port masks

Many data types uses a pre-processor define to specify the length of an array. That will again cause the ABI of the header to depend on the actual configuration, because most platforms has a different port count.

An example of this can be see below:

vtss_l2_api.h

```
typedef struct {
  BOOL port_list[VTSS_PORT_ARRAY_SIZE];
  vtss_vce_mac_t mac;
  vtss_vce_tag_t tag;
  vtss_vce_type_t type;
  union { ...  } frame;
} vtss_vce_key_t;
```

Below is illustrated how the same behavioral can be achieved without making the ABI depending on the actual configuration.

mscc/ethernet/switch/api/l2.h

```
typedef struct {
  mesa_port_mask_t port_list;
  vtss_vce_mac_t mac;
  vtss_vce_tag_t tag;
  vtss_vce_type_t type;
  union { ...  } frame;
} vtss_vce_key_t;
```

Instead of using an array of booleans, a port-mask is used instead. The port-mask can be dimensioned for worst case (current implementation is using 64bits, supporting up-to 64 ports), still be more space effective than the array of booleans.

An ABI compatible C header files can be used to emulate the old behavioral such that users in a C context will not notice that they are operating on a mask instead of an array. The trick is: if the header file defining `vtss_port_mask_t` is included by a C++ compiler, then it defines the `operator[]` without changing the API.

3.2.3.2. Other arrays

Creating bit masks to replace arrays, will only work for port-lists. Doing worst case allocation for other types will cause a significant overhead in terms of memory, which can not be accepted.

Instead of doing worst-case allocation, the date structures needs to be re-organized and/or the application need to dynamic allocate arrays based on the capability values returned by `mesa_capability`.

Lets consider an other example where the number of ports is being used to specify the length of an array. The following two snippets shows how a function in the Unified-API, and the MESA API.

vtss_l2_api.h

```
vtss_rc vtss_vlan_tx_tag_set(
    const vtss_inst_t        inst,
    const vtss_vid_t         vid,
    const vtss_vlan_tx_tag_t tx_tag[VTSS_PORT_ARRAY_SIZE]);
```

mscc/ethernet/switch/api/l2.h

```
mesa_rc mesa_vlan_tx_tag_set(const mesa_inst_t        inst,
                             const mesa_vid_t         vid,
                             const uint32_t           count,
                             const mesa_vlan_tx_tag_t *const tx_tag);
```

The Unified-API know the dimension at compile time, and can therefore use it directly in the prototype. This is not the case in the MEAS library, where it is parsed as a pointer, along with a `count`.

To call the MESA function from a `C` application will require that either the application know the size of the array, or that it uses the capability system to determinate it. A `C++` application can benefit from using the `CapArray` to take care of the memory handling.

Lets have a look at the three different scenarios (all examples assume the API has initialized before use):

c_application_with_static_sizes.c

```
#define PORT_CNT 8
int some_function() {
    mesa_vlan_tx_tag_t tags[PORT_CNT];
    memset(tags, PORT_CNT, sizeof(mesa_vlan_tx_tag_t));
    // fill in the data
    return mesa_vlan_tx_tag_set(NULL, 1, PORT_CNT, tags);
}
```

c_application_with_dynamic_sizes.c

```
int some_function() {
    int res = 0;
    int cnt = mesa_capability(NULL, MESA_CAP_PORT_CNT);
    mesa_vlan_tx_tag_t *tags = calloc(cnt, sizeof(mesa_vlan_tx_tag_t));
    if (!tags) {
        return -1;
    }

    // fill in the data

    res = mesa_vlan_tx_tag_set(NULL, 1, PORT_CNT, tags);

    free(tags);

    return res;
}
```

caparray_application.cxx

```
int some_function() {
    CapArray<mesa_vlan_tx_tag_t, MESA_CAP_PORT_CNT> tags;
    // fill in the data
    return mesa_vlan_tx_tag_set(NULL, 1, tags.size(), tags);
}
```

The three examples does the same thing, it is just different ways of doing it. Application should choose the approach that fits best into the current design, and the one they are most comfortable with.

3.2.4. Case study - QOS

Following is a more complete example taken from the QOS module, more specific the `vtss_qos_port_conf_t` structure. The structure has lots of `ifdef`, arrays with target specific constants, and features that takes up significant amount of space that is kind of unrelated. See the original code below:

vtss_qos_api.h

```
typedef struct {
    vtss_policer_t policer_port[VTSS_PORT_POLICERS];
    vtss_policer_ext_t policer_ext_port[VTSS_PORT_POLICERS];
    vtss_policer_t     policer_queue[VTSS_QUEUE_ARRAY_SIZE];
    vtss_shaper_t  shaper_port;
    vtss_shaper_t shaper_queue[VTSS_QUEUE_ARRAY_SIZE];
#if defined(VTSS_FEATURE_QOS_EGRESS_QUEUE_SHAPERS_EB)
    BOOL          excess_enable[VTSS_QUEUE_ARRAY_SIZE];
#endif
#if defined(VTSS_FEATURE_QOS_EGRESS_QUEUE_SHAPERS_CRB)
    BOOL          credit_enable[VTSS_QUEUE_ARRAY_SIZE];
#endif
#if defined(VTSS_FEATURE_QOS_EGRESS_QUEUE_CUT_THROUGH)
    BOOL          cut_through_enable[VTSS_QUEUE_ARRAY_SIZE];
#endif
#if defined(VTSS_FEATURE_QOS_FRAME_PREEMPTION)
    vtss_qos_fp_port_conf_t fp;
    BOOL         frame_preemption_enable[VTSS_QUEUE_ARRAY_SIZE];
#endif
    vtss_prio_t    default_prio;
    vtss_tagprio_t usr_prio;
    vtss_dp_level_t   default_dpl;
    vtss_dei_t        default_dei;
    BOOL              tag_class_enable;
    vtss_prio_t       qos_class_map[VTSS_PCP_ARRAY_SIZE][VTSS_DEI_ARRAY_SIZE];
    vtss_dp_level_t   dp_level_map[VTSS_PCP_ARRAY_SIZE][VTSS_DEI_ARRAY_SIZE];
    BOOL              dscp_class_enable;
    vtss_dscp_mode_t  dscp_mode;
    vtss_dscp_emode_t dscp_emode;
    BOOL              dscp_translate;
    vtss_tag_remark_mode_t tag_remark_mode;
    vtss_tagprio_t         tag_default_pcp;
    vtss_dei_t             tag_default_dei;
    vtss_tagprio_t         tag_pcp_map[VTSS_PRIO_ARRAY_SIZE][2];
    vtss_dei_t             tag_dei_map[VTSS_PRIO_ARRAY_SIZE][2];
    BOOL        dwrr_enable;
#if defined(VTSS_FEATURE_QOS_SCHEDULER_DWRR_CNT)
    u8          dwrr_cnt;
#endif
    vtss_pct_t queue_pct[VTSS_QUEUE_ARRAY_SIZE];
#if defined(VTSS_FEATURE_QCL_DMAC_DIP)
    BOOL        dmac_dip;
#endif
#if defined(VTSS_FEATURE_QCL_KEY_TYPE)
    vtss_vcap_key_type_t key_type;
#endif
#if defined(VTSS_FEATURE_QOS_WRED_V3)
    vtss_wred_group_t wred_group;
#endif
#if (defined VTSS_FEATURE_QOS_COSID_CLASSIFICATION)
    vtss_cosid_t cosid;
#endif
#if (defined VTSS_FEATURE_QOS_INGRESS_MAP)
    vtss_qos_ingress_map_id_t ingress_map;
#endif
#if (defined VTSS_FEATURE_QOS_EGRESS_MAP)
    vtss_qos_egress_map_id_t egress_map;
#endif
#if defined(VTSS_FEATURE_QOS_EGRESS_QUEUE_SHAPERS_TAS)
    vtss_qos_qbv_port_conf_t qbv;
#endif
} vtss_qos_port_conf_t;
```

mscc/ethernet/switch/api/qos.h

```
typedef struct {
    mesa_shaper_t              shaper;
    mesa_prio_t                default_prio;
    mesa_dp_level_t            default_dpl;
    mesa_qos_port_tag_conf_t   tag;
    mesa_qos_port_dscp_conf_t  dscp;
    mesa_bool_t                dwrr_enable;
    uint8_t                    dwrr_cnt CAP(QOS_SCHEDULER_CNT_DWRR);
    mesa_bool_t                dmac_dip CAP(QOS_QCL_DMAC_DIP);
    mesa_vcap_key_type_t       key_type CAP(QOS_QCL_KEY_TYPE);
    mesa_wred_group_t          wred_group CAP(QOS_WRED_GROUP_CNT);
    mesa_cosid_t               cosid CAP(QOS_COSID_CLASSIFICATION);
    mesa_qos_ingress_map_id_t  ingress_map CAP(QOS_INGRESS_MAP_CNT);
    mesa_qos_egress_map_id_t   egress_map CAP(QOS_EGRESS_MAP_CNT);
    mesa_qos_qbv_port_conf_t   qbv CAP(QOS_EGRESS_QUEUE_SHAPERS_TAS);
    mesa_qos_fp_port_conf_t    fp CAP(QOS_FRAME_PREEMPTION);
    mesa_qos_port_queue_conf_t queue[MESA_QUEUE_ARRAY_SIZE];
} mesa_qos_port_conf_t;

mesa_rc mesa_qos_port_conf_set(const mesa_inst_t inst,
                               const mesa_port_no_t port_no,
                               mesa_qos_port_conf_t *const conf);

typedef struct {
    mesa_policer_t  policer;
    mesa_bool_t     frame_rate;
    mesa_dp_level_t dp_bypass_level CAP(QOS_PORT_POLICER_EXT_DPBL);
    mesa_bool_t known_unicast        CAP(QOS_PORT_POLICER_EXT_TTM);
    mesa_bool_t known_multicast      CAP(QOS_PORT_POLICER_EXT_TTM);
    mesa_bool_t known_broadcast      CAP(QOS_PORT_POLICER_EXT_TTM);
    mesa_bool_t unknown_unicast      CAP(QOS_PORT_POLICER_EXT_TTM);
    mesa_bool_t unknown_multicast    CAP(QOS_PORT_POLICER_EXT_TTM);
    mesa_bool_t unknown_broadcast    CAP(QOS_PORT_POLICER_EXT_TTM);
    mesa_bool_t learning             CAP(QOS_PORT_POLICER_EXT_TTM);
    mesa_bool_t limit_noncpu_traffic CAP(QOS_PORT_POLICER_EXT_TTM);
    mesa_bool_t limit_cpu_traffic    CAP(QOS_PORT_POLICER_EXT_TTM);
    mesa_bool_t flow_control;
} mesa_qos_port_policer_conf_t;

mesa_rc mesa_qos_port_police_conf_set(const mesa_inst_t inst,
                                      mesa_port_no_t port_no, uint32_t cnt,
                                      mesa_qos_port_policer_conf_t *const pol);
```

Following is a listing of the different techniques which has been applied in the example. Note that this example only covers the changes to `vtss_qos_port_conf_t`, similar initiatives will be need in almost all other structures defined in the `vtss_qos_api.h` header file.

1. Defines that is not active anymore has been removed. The QOS module has a long history, and earlier versions supported Luton28 and JR1 targets which is significant different from the current collection of chips. These old chips is not supported anymore, which means that many of the `ifdef` does not have any effect.
2. Members/variables that is only available on specific targets has been guarded with the `CAP()` macro.
3. Queue and police attributes were represented as chip dependent arrays in the current version, they have been separated out and is now handled by dedicated `get`/`set` functions that will allow run-time definitions of length parameter.

The result is an interface that is `ABI` compatible across the different targets, it does not cause any significant overhead and the changes in how the API must be accessed are minimal.

4. Build system, auto-generation and migration

The Unified-API and MESA library uses CMake as build system. The CMake system allows to specify what targets to include in a given build, and to use an existing cross compiler.

4.1. Development environment

Working with the source code raises some requirements to the development environment. This section will provide instructions on how to set-up a development machine based on x86_64 Ubuntu 16.04LTS installation. Other (recent) Linux distributions can be used, but that is not supported by MSCC. Setting up the development environment requires `root` access through the `sudo` command.

First step is to install a bunch of required packages using the package system provided by Ubuntu:

```
$ sudo apt-get install bc build-essential bzip2 coreutils cpio findutils gawk git
grep gzip libc6-i386 libcrypt-openssl-rsa-perl libncurses5-dev patch perl python
ruby sed squashfs-tools tcl tar wget libyaml-tiny-perl libcgi-fast-perl ruby-parslet
cmake
```

Next step is to download and install the binary BSP. This example will be using `2017.02-035` as example, but future releases may depend on newer versions.

Use the following link to browse the released BSPs: http://mscc-ent-open-source.s3-website-eu-west-1.amazonaws.com/. The steps below will download, install and test that the installed binaries work:

```
$ cd
$ wget -q http://mscc-ent-open-source.s3-eu-west-1.amazonaws.com/public_root/bsp/
mscc-brsdk-mips-2017.02-035.tar.gz
$ sudo mkdir -p /opt/mscc
$ sudo tar xf mscc-brsdk-mips-2017.02-035.tar.gz -C /opt/mscc
$ /opt/mscc/mscc-brsdk-mips-2017.02-035/stage2/smb/x86_64-linux/usr/bin/
mipsel-buildroot-linux-gnu-gcc --version
mipsel-linux-gcc.br_real (Buildroot 2016.05-git) 5.3.0
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

The final step is to extract the API sources. The example below shows how to extract API.

```
$ cd
$ tar -xf API_x.yy.tar.gz # File name/version may differ, depending on the release
```

4.2. Building the sources

Building sources using CMake, is a two-step operation. First step is to configure the project. To do that the `cmake` or the `ccmake` command is used. The latter provide a curses GUI, where the different options can be selected, while the first require you to specify all the options on the command line. For more details, read the manuals provided along with CMake.

When CMake has configured a project, it will generate a set of build files, in this case Makefile’s, which can be used to do the actual build.

This example is configuring a project to use the compiler from the BSP which was just installed, and is enabling the ocelot demo target.

The toolchain (and thereby the cross compiler) must be specified on the command line the first time CMake is invoked. The only way to change the toolchain is to delete the build folded, and start over.

```
$ cd
$ cd API_X.Y/vtss_api    # Enter the source folder
$ mkdir build; cd build   # Create and enter build directory
$ cmake -DCMAKE_TOOLCHAIN_FILE=/opt/mscc/mscc-brsdk-mips-2017.02-035/stage2/smb/x86_64-linux/usr/share/buildroot/toolchainfile.cmake -Dmesa_demo_ocelot_vsc7514=ON ..
-- The C compiler identification is GNU 6.3.0
-- The CXX compiler identification is GNU 6.3.0
-- Check for working C compiler: /opt/mscc/mscc-brsdk-mips-2017.02-035/stage2/smb/x86_64-linux/usr/bin/mipsel-buildroot-linux-gnu-gcc
-- Check for working C compiler: /opt/mscc/mscc-brsdk-mips-2017.02-035/stage2/smb/x86_64-linux/usr/bin/mipsel-buildroot-linux-gnu-gcc - works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /opt/mscc/mscc-brsdk-mips-2017.02-035/stage2/smb/x86_64-linux/usr/bin/mipsel-buildroot-linux-gnu-g++
-- Check for working CXX compiler: /opt/mscc/mscc-brsdk-mips-2017.02-035/stage2/smb/x86_64-linux/usr/bin/mipsel-buildroot-linux-gnu-g++ - works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Project name = vtss_api
--   Type        = Release
--   cxx_flags   = -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -Os -std=c++11
--   c_flags     = -D_LARGEFILE_SOURCE -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -Os -Wall -Wno-array-bounds -fasynchronous-unwind-tables -std=c99 -D_POSIX_C_SOURCE=200809L -D_BSD_SOURCE -D_DEFAULT_SOURCE
-- Looking for include file endian.h
-- Looking for include file endian.h - found
-- Looking for include file asm/byteorder.h
-- Looking for include file asm/byteorder.h - found
-- MESA layer: ON
-- Configuring done
-- Generating done
-- Build files have been written to: /home/anielsen/API_X.Y/mesa/build
$ make -j 10              # Build the sources
...
$ ls lib*.so mesa/demo/*.mfi
libvsc7514_aqr.so mesa/demo/mesa_demo_ocelot_vsc7514.mfi
```

The build is producing a MESA library matching the vsc7514 target, and a demo images which can be used on the Ocelot reference boards.

4.2.1. Auto generated content

During the build-process, most of the MESA source code is auto-generated. The auto generator is parsing the header files (both Unified-API and MESA), and check if it can implement the body of the MESA functions automatic. If that is possible (because the differences follows simple rules), then it will do so. If not then it will skip implementing that function. Functions that can not be auto generated, needs to be generated manually.

The linker will ensure that the union of the manual implemented and the auto generated functions covers all functions in the MESA library.

The auto generated is found here (inside the build folder):

```
$ ls mesa-ag/
include   log.txt   mesa   mesa.cxx   mesa.hxx   mesa_pp_rename.h   rename
```

The `mesa.cxx` and `mesa.hxx` is the auto generated part of the mesa implementation, it is just compiled into the MESA library as everything else. The `rename` and `mesa_pp_rename.h` is a complete list of symbols renamed in the MESA library, they are intended to be used for migrating applications - this is covered in the next section.

5. Application migration

Existing application written for the Unified API can migrated to using the MESA library. To ease that a list of symbol renames is provided by the MESA building system. MSCC has done the migration of the WebStaX product family, by using the same rename script in combination with the CapArrays. Depending on the size of the application, such a migration can be a considerable effort.

> Customers are not forced to migrate to MESA, if they do not see any benefits in doing so, they are encourage to avoid it. MSCC has converted the WebStaX family to use the MESA layer, because it allows significant build time performance when building for many targets, and because it provides better separation of the application and API. The existing Unified API is still supported, and offers the same feature set and is supported in the same way.

The auto generator (`ag.rb`) is parsing the header files of the MESA library and the Unified API. This means that the `ag.rb` tool has a complete overview of all the symbols defined in the two header files. Having that overview, makes it easy to create a list of all the renames needed to migrate an application from using the Unified API to the MESA library.

The auto generator is creating a list, in two formats. One that can be used by `sed` to perform the renames in place (called `rename`), and another which can be included in a compilation unit (but in the top of a file) to set the pre-processor to do the rename (called `mesa_pp_rename.h`).

The following command can be used to do the rename in place, for a large number of files. (Remember to have a backup!):

```
$ cd /root/of/repository/to/perform/rename/on
$ find . \( -name "*.h"   -or -name "*.c" -or  \
            -name "*.cxx" -or -name "*.cxx" \) \
            -exec sed -i /path/to/auto/generated/rename {} \;
```

6. Reference boards and demo system

The MESA project includes a simple demo application which is using the MESA and MEBA libraries. The application includes a number of modules, each implemented in a single C file. These modules include the following functionality:

- Main control and initialization of MEBA/MESA
- Port and PHY control
- MAC Address table
- VLAN control
- IP management via selected switch port
- Code trace
- Test commands
- Debug functions
- Command Line Interface (CLI)

The application binary (`mesa_demo`) supports a number of startup options, for example trace level control. It runs as a daemon, allowing the user to execute shell commands.

6.1. CLI Application

The CLI Application is simple client program, which communicates with the demo application using a socket interface. The CLI binary `cli` does the following:

- Reads an input command
- Sends the command to the demo application
- Writes the response to the terminal
- Exits

6.2. Building the demo application

The section called 'Building the sources' describes how to build the MESA library. This build job will also build the demo application, and organize it all into a `mfi` file that can be loaded on the corresponding reference board, if that has been enabled.

To expose what reference systems is supported, use the `ccmake .` command, and look for the `mesa_demo_*` settings. In this case `mesa_demo_ocelot_vsc7514` will be `ON`. Other components required by this demo application will also be `ON`.

The following sections assume that the `mfi` file is already build (in section 'Building the sources').

6.3. Download

The MFI file can now be loaded into RAM on a reference board. This requires the following:

- The reference board must have a Flash image, which can load the MFI file.

- The reference board must be setup with IP access to a TFTP server.
- The MFI file must be copied to the TFTP server.

The example below assumes that the MFI file has been copied to `new.img` in the TFTP home directory on a TFTP server with IP address 1.1.1.2. The switch IP address is set to 1.1.1.1/24 in the default VLAN. The configuration is stored as startup configuration, so that the system is ready to load the MFI file after reboot.

Note that the system is stopped in the bootloader (using `^C`) before the `ramload` command is given.

```
Username: admin
Password:
# conf term
config # int vlan 1
config-if-vlan # ip addr 1.1.1.1 255.255.255.0
config-if-vlan # end
# copy running-config startup-config
# platform debug allow
< Output from command not shown >
# debug firmware ramload tftp://1.1.1.2/new.img
< Output from load and boot process not shown >
^C
ramload
< Output from boot process and Linux startup not shown >
login: root
```

6.4. Running

After logging in as `root`, the demo application can be started and controlled using the CLI Application.

```
# mesa_demo -h
mesa_demo options:

-h                          : Show this help text
-p <port>                   : Enable IP management port
-t <module>:<group>:<level> : Set trace level for <module> and <group>, use '*' for
wildcard
# mesa_demo
# cli
Available Commands:

Help
Exit
IP Status
MAC Add <mac_addr> <port_list> [<vid>]
MAC Agetime [<age_time>]
MAC Delete <mac_addr> [<vid>]
MAC Dump
MAC Flush
MAC Lookup <mac_addr> [<vid>]
Port Flow Control [<port_list>] [enable|disable]
Port MaxFrame [<port_list>] [<max_frame>]
Port Mode [<port_list>] [10hdx|10fdx|100hdx|100fdx|1000fdx|auto]
Port State [<port_list>] [enable|disable]
Port Statistics [<port_list>] [clear|packets|bytes|errors|discards]
Test [<test_no>]
VLAN Add <vid> <port_list>
VLAN Delete <vid>
VLAN Filter [<port_list>] [enable|disable]
VLAN Frame [<port_list>] [all|tagged|untagged]
VLAN PVID [<port_list>] [<vid>]
VLAN Type [<port_list>] [unaware|c-port|s-port]
VLAN UVID [<port_list>] [all|none|pvid]
Debug API [<layer>] [<group>] [<port_list>] [full] [clear]
Debug Chip ID
Debug I2C Read <port_list> <i2c_addr> <addr> [<count>]
Debug I2C Write <port_list> <i2c_addr> <addr> <value>
Debug MMD Read <port_list> <mmd_list> <mmd_addr>
Debug MMD Write <port_list> <mmd_list> <mmd_addr> <value>
Debug PHY Read <port_list> <addr_list> [<page>]
Debug PHY Write <port_list> <addr_list> <value> [<page>]
Debug Port Polling [enable|disable]
Debug Sym Query <word128>
Debug Sym Read <word128>
Debug Sym Write <word128> <value32>
Debug Trace [<module>] [<group>] [off|error|info|debug|noise]
# cli port mode
Port  State     Mode    Flow Control  Rx Pause  Tx Pause  MaxFrame  Link
----  --------  ------  ------------  --------  --------  --------  --------
1     Enabled   Auto    Enabled       Disabled  Disabled  1518      1Gfdx
2     Enabled   Auto    Enabled       Disabled  Disabled  1518      1Gfdx
3     Enabled   1Gfdx   Disabled      Disabled  Disabled  1518      Down
4     Enabled   1Gfdx   Disabled      Disabled  Disabled  1518      1Gfdx
5     Enabled   1Gfdx   Disabled      Disabled  Disabled  1518      Down
6     Enabled   1Gfdx   Disabled      Disabled  Disabled  1518      Down
```

```
7     Enabled   10Gfdx  Disabled      Disabled  Disabled  1518      Down
8     Enabled   10Gfdx  Disabled      Disabled  Disabled  1518      Down
9     Enabled   Auto    Enabled       Disabled  Disabled  1518      1Gfdx
```

6.5. Adding a Test

The demo application includes a test module, which registers a CLI command `cli test` into the system. The command takes an integer argument identifying the test case to be executed. Additional test cases can be added in the `vtss_api/mesa/demo/test.c` file. The code below shows how the function `my_test` is added to the `test_table` in addition to existing ACL and EVC test cases. The test simply prints the number of packets received on the first switch port.

```
static mesa_rc my_test(void)
{
    mesa_port_counters_t counters;
    MESA_RC(mesa_port_counters_get(NULL, 0, &counters));
    cli_printf("Rx Packets: %llu\n", counters.rmon.rx_etherStatsPkts);
    return MESA_RC_OK;
}

static test_entry_t test_table[] = {
    {
        "ACL test",
        test_acl
    },
    {
        "EVC test",
        test_evc
    },
    {
        "My test",
        my_test
    },
};
```

After building, downloading and starting the demo application, the new command can be executed as shown below.

```
# cli test
Number  Description
------  -----------
0       ACL test
1       EVC test
2       My test
# cli test 2
Rx Packets: 10
```

Appendix A: C++ CapArray

The CapArray is included as an example to the user (application developer), to show way of integrating the capabilities in the application. It is not a part of the MESA-API, and therefore not part of the UnifiedAPI/MESA support agreement.

The WebStaX family is using the CapArray heavily, and it is an official part of the WebStaX product, which is also supporting issues that may be caused by CapArray.

> This is a C++ library, it depends on RAII, which means that it will not work (give wrong result, leak or crash) if combined with `memset`, `memcpy`, `memcmp` and other traditional `C` constructions. Do not use it, if you do not understand it!

The purpose of the CapArray, is mimic traditional `C` arrays as much as possible, but allow the array dimensions to be runtime determinate. It is not an alternative to `std::vector` as the size of the array is fixed (in oppesite to `std::vector` which is providing dynamic size).

The Current example supports up to 4 dimensional arrays, but can easily be extended further. Here is a small example using it:

vtss_appl/main/caparray.hxx

```
#include "caparray.hxx"
#include "mscc/ethernet/switch/api/capability.h"

struct Foo {
    int a, b;
};

int main() {
    CapArray<Foo, MESA_CAP_PORT_CNT> foo_array;

    // Make sure that the API is initialized correctly before using.

    for (size_t i = 0; i < foo_array.size(); i++) {
        foo_array[i].a = i;
        foo_array[i].b = 1;
    }

    CapArray<Foo, MESA_CAP_PORT_CNT, MESA_CAP_L3_RLEG_CNT> foo_array_array;
    for (size_t i = 0; i < foo_array.size(); i++) {
        for (size_t j = 0; j < foo_array[i].size(); j++) {
            foo_array[i][j].a = i;
            foo_array[i][j].b = j;
        }
    }

}
```

Following is the CapArray sources.

vtss_appl/main/caparray.hxx

```
/*
Copyright (c) 2004-2017 Microsemi Corporation "Microsemi".

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.

*/

template <typename T>
class A1 {
  public:
    A1(const A1 &) = delete;
    A1 &operator=(const A1 &rhs) = delete;
    A1(T *d, size_t _1) : data_(d), d1(_1) {}
    A1(A1 &&rhs) : data_(rhs.data_), d1(rhs.d1) {}

    // Needed to make it non-POD, for static analysis to catch ARRSZ problem
    ~A1(){}

    T &operator[](size_t idx) {
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return data_[idx];
    }
    const T &operator[](size_t idx) const {
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return data_[idx];
    }
    size_t size() const { return d1; }
    size_t mem_size() const { return d1 * sizeof(T); }
    const T *data() const { return data_; }
    T *data() { return data_; }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, void>::type clear() {
        memset(data_, 0, mem_size());
    }

  protected:
    T *const data_;
    const size_t d1;
};

template <typename T>
class A2 {
  public:
    A2(const A2 &) = delete;
    A2 &operator=(const A2 &rhs) = delete;
    A2(A2 &&rhs) : data_(rhs.data_), d1(rhs.d1), d2(rhs.d2) {}
    A2(T *d, size_t _1, size_t _2) : data_(d), d1(_1), d2(_2) {}

    // Needed to make it non-POD, for static analysis to catch ARRSZ problem
    ~A2(){}

    A1<T> operator[](size_t idx) {
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return A1<T>(data_ + (idx * d2), d2);
    }
    const A1<T> operator[](size_t idx) const {
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return A1<T>(data_ + (idx * d2), d2);
    }
    size_t size() const { return d1; }
    size_t mem_size() const { return d1 * d2 * sizeof(T); }
    const T *data() const { return data_; }
    T *data() { return data_; }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, void>::type clear() {
        memset(data_, 0, mem_size());
    }


  protected:
    T *data_ = nullptr;
    size_t d1 = 0, d2 = 0;
};

template <typename T>
class A3 {
  public:
    A3(const A3 &) = delete;
    A3 &operator=(const A3 &rhs) = delete;
    A3(A3 &&rhs) : data_(rhs.data_), d1(rhs.d1), d2(rhs.d2), d3(rhs.d3) {}
    A3(T *d, size_t x, size_t y, size_t z) : data_(d), d1(x), d2(y), d3(z) {}

    // Needed to make it non-POD, for static analysis to catch ARRSZ problem
    ~A3(){}

    A2<T> operator[](size_t idx) {
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return A2<T>(data_ + (idx * d2 * d3), d2, d3);
    }
    const A2<T> operator[](size_t idx) const {
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return A2<T>(data_ + (idx * d2 * d3), d2, d3);
    }
    size_t size() const { return d1; }
    size_t mem_size() const { return d1 * d2 * d3 * sizeof(T); }
    const T *data() const { return data_; }
    T *data() { return data_; }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, void>::type clear() {
        memset(data_, 0, mem_size());
    }

  protected:
    T *data_ = nullptr;
    size_t d1 = 0, d2 = 0, d3 = 0;
};

template <typename T, int... CAPS>
class CapArray;

template <typename T, int C1>
class CapArray<T, C1> {
  public:
    CapArray() {}
    CapArray &operator=(const CapArray &rhs) {
        init();
        for (size_t i = 0; i < d1; ++i) data_[i] = rhs.data()[i];
        return *this;
    }

    CapArray(const CapArray &rhs) {
        init();
        for (size_t i = 0; i < d1; ++i) data_[i] = rhs.data()[i];
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, bool>::type operator==(
            const CapArray &rhs) const {
        init();
        return memcmp(data(), rhs.data(), mem_size()) == 0;
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, bool>::type operator!=(
            const CapArray &rhs) const {
        return !this->operator==(rhs);
    }

    void init() const {
        if (data_) return;
        d1 = mesa_capability(nullptr, C1);
        data_ = (T *)VTSS_CALLOC_MODID(VTSS_MODULE_ID, d1, sizeof(T), __FILE__,
                                       __LINE__);
        if (!data_) return;
        for (size_t i = 0; i < d1; ++i) new (&(data_[i])) T();
    }

    ~CapArray() {
        if (!data_) return;
        for (size_t i = 0; i < d1; ++i) data_[i].~T();
        VTSS_FREE(data_);
    }

    T &operator[](size_t idx) {
        init();
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return data_[idx];
    }

    const T &operator[](size_t idx) const {
        init();
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return data_[idx];
    }

    size_t size() const {
        init();
        return d1;
    }

    size_t mem_size() const {
        init();
        return d1 * sizeof(T);
    }

    const T *data() const {
        init();
        return data_;
    }

    T *data() {
        init();
        return data_;
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, void>::type clear() {
        init();
        memset(data_, 0, mem_size());
    }

  private:
    mutable size_t d1;
    mutable T *data_ = nullptr;
};

template <typename T, int C1, int C2>
class CapArray<T, C1, C2> {
  public:
    CapArray() {}
    CapArray &operator=(const CapArray &rhs) {
        init();
        for (size_t i = 0; i < d1 * d2; ++i) data_[i] = rhs.data()[i];
        return *this;
    }

    CapArray(const CapArray &rhs) {
        init();
        for (size_t i = 0; i < d1 * d2; ++i) data_[i] = rhs.data()[i];
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, bool>::type operator==(
            const CapArray &rhs) const {
        init();
        return memcmp(data(), rhs.data(), mem_size()) == 0;
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, bool>::type operator!=(
            const CapArray &rhs) const {
        return !this->operator==(rhs);
    }

    void init() const {
        if (data_) return;
        d1 = mesa_capability(nullptr, C1);
        d2 = mesa_capability(nullptr, C2);
        data_ = (T *)VTSS_CALLOC_MODID(VTSS_MODULE_ID, d1 * d2, sizeof(T),
                                       __FILE__, __LINE__);
        if (!data_) return;
        for (size_t i = 0; i < d1 * d2; ++i) new (&(data_[i])) T();
    }

    ~CapArray() {
        if (!data_) return;
        for (size_t i = 0; i < d1 * d2; ++i) data_[i].~T();
        VTSS_FREE(data_);
    }

    A1<T> operator[](size_t idx) {
        init();
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return A1<T>(data_ + (idx * d2), d2);
    }

    const A1<const T> operator[](size_t idx) const {
        init();
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return A1<const T>(data_ + (idx * d2), d2);
    }

    size_t mem_size() const {
        init();
        return d1 * d2 * sizeof(T);
    }

    size_t size() const {
        init();
        return d1;
    }

    const T *data() const {
        init();
        return data_;
    }

    T *data() {
        init();
        return data_;
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, void>::type clear() {
        init();
        memset(data_, 0, mem_size());
    }

  private:
    mutable size_t d1, d2;
    mutable T *data_ = nullptr;
};

template <typename T, int C1, int C2, int C3>
class CapArray<T, C1, C2, C3> {
  public:
    CapArray() {}
    CapArray &operator=(const CapArray &rhs) {
        init();
        size_t n = d1 * d2 * d3;
        for (size_t i = 0; i < n; ++i) data_[i] = rhs.data()[i];
        return *this;
    }

    CapArray(const CapArray &rhs) {
        init();
        size_t n = d1 * d2 * d3;
        for (size_t i = 0; i < n; ++i) data_[i] = rhs.data()[i];
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, bool>::type operator==(
            const CapArray &rhs) const {
        init();
        return memcmp(data(), rhs.data(), mem_size()) == 0;
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, bool>::type operator!=(
            const CapArray &rhs) const {
```

```
        return !this->operator==(rhs);
    }

    void init() {
        if (data_) return;
        d1 = mesa_capability(nullptr, C1);
        d2 = mesa_capability(nullptr, C2);
        d3 = mesa_capability(nullptr, C3);
        size_t n = d1 * d2 * d3;

        data_ = (T *)VTSS_CALLOC_MODID(VTSS_MODULE_ID, n, sizeof(T), __FILE__,
                                       __LINE__);
        if (!data_) return;
        for (size_t i = 0; i < n; ++i) new (&(data_[i])) T();
    }

    ~CapArray() {
        if (!data_) return;
        size_t n = d1 * d2 * d3;
        for (size_t i = 0; i < n; ++i) data_[i].~T();
        VTSS_FREE(data_);
    }

    A2<T> operator[](size_t idx) {
        init();
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return A2<T>(data_ + (idx * d2 * d3), d2, d3);
    }

    const A2<T> operator[](size_t idx) const {
        init();
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return A2<const T>(data_ + (idx * d2 * d3), d2, d3);
    }

    size_t mem_size() const {
        init();
        return d1 * d2 * d3 * sizeof(T);
    }

    size_t size() const {
        init();
        return d1;
    }

    const T *data() const {
        init();
        return data_;
    }

    T *data() {
        init();
        return data_;
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, void>::type clear() {
        init();
        memset(data_, 0, mem_size());
    }


  private:
    mutable size_t d1, d2, d3;
    mutable T *data_ = nullptr;
};

template <typename T, int C1, int C2, int C3, int C4>
class CapArray<T, C1, C2, C3, C4> {
  public:
    CapArray() {}
    CapArray &operator=(const CapArray &rhs) {
        init();
        size_t n = d1 * d2 * d3 * d4;
        for (size_t i = 0; i < n; ++i) data_[i] = rhs.data()[i];
        return *this;
    }

    CapArray(const CapArray &rhs) {
        init();
        size_t n = d1 * d2 * d3 * d4;
        for (size_t i = 0; i < n; ++i) data_[i] = rhs.data()[i];
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, bool>::type operator==(
            const CapArray &rhs) const {
        init();
        return memcmp(data(), rhs.data(), mem_size()) == 0;
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, bool>::type operator!=(
            const CapArray &rhs) const {
        return !this->operator==(rhs);
    }

    void init() {
        if (data_) return;
        d1 = mesa_capability(nullptr, C1);
        d2 = mesa_capability(nullptr, C2);
        d3 = mesa_capability(nullptr, C3);
        d4 = mesa_capability(nullptr, C4);
        size_t n = d1 * d2 * d3 * d4;

        data_ = (T *)VTSS_CALLOC_MODID(VTSS_MODULE_ID, n, sizeof(T), __FILE__,
                                       __LINE__);
        if (!data_) return;
        for (size_t i = 0; i < n; ++i) new (&(data_[i])) T();
    }

    ~CapArray() {
        if (!data_) return;
        size_t n = d1 * d2 * d3 * d4;
        for (size_t i = 0; i < n; ++i) data_[i].~T();
        VTSS_FREE(data_);
    }

    A3<T> operator[](size_t idx) {
        init();
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return A3<T>(data_ + (idx * d2 * d3 * d4), d2, d3, d4);
    }

    const A3<const T> operator[](size_t idx) const {
        init();
        CAP_ARRAY_CHECK_DIM(idx, d1);
        return A3<const T>(data_ + (idx * d2 * d3 * d4), d2, d3, d4);
    }

    size_t mem_size() const {
        init();
        return d1 * d2 * d3 * d4 * sizeof(T);
    }

    size_t size() const {
        init();
        return d1;
    }

    const T *data() const {
        init();
        return data_;
    }

    T *data() {
        init();
        return data_;
    }

    template <class Q = T>
    typename std::enable_if<std::is_pod<Q>::value, void>::type clear() {
        init();
        memset(data_, 0, mem_size());
    }


  private:
    mutable size_t d1, d2, d3, d4;
    mutable T *data_ = nullptr;
};
```

References

- [AN1007] MSCC Ethernet API Software
- [ug1068] SW Introduction to WebStaX under Linux
- [ug1069] MEBA Programmers Guide