NETRONOME

Agilio OvS 2.2 Programmer's Reference Manual

Netronome Intelligent Server Adapters

- Proprietary and Confidential -

Product code
040-00023-000

Agilio OvS 2.2 Programmer's Reference Manual: Netronome Intelligent Server Adapters

COPYRIGHT

No part of this publication or documentation accompanying this Product may be reproduced in any form or by any means or used to make any derivative work by any means including but not limited to by translation, transformation or adaptation without permission from Netronome Systems, Inc., as stipulated by the United States Copyright Act of 1976. Contents are subject to change without prior notice.

WARRANTY

Netronome warrants that any media on which this documentation is provided will be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of shipment. If a defect in any such media should occur during this 90-day period, the media may be returned to Netronome for a replacement.

NETRONOME DOES NOT WARRANT THAT THE DOCUMENTATION SHALL BE ERROR-FREE. THIS LIMITED WARRANTY SHALL NOT APPLY IF THE DOCUMENTATION OR MEDIA HAS BEEN (I) ALTERED OR MODIFIED; (II) SUBJECTED TO NEGLIGENCE, COMPUTER OR ELECTRICAL MALFUNCTION; OR (III) USED, ADJUSTED, OR INSTALLED OTHER THAN IN ACCORDANCE WITH INSTRUCTIONS FURNISHED BY NETRONOME OR IN AN ENVIRONMENT OTHER THAN THAT INTENDED OR RECOMMENDED BY NETRONOME.

EXCEPT FOR WARRANTIES SPECIFICALLY STATED IN THIS SECTION, NETRONOME HEREBY DISCLAIMS ALL EXPRESS OR IMPLIED WARRANTIES OF ANY KIND, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE.

Some jurisdictions do not allow the exclusion of implied warranties, so the above exclusion may not apply to some users of this documentation. This limited warranty gives users of this documentation specific legal rights, and users of this documentation may also have other rights which vary from jurisdiction to jurisdiction.

LIABILITY

Regardless of the form of any claim or action, Netronome’s total liability to any user of this documentation for all occurrences combined, for claims, costs, damages or liability based on any cause whatsoever and arising from or in connection with this documentation shall not exceed the purchase price (without interest) paid by such user.

IN NO EVENT SHALL NETRONOME OR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THE DOCUMENTATION BE LIABLE FOR ANY LOSS OF DATA, LOSS OF PROFITS OR LOSS OF USE OF THE DOCUMENTATION OR FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, EXEMPLARY, PUNITIVE, MULTIPLE OR OTHER DAMAGES, ARISING FROM OR IN CONNECTION WITH THE DOCUMENTATION EVEN IF NETRONOME HAS BEEN MADE AWARE OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL NETRONOME OR ANYONE ELSE WHO HAS BEEN INVOLVED IN THE CREATION, PRODUCTION, OR DELIVERY OF THE DOCUMENTATION BE LIABLE TO ANYONE FOR ANY CLAIMS, COSTS, DAMAGES OR LIABILITIES CAUSED BY IMPROPER USE OF THE DOCUMENTATION OR USE WHERE ANY PARTY HAS SUBSTITUTED PROCEDURES NOT SPECIFIED BY NETRONOME.

Revision History

Date	Revision	Description
24 February 2016	000	First DRAFT for Agilio OvS 2.2.

Table of Contents

List of Tables

1. Introduction

1.1 Scope

This manual details the host APIs for programmatically manipulating aspects of the Netronome Systems Agilio OvS 2.2 product. The intended audience for this book includes developers and system programmers.

This manual is organized as follows:

- Chapter 2 provides a high-level description of the local Flow API for manipulating the Agilio OvS 2.2 Flow Tables.
- Chapter 3 provides a high-level description of the group API for manipulating the Agilio OvS 2.2 group entries.
- Chapter 4 provides a high-level description of the health API.
- Chapter 5 provides a complete reference of the APIs.
- Chapter 6 provides information on technical support.

1.2 Related Documents

Descriptive Name	Description
Netronome Agilio OvS 2.2 User's Guide	Provides a general overview of the Agilio OvS firmware.
Netronome Agilio OvS 2.2 Getting Started Guide	A guide to new users of Netronome's OpenFlow switch software.
OpenFlow Switch Specification 1.3.3	Control protocol standardized by Open Networking Foundation.

1.3 Terminology

Acronym	Description
API	Application Programming Interface
CIDR	Classless Inter-Domain Routing
CSV	Comma Separated Values, a simple text file format
FST	Flow State Table
IPv4	Internet Protocol version 4
LB	Load Balancing

Acronym	Description
MAC	Media Access Control
MPLS	MultiProtocol Label Switching
NFD	Network Flow Driver
NFP	Netronome Network Flow Processor
NIC	Network Interface Card
OvS	Open vSwitch
PCIe	PCI Express
SDN	Software Defined Networking
TCP	Transmission Control Protocol
TTL	Time To Live
VLAN	Virtual Local Area Network
VM	Virtual Machine
VNIC	Virtual Network Interface Card

2. Local Flow API

2.1 Purpose

The local Flow API allows the user to manipulate (add, modify, delete) flow entries in Netronome Agilio OvS 2.2 flow tables. These flow entry operations closely mirror OpenFlow semantics as described in the OpenFlow 1.3 specification[1].

2.2 Overview

A flow entry typically consists of the following information:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

The API function prototypes are declared in the `ofp_flow.h` file[2], and these functions allow the user to add, modify and delete flow entries in specified tables. The flow entry fields (as in the table definition above, with the exception of Counters) can also be manipulated by these APIs. During flow processing, the Match Fields are compared against packet headers, ingress port and other fields such as metadata from a previous table match. The Priority specifies the precedence of the flow entry. A flow table entry is uniquely identified by the Match Fields and Priority combination. The flow entry that wildcards all fields (all fields omitted) and has priority set to 0 is called the table-miss flow entry. The Cookie field is an opaque value chosen by the user, not used when processing packets but used to later filter flow entries for statistics, flow modification and flow removal.

The Instructions field consists of an ordered set comprising at most one of each of the following:

- Meter *meter_id*
- Apply-Actions *action_list*
- Clear-Actions
- Write-Actions *action_set*
- Write-Metadata *metadata/mask*
- Goto-Table *next-table-id*

APIs are provided to enable each of these instructions in the flow entry. The APIs for the actions referred to by the two action instructions need to be called in an apply or write mode, to distinguish between the ordered list required by the Apply-Actions instruction, and the set required by the Write-Actions instruction. To this end, the user must call `ofp_action_create()` prior to calling a series of `ofp_action_*` functions, and finally the apply or write instruction can be enabled. The general usage of the API is as follows:

1. Initialize an `ofp_flow` subsystem handle (`ofp_sdn_sys_init()`). An error callback function pointer may be registered, this will be invoked in the event of an error indication from the switch, and provides a transaction id, error type and error code (details can found in the OpenFlow specification).

[1] Refer to the section named "OpenFlow Tables" in the OpenFlow Switch Specification 1.3.3 on the www.opennetworking.org website.
[2] This file is included in the Agilio OvS 2.2 installation, it can be found in the `/opt/netronome/include` directory.

2. Create an `ofp_flow_entry` handle (`ofp_flow_entry_create()`), specifying table type, whether the entry is for add, delete or modify operation, and operation-specific flags. Some table types do not support the full OpenFlow semantics (see Section 2.3).
3. Set the desired packet match fields. If left unspecified, the flow entry will match all packets (`ofp_flow_entry_match_*` APIs).
4. Set the priority, defaults to 0 if left unspecified (`ofp_flow_entry_priority()`). Note: Omitting both steps 3 and 4 will result in the flow entry being regarded as the table-miss flow entry. This special entry specifies how to process packets unmatched by other flow entries in the table.
5. Set the timeouts and cookie fields as desired (`ofp_flow_entry_timeout()` , `ofp_flow_entry_cookie()`).
6. Add actions, after creating a new handle (`ofp_action_create()`), to be used next with Write-Actions or Apply-Actions instruction (`ofp_action_*` APIs).
7. Add instructions to the entry (`ofp_flow_entry_instr_*` APIs).
8. Optionally, a transaction ID may be specified (`ofp_flow_entry_xid()`). This is an arbitrary user defined number which can be meaningful in the event of an error callback on a particular flow entry operation. If left unspecified it will auto-increment from the previous value (starting at 0 for the very first transaction).
9. Commit the entry to a specified table number (`ofp_flow_entry_commit()`).
10. Repeat steps 2-8 as often as necessary.
11. Terminate the ofp_flow subsystem handle (`ofp_sdn_sys_shutdown()`).

2.3 Table Types

Flow entries are created in numbered tables of different types in Agilio OvS 2.2 (see `ofp_flow_entry_create()`). Two table types are enumerated:

1. Open vSwitch table via OpenFlow protocol.
2. NFP hardware flow cache table (not implemented currently).

The Open vSwitch tables are programmed via the OpenFlow protocol over a TCP connection. The API library acts as an OpenFlow controller, and Open vSwitch needs to be configured to allow so-called passive TCP connections:

```
ovs-vsctl set-controller br0 ptcp:6653
```

In particular, the API uses OpenFlow version 1.3, and Open vSwitch needs to be configured to not exclude that version. This can be accomplished by not restricting the protocol versions at all (which is also the default setting):

```
ovs-vsctl set Bridge br0 protocols=[]
```

or specifically setting it to allow version 1.3:

```
ovs-vsctl set Bridge br0 protocols=OpenFlow13
```

2.4 Sample Applications

Sample applications using the Flow API are included in the Agilio OvS 2.2 installation. The source files are located in `/opt/netronome/samples/user_api` and can be compiled using the following command:

```
make
```

2.4.1 CSV Sample Application

The application (source in `ofp_flow_csv.c`) adds OvS table 0 entries from a fixed format CSV file containing entries for matching source IPv4 address, destination IPv4 address and an outport port action (referring to OpenFlow port number). The IPv4 addresses may optionally include netmasks in CIDR notation or dotted-decimal format, e.g. `192.168.0.0/16` or `192.168.0.0/255.255.0.0` would equivalently match only the top 16 bits of the IP address. The output port (3rd item in each CSV entry) can be set to 0 to indicate drop (i.e. no output action), or -1 to indicate switch normal processing (in effect, L2 learning with Open vSwitch).

The default table-miss rule can be specified by adding a CSV entry containing `0.0.0.0/0` for source and destination IP address, with output port set as desired (or 0 to drop).

2.4.2 Command Line Control Sample Application

This sample (`ofp_flow_ctl.c`) provides more exhaustive coverage of the Flow API by implementing a set of command line options to manipulate (add, delete or modify) the flow tables. The complete usage information can be obtained by typing: `ofp_flow_ctl -?`

The user can for example add the table-miss flow entry (by omitting match fields and priority) to output to the OpenFlow NORMAL port as follows:

```
ofp_flow_ctl --add --write-actions output:0xfffffffa
```

or add a flow entry with masked source/destination IPv4 address (the field '`ip`' expands internally to match ethernet type 0x0800):

```
ofp_flow_ctl --add --match-fields ip,ip_src=10.0.0.0/8,ip_dst=10.0.0.0/8 \
--write-actions output:0xfffffffa
```

A slightly more complete example might set up a chain of two table entries, the first entry (with a timeout) matching some IP range and setting metadata, and the second matching on that metadata and performing an output action to OpenFlow port 1:

```
ofp_flow_ctl --add --table_num 0 --match-fields ip,ip_src=10.0.0.0/8,ip_dst=10.0.0.0/8\
--priority 5 --timeout 60 --cookie 1234 --write-metadata 42 --goto-table 1
ofp_flow_ctl --add --table_num 1 --match-fields metadata=42 --write-actions output:1
```

3. Group API

3.1 Purpose

The Group API allows the user to manipulate (add, modify, delete) group entries in Netronome Agilio OvS 2.2 group tables. These operations closely mirror OpenFlow semantics as described in the OpenFlow 1.3 specification[1].

3.2 Overview

A group entry typically consists of the following information:

Identifier	Type	Counters	Action buckets

The API function prototypes are declared in the `ns_group.h` file[2], and these functions allow the user to add, modify and delete group entries.

The general usage of the API is as follows:

1. Create an `ns_group` handle (`ofp_group_create()`), specifying the command (add, modify or delete), group type (all, select, indirect, fast failover), and group identifier.
2. Create a new action set (`ofp_action_create()`), populate it with desired actions.
3. Add the action set to the group as an action bucket (`ofp_group_add_action_bucket()`) with a specified weight.
4. Repeat steps 2-3 as often as necessary. Indirect groups may only have a single action bucket.
5. Commit the group modifier (`ofp_group_commit()`).

3.3 Sample Application

A minimal Group API sample application is included in the Agilio OvS 2.2 installation. The source file is located in `/opt/netronome/samples/user_api/ofp_group_tst.c` and can be compiled using the following command:

```
make
```

The sample creates a load balancing group outputting to OpenFlow ports 2 and 3.

[1] Refer to the section named "OpenFlow Tables" in the OpenFlow Switch Specification 1.3.3 on the www.opennetworking.org website.
[2] This file is included in the Agilio OvS 2.2 installation, it can be found in the `/opt/netronome/include` directory.

4. Health Monitoring API

4.1 Purpose

The health monitoring API allows the user to monitor the system health on the host operating system. The health monitoring API allows the user to create their own health monitoring application suitable to the user's operating environment.

4.2 Overview

The API function prototypes are declared in the `ns_sdn_health.h` file[1], and these functions allow the user to create an application to monitor the health of the Agilio OvS system.

The general usage of the API is as follows:

1. Initialize an `ns_sdn_health_handle_t` instance.
2. Execute (`ns_sdn_health_check()`) to run all the health checks on the system. This function can be called periodically if needed.
3. After the health checks have been executed, the results can be processed by calling (`ns_sdn_health_next_failure()`) successively until no more failures are returned.
4. Destroy the `ns_sdn_health_handle_t` instance (`ns_sdn_health_destroy()`).

4.3 Sample Applications

A sample applications using the Health Monitoring API are included in the Agilio OvS 2.2 installation. The source files are located in `/opt/netronome/samples/sdn_health` and can be compiled using the following command:

```
make
```

The included sample provide a few possible options to display the current system health. Refer to the usage output of the sample.

[1] This file is included in the Agilio OvS 2.2 installation, it can be found in the `/opt/netronome/include` directory.

5. Agilio OVS 2.2 Host API Reference

5.1 ofp_flow

5.1.1 Defines

Table 5.1. ofp_flow.h Defines

Defined	Definition
`ofp_flow_entry_timeout`	`ofp_flow_entry_hard_timeout`

5.1.2 Enumerations

5.1.2.1 ofp_flow_mod_command

Table 5.2. enum ofp_flow_mod_command

Name	Description
`OFPFC_ADD`	New flow.
`OFPFC_MODIFY`	Modify all matching flows.
`OFPFC_MODIFY_STRICT`	Modify entry strictly matching wildcards and priority.
`OFPFC_DELETE`	Delete all matching flows.
`OFPFC_DELETE_STRICT`	Delete entry strictly matching wildcards and priority.
`STATS_FAKE_CMD`	Fake command to be used when creating a flow entry to be used by `ofp_flow_entry_stats()`.

5.1.2.2 ofp_flow_table_type

Table 5.3. enum ofp_flow_table_type

Name	Description
`OFP_FLOW_TABLE_OVS_OPENFLOW`	Open vSwitch table via OpenFlow protocol.
`OFP_FLOW_TABLE_NFP_CACHE`	NFP hardware flow cache (not implemented).

5.1.2.3 ofp_group_no

Table 5.4. enum ofp_group_no

Name	Description
`OFPG_MAX`	Last usable group number.
`OFPG_ALL`	Represents all groups for group delete commands.
`OFPG_ANY`	Wildcard group used only for flow stats requests. Selects all flows regardless of group (including flows with no group).

5.1.2.4 ofp_port_no

Table 5.5. enum ofp_port_no

Name	Description
`OFPP_MAX`	Maximum number of physical and logical switch ports.
`OFPP_IN_PORT`	Send the packet out the input port. This reserved port must be explicitly used in order to send back out of the input port.
`OFPP_TABLE`	Submit the packet to the first flow table NB: This destination port can only be used in packet-out messages.
`OFPP_NORMAL`	Process with normal L2/L3 switching.
`OFPP_FLOOD`	All physical ports in VLAN, except input port and those blocked or link down.
`OFPP_ALL`	All physical ports except input port.
`OFPP_CONTROLLER`	Send to controller.
`OFPP_LOCAL`	Local openflow "port".
`OFPP_ANY`	Wildcard port used only for flow mod (delete) and flow stats requests. Selects all flows regardless of output port (including flows with no output port).

5.1.2.5 ofp_table

Table 5.6. enum ofp_table

Name	Description
`OFPTT_MAX`	Last usable table number.
`OFPTT_ALL`	Wildcard table used for table config, flow stats and flow deletes.

5.1.3 Typedefs

5.1.3.1 openflow_error_fptr_t

Error callback function pointer.

OpenFlow error indications are sent asynchronously by the switch. Registering this error callback in the ofp_sdn_config struct passed to `ofp_sdn_sys_init()` allows the user to be notified of these errors. The user_context (also initially configured in ofp_sdn_config) is given back to the user, along with xid (a transaction ID which can be set on any flow entry using the `ofp_flow_entry_xid()` function). The errtype and errcode are received from the switch, and refer to OpenFlow errors which can be found in the OpenFlow specification (Section 7.4.4 in OpenFlow 1.3.4). The type value indicates the high-level type of error, while the code value is interpreted based on the type.

Table 5.7. typedef openflow_error_fptr_t

Type	Definition
`openflow_error_fptr_t`	`void(* openflow_error_fptr_t)(uint64_t user_context, uint32_t xid, uint16_t errt`

See Also:

- `ofp_sdn_barrier()`

5.1.4 Functions

5.1.4.1 ofp_sdn_sys_init

Prototype:

```
struct ofp_sdn* ofp_sdn_sys_init(struct ofp_sdn_config* conf)
```

Description:

ofp_sdn API system initialization.

This function needs to be called before any flow modifications can be performed. The resulting handle can be passed repeatedly to `ofp_flow_entry_create()`, before eventually calling `ofp_sdn_sys_shutdown()` when done.

Open vSwitch needs to be configured to allow controller connections, e.g. ovs-vsctl set-controller br0 ptcp:6653

Table 5.8. ofp_sdn_sys_init parameters

Name	Description
`conf`	Pointer to struct containing config. If ip_addr[0]==0, the /etc/netronome.conf file will be parsed, and if that file is not found, IP 127.0.0.1 and port 6653 will be used.

Returns:

The initialized handle is returned on success, NULL otherwise.

See Also:

- `ofp_sdn_sys_shutdown()`
- ofp_sdn_entry_create()

5.1.4.2 ofp_sdn_sys_shutdown

Prototype:

```
void ofp_sdn_sys_shutdown(struct ofp_sdn* nssdn)
```

Description:

API system cleanup

Reverts all the initialization done by `ofp_sdn_sys_init()`, this must be called when done with the API.

5.1.4.3 ofp_sdn_barrier

Prototype:

```
int ofp_sdn_barrier(struct ofp_sdn* nssdn)
```

Description:

OpenFlow transaction barrier.

This call sends a barrier request and waits for a barrier reply. When the reply has been received, the user can be assured that all previous asynchronous flow requests have been completed.

Table 5.9. ofp_sdn_barrier parameters

Name	Description
nssdn	ofp_sdn API handle

Returns:

non-zero indicates failure

5.1.4.4 ofp_sdn_barrier_timeout

Prototype:

```
int ofp_sdn_barrier_timeout(struct ofp_sdn* nssdn, uint32_t timeout_ms)
```

Description:

OpenFlow transaction barrier with timeout.

This call sends a barrier request and waits (with timeout) for a barrier reply. When the reply has been received, the user can be assured that all previous asynchronous flow requests have been completed.

Table 5.10. ofp_sdn_barrier_timeout parameters

Name	Description
nssdn	ofp_sdn API handle
timeout_ms	Timeout period in milliseconds

Returns:

non-zero indicates failure or timeout

5.1.4.5 ofp_flow_entry_create

Prototype:

```
struct ofp_flow_entry* ofp_flow_entry_create(struct ofp_sdn* nssdn, enum
ofp_flow_table_type ttype, enum ofp_flow_mod_command cmd, unsigned flags)
```

Description:

Create Flow entry handle

Allocates memory and initializes a Flow entry handle to be used with all match APIs. Initializes 'match' as a "catch-all" match that matches every packet. The returned handle must eventually be committed, or freed.

> **Note**
>
> The distinction between DELETE and DELETE_STRICT, as well as between MODIFY and MODIFY_STRICT, can be found in the OpenFlow specification as follows: Modify and delete flow mod commands have non-strict versions (OFPFC_MODIFY and OFPFC_DELETE) and strict versions (OFPFC_MODIFY_STRICT or OFPFC_DELETE_STRICT). In the strict versions, the set of match fields, all match fields, including their masks, and the priority, are strictly matched against the entry, and only an identical flow entry is modified or removed. For example, if a message to remove entries is sent that has no match fields included, the OFPFC_DELETE command would delete all flow entries from the tables, while the OFPFC_DELETE_STRICT command would only delete a flow entry that applies to all packets at the specified priority
>
> The meanings of the flag values can be found in the OpenFlow specification.

Example (add a new flow entry to table 0 to match a specified IPv4 src address and send it out on the NORMAL virtual port):

```
struct ofp_sdn *nssdn = ofp_sdn_sys_init(argv[0]); assert(nsflow);
struct ofp_flow_entry *fm =
  ofp_flow_entry_create(nsflow, OFP_FLOW_TABLE_OVS_OPENFLOW, OFPFC_ADD, 0); assert(fm);
```

```
ofp_flow_entry_match_field(fm, OFPXMT_OFB_IPV4_SRC, &val);
struct ofp_actions *act = ofp_action_create(OFP_ACTION_SET);
ofp_action_output(act, OFPP_NORMAL);
ofp_flow_entry_instr_write_actions(fm, act);
ofp_flow_entry_commit(fm);
ofp_sdn_sys_shutdown(nsflow);
```

Example (delete all flow entries from table 0):

```
struct ofp_flow_entry *fm =
  ofp_flow_entry_create(nsflow, OFP_FLOW_TABLE_OVS_OPENFLOW, OFPFC_DELETE, 0); assert(fm);
ofp_flow_entry_commit(fm, 0);
```

Table 5.11. ofp_flow_entry_create parameters

Name	Description
nssdn	ofp_sdn API handle
ttype	enum ofp_flow_table_type (OFP_FLOW_TABLE_OVS_OPENFLOW, OFP_FLOW_TABLE_NFP_CACHE)
cmd	enum ofp_flow_mod_command (OFPFC_ADD, OFPFC_MODIFY, OFPFC_MODIFY_STRICT, OFPFC_DELETE, OFPFC_DELETE_STRICT)
flags	Combination of values from enum ofp_flow_mod_flags, ofp10_flow_mod_flags, ofp12_flow_mod_flags or ofp13_flow_mod_flags: OFPFF_SEND_FLOW_REM, OFPFF_CHECK_OVERLAP, OFPFF10_EMERG, OFPFF_RESET_COUNTS, OFPFF_NO_PKT_COUNTS, OFPFF_NO_BYT_COUNTS

Returns:

ofp_flow_entry handle

See Also:

- `ofp_sdn_sys_init()`
- `ofp_flow_entry_commit()`
- `ofp_flow_entry_free()`

5.1.4.6 ofp_flow_entry_free

Prototype:

```
void ofp_flow_entry_free(struct ofp_flow_entry* fe)
```

Description:

Free allocated memory for flow mod buffer. Used to destroy a flow handle which has not been committed.

Table 5.12. ofp_flow_entry_free parameters

Name	Description
fe	Flow entry handle

See Also:

- ofp_flow_entry_create()

5.1.4.7 ofp_flow_entry_match_field

Prototype:

```
int ofp_flow_entry_match_field(struct ofp_flow_entry* fe, enum oxm_ofb_match_fields
match_field, const union action_value* value)
```

Description:

Add match field to flow mod (without wildcard mask).

Table 5.13. ofp_flow_entry_match_field parameters

Name	Description
fe	Flow entry handle
match_field	Member of enum oxm_ofb_match_fields specifying field to match
value	Pointer to value of match field. All of the 16 and 32bit network packet fields must be specified in network byte order. Other fields such as metadata, cookie, port numbers must be specified in x86 host byte order.

Returns:

non-zero indicates failure

See Also:

- ofp_flow_entry_create()

5.1.4.8 ofp_flow_entry_match_field_mask

Prototype:

```
int ofp_flow_entry_match_field_mask(struct ofp_flow_entry* fe, enum
oxm_ofb_match_fields match_field, const union action_value* value, const union
action_value* mask_value)
```

Description:

Adds match field to flow mod (with wildcard mask).

Table 5.14. ofp_flow_entry_match_field_mask parameters

Name	Description
fe	Flow entry handle
match_field	Member of enum oxm_ofb_match_fields specifying field to match
value	Pointer to value of match field. All of the 16 and 32bit network packet fields must be specified in network byte order. Other fields such as metadata, cookie, port numbers must be specified in x86 host byte order.
mask_value	Pointer to mask value for match_field value

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.9 ofp_flow_entry_cookie

Prototype:

```
int ofp_flow_entry_cookie(struct ofp_flow_entry* fe, uint64_t cookie, uint64_t mask)
```

Description:

Populates Cookie field in flow entry

Note

Modify and delete commands can also be filtered by cookie value, if the cookie_mask field contains a value other than 0. This constraint is that the bits specified by the cookie_mask in both the cookie field of the flow mod and a flow entry's cookie value must be equal. In other words, (flow_entry.cookie & flow mod.cookie_mask) == (flow mod.cookie & flow mod.cookie_mask).

Table 5.15. ofp_flow_entry_cookie parameters

Name	Description
fe	Flow entry handle
cookie	Cookie value, in x86 host byte order
mask	Cookie mask, only relevant for modify and delete flow entries

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.10 ofp_flow_entry_xid

Prototype:

```
int ofp_flow_entry_xid(struct ofp_flow_entry* fe, uint32_t xid)
```

Description:

Set the transaction ID. This is an arbitrary number associated with a transaction. This number is included in any responses, such as error callback, to facilitate pairing. It is thus recommended to be set to a unique value per transaction. It defaults to 0 when a new ofp_sdn instance is created, and auto-increments with each transaction. This function resets the current value, which auto-increments again for subsequent transactions if set again.

Table 5.16. ofp_flow_entry_xid parameters

Name	Description
fe	Flow entry handle
xid	Transaction ID

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.11 ofp_flow_entry_hard_timeout

Prototype:

```
int ofp_flow_entry_hard_timeout(struct ofp_flow_entry* fe, uint16_t hard_timeout)
```

Description:

Populates Hard Timeout in flow entry (for temporary flow entries which can time out)

Table 5.17. ofp_flow_entry_hard_timeout parameters

Name	Description
fe	Flow entry handle
hard_timeout	Hard timeout Value

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.12 ofp_flow_entry_idle_timeout

Prototype:

```
int ofp_flow_entry_idle_timeout(struct ofp_flow_entry* fe, uint16_t idle_timeout)
```

Description:

Populates Idle Timeout in flow entry (for temporary flow entries which can time out)

Table 5.18. ofp_flow_entry_idle_timeout parameters

Name	Description
`fe`	Flow entry handle
`idle_timeout`	Idle timeout Value

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.13 ofp_flow_entry_priority

Prototype:

```
int ofp_flow_entry_priority(struct ofp_flow_entry* fe, uint16_t priority)
```

Description:

Populates Priority in flow entry

Table 5.19. ofp_flow_entry_priority parameters

Name	Description
`fe`	Flow entry handle
`priority`	Priority Value

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.14 ofp_flow_entry_out_port

Prototype:

```
int ofp_flow_entry_out_port(struct ofp_flow_entry* fe, uint32_t out_port)
```

Description:

Populates out_port in flow entry

"The out_port and out_group fields optionally filter the scope of OFPFC_DELETE and OFPFC_DELETE_STRICT messages by output port and group. If either out_port or out_group contains a value other than OFPP_ANY or OFPG_ANY respectively, it introduces a constraint when matching. This constraint is that the flow entry must contain an output action directed at that port or group. Other constraints such as ofp_match structs and priorities are still used; this is purely an additional constraint. Note that to disable output filtering, both out_port and out_group must be set to OFPP_ANY and OFPG_ANY respectively. These fields are ignored by OFPFC_ADD, OFPFC_MODIFY or OFPFC_MODIFY_STRICT messages."

Table 5.20. ofp_flow_entry_out_port parameters

Name	Description
fe	Flow entry handle
out_port	Output Port Number

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.15 ofp_flow_entry_out_group

Prototype:

```
int ofp_flow_entry_out_group(struct ofp_flow_entry* fe, uint32_t out_group)
```

Description:

Populates out_group in flow entry

"The out_port and out_group fields optionally filter the scope of OFPFC_DELETE and OFPFC_DELETE_STRICT messages by output port and group. If either out_port or out_group contains a value other than OFPP_ANY or OFPG_ANY respectively, it introduces a constraint when matching. This constraint is that the flow entry must contain an output action directed at that port or group. Other constraints such as ofp_match structs and priorities are still used; this is purely an additional constraint. Note that to disable output filtering, both out_port and out_group must be set to OFPP_ANY and OFPG_ANY respectively. These fields are ignored by OFPFC_ADD, OFPFC_MODIFY or OFPFC_MODIFY_STRICT messages."

Table 5.21. ofp_flow_entry_out_group parameters

Name	Description
fe	Flow entry handle
out_group	Output Group Number

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.16 ofp_flow_entry_table_num

Prototype:

```
int ofp_flow_entry_table_num(struct ofp_flow_entry* fe, uint8_t tablenum)
```

Description:

Populates table num in flow entry

Table 5.22. ofp_flow_entry_table_num parameters

Name	Description
fe	Flow entry handle
tablenum	Table number; OFPTT_ALL can be used when deleting flow entries to indicate that matching flow entries are to be deleted from all flow tables. Note that 254 is reserved for internal use by Open vSwitch.

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.17 ofp_flow_entry_instr_goto_table

Prototype:

```
int ofp_flow_entry_instr_goto_table(struct ofp_flow_entry* fe, uint8_t table_id)
```

Description:

Adds Goto Table instruction to flow entry

Table 5.23. ofp_flow_entry_instr_goto_table parameters

Name	Description
fe	Flow entry handle
table_id	Table ID value

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.18 ofp_flow_entry_instr_write_metadata

Prototype:

```
int ofp_flow_entry_instr_write_metadata(struct ofp_flow_entry* fe, uint64_t metadata,
uint64_t metadata_mask)
```

Description:

Adds Write Metadata instruction to flow entry (with metadata_mask value)

Table 5.24. ofp_flow_entry_instr_write_metadata parameters

Name	Description
fe	Flow entry handle
metadata	Metadata value, in x86 host byte order
metadata_mask	Metadata mask value, ((uint64_t)(-1)) to match all bits

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.19 ofp_flow_entry_instr_clear_actions

Prototype:

```
int ofp_flow_entry_instr_clear_actions(struct ofp_flow_entry* fe)
```

Description:

Adds Clear Actions instruction to flow entry, to clear the Action Set associated with the packet.

> **Note**
>
> In OpenFlow semantics, the Clear-Actions instruction will always be executed prior to the Write-Actions instruction.

Table 5.25. ofp_flow_entry_instr_clear_actions parameters

Name	Description
fe	Flow entry handle

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.20 ofp_flow_entry_instr_meter

Prototype:

```
int ofp_flow_entry_instr_meter(struct ofp_flow_entry* fe, uint32_t meter_id)
```

Description:

Adds Meter instruction to flow entry

Table 5.26. ofp_flow_entry_instr_meter parameters

Name	Description
fe	Flow entry handle
meter_id	Meter ID value, in x86 host byte order

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.21 ofp_flow_entry_instr_write_actions

Prototype:

```
int ofp_flow_entry_instr_write_actions(struct ofp_flow_entry* fe, struct ofp_actions*
act)
```

Description:

Add the Write-Actions instruction to the flow entry, which merges the specified action set into the current action set associated with the packet. The action set is specified by calls to ofp_action_* APIs while in the action set mode, the order in which those APIs are called is not significant (unless the same action is (incorrectly) specified more than once, in which case the last one will be the only one that takes effect).

> **Note**
>
> In accordance with OpenFlow semantics, if a user wishes to completely specify the action set instead of merging, the Clear-Actions instruction should be enabled on the flow entry in addition to the Write-Actions instruction. The Clear-Actions instruction will always be executed before the Write-Actions instruction.
>
> The action set accumulated on a packet after matching (possibly more than one) flow entries in flow tables is only executed when the final matching flow entry does not contain a Goto-Table instruction. The order of execution of these actions is determined by the OpenFlow specification. Contrast this with the Apply-Actions instruction, where an ordered action list is executed as soon as a packet matches a flow entry.

Table 5.27. ofp_flow_entry_instr_write_actions parameters

Name	Description
fe	Flow entry handle
act	Actions handle, ownership transfers to this API (caller must not access or free)

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.1.4.22 ofp_flow_entry_instr_apply_actions

Prototype:

```
int ofp_flow_entry_instr_apply_actions(struct ofp_flow_entry* fe, struct ofp_actions*
act)
```

Description:

Add the Apply-Actions instruction to the flow entry. The action list is populated by calls to ofp_action_* APIs while in the action list mode, and sequential order of those API calls is significant.

Note

When a packet matches a flow entry with an Apply-Actions instruction, the ordered action list is executed immediately. Contrast this with the Write-Actions instruction, which only executes actions after the last matching flow entry does not have a Goto-Table instruction.

Table 5.28. ofp_flow_entry_instr_apply_actions parameters

Name	Description
fe	Flow entry handle
act	Actions handle, ownership transfers to this API (caller must not access or free)

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.1.4.23 ofp_flow_entry_commit

Prototype:

```
int ofp_flow_entry_commit(struct ofp_flow_entry* fe)
```

Description:

Commit the flow entry to the specified table number.

Table 5.29. ofp_flow_entry_commit parameters

Name	Description
fe	Flow entry handle

Returns:

non-zero indicates failure

See Also:

- `ofp_flow_entry_create()`

5.1.4.24 ofp_flow_entry_stats

Prototype:

```
int ofp_flow_entry_stats(struct ofp_flow_entry* fe, uint64_t* bytes, uint64_t* packets,
uint32_t* flows)
```

Description:

Request flow statistics. The total bytes, packets and number of flows matching the criteria in fe are returned. By default table_id is OFPTT_ALL, out_port is OFPP_ANY and out_group is OFPG_ANY. Only the match fields in fe are taken into consideration, any instructions+actions that may have been specified are ignored.

Table 5.30. ofp_flow_entry_stats parameters

Name	Description
fe	Flow entry handle, must be created with the special STATS_FAKE_CMD command type
bytes	Pointer to variable in which resulting bytes are stored
packets	Pointer to variable in which resulting packets are stored
flows	Pointer to variable in which resulting flows are stored

Returns:

non-zero indicates failure

5.2 ofp_group

5.2.1 Enumerations

5.2.1.1 ofp_group_mod_command

Table 5.31. enum ofp_group_mod_command

Name	Description
OFPGC_ADD	
OFPGC_MODIFY	
OFPGC_DELETE	

5.2.1.2 ofp_group_type

Table 5.32. enum ofp_group_type

Name	Description
OFPGT_ALL	
OFPGT_SELECT	
OFPGT_INDIRECT	

Name	Description
`OFPGT_FF`	

5.2.2 Functions

5.2.2.1 ofp_group_create

Prototype:

```
struct ofp_group* ofp_group_create(struct ofp_sdn* nssdn, enum ofp_group_mod_command
cmd, enum ofp_group_type typ, uint32_t id)
```

Description:

Create a new group modifier. The returned handle must eventually be committed, or freed.

Table 5.33. ofp_group_create parameters

Name	Description
nssdn	
cmd	Add, modify or delete.
typ	Group type.
id	The group identifier.

Returns:

struct ofp_group* handle to create group modifier

5.2.2.2 ofp_group_add_action_bucket

Prototype:

```
int ofp_group_add_action_bucket(struct ofp_group* grp, uint16_t weight, uint32_t
watch_port, uint32_t watch_group, struct ofp_actions* actions)
```

Description:

Add a new bucket to the list, with specified weight and action set.

Table 5.34. ofp_group_add_action_bucket parameters

Name	Description
grp	Group modifier handle.

Name	Description
weight	Relative weight of bucket.
watch_port	Port whose state affects this bucket (only for fast failover groups).
watch_group	Group whose state affects this bucket (only for fast failover groups).
actions	Action set created by ofp_action APIs.

Returns:

int Non-zero on failure.

5.2.2.3 ofp_group_commit

Prototype:

```
int ofp_group_commit(struct ofp_group* grp)
```

Description:

Commit the group modifier.

Table 5.35. ofp_group_commit parameters

Name	Description
grp	Group modifier handle.

Returns:

int Non-zero on failure.

5.2.2.4 ofp_group_free

Prototype:

```
int ofp_group_free(struct ofp_group* grp)
```

Description:

Free the group modifier. Used only on group handles that haven't been committed.

Table 5.36. ofp_group_free parameters

Name	Description
grp	Group modifier handle.

Returns:

int Non-zero on failure.

5.3 ofp_action

5.3.1 Defines

Table 5.37. ofp_action.h Defines

Defined	Definition
ETH_ADDR_LEN	6

5.3.2 Enumerations

5.3.2.1 ofp_action_mode

Table 5.38. enum ofp_action_mode

Name	Description
OFP_ACTION_SET	Add subsequent actions to an Action Set, as used by the Write-Actions instruction.
OFP_ACTION_LIST	Add subsequent actions to an Action List, as used by the Apply-Actions instruction.

5.3.2.2 oxm_ofb_match_fields

Table 5.39. enum oxm_ofb_match_fields

Name	Description
OFPXMT_OFB_IN_PORT	
OFPXMT_OFB_IN_PHY_PORT	
OFPXMT_OFB_METADATA	
OFPXMT_OFB_ETH_DST	
OFPXMT_OFB_ETH_SRC	
OFPXMT_OFB_ETH_TYPE	
OFPXMT_OFB_VLAN_VID	
OFPXMT_OFB_VLAN_PCP	
OFPXMT_OFB_IP_DSCP	
OFPXMT_OFB_IP_ECN	
OFPXMT_OFB_IP_PROTO	

Name	Description
OFPXMT_OFB_IPV4_SRC	
OFPXMT_OFB_IPV4_DST	
OFPXMT_OFB_TCP_SRC	
OFPXMT_OFB_TCP_DST	
OFPXMT_OFB_UDP_SRC	
OFPXMT_OFB_UDP_DST	
OFPXMT_OFB_SCTP_SRC	
OFPXMT_OFB_SCTP_DST	
OFPXMT_OFB_ICMPV4_TYPE	
OFPXMT_OFB_ICMPV4_CODE	
OFPXMT_OFB_ARP_OP	
OFPXMT_OFB_ARP_SPA	
OFPXMT_OFB_ARP_TPA	
OFPXMT_OFB_ARP_SHA	
OFPXMT_OFB_ARP_THA	
OFPXMT_OFB_IPV6_SRC	
OFPXMT_OFB_IPV6_DST	
OFPXMT_OFB_IPV6_FLABEL	
OFPXMT_OFB_ICMPV6_TYPE	
OFPXMT_OFB_ICMPV6_CODE	
OFPXMT_OFB_IPV6_ND_TARGET	
OFPXMT_OFB_IPV6_ND_SLL	
OFPXMT_OFB_IPV6_ND_TLL	
OFPXMT_OFB_MPLS_LABEL	
OFPXMT_OFB_MPLS_TC	
OFPXMT_OFB_MPLS_BOS	
OFPXMT_OFB_PBB_ISID	
OFPXMT_OFB_TUNNEL_ID	
OFPXMT_OFB_IPV6_EXTHDR	

5.3.3 Functions

5.3.3.1 ofp_action_create

Prototype:

```
struct ofp_actions* ofp_action_create(enum ofp_action_mode a)
```

Description:

Create an action handle for use with ofp_flow_entry or ofp_group APIs. The supplied mode is used to distinguish between adding actions to a set or an ordered list.

Table 5.40. ofp_action_create parameters

Name	Description
a	ofp_action_mode enum

See Also:

- `ofp_flow_entry_instr_write_actions()`
- `ofp_flow_entry_instr_apply_actions()`
- ofp_group_mod_add_action_bucket()

5.3.3.2 ofp_action_free

Prototype:

```
void ofp_action_free(struct ofp_actions* act)
```

Description:

Free allocated memory for actions buffer. Only necessary if the action handle has not had ownership transferred to ofp_flow_entry or ofp_group API instances.

Table 5.41. ofp_action_free parameters

Name	Description
act	Action handle

See Also:

- ofp_action_create()

5.3.3.3 ofp_action_output

Prototype:

```
int ofp_action_output(struct ofp_actions* act, uint32_t output_port)
```

Description:

Adds Output action. The outport port value can also include the special OpenFlow values: OFPP_IN_PORT Send the packet back out the input port. OFPP_NORMAL Permit normal forwarding to select the output port. OFPP_CONTROLLER Send to controller.

Table 5.42. ofp_action_output parameters

Name	Description
act	Action handle
output_port	Output port value in host byte order (x86 little endian)

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.3.3.4 ofp_action_group

Prototype:

```
int ofp_action_group(struct ofp_actions* act, uint32_t group_id)
```

Description:

Adds Group action.

Table 5.43. ofp_action_group parameters

Name	Description
act	Action handle
group_id	Group ID value in host byte order (x86 little endian)

Returns:

non-zero indicates failure

See Also:

- ofp_action_create(), `ofp_group_create()`

5.3.3.5 ofp_action_mpls_ttl

Prototype:

```
int ofp_action_mpls_ttl(struct ofp_actions* act, uint8_t mpls_ttl)
```

Description:

Add Set MPLS TTL action

Table 5.44. ofp_action_mpls_ttl parameters

Name	Description
act	Action handle
mpls_ttl	mpls_ttl value

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.3.3.6 ofp_action_dec_mpls_ttl

Prototype:

```
int ofp_action_dec_mpls_ttl(struct ofp_actions* act)
```

Description:

Add Decrement MPLS TTL action

Table 5.45. ofp_action_dec_mpls_ttl parameters

Name	Description
act	Action handle

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.3.3.7 ofp_action_ip_ttl

Prototype:

```
int ofp_action_ip_ttl(struct ofp_actions* act, uint8_t ip_ttl)
```

Description:

Add Set IP TTL action

Table 5.46. ofp_action_ip_ttl parameters

Name	Description
act	Action handle

Name	Description
ip_ttl	ip_ttl value

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.3.3.8 ofp_action_dec_ip_ttl

Prototype:

```
int ofp_action_dec_ip_ttl(struct ofp_actions* act)
```

Description:

Add Decrement IP TTL action

Table 5.47. ofp_action_dec_ip_ttl parameters

Name	Description
act	Action handle

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.3.3.9 ofp_action_push_mpls

Prototype:

```
int ofp_action_push_mpls(struct ofp_actions* act, uint16_t ethertype)
```

Description:

Add Push MPLS action

Table 5.48. ofp_action_push_mpls parameters

Name	Description
act	Action handle

Name	Description
ethertype	ethertype value, in network byte order

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.3.3.10 ofp_action_pop_mpls

Prototype:

```
int ofp_action_pop_mpls(struct ofp_actions* act)
```

Description:

Add Pop MPLS action

Table 5.49. ofp_action_pop_mpls parameters

Name	Description
act	Action handle

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.3.3.11 ofp_action_set_queue

Prototype:

```
int ofp_action_set_queue(struct ofp_actions* act, uint32_t queue_id)
```

Description:

Add Set Queue action

Table 5.50. ofp_action_set_queue parameters

Name	Description
act	Action handle

Name	Description
queue_id	queue_id value

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.3.3.12 ofp_action_push_vlan

Prototype:

```
int ofp_action_push_vlan(struct ofp_actions* act, uint16_t ethertype)
```

Description:

Adds Push VLAN action

Table 5.51. ofp_action_push_vlan parameters

Name	Description
act	Action handle
ethertype	ethertype value in network byte order (only htons(0x8100) is currently supported)

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.3.3.13 ofp_action_pop_vlan

Prototype:

```
int ofp_action_pop_vlan(struct ofp_actions* act)
```

Description:

Adds Pop VLAN action

Table 5.52. ofp_action_pop_vlan parameters

Name	Description
act	Action handle

Returns:

non-zero indicates failure

See Also:

- ofp_action_create()

5.3.3.14 ofp_action_set_field

Prototype:

```
int ofp_action_set_field(struct ofp_actions* act, enum oxm_ofb_match_fields field,
union action_value* value)
```

Description:

Add a set field action

Table 5.53. ofp_action_set_field parameters

Name	Description
act	Action handle
field	Member of enum oxm12_ofb_match_fields specifying field to set
value	Pointer to value of match field. All of the 16 and 32bit network packet fields must be specified in network byte order. Other fields such as metadata, cookie, port numbers must be specified in x86 host byte order.

Returns:

non-zero indicates failure

5.4 ns_sdn_health

5.4.1 Defines

Table 5.54. ns_sdn_health.h Defines

Defined	Definition
NS_SDN_HEALTH_FLAG_ENABLE_COLOR_OUTPUT	2 Initialization flag: If specified, all text report generated by the library will be color coded for display purposes.
NS_SDN_HEALTH_FLAG_ENABLE_SYSLOG	1

Defined	Definition
	Initialization flag: If specified, all failures will be logged via syslog when 'ns_sdn_health_check' is called. Syslog messages are generated with a syslog id of "ns_sdn_health" and logged to the LOCAL7 facility.
`NS_SDN_HEALTH_FLAG_FAIL_ON_WARN`	`4` Initialization flag: If specified, warnings are treated as failures.
`NS_SDN_HEALTH_OPTIONAL_DISABLE_ALL`	`(0)`
`NS_SDN_HEALTH_OPTIONAL_ENABLE_ALL`	`(~(UINT32_MAX << NS_SDN_HEALTH_OPTIONAL_MAX))`
`NS_SDN_HEALTH_PID_ABS_FILE`	`NS_SDN_HEALTH_PID_PATH"/"NS_SDN_HEALTH_PID_FILE`
`NS_SDN_HEALTH_PID_FILE`	`"lock.pid"`
`NS_SDN_HEALTH_PID_PATH`	`"/tmp/ovs.lock.v2.2"`

5.4.2 Enumerations

5.4.2.1 ns_sdn_health_failure_code_t

Table 5.55. enum ns_sdn_health_failure_code_t

Name	Description
`NS_SDN_HEALTH_NO_FAILURE`	No failure.
`NS_SDN_HEALTH_OVSDB_SERVER_FAILED`	OVSDB server process failed.
`NS_SDN_HEALTH_OVS_VSWITCHD_FAILED`	OVS VSWITCHD process failed.
`NS_SDN_HEALTH_VIRTIORELAYD_FAILED`	Virtiorelayd process failed.
`NS_SDN_HEALTH_NFP_MODULE_NOT_LOADED`	Missing NFP kernel module.
`NS_SDN_HEALTH_CMSG_MODULE_NOT_LOADED`	Missing NFP control message module.
`NS_SDN_HEALTH_FALLBACK_MODULE_NOT_LOADED`	Missing NFP Fallback module.
`NS_SDN_HEALTH_OFFLOADS_MODULE_NOT_LOADED`	Missing NFP Offload module.
`NS_SDN_HEALTH_OVS_MODULE_NOT_LOADED`	Missing Open vSwitch kernel module.
`NS_SDN_HEALTH_FIRMWARE_NOT_LOADED`	Firmware not loaded.
`NS_SDN_HEALTH_NFP_FPC_UNRESPONSIVE`	Flow processing cores are unresponsive.
`NS_SDN_HEALTH_NFP_CTRL_CHAN_UNRESPONSIVE`	NFP Control channel is unresponsive.
`NS_SDN_HEALTH_TRAFFIC_DIVERTED`	Traffic diverted to the fallback channel.
`NS_SDN_HEALTH_NFP_INGRESS_NBI_BACKED_UP`	Ingress NBI processing cores are possibly locked up.
`NS_SDN_HEALTH_NFP_NOT_DETECTED`	NFP not detected.
`NS_SDN_HEALTH_ECC_ERRORS_DETECTED`	ECC hardware errors detected.
`NS_SDN_HEALTH_PARITY_ERRORS_DETECTED`	Parity hardware errors detected.

Name	Description
`NS_SDN_HEALTH_CONFIGURATOR_CHECK`	Check for up to date configurator.
`NS_SDN_HEALTH_ERR47_WORKAROUND`	ERR47 workaround not detected.
`NS_SDN_HEALTH_MAX`	Marker to indicate maximum number of enums.

5.4.2.2 ns_sdn_health_optional_checks_t

Table 5.56. enum ns_sdn_health_optional_checks_t

Name	Description
`NS_SDN_HEALTH_OPTIONAL_VIRTIORELAYD`	Virtiorelayd process check.
`NS_SDN_HEALTH_OPTIONAL_MAX`	Marker to indicate maximum number of enums.

5.4.3 Typedefs

5.4.3.1 ns_sdn_health_failure_t

Table 5.57. typedef ns_sdn_health_failure_t

Type	Definition
`ns_sdn_health_failure_t`	`struct ns_sdn_health_failure*`

5.4.4 Functions

5.4.4.1 ns_sdn_health_init

Prototype:

```
ns_sdn_health_handle_t ns_sdn_health_init(unsigned int flags, unsigned int
enabled_optional_checks)
```

Description:

Initialize a new instance of the SDN health check API.

Table 5.58. ns_sdn_health_init parameters

Name	Description
`flags`	Bitmap of initialization flags

Name	Description
enabled_optional_checks	Bitmap of optional checks which are enabled. Bitmap must be created with a '1' value bit shifted with the corresponding enum value in the 'ns_sdn_health_optional_checks_t' enum.

Returns:

New instance. Returns NULL if function call fails. In case of failure, sets ERRNO value appropriately. ERRNO value of 'EBUSY' indicates the existence of the PID file /tmp/sdn_health.pid. The memory allocated needs to be freed with a call to 'ns_sdn_health_destroy' once done.

5.4.4.2 ns_sdn_health_destroy

Prototype:

```
void ns_sdn_health_destroy(ns_sdn_health_handle_t handle)
```

Description:

Destroys an instance of the SDN health check API.

Table 5.59. ns_sdn_health_destroy parameters

Name	Description
handle	Health check handle

5.4.4.3 ns_sdn_health_check

Prototype:

```
int ns_sdn_health_check(ns_sdn_health_handle_t handle)
```

Description:

Executes the health check on the SDN 2.2 system. This function should be executed periodically or whenever the current system health needs to be determined.

Previous health failures will be cleared with each call to this function.

Table 5.60. ns_sdn_health_check parameters

Name	Description
handle	Health check handle

Returns:

The number of failures detected. Returns -1 when unable to perform health check due to errors. Warnings are not counted unless the 'NS_SDN_HEALTH_FLAG_FAIL_ON_WARN' flag is specified.

5.4.4.4 ns_sdn_health_failure_count

Prototype:

```
int ns_sdn_health_failure_count(ns_sdn_health_handle_t handle)
```

Description:

Returns the number of failures detected during the last execution of 'ns_sdn_health_check'.

Table 5.61. ns_sdn_health_failure_count parameters

Name	Description
handle	Health check handle

Returns:

The number of failures detected. Will return 0 if 'ns_sdn_health_check' has not been executed yet. Warnings are not counted unless the 'NS_SDN_HEALTH_FLAG_FAIL_ON_WARN' flag is specified. Returns -1 when unable to perform health check due to errors.

5.4.4.5 ns_sdn_health_warning_count

Prototype:

```
int ns_sdn_health_warning_count(ns_sdn_health_handle_t handle)
```

Description:

Returns the number of warnings detected during the last execution of 'ns_sdn_health_check'.

Table 5.62. ns_sdn_health_warning_count parameters

Name	Description
handle	Health check handle

Returns:

The number of warnings detected. Will return 0 if 'ns_sdn_health_check' has not been executed yet. Returns -1 when unable to perform health check due to errors.

5.4.4.6 ns_sdn_health_next_failure

Prototype:

```
int ns_sdn_health_next_failure(ns_sdn_health_handle_t handle, ns_sdn_health_failure_t*
fail, ns_sdn_health_failure_code_t* failcode, const char** failstr, const char**
affected_component, const char** recommended_action)
```

Description:

Returns the failure structure of the next available health check failure. The number of health check failures available is determined from the call to 'ns_sdn_health_check'.

Table 5.63. ns_sdn_health_next_failure parameters

Name	Description
handle	Health check handle
fail	Output pointer to failure structure. Can be NULL if the caller doesn't want to receive this data. Caller must not free this pointer. Library sets this to NULL if there are no more failures available.
failcode	Output pointer to the failure code. Can be NULL if the caller doesn't want to receive this data.
failstr	Output pointer to failure string. Can be NULL if the caller doesn't want to receive this data. Caller must not free this pointer. Library sets this to NULL if there are no more failures available.
affected_component	Output pointer to string indicating affected component. Can be NULL if the caller doesn't want to receive this data. Caller must not free this pointer. Library sets this to NULL if there are no more failures available.
recommended_action	Output pointer to string indicating recommended action. Can be NULL if the caller doesn't want to receive this data. Caller must not free this pointer. Library sets this to NULL if there are no more failures available.

Returns:

0 on success, 1 on no more failures and -1 on error

5.4.4.7 ns_sdn_health_generate_report

Prototype:

```
int ns_sdn_health_generate_report(ns_sdn_health_handle_t handle, char* buffer, size_t
buffer_size, int verbose)
```

Description:

Generate a health report.

Table 5.64. ns_sdn_health_generate_report parameters

Name	Description
handle	Health check handle
buffer	User provided buffer where report will be written to.
buffer_size	Size of the user provided buffer. Library will not write more than this many bytes to the buffer.

Name	Description
verbose	If non-zero writes details of which subcomponents failed if a health check failed, e.g. register names and values are written to the buffer.

Returns:

Returns positive value with amount of bytes written to buffer or -1 on error.

6. Technical Support

To obtain additional information, or to provide feedback, please email `<support@netronome.com>` or contact the nearest **Netronome** technical support representative.

6.1 Related Documents

Descriptive Name	Description
Netronome Network Flow Processor: Agilio OvS 2.2 Getting Started Guide	A guide to new users of Netronome's Agilio OvS software.
Netronome Network Flow Processor: Agilio OvS 2.2 Programmer's Reference Manual	A reference for programming and system design using the Agilio OvS software.
Intelligent Server Adapters: Hardware User Manual	Contains summary information on the Netronome Intelligent Server Adpater (ISA) PCIe card including card physical descriptions.
Netronome Network Flow Processor: Datasheet	Contains summary information on the Netronome Network Flow Processor NFP including a functional description, signal descriptions, electrical specifications, and mechanical specifications.
Netronome Network Flow Processor: Databook	Contains detailed reference information on the Netronome Network Flow Processor NFP.
Netronome Network Flow Processor: Development Tools User's Guide	Describes Programmer Studio and the development tools that can be accessed through Programmer Studio.
Netronome Network Flow Processor: Network Flow Assembler System User's Guide	Describes the syntax of the NFP's assembly language, supplies assembler usage information, and lists assembler warnings and errors.
Netronome Network Flow Processor: Microengine Programmer's Reference Manual	A reference for microcode programming of the Netronome Network Flow Processor NFP.
Netronome Network Flow Processor: Network Flow C Compiler User's Guide	Presents information, language structures and extensions to the language specific to the Netronome Network Flow C Compiler for Netronome NFP.
Netronome Network Flow Compiler LibC: Reference Manual	Specifies the subset and the extensions to the language that support the unique features of the Netronome Network Flow Processor NFP product line.
Open vSwitch Software Documentation	Agilio OvS software offers acceleration of Open vSwitch software. Refer to http://openvswitch.org/ for more details on Open vSwitch.
OpenFlow Specification	Open vSwitch (which is accelerated by Agilio OvS software) is an OpenFlow switch implementation. Refer to https://www.opennetworking.org/sdn-

Descriptive Name	Description
	resources/openflow for more details on this specification.
Data Plane Development Kit Documentation	DPDK related documentation is available at http://dpdk.org.