

# Crossbow Interfaces

## 1 Libraries

The libdladm library is currently used to implement data-link administration for all GLDv3 data-links, including link aggregation and wireless. Project Crossbow extends the libdladm library to provide a programming interface for VNICs and flows management.

Libdladm is a consolidation private library. This phase of Crossbow will not alter that classification.

### ○ **VNICs and Etherstubs**

The following functions are new to libdladm. They are declared in <libdlvnic.h>:

- `dladm_vnic_create()`

```
dladm_status_t dladm_vnic_create(const char *vnic, datalink_id_t linkid,  
                                dladm_vnic_mac_addr_type_t mac_addr_type, uchar_t *mac_addr, int mac_len,  
                                int *mac_slot, uint_t mac_prefix_len, uint16_t vid, datalink_id_t *vnic_id_out,  
                                uint32_t flags)
```

Creates a VNIC with the specified *vnic* name on top of the data-link specified by *linkid*. This function is invoked by the `dladm(1M)` `create-vnic` subcommand, and when creating a VLAN data-link. The *mac\_addr\_type* specifies the type of MAC address to be assigned to the VNIC, and corresponds to the keywords allowed by the `dladm create-vnic` sub-command, see `dladm(1m)` draft man page.

The *vnic* argument can be set to NULL. In that case, a name will be picked for the VNIC, and can be obtained by the caller through *vnic\_id\_out*. When a new random MAC address is requested, the desired prefix is passed through *mac\_addr*, the prefix length through *mac\_prefix\_len*, and *mac\_len* is set to zero.

The generated MAC address is passed back through the *mac\_addr* argument.

The following flags are supported:

- `DLADM_OPT_PERSIST`:

The VNIC configuration will be stored in the data-link repository.

- `DLADM_OPT_ANCHOR`:

Create an anchor VNIC for an etherstub, see Draft man page.

- `DLADM_OPT_VLAN`:

Create VLAN data-link, specified when creating a VLAN using the `dladm(1M)` `create-vlan` subcommand.

- `DLADM_OPT_NODUPCHECK`:

The library will not enforce that the VNIC MAC address is unique on the same underlying data-link.

- `dladm_vnic_delete()`

```
dladm_status_t dladm_vnic_delete(datalink_id_t linkid, uint32_t flags);
```

Delete the specified VNIC. The `DLADM_OPT_PERSIST` and `DLADM_OPT_ACTIVE` are supported.

- `dladm_vnic_up()`

```
dladm_status_t dladm_vnic_up(datalink_id_t linkid, uint32_t flags);
```

Brings up the specified VNIC, or all VNICs if *linkid* is set to `DATALINK_ALL_LINKID`. This function is invoked via `dladm(1M)` at boot time to instantiate all VNICs. The `DLADM_OPT_VLAN` flag is specified to bring only data-links of type VLAN.

## ○ **Resource Controls Link Properties**

Resource controls for data links in general (physical links, VNICs, link aggregations) are expressed as link properties. They are passed using the existing `dladm_{set, get}_prop()` functions of the `libdladm`

The VNIC administration library provides a consolidation private API which can be used by consumers such as `dladm(1M)` to create, delete, and query information about VNICs and Etherstubs. It also maintains a persistent repository which allows VNIC configuration to be stored across reboots.

## ○ **Flows**

Flows and their properties are declared in `<flow.h>`. The following functions are new to `libdladm` and are declared in `libdlflow.h`

- `dladm_flow_add()`

```
dladm_status_t dladm_flow_add(datalink_id_t linkid, flow_desc_t *flowdesc,
                             dladm_prop_list_t *proplist, char *flowname, boolean_t tempop, const char *root)
```

Creates a flow with the specified *flowname* on top of the data-link specified by *linkid*. This function is invoked by the `flowadm(1M)` `add-flow` subcommand. see `flowadm(1m)` draft man page.

The *flowdesc* argument is a data structure defined in `<flow.h>`. It describes the criteria for packets to match the specified flow. These criteria are essentially the header fields that need to be matched (IP source or destination addresses, transport type, port number, etc), and their values. `flowadm add-flow` translates the flow attributes passes to it in the CLI into a `flow_desc_t` structure.

The *proplist* is the array of flow properties specified at creation time.

The *tempop* indicated whether a temporary or a persistent flow is requested.

When not NULL, the *root* argument is the path name for an alternative root, coming from the -R option of the CLI.

- `dladm_flow_remove()`

```
dladm_status_t dladm_flow_remove(char *flowname, boolean_t tempop, const char *root)
```

Removed the flow called *lowname* from the data-link over which it is currently defined. This function is invoked by the `flowadm(1M)` `remove-flow` subcommand. see `flowadm(1m)` draft man page.

- `dladm_flow_info()`

```
dladm_status_t dladm_flow_info(const char *flowname, dladm_flow_attr_t attr)
```

This routine retrieves the attributes defining the flow.

- `dladm_set_flowprop()`

```
dladm_status_t dladm_set_flowprop(const char *flow, const char *prop_name, char **prop_val, uint_t val_cnt, uint_t flags, char **errprop)
```

This routine sets the properties (`maxbw`, `cpus`, `fanout`, `priority`) on the flow.

A call to `flowadm set-flowprop` is translated directly into an invocation of this function. All the arguments parallel the syntax and semantics of `dladm_set_linkprop()`. The only difference is the fact that this routine operates on a flow specified by its name *flow* instead of a *linkid*.

- `dladm_get_flowprop()`

```
dladm_status_t dladm_get_flowprop(const char *flow, uint32_t type, const char *prop_name, char **prop_val, uint_t *val_cnt)
```

This routine reports the properties (`maxbw`, `cpus`, `fanout`, `priority`) of the specified flow.

## 2 MAC Client API

The project defines the following Consolidation Private MAC Client APIs. These interfaces are used by internal components of Crossbow, such as DLS, Xen/XVM `xnbo` back-end driver, and LDom's `vswitch` are also consumers of these interfaces, and were both ported to the new API as part of Crossbow.

The full and detailed description is provided in section 5.2 – Consolidation Private MAC API - of the Crossbow MAC Virtualization document ([References/crossbow-virt.pdf](#)), included as a reference with the project's materials.

The declaration is also included in `<sys/mac.h>`

```
int mac_client_open(mac_handle_t *mh, char *name, mac_client_handle_t *mchp,
```

uint16\_t flags);

void mac\_client\_close(mac\_client\_handle\_t mch, uint16\_t flags);

mac\_client\_open() must be called by a MAC client before being able to send a receive data through a MAC. Each call to mac\_client\_open() returns a new opaque mac\_client\_handle\_t used by the caller for all subsequent operations with this client.

int c\_client\_set\_resources(mac\_client\_handle\_t mch, net\_bind\_cpu\_t \*bind\_cpu, net\_resource\_ctl\_t \*res\_ctl, uint32\_t modify\_mask, uint32\_t flags);

void mac\_client\_get\_resources(mac\_client\_handle\_t mch, net\_bind\_cpu\_t \*bind\_cpu, net\_resource\_ctl\_t \*res\_ctl);

Sets or retrieves the list of resources used to process the traffic associated with the specified MAC client. modify\_mask specifies the type of resource being set.

int mac\_unicast\_handle\_t mac\_unicast\_add( mac\_client\_handle\_t mch, uchar\_t \*mac\_addr, uint32\_t flags, mac\_unicast\_handle\_t &mah, uint16\_t vid, mac\_diag\_t \*diag);

void mac\_unicast\_remove(mac\_unicast\_handle\_t);

Add and Remove a unicast MAC address associated with the client.

typedef void (\*mac\_rx\_t)(void \*arg, mac\_resource\_handle\_t ring, mblk\_t \*mp);

int mac\_rx\_set(mac\_client\_handle\_t mch, mac\_rx\_fn\_t rx\_fn, void \*arg);

int mac\_rx\_clear(mac\_client\_handle\_t mch);

Set the receive function to be invoked for the MAC client specified by mch. The client must have previously set a MAC address with mac\_addr\_set() before calling this function

int mac\_multicast\_add(mac\_client\_handle\_t mch, const uchar\_t \*addr);

int mac\_multicast\_remove(mac\_client\_handle\_t mch, const uchar\_t \*addr);

Add or remove a multicast address to or from a MAC client's list of multicast addresses. Packets for the specified multicast address will be sent to the receive callback specified by mac\_rx\_set()

mblk\_t \* mac\_tx(mac\_client\_handle\_t \*mch, mblk\_t \*mp\_chain, uint64\_t hint, uint16\_t flag, mblk\_t \*\*ret\_mp);

Sends the specified packet chain from the MAC client associated with the specified handle.

int mac\_promisc\_add(mac\_client\_handle\_t mch, mac\_promisc\_type promisc\_type,

```
mac_promisc_fn_t promisc_fn, void *arg, mac_promisc_handle_t *php);
int mac_promisc_remove(mac_client_handle_t mch, mac_promisc_handle_t *ph);
```

Register a promiscuous mode callback for the MAC client specified by mch. The promisc\_fn() function will be given a copy sent and received packets

Further of Project Private MAC Client APIs were defined, and they are described in section 5.3 – Project Private MAC API - of the same document (Crossbow MAC Virtualization).

### 3 MAC Driver API

The project extends the Consolidation Private driver interface to express the modern hardware capabilities that support virtualization.. The virtualized MAC layer produced an abstraction of the drivers capabilities, insulating the MAC clients from the diversity of NICs and offering them a common way to access the links. With Crossbow, the MAC framework became the only consumer of the driver interface.

All start with the MAC\_CAPAB\_RINGS which allows the MAC framework to query and keep an up-to-date an inventory of all groups, their type and current state, and their ring members.

The interface defines the per-group and per-ring control operations such as MAC addresses add/remove, and interrupt throttling. It also exposes the per-ring data transfer entry points (transmit and poll)

Section 2 – Provider Interfaces – of the Crossbow Provider Model And Interface gives a full and detailed description of the drivers interface. Section 3 uses two examples of commercially available 10Gb NICs to analyze the main design choices a MIC driver writer may face when decides on how to expose the hardware's rings and ring groups.

### 4 Deprecated Interfaces

The following two Consolidation Private Interfaces are deprecated by the Crossbow project

- The mc\_resources of the mac\_callbacks\_t  
the mc\_resources used to be the only way for a driver capable of multiple receive rings to advertise them. MAC\_CAPAB\_RINGSs is now used instead.
- MAC\_CAPAB\_MULTIADDRESS was the first attempt at offering clients a way to ask the driver for filtering on more than a single unicast MAC address. That is now replaced by the adding of MAC addresses to ring groups.

- VLAN PPA “hack”. The naming convention for creating VLAN interfaces documented in the following section of dlpi(7) is being deprecated. dladm(1m) is the only supported way to create VLAN interfaces. Attempts to create a VLAN interface by simply opening a device according to the deprecated way (e.g. From ifconfig(1m)) will fail with ENODEV.

#### VLAN Support

##### VLAN PPA Access

Some DL\_ETHER DLPI providers support IEEE 802.1Q Virtual LANs (VLAN). For these providers, traffic for a particular VLAN can be accessed by attaching to (or opening) a special VLAN PPA that is calculated from the VLAN ID and the actual hardware PPA of the device. This VLAN PPA is calculated by multiplying the VLAN ID by 1000 and adding the hardware PPA. For example, VLAN PPA 2001 provides access to VLAN ID 2 on hardware PPA 1.

- dladm show-dev was made redundant by the introduction of show-phys. It will be removed in this project.