OpenMuL Controller – Mid-Level API Description

open MUL

Table of Contents

I. Introduction

This document aims to provide adequate information for Developers to quickly start developing in MuL Controller Code base.

All salient features or examples in this document references MuL Controller unless specified otherwise.

The organization of rest of the text is as follows:

MuL Code Base – This chapter outlines the file system structure of MuL Controller Code Base and ML API App directory.

MuL Core API – This chapter outlines functionalities provided by MuL Controller Core. More emphasis will be put on list of the functions that is available to ML API App Developers, and various conventions that is implicitly observed by currently available MuL ML API Apps.

MuL Service API – This chapter outlines Service API exposed by currently available MuL ML API Apps.

3rd Party Libraries – This chapter outlines commonly implemented features on ML API Apps that depends on 3rd party libraries. Emphasis is again on list of the functions and the conventions implicitly observed by currently available ML API Apps.

Adding ML API App to Code Base – This chapter outlines standard procedure to add new ML API App into MuL Codebase and Autoconf Build System.

II. MuL Code Base

Below table is the list of important directories inside MuL Controller Code Base

Location	Description
/application	ML API Apps
/common	Common headers and source files that are essential for ML API Apps
/common-lib/	Other libraries used by MuL
/mul	Core Controller Code
/services	Service Provider Apps that will be used by multiple ML API Apps

III. MuL Core API

Core API encompasses several features that are absolutely essential for any ML API apps to function properly.

Developers can import “mul_app_interface.h” to gain access to functions listed below.

1. Event Handler API

One of important responsibility of Controller Core is to provide mechanisms for Apps to register themselves to Data Plane Events. Alongside DP Events, Event Handler API allows App to register and listen to controller specific events as well.

1.1 Register/Unregister

1.1.1 mul_register_app_cb

Arguments:

Type	Name	Description
void *	app_arg	Opaque app handle (provided as an argument to module_init function or event callback function)
char *	app_name	Human readable ML-API Name (Limit: 64 characters)
uint32_t	app_flags	Bit Field combination of { C_APP_ALL_SW, C_APP_REMOTE, C_APP_AUX_REMOTE}
uint32_t	ev_mask	Bit Field of events that registering app wishes to listen to. Applicable values are: (1 << { any of enum c_app_event constats }) or C_APP_ALL_EVENTS (All values are self-explanatory)
uint32_t	n_dpid	Number of switches in the dpid_list in case C_APP_ALL_SW has not been specified.
uint64_t *	dpid_list	If C_APP_ALL_SW is not on then it means the app wishes to subscribe to events of only some of the switches controlled by the controller. The app can identify switches it is interested in.
struct mul_app_client_cb *	app_cbs	Set of Callback functions to be called when an event has occurred.

Returns: 0

This function creates App Instance and registers the app to the Controller Core.

The ML API App Developer usually calls this function twice in there code. Once at the end of module_init function (specified by module_init macro) and when C_OFPT_RECONN_APP event has been notified (app was disconnected to controller, but now connection is back online) through registered event callback function.

1.1.2 mul_unregister_app

Arguments:

Type	Name	Description
char *	app_name	Human readable ML-API Name (Limit: 64 characters)

Returns: 0

This function unregisters specified App Instance from the Controller Core.

1.2 Event callback functions

1.2.1 switch_priv_alloc

This function is called when switch context is created for registered APP to store private data.

Arguments:

Type	Name	Description
void **	switch_ptr	Pointer to private data to be stored at the time of Switch Context creation.

1.2.2 switch_priv_free

Arguments:

Type	Name	Description
void **	switch_ptr	Pointer to private data to be stored at the time of Switch Context creation.

1.2.3 switch_add_cb

This function is called when a switch is connected to the controller core.

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.

1.2.4 switch_del_cb

This function is called when a switch is disconnected the controller core.

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.

1.2.5 switch_priv_port_alloc

This function is called when a port context is created.

Arguments:

Type	Name	Description
void **	port_ptr	Pointer to private data pointer in port context.

1.2.6 switch_priv_port_free

This function is called when a port context is created.

Arguments:

Type	Name	Description
void **	port_ptr	Pointer to private data pointer in port context.

1.2.7 switch_port_add_cb

This function is called when a port is added to a particular switch.

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.
Mul_port_t *	port_ptr	Pointer to port context.

1.2.8 switch_port_del_cb

This function is called when a port is removed from a particular switch.

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.
Mul_port_t *	port_ptr	Pointer to port context.

1.2.9 switch_port_chg

This function is called when a port's state is modified.

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.
mul_port_t *	port_ptr	Pointer to port context.
bool	adm	This flag variable tells the administrative state of a port.
bool	link	This flag variable tells the physical state of a port.

1.2.10 switch_packet_in

This function is called when a packet is received from a switch.

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.
Struct flow *	Flow_ptr	Pointer to matched flow context.
uint32_t	in_port	This variable holds the port number from which the switch has received this packet.
uint32_t	buffer_id	This variable contains the index value for packet buffer. Valid if switch does not send complete packet.
uint8_t*	pkt	Pointer to packet data.
size_t	pkt_len	This variable contains the packet length.

1.2.11 switch_error

This function is called when an error is returned from a switch.

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.
uint16_t	type	Variable contains type of error.
uint16_t	code	Variable contains code for error type.
uint8_t*	pkt	Pointer to packet data.
size_t	pkt_len	This variable contains the packet length.

1.2.12 switch_fl_mod_err

This function is called when an error is returned from a switch after sending flow add /del (flow_mod).

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.
uint16_t	type	Variable contains type of error.
uint16_t	code	Variable contains code for error type.
c_ofp_flow_mod_t *	flow_mod	Pointer to flow mod information sent to switch.

1.2.13 switch_group_mod_err

This function is called when an error is returned from a switch after sending group add /del (flow_mod).

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.
uint16_t	type	Variable contains type of error.
uint16_t	code	Variable contains code for error type.
c_ofp_group_mod_t *	group_mod	Pointer to group mod information sent to switch.

1.2.14 switch_meter_mod_err

This function is called when an error is returned from a switch after sending meter add /del (flow_mod).

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.
uint16_t	type	Variable contains type of error.
uint16_t	code	Variable contains code for error type.
c_ofp_meter_mod_t *	meter_mod	Pointer to meter mod information sent to switch.

1.2.15 core_conn_closed

This function is called when connection is closed between Core and Application.

Arguments: N.A.

1.2.16 core_conn_reconn

This function is called when connection is re-established between Core and Application.

Arguments: N.A.

1.2.17 app_ha_state

This function is called there is a change in HA state for controller core.

Arguments:

Type	Name	Description
uint32_t	sys_id	Variable contains value for system ID returned by switch.
uint32_t	ha_state	Variable contains new HA state.

1.2.18 process_vendor_msg_cb

This function is called vendor specific message is received from a switch.

Arguments:

Type	Name	Description
mul_switch_t *	switch_ptr	Pointer to switch context.
uint8_t*	msg	Pointer to message data.
size_t	msg_len	This variable contains the message length.

1.2.19 topo_route_status_cb

This function is called topology is converged.

Arguments:

Type	Name	Description
uint64_t	status	This variable contains the status code.

1.3 Examples

Below is a simple example application that uses MuL Core API

```
/*
 * hello.c : sample application to demonstrate
 * mul controller usage
 */
#include "mul_common.h"
#define APP_NAME "hello"
struct mul_app_client_cb prism_app_cbs = {
.switch_priv_alloc = NULL,
.switch_priv_free = NULL,
.switch_add_cb =  switch_add,
.switch_del_cb =  switch_del,
.switch_priv_port_alloc = NULL,
.switch_priv_port_free = NULL,
.switch_port_add_cb = NULL,
.switch_port_del_cb = NULL,
.switch_port_chg = NULL,
.switch_packet_in = NULL,
.switch_error = NULL,
.core_conn_closed = NULL,
.core_conn_reconn = NULL,
}
/*
 * SWITCH_ADD Callback|
 */
static void switch_add(uint64_t dpid)
{
     c_log_debug("switch 0x%016llx connected", dpid);
}
/*
 * SWITCH_DELETE Callback
 */
static void switch_delete(uint64_t dpid)
{
     c_log_debug("switch 0x%016llx disconnected", dpid);
}

void hello_app_init(void *ctx)
{
     mul_register_app_cb(NULL, APP_NAME,
```

```
                        C_APP_ALL_SW, C_APP_ALL_EVENTS,
                        0, NULL, hello_app_cbs);
}
module_init(hello_app_init);
```

2. Servlet Handler API

Sometimes it is essential for an App to communicate with other modules or App via Service API for better modularity and to promote code reuse. Servlet Handler API refers to set of functions provided by MuL Controller Core to facilitate App-to-App communication.

2.1 List of service lifecycle functions

2.1.1 mul_app_create_service

Arguments:

Type	Name	Description
char *	name	Human readable name of new service
func ptr	service_handler	Callback function to be called when servlet request has arrived

Returns: Handle to Service Instance Object

Service Provider Apps, which wish to expose RPC-like Service API to other ML API Apps can use this function to register itself as a handler for a service called *"name"*.

The callback function will be called whenever service request arrives.

2.1.2 service_handler callback function

Arguments:

Type	Name	Description
void *	service	Handle to Service Instance Object
struct cbuf *msg	msg	Body of Service Request

When Service Request has arrived, App receives actual service request message as 'msg' argument.

Although callback function will be given struct cbuf * as a parameter, it is customary to format the content according to struct c_ofp_auxapp_cmd.

Thus, one can and should always write:

```
struct c_ofp_auxapp_cmd *cofp_aac = (void *)(b->data);
```

and inspect cofp_aac->cmd_code to determine specific command that Service Consumer wishes to execute.

2.1.3 mul_app_get_service

Arguments:

Type	Name	Description
char *	name	Human readable name of new service

Returns: Handle to Service Instance Object

Service Consumer Apps, which wish to use Service API exposed by other Apps can use this function to get a handle to a service called “name”.

However, unlike mul_app_create_service, by calling this function alone will not dynamically generate new services. Please refer to next section (Implementation Issue) for more details.

2.1.4 mul_app_get_service_notify

Arguments:

Type	Name	Description
char *	name	Human readable name of new service
func ptr	conn_update	Callback function to be called when there is change in connectivity status (i.e. service down or up)
Bool	Retry_conn	If *true*, mul_app_get_service_notify will repeatedly attempt to connect to this service. Otherwise, no retry will be performed.

Returns: Handle to Service Instance Object

This function achieves essentially same objective as mul_app_get_service. The difference is the availability of conn_update and retry_conn argument.

2.1.5 conn_update call back function

Arguments:

Type	Name	Description
void *	service	Handle to Service Instance Object
unsigned char	conn_event	MUL_SERVICE_UP, MUL_SERVICE_DOWN

Callback function to be called when there has been a change in connectivity status.

2.1.6 mul_app_destroy_service

Arguments:

Type	Name	Description
void *	Service	Handle to Service Instance Object of terminating service

Regardless whether caller is a Service Provider or Service Consumer, this will close all connections and free all resources relevant to this service.

2.1.7 List of functions for handler implementation

This section of the text outlines two functions that plays key role in actual information exchange between Service Consumer and Provider.

Key ideas:

All Service API interactions are Service Consumer (Client App) initiated exchanges.

No Service Provider (App/Core) should send any message using c_service_send unless there has been request from consumer

If there has been a request, there must be a response (response to service request or error message)

Current design and conventions adopted by stub code will block Service Consumer execution unless it receives a response from the Service Provider.

2.1.8 c_service_send

Arguments:

Type	Name	Description
mul_service_t *	service	Handle to RPC-like Service Instance Object
struct cbuf *	b	Buffer containing request message

Both Service Provider and Consumer may send message to each other using this function. Consumer will use this function to request some service to Provider (although this will be most likely done by stub code inside a _servlet.c/h files), and Provider may send a reply to a request using this function.

Currently, only one service request at a time should be generated.

The format of message will be *struct c_ofp_auxapp_cmd* both ways.

2.1.9 c_service_wait_response

Arguments:

Type	Name	Description
mul_service_t *	service	Handle to RPC-like Service Instance Object

Return: struct cbuf containing the reply message.

Consumer (or the stub code) should call this function to wait for Producer to reply the last service API request. This function will block until a response has been received.

The return message will be formatted according to struct c_ofp_auxapp_cmd.

2.2 Implementation

Unlike Core API, implementation of ML API App which consumes/provides service requires additional effort beyond mere usage of functions outlined in previous section. In this section, we go over all necessary steps to complete such implementation.

2.2.1 Modification of mul_app_main.c

Although this document focuses on RPC-like Service API, there are actually two breeds of so-called "Service API" in MuL Controller Application Framework. The other type is implemented using shared-memory constructs (shm_open/mmap etc) that is frequently used in general inter-process communication.

Because of this peculiarity, a table with information on type of service must be maintained. Currently, this is not dynamic and this information is hard-coded within a common source file ("mul_app_main.c") which must be linked with any ML API App that is built as separate process.

This is an excerpt of code in "mul_app_main.c"

```
struct c_app_service {
   char service_name[MAX_SERV_NAME_LEN];
   uint16_t  port;
   void * (*service_priv_init)(void);
}c_app_service_tbl[] = {
    { MUL_TR_SERVICE_NAME, MUL_TR_SERVICE_PORT, NULL },
    { MUL_ROUTE_SERVICE_NAME, 0, mul_route_service_get },
    { MUL_CORE_SERVICE_NAME, C_AUX_APP_PORT, NULL },
    { MUL_FAB_CLI_SERVICE_NAME, MUL_FAB_CLI_PORT, NULL }
};
```

When new service is developed, c_app_service_tbl must be updated with a new entry for new service.

In case of RPC-like Service Provider, the entry should look like

```
    { NEW_SERVICE_HUMAN_READABLE_NAME, NEW_SERVICE_LISTENING_PORT, NULL }
```

2.2.2 _servlet.h and _servlet.c

While it is not mandatory, it is customary for Service Producer App developers to provide stub code (in the RPC sense) in *appname*_servlet.h (to be imported by consumers) and *appname*_servlet.c (actual implementation) to facilitate the use of Service API.

Here is an example stub function (provided in mul_servlet.c)

```
/**
 * mul_get_switch_detail -
 *
 * Get detail switch info connected to mul
 */
struct cbuf *
mul_get_switch_detail(void *service, uint64_t dpid)
{
    struct cbuf *b;
    struct c_ofp_auxapp_cmd *cofp_auc;
    struct c_ofp_req_dpid_attr *cofp_rda;
    struct ofp_header *h;

    if (!service) return NULL;

    b = of_prep_msg(sizeof(*cofp_auc) + sizeof(*cofp_rda),
                    C_OFPT_AUX_CMD, 0);

    cofp_auc = (void *)(b->data);
    cofp_auc->cmd_code = htonl(C_AUX_CMD_MUL_GET_SWITCH_DETAIL);
    cofp_rda = (void *)(cofp_auc->data);
    cofp_rda->datapath_id = htonll(dpid);

    c_service_send(service, b);
    b = c_service_wait_response(service);
    if (b) {
        h = (void *)(b->data);
        if (h->type != C_OFPT_SWITCH_ADD ||
            ntohs(h->length) < sizeof(struct ofp_switch_features)) {
            c_log_err("%s: Failed", FN);
            free_cbuf(b);
            return NULL;
        }
    }

    return b;
}
```

As evident in the code, majority of the stub code should be used to prepare the request packet (struct c_ofp_auxapp_cmd), encapsulate arguments if necessary, and sending and waiting for response.

2.3 Example

In this section, we provide simple implementations of Service Provider and Consumer

2.3.1 Service Provider

```
/*
 * service_provider.c : sample application to demonstrate
 * Service API usage
 */
#include "mul_common.h"
#define APP_NAME "mul_service_provider"
#define SERVICE_NAME APP_NAME

#define C_AUX_CMD_SVC_1 1

/**
 * example_service_error -
 *
 * Sends error message to service requester in case of error
 */
static void
example_service_error(void *service, struct cbuf *b,
                      uint16_t type, uint16_t code)
{
    struct cbuf       *new_b;
    c_ofp_error_msg_t *cofp_em;
    void              *data;
    size_t            data_len;

    data_len = b->len > C_OFP_MAX_ERR_LEN?
                   C_OFP_MAX_ERR_LEN : b->len;

    new_b = of_prep_msg(sizeof(*cofp_em) + data_len, C_OFPT_ERR_MSG, 0);

    cofp_em = (void *)(new_b->data);
    cofp_em->type = htons(type);
    cofp_em->code = htonl(code);

    data = (void *)(cofp_em + 1);
    memcpy(data, b->data, data_len);

    c_service_send(service, new_b);
}
/**
 * example_cmd_1_handler -
 *
 * Sends error message to service requester in case of error
 */
static void
example_cmd_1_handler(void *service)
{
    struct c_ofp_auxapp_cmd *cofp_aac;
    struct cbuf *b;
    char *buf;
    uint64_t dpid = 0;
```

```
    b = of_prep_msg(sizeof(*cofp_aac) + 12, C_OFPT_AUX_CMD, 0);
    cofp_aac = (void *)(b->data);
    cofp_aac->cmd_code = htonl(C_AUX_CMD_SVC_1);
    buf = (void *)(cofp_aac->data);

    snprintf(buf, 12, "Hello, World");
    c_service_send(fab_service, b);
}

/*
 *  Servlet Request handler
 */
static void example_service_handler(void *service, struct cbuf *b)
{
    struct c_ofp_auxapp_cmd *cofp_aac = (void *)(b->data);

    switch(ntohl(cofp_aac->cmd_code)) {
    case C_AUX_CMD_SVC_1:
        example_cmd_1_handler(service);
    default:
        example_service_error(service, b, OFPET_BAD_REQUEST,
                              OFPBRC_BAD_GENERIC);
    }
}
/*
 *  NOOP Event handler
 */
static void event_notifier(void *app_arg, void *pkt_arg)
{
    struct cbuf         *b = pkt_arg;
    struct ofp_header   *hdr;
    if (!b) {
        c_log_err("%s: Invalid arg", FN);
        return;
    }
    hdr = (void *)(b->data);
    switch(hdr->type) {
    /* No action on any event */
    case C_OFPT_RECONN_APP:
        mul_register_app(NULL, APP_NAME,
                         C_APP_ALL_SW, C_APP_ALL_EVENTS,
                         0, NULL, event_notifier);
    }
}
void hello_app_init(void *ctx)
{
     void *service = mul_app_create_service(SERVICE_NAME,
                                            example_service_handler);
     /* make sure service is created */
     assert(service);
     mul_register_app(NULL, APP_NAME,
                         C_APP_ALL_SW, C_APP_ALL_EVENTS,
                         0, NULL, event_notifier);
}
```

```
module_init(hello_app_init);
```

2.3.2 Service Consumer

```
/**
 * mul_get_error_detail -
 *
 * Get detail error info connected to mul
 */
struct cbuf *
mul_get_error_detail(void *service, uint64_t dpid)
{
    struct cbuf *b;
    struct c_ofp_auxapp_cmd *cofp_auc;
    struct c_ofp_req_dpid_attr *cofp_rda;
    struct ofp_header *h;

    if (!service) return NULL;

    b = of_prep_msg(sizeof(*cofp_auc) + sizeof(*cofp_rda),
                    C_OFPT_AUX_CMD, 0);

    cofp_auc = (void *)(b->data);
    cofp_auc->cmd_code = htonl(C_AUX_CMD_SVC_1);
    cofp_rda = (void *)(cofp_auc->data);
    cofp_rda->datapath_id = htonll(dpid);

    c_service_send(service, b);
    b = c_service_wait_response(service);
    if (b) {
        h = (void *)(b->data);
        if (h->type != C_AUX_CMD_SVC_1 ||
            ntohs(h->length) <= 0)) {
            c_log_err("%s: Failed", FN);
            free_cbuf(b);
            return NULL;
        }
    }

    return b;
}
```

3. Flow API

Software Defined Networking is all about controller Apps being able to access flow table of each datapath devices. MuL exposes such features through set of Flow API functions.

3.1 List of function APIs

3.1.1 mul_app_send_flow_add

Arguments:

Type	Name	Description
char *	app_name	Human readable name of app
void *	sw_arg	Opaque handle to internal switch struct. This can be NULL
uint64_t	dpid	64bit Datapath ID of device to insert flow
struct flow *	fl	Match parameters to match the flow. A field will be ignored if masked in wildcard field.
uint32_t	buffer_id	Buffer id
void *	actions	Pointer to buffer containing array of struct ofp_action_headers
size_t	action_len	Size of actions buffer
uint16_t	itimeo	
uint16_t	htimeo	
uint32_t	wildcards	Bit field of wildcards (compatible to Openflow 1.0 wildcards). A match parameter is “don’t care” if the corresponding field is set.
uint16_t	prio	Priority of match field. When a packet can be matched to multiple flows, this field will be used as a tie-breaker. If there are multiple matching flows with same priority flow selection is done arbitrarily by data-plane device.
uint8_t	flag	

Returns: 0

This function adds the flow to the flow table.

3.1.2 mul_app_send_flow_del

Arguments:

Type	Name	Description
char *	app_name	Human readable name of app
void *	sw_arg	Opaque handle to internal switch struct. This can be NULL
uint64_t	dpid	64bit Datapath ID of device
struct flow *	fl	Match parameters to match the flow. A field will be ignored if masked in wildcard field.
uint32_t	wildcards	Bit field of wildcards (compatible to Openflow 1.0 wildcards). A match parameter is “don’t care” if the corresponding field is set.
uint16_t	port	Output port
uint16_t	prio	Priority of match field. When a packet can be matched to multiple flows, this field will be used as a tie-breaker. If there are multiple matching flows with same priority flow selection is done arbitrarily by data-plane device.
uint8_t	flag	

Returns: 0

This function removes the flow from the flow table.

3.2 Example

3.2.1 Flow Add

Below Code Snippet explains the usage of Flow Add API.

```
void
app_add_flow(struct flow *in_flow, uint32_t port, uint32_t out_port, uint64_t dpid) {
   struct mul_act_mdata mdata;
   struct flow fl;
   struct flow mask;
   memset(&fl, 0, sizeof(fl));
   of_mask_set_dc_all(&mask);
   of_mask_set_dl_dst(&mask);
   of_mask_set_in_port(&mask);
   memcpy(fl.dl_dst, in_fl->dl_dst, OFP_ETH_ALEN);
   fl.in_port = htonl(port);
   mul_app_act_alloc(&mdata);
   mul_app_act_set_ctors(&mdata, dpid);
   mul_app_action_output(&mdata, out_port);
   mul_app_send_flow_add(APP_NAME, NULL,dpid, &fl,
                         &mask, 0xffffffff,
                         mdata.act_base, mul_app_act_len(&mdata),
                         0, 0, 2, 0);
   mul_app_act_free(&mdata);
}
```

3.2.2 Flow Delete

Below Code Snippet explains the usage of Flow Delete API.

```
void
app_del_flow(uint64_t dpid, uint8_t *mac_da) {
   struct flow        fl;
   struct flow        mask;
   memset(&fl, 0, sizeof(fl));
   of_mask_set_dc_all(&mask);
   of_mask_set_dl_dst(&mask);
   memcpy(&fl.dl_dst, mac_da, OFP_ETH_ALEN);
   mul_app_send_flow_del(APP_NAME, NULL, dpid, &fl,
                         &mask, 0, 0, 2, 0);
}
```

IV. MuL Service API

This chapter outlines Service API provided by existing ML API App in the code base.

1. TR_SERVICE

TR stands for topology & routing. However, currently only topology information is provided in this RPC-like Service API

```
#include “mul_tr_servlet.h”
 struct cbuf *mul_neigh_get(void *service, uint64_t dpid);
```

This function returns all neighbors of the switch given by dpid. The return message is in struct c_ofp_switch_neigh format with list of struct c_ofp_port_neigh attached behind.

2. ROUTE_SERVICE

Unlike other services, this service is shared memory construct based Service. Any of Service API functions should not be used. (Code will seg-fault).

```
#include “mul_route.h”
size_t mul_route_get_nodes(void *rt_service);
```

List of nodes in the network

GSList *mul_route_get(void *rt_service, int src_sw, int dest_sw);

Get Route between src_sw and dest_sw. Input parameters are switch aliases instead of DPID. List is populated with struct rt_path_elem_ entries.

void *mul_route_service_get(void);

mul_service_get equivalent for route service.

void *mul_route_service_destroy(void *rt_service);

mul_service_detroy equivalent for route service.

3. MUL_SERVICE

This is the Service exposed by MuL Controller Core.

#include “mul_servlet.h”

struct cbuf *mul_get_switches_brief(void *service);

struct cbuf *mul_get_switch_detail(void *service, uint64_t dpid);

int mul_get_flow_info(void *service, uint64_t dpid, bool flow_self,

bool dump_cmd, bool nbapi_cmd, void *arg,

void (*cb_fn)(void *arg, void *pbuf));

Example:

1) Group Add

Below code snippet explains how to use MUL_SERVICE to add a group in a switch.

int

app_group_install()

{

```
struct mul_act_mdata mdata;
struct of_act_vec_elem *act_elem = NULL;


struct of_group_mod_params *g_parms;
/* Allocate the group ID from Switch Group ID pool */
if ((group_id = ipool_get(sw->group_ipool, NULL)) < 0) {
   /* Unable to allocate group ID, we cannot proceed further.
    * Exit with Internal error as Code*/
   app_log_err("%s: Unable to allocate Group ID ", FN);
   /* Group ID cannot be allocated, No need to hold the switch any more*/
   return APP_INTERNAL_ERROR;
}
g_parms = calloc(1, sizeof(struct of_group_mod_params));
memset(g_parms, 0, sizeof(*g_parms));
g_parms->group =  mul_app_group_id_alloc(group_id);
g_parms->type = OFPGT_ALL;
g_parms->flags = C_GRP_STATIC | C_GRP_BARRIER_EN | C_GRP_GSTATS;
mul_app_act_alloc(&mdata);
mdata.only_acts = true;
mul_app_act_set_ctors(&mdata, dpid);
/* Set source MAC*/
mul_app_action_set_smac(&mdata, hw_addr);
/* Set destination MAC*/
mul_app_action_set_dmac(&mdata, dl_dst_mac);
```

```
    /* Decrementing Network TTL*/
    mul_app_action_dec_nw_ttl(&mdata);
    /* Set the output port */
    mul_app_action_output(&mdata, outport);

    act_elem = calloc(1, sizeof(*act_elem));

    act_elem->actions = mdata.act_base;
    act_elem->action_len = of_mact_len(&mdata);
    g_parms->act_vectors[0] = act_elem;
    g_parms->act_vec_len = 1;

    /* Send group add to MUL Core*/
    mul_service_send_group_add(mul_service, dpid, g_parms);
    /* Must Wait for response after sending a request using service */
    if (c_service_timed_wait_response(mul_service) > 0) {
        app_log_err("%s: Failed to add a group %u. Check log messages",
                FN, g_parms->group);
        return -1;
    }
    return 0;
}
```

2) Group Delete

Below code snippet explains how to use MUL_SERVICE to delete a group in a switch.

```
void
app_group_uninstall(uint32_t group_id)
{
    struct of_group_mod_params *g_parms;
    g_parms = calloc(1, sizeof(struct of_group_mod_params));
    memset(g_parms, 0, sizeof(*g_parms));
    g_parms->group =  group_id;
    g_parms->type = OFPGT_ALL;
    g_parms->flags = C_GRP_STATIC | C_GRP_BARRIER_EN | C_GRP_GSTATS;

   /* Send group add to MUL Core*/
   mul_service_send_group_del(mul_service, dpid, gp_parms);
   if (c_service_timed_wait_response(mul_service) > 0) {
       app_log_err("%s: Failed to delete a group %u. Check log messages",
               FN, gp_parms->group);
   }
}
```

3) Meter Add

Below code snippet explains how to use MUL_SERVICE to add a meter in a switch.

```
int
app_meter_install(uint32_t meter_id, uint64_t dpid) {
   struct mul_act_mdata mdata[2];
   struct of_meter_band_parms meter_band_params[2];
   struct of_meter_mod_params m_parms;
   uint16_t num_bands = 0;
   uint16_t band = 0;
   uint8_t version = 0;

   if (mul_app_act_set_ctors(&mdata[0], dpid)) {
      app_log_err( "Switch 0x%llx does not exist\r\n", U642ULL(dpid));
      return -1;
   }

   version = c_app_switch_get_version_with_id(dpid);
   if (version !=  OFP_VERSION_131 && version != OFP_VERSION_140) {
      app_log_err( "Switch 0x%llx does not support meter\r\n", U642ULL(dpid));
      return -1;
   }

   mdata[0].only_acts = true;
   meter_band_params[0].rate = 12345;
   meter_band_params[0].burst_size = 123;
```

```
/* Add drop meter band*/
mul_app_set_band_drop(&mdata[0],&meter_band_params[0]);
num_bands++;

if (mul_app_act_set_ctors(&mdata[1], dpid)) {
   app_log_err( "Switch 0x%llx does not exist\r\n", U642ULL(dpid));
   return -1;
}

mdata[1].only_acts = true;
meter_band_params[1].rate = 55555;
meter_band_params[1].burst_size = 555;
meter_band_params[1].prec_level = 5;
/* Add drop meter band*/
 mul_app_set_band_dscp(&mdata[1], &meter_band_params[1]);
num_bands++;

memset(&m_parms, 0, sizeof(m_parms));
m_parms.meter = meter_id; /*meter_id*/
m_parms.flags = OFPMF_KBPS; /*Meter type*/
m_parms.cflags = C_METER_STATIC;  /* Controller's Flag*/
m_parms.cflags |= C_METER_GSTATS; /* meter with statistics*/

for (band = 0; band < num_bands; band++) {
   band_elem = calloc(1, sizeof(*band_elem));
```

```
        band_elem->band =mdata[band].act_base;
        band_elem->band_len = of_mact_len(&mdata[band]);
        m_parms.meter_bands[band] = band_elem;
    }
    m_parms.meter_nbands = num_bands;
    mul_service_send_meter_add(mul_service,dpid, &m_parms);
    if (c_service_timed_wait_response(mul_service) > 0) {
        app_log_err( "Failed to add meter. Check log messages");
        return -2;
    }
    for (band = 0; band < nu,_bands; band++) {
        of_mact_free(&mdata[band]);
        free(m_parms.meter_bands[band]);
    }
   return 0;
}
```

4) Meter Delete

```
int
app_meter_del(uint32_t meter_id, uint64_t dpid) {
    struct of_meter_mod_params m_parms;
    uint64_t dpid;
    uint32_t meter;
    uint8_t version;
```

```
    memset(&m_parms, 0, sizeof(m_parms));


   version = c_app_switch_get_version_with_id(dpid);
    if (version !=  OFP_VERSION_131 && version != OFP_VERSION_140) {
        app_log_err( "Switch 0x%llx does not support meter\r\n", U642ULL(dpid));
        return -1;
    }


    m_parms.meter = meter_id;
    m_parms.cflags = C_METER_STATIC;


    mul_service_send_meter_del(mul_service, dpid, &m_parms);
    if (c_service_timed_wait_response(mul_service) > 0) {
        app_log_err("Failed to del the meter. Check log messages%s");
        return -2;
    }
    return 0;
}
```