

# FNA Reference Manual

Generated by Doxygen 1.4.6

Tue Mar 13 18:57:27 2007



# Contents

<b>1</b>	<b>FNA Module Index</b>	<b>1</b>
1.1	FNA Modules . . . . .	1
<b>2</b>	<b>FNA File Index</b>	<b>3</b>
2.1	FNA File List . . . . .	3
<b>3</b>	<b>FNA Module Documentation</b>	<b>5</b>
3.1	FNA Private Interface . . . . .	5
3.2	FNA Initialization . . . . .	6
3.3	FNA Virtual Resources . . . . .	7
3.4	FNA Spare Capacity . . . . .	13
3.5	FNA Send and Receive . . . . .	15
3.6	FNA Network Configuration . . . . .	19
3.7	FNA Public Interface . . . . .	21
3.8	FNA Mapping Functions . . . . .	22
3.9	FNA Negotiation Service Parameters . . . . .	24
<b>4</b>	<b>FNA File Documentation</b>	<b>27</b>
4.1	fna.h File Reference . . . . .	27
4.2	frsh_fna.h File Reference . . . . .	29



# Chapter 1

## FNA Module Index

### 1.1 FNA Modules

Here is a list of all modules:

FNA Private Interface . . . . .	5
FNA Initialization . . . . .	6
FNA Virtual Resources . . . . .	7
FNA Spare Capacity . . . . .	13
FNA Send and Receive . . . . .	15
FNA Network Configuration . . . . .	19
FNA Public Interface . . . . .	21
FNA Mapping Functions . . . . .	22
FNA Negotiation Service Parameters . . . . .	24



# Chapter 2

## FNA File Index

### 2.1 FNA File List

Here is a list of all files with brief descriptions:

<a href="#">fna.h</a> . . . . .	27
<a href="#">frsh_fna.h</a> . . . . .	29





# Chapter 3

## FNA Module Documentation

### 3.1 FNA Private Interface

#### Modules

- [FNA Initialization](#)
- [FNA Virtual Resources](#)
- [FNA Spare Capacity](#)
- [FNA Send and Receive](#)
- [FNA Network Configuration](#)

#### 3.1.1 Detailed Description

FNA is a Network adaption layer that allows to plugin new network protocols to the distributed module.

It is divided in two parts:

- FRSH\_FNA: public types and functions for the FRSH API
- FNA: private functions only used within FRSH.

## 3.2 FNA Initialization

### Functions

- int [fna\\_init](#) (const frsh\_resource\_id\_t resource\_id)

#### 3.2.1 Detailed Description

These functions need to be called before using any network

#### 3.2.2 Function Documentation

##### 3.2.2.1 int [fna\\_init](#) (const frsh\_resource\_id\_t *resource\_id*)

###### [fna\\_init](#)()

This function will be hooked to the frsh\_init function and it is intended to initialize the protocol and its structures.

###### Parameters:

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)

###### Returns:

- FNA\_NO\_ERROR
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_ALREADY\_INITIALIZED: if the function has already been called before (with success)

## 3.3 FNA Virtual Resources

### Typedefs

- typedef void \* [fna\\_vres\\_id\\_t](#)

### Functions

- int [fna\\_vres\\_create](#) (const frsh\_resource\_id\_t resource\_id, const frsh\_contract\_t \*contract, [fna\\_vres\\_id\\_t](#) \*vres)
- int [fna\\_vres\\_renegotiate\\_sync](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, const frsh\_contract\_t \*new\_contract)
- int [fna\\_vres\\_renegotiate\\_async](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, const frsh\_contract\_t \*new\_contract, frsh\_signal\_t signal\_to\_notify, frsh\_signal\_info\_t signal\_info)
- int [fna\\_vres\\_get\\_renegotiation\\_status](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, frsh\_renegotiation\_status\_t \*renegotiation\_status)
- int [fna\\_vres\\_destroy](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres)
- int [fna\\_vres\\_get\\_contract](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, frsh\_contract\_t \*contract)
- int [fna\\_vres\\_get\\_usage](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, struct timespec \*usage)
- int [fna\\_vres\\_get\\_remaining\\_budget](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, struct timespec \*remaining\_budget)
- int [fna\\_vres\\_get\\_budget\\_and\\_period](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, struct timespec \*budget, struct timespec \*period)

### 3.3.1 Detailed Description

The following functions are used to negotiate, renegotiate and cancel virtual network resources.

### 3.3.2 Typedef Documentation

#### 3.3.2.1 typedef void\* [fna\\_vres\\_id\\_t](#)

[fna\\_vres\\_id\\_t](#)

Internal virtual resource id. The type [fna\\_vres\\_id\\_t](#) is a pointer to void. The FRSB layer will keep a map between the [frsh\\_vres\\_id\\_t](#), this pointer and the [resource\\_id](#). This pointer could be used as the ID itself using casting, or as internal pointer to any structure.

Definition at line 148 of file [fna.h](#).

### 3.3.3 Function Documentation

#### 3.3.3.1 int [fna\\_vres\\_create](#) (const [frsh\\_resource\\_id\\_t](#) *resource\_id*, const [frsh\\_contract\\_t](#) \* *contract*, [fna\\_vres\\_id\\_t](#) \* *vres*)

[fna\\_vres\\_create\(\)](#)

The operation negotiates a contract and if accepted it will return a [fna\\_vres\\_id\\_t](#). It will also check that the given [contract\\_id](#) is unique within the network.

In object oriented terminology it is similar to a constructor of the virtual resource class.

If the on-line admission test is enabled, it determines whether the contract can be admitted or not based on the current contracts established in the network. Then it creates the vres and recalculates all necessary parameters for the contracts already present in the system.

This is a potentially blocking operation, it returns when the system has either rejected the contract, or admitted it and made it effective.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *contract* The contract parameters to negotiate
- *vres* The internal virtual resource id

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_TOO\_MANY\_VRES: if there is no space for more vres
- FNA\_ERR\_CONTRACT\_ID\_ALREADY\_EXISTS: contract\_id is not unique
- FNA\_ERR\_CONTRACT\_REJECTED: if the contract is not accepted
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

**3.3.3.2 int fna\_vres\_destroy (const frsh\_resource\_id\_t resource\_id, const fna\_vres\_id\_t vres)**

[fna\\_vres\\_destroy\(\)](#)

The operation eliminates the specified vres and recalculates all necessary parameters for the contracts remaining in the system. This is a potentially blocking operation; it returns when the system has made the changes effective.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *vres* The internal virtual resource id to destroy

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_CONTRACTED\_VRES: if the vres is not contracted
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

**3.3.3.3 int fna\_vres\_get\_budget\_and\_period (const frsh\_resource\_id\_t resource\_id, const fna\_vres\_id\_t vres, struct timespec \* budget, struct timespec \* period)**

[fna\\_vres\\_get\\_budget\\_and\\_period\(\)](#)

This function gets the budget and period associated with the specified vres for each period. If one of these pointers is NULL, the corresponding information is not stored.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *vres* The internal virtual resource id
- *budget* The budget associated to vres
- *period* The period associated to vres

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_CONTRACTED\_VRES: if the vres is not contracted
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if both pointers are NULL

### 3.3.3.4 int fna\_vres\_get\_contract (const frsh\_resource\_id\_t resource\_id, const fna\_vres\_id\_t vres, frsh\_contract\_t \* contract)

#### [fna\\_vres\\_get\\_contract\(\)](#)

This operation stores the contract parameters currently associated with the specified vres in the variable pointed to by contract. It returns an error if the vres\_id is not recognised.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *vres* The internal virtual resource id
- *contract* The contract parameters that we want

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_CONTRACTED\_VRES: if the vres is not contracted
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

### 3.3.3.5 int fna\_vres\_get\_remaining\_budget (const frsh\_resource\_id\_t resource\_id, const fna\_vres\_id\_t vres, struct timespec \* remaining\_budget)

#### [fna\\_vres\\_get\\_remaining\\_budget\(\)](#)

This function stores in the variable pointed to by budget the remaining execution-time budget associated with the specified vres in the present period.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *vres* The internal virtual resource id
- *remaining\_budget* The remaining budget for this period

**Returns:**

FNA\_NO\_ERROR: in this case it also means contract accepted  
 FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors  
 FNA\_ERR\_NOT\_CONTRACTED\_VRES: if the vres is not contracted  
 FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized  
 FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id  
 FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

### 3.3.3.6 `int fna_vres_get_renegotiation_status (const frsh_resource_id_t resource_id, const fna_vres_id_t vres, frsh_renegotiation_status_t *renegotiation_status)`

#### `fna_vres_get_renegotiation_status()`

The operation reports on the status of the last renegotiation operation enqueued for the specified vres. It is callable even after notification of the completion of such operation, if requested.

If the vres is not and has not been involved in any of the `frsh_contract_renegotiate_async()` or `frsh_group_change_mode_async()` operations, the status returned is FNA\_NOT\_REQUESTED

**Parameters:**

← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)  
 ← *vres* The internal virtual resource id we want the status from  
 ← *renegotiation\_status* The status of the last renegotiation on vres (FRSH\_RS\_IN\_PROGRESS, FRSH\_RS\_REJECTED, FRSH\_RS\_ADMITTED, FRSH\_RS\_NOT\_REQUESTED)

**Returns:**

FNA\_NO\_ERROR: in this case it also means contract accepted  
 FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors  
 FNA\_ERR\_NOT\_CONTRACTED\_VRES: if the vres is not contracted  
 FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized  
 FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id  
 FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

### 3.3.3.7 `int fna_vres_get_usage (const frsh_resource_id_t resource_id, const fna_vres_id_t vres, struct timespec *usage)`

#### `fna_vres_get_usage()`

This function gets the execution time spent by all messages that have been sent through the specified vres.

**Parameters:**

← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)  
 ← *vres* The internal virtual resource id  
 → *usage* Execution time spent by this vres

**Returns:**

FNA\_NO\_ERROR: in this case it also means contract accepted  
 FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors  
 FNA\_ERR\_NOT\_CONTRACTED\_VRES: if the vres is not contracted

FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized  
 FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id  
 FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

**3.3.3.8** `int fna_vres_renegotiate_async (const frsh_resource_id_t resource_id, const fna_vres_id_t vres, const frsh_contract_t * new_contract, frsh_signal_t signal_to_notify, frsh_signal_info_t signal_info)`

[fna\\_vres\\_renegotiate\\_async\(\)](#)

The operation enqueues a renegotiate operation for an existing vres, and returns immediately. The renegotiate operation is performed asynchronously, as soon as it is practical; meanwhile the system operation will continue normally. When the renegotiation is made, if the on-line admission test is enabled it determines whether the contract can be admitted or not based on the current contracts established in the system. If it cannot be admitted, the old contract remains in effect. If it can be admitted, it recalculates all necessary parameters for the contracts already present in the system.

When the operation is completed, notification is made to the caller, if requested, via a signal. The status of the operation (in progress, admitted, rejected) can be checked with the `frsh_vres_get_renegotiation_status()` operation. The argument `sig_notify` can be `FRSH_NULL_SIGNAL` (no notification), or any `FRSH` signal value and in this case `signal_info` is to be sent with the signal.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *vres* The internal virtual resource id to renegotiate
- ← *new\_contract* The new contract
- ← *signal\_to\_notify* Signal number to use to notify vres of the negotiation result. If `FRSH_NULL_SIGNAL`, no signal will be raised.
- ← *signal\_info*: Associated info that will come with the signal. This parameter will be ignored if `signal_to_notify == FRSH_NULL_SIGNAL`.

**Returns:**

FNA\_NO\_ERROR: in this case it also means contract accepted  
 FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors  
 FNA\_ERR\_NOT\_CONTRACTED\_VRES: if the vres is not contracted  
 FNA\_ERR\_CONTRACT\_ID\_ALREADY\_EXISTS: contract\_id is not unique  
 FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized  
 FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id  
 FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL, or sig\_notify is neither NULL nor a valid POSIX signal

**3.3.3.9** `int fna_vres_renegotiate_sync (const frsh_resource_id_t resource_id, const fna_vres_id_t vres, const frsh_contract_t * new_contract)`

[fna\\_vres\\_renegotiate\\_sync\(\)](#)

The operation renegotiates a contract for an existing vres. If the on-line admission test is enabled it determines whether the contract can be admitted or not based on the current contracts established in the system. If it cannot be admitted, the old contract remains in effect and an error is returned. If it can be admitted, it recalculates all necessary parameters for the contracts already present in the system and returns zero.

This is a potentially blocking operation; it returns when the system has either rejected the new contract, or admitted it and made it effective.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *vres* The internal virtual resource id to renegotiate
- ← *new\_contract* The new contract

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_CONTRACTED\_VRES: if the vres is not contracted
- FNA\_ERR\_CONTRACT\_ID\_ALREADY\_EXISTS: contract\_id is not unique
- FNA\_ERR\_CONTRACT\_REJECTED: if the contract is not accepted
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL



## 3.4 FNA Spare Capacity

### Functions

- `int fna_spare_get_capacity` (const frsh\_resource\_id\_t resource\_id, const int importance, uint32\_t \*capacity)
- `int fna_spare_get_total_weight` (const frsh\_resource\_id\_t resource\_id, const int importance, int \*total\_weight)
- `int fna_spare_release_capacity` (const frsh\_resource\_id\_t resource\_id, const fna\_vres\_id\_t vres, const struct timespec new\_budget, const struct timespec new\_period)

### 3.4.1 Detailed Description

The following functions are used to get spare capacity data

### 3.4.2 Function Documentation

#### 3.4.2.1 `int fna_spare_get_capacity` (const frsh\_resource\_id\_t *resource\_id*, const int *importance*, uint32\_t \* *capacity*)

`fna_spare_get_capacity()`

This operation gets the spare capacity currently assigned to a importance level. If we divide this value by UINT32\_MAX we will get the network utilization associated to the spare capacity of a importance level.

The following is typically in stdint.h:

- `typedef unsigned int uint32_t;`
- `# define UINT32_MAX (4294967295U)`

#### Parameters:

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *importance* The importance we want the capacity of
- *capacity* The spare capacity for that importance level

#### Returns:

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

#### 3.4.2.2 `int fna_spare_get_total_weight` (const frsh\_resource\_id\_t *resource\_id*, const int *importance*, int \* *total\_weight*)

`fna_spare_get_total_weight()`

This function gets the sum of the weight parameters for all vres in a network of an importance level.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *importance* The importance we want the total weight of
- *total\_weight* The total weight for that importance level

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

### 3.4.2.3 `int fna_spare_release_capacity (const frsh_resource_id_t resource_id, const fna_vres_id_t vres, const struct timespec new_budget, const struct timespec new_period)`

#### `fna_spare_release_capacity()`

This function allows to ask for less budget and period than what we received. The request must be compatible with the rest of contract parameters of the vres. If we want to recover the released capacity we will need to renegotiate.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *vres* The internal virtual resource id
- ← *new\_budget* The new\_budget
- ← *new\_period* The new Period

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_CONTRACTED\_VRES: if the vres is not contracted
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL
- FNA\_ERR\_CONTRACT\_REJECTED: if it is incompatible with the current contract

## 3.5 FNA Send and Receive

### Functions

- `int fna_send_sync` (const frsh\_send\_endpoint\_t \*endpoint, const void \*msg, const size\_t size)
- `int fna_send_async` (const frsh\_send\_endpoint\_t \*endpoint, const void \*msg, const size\_t size)
- `int fna_receive_sync` (const frsh\_receive\_endpoint\_t \*endpoint, void \*buffer, const size\_t buffer\_size, size\_t \*received\_bytes)
- `int fna_receive_async` (const frsh\_receive\_endpoint\_t \*endpoint, void \*buffer, const size\_t buffer\_size, size\_t \*received\_bytes)
- `int fna_send_endpoint_get_status` (const frsh\_send\_endpoint\_t \*endpoint, int \*number\_of\_pending\_messages, frsh\_endpoint\_network\_status \*network\_status, frsh\_protocol\_status\_t \*protocol\_status)
- `int fna_receive_endpoint_create` (const frsh\_resource\_id\_t resource\_id, const frsh\_stream\_id\_t stream\_id, const frsh\_endpoint\_queueing\_info\_t queueing\_info, const frsh\_protocol\_info\_t protocol\_info, frsh\_receive\_endpoint\_t \*endpoint)
- `int fna_receive_endpoint_get_status` (const frsh\_receive\_endpoint\_t \*endpoint, int \*number\_of\_pending\_messages, frsh\_endpoint\_network\_status \*network\_status, frsh\_protocol\_status\_t \*protocol\_status)

### 3.5.1 Detailed Description

The following functions are used to send and receive

### 3.5.2 Function Documentation

#### 3.5.2.1 `int fna_receive_async` (const frsh\_receive\_endpoint\_t \* endpoint, void \* buffer, const size\_t buffer\_size, size\_t \* received\_bytes)

`fna_receive_async()`

This operation is similar to the previous one but it works in a non blocking (asynchronous) fashion. If no message is available it returns with error FNA\_NO\_MESSAGE.

#### Parameters:

- ← *endpoint* The receive endpoint we are receiving from. (resource\_id is in the endpoint).
- *buffer* Buffer for storing the received message
- ← *buffer\_size* The size in bytes of this buffer
- *received\_bytes* The actual number of received bytes

#### Returns:

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL
- FNA\_ERR\_NO\_SPACE: if the message size is bigger than the provided buffer.
- FNA\_NO\_MESSAGE: if no messages are available in the queue.

**3.5.2.2** `int fna_receive_endpoint_create (const frsh_resource_id_t resource_id, const frsh_stream_id_t stream_id, const frsh_endpoint_queueing_info_t queueing_info, const frsh_protocol_info_t protocol_info, frsh_receive_endpoint_t * endpoint)`

`fna_receive_endpoint_create()`

This operation creates a receive endpoint associated with a unidirectional stream within a network interface of the node.

Receiving endpoints are not bound to any network vres, this is because don't originate any traffic.

Note that the protocol address is not needed for reception because it can be determined internally by FRSB based on the network\_id.

Note also that messages may come from diferent originators.

**Parameters:**

- ← *resource\_id* Id of the network from which we listen.
- ← *stream\_id* Id of the stream within the network.
- ← *queueing\_info* Buffering information(queue size and policy).
- ← *protocol\_info* Extra protocol info opaque for the application.
- *endpoint* Placeholder for the endpoint object.

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

**3.5.2.3** `int fna_receive_endpoint_get_status (const frsh_receive_endpoint_t * endpoint, int * number_of_pending_messages, frsh_endpoint_network_status * network_status, frsh_protocol_status_t * protocol_status)`

`fna_receive_endpoint_get_pending_messages`

This function tells the number of messages still pending in the endpoint queue, whether the network is up or down and some optional information which is protocol dependent.

**Parameters:**

- ← *endpoint* The receive endpoint (resource\_id is in the endpoint).
- *number\_of\_pending\_messages* The number of pending messages
- *network\_status* How is the network (up, down..)
- *protocol\_status* Protocol dependent status info

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

### 3.5.2.4 `int fna_receive_sync (const frsh_receive_endpoint_t * endpoint, void * buffer, const size_t buffer_size, size_t * received_bytes)`

#### `fna_receive_sync()`

This operation is used to receive messages from the network with a blocking behavior (if there are no messages this operation blocks the calling thread).

When a message is available, it is copied to buffer (up to its size). The number of bytes copied is returned in `received_bytes`. The rest of the bytes of that message will be lost or not depending on the protocol (FNA\_ERR\_NO\_SPACE will be returned if it is).

The function fails with FNA\_ERR\_NO\_SPACE if the buffersize is too small for the message received. In this case the message is lost.

Messages arriving at a receiver buffer that is full will be handled according to the queuing policy of the endpoint (overwrite oldest, discard it,etc).

#### Parameters:

- ← *endpoint* The receive endpoint we are receiving from. (`resource_id` is in the endpoint).
- *buffer* Buffer for storing the received message
- ← *buffer\_size* The size in bytes of this buffer
- *received\_bytes* The actual number of received bytes

#### Returns:

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of `resource_id`
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL
- FNA\_ERR\_NO\_SPACE: if the message size is bigger than the provided buffer.

### 3.5.2.5 `int fna_send_async (const frsh_send_endpoint_t * endpoint, const void * msg, const size_t size)`

#### `fna_send_async()`

This operation sends a message stored in `msg` and of length `size` through the given send endpoint. The operation is non-blocking and returns immediately.

#### Parameters:

- ← *endpoint* The send endpoint we are sending through. It must be bound to a virtual resource (`resource_id` is in the endpoint).
- ← *msg* The message we want to send
- ← *size* The size in bytes of the message

#### Returns:

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_BOUND: if endpoint is not bound to a valid vres
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of `resource_id`
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

FNA\_ERR\_TOO\_LARGE: if the message is too large for the network protocol  
 FNA\_ERR\_BUFFER\_FULL: if the message has been discarded because the queue is full (and does not have the policy FNA\_QP\_OLDEST)

### 3.5.2.6 `int fna_send_endpoint_get_status (const frsh_send_endpoint_t * endpoint, int * number_of_pending_messages, frsh_endpoint_network_status * network_status, frsh_protocol_status_t * protocol_status)`

[fna\\_send\\_endpoint\\_get\\_status\(\)](#)

This function tells the number of messages still pending in the endpoint queue, whether the network is up or down with some optional information which is protocol\_dependent.

#### Parameters:

- ← *endpoint* The send endpoint (resource\_id is in the endpoint).
- *number\_of\_pending\_messages* The number of pending messages
- *network\_status* How is the network (up, down..)
- *protocol\_status* Protocol dependent status info

#### Returns:

FNA\_NO\_ERROR: in this case it also means contract accepted  
 FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors  
 FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized  
 FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id  
 FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

### 3.5.2.7 `int fna_send_sync (const frsh_send_endpoint_t * endpoint, const void * msg, const size_t size)`

[fna\\_send\\_sync\(\)](#)

Similar to previous function but now the sending thread gets blocked until the message is already sent to the network.

#### Parameters:

- ← *endpoint* The send endpoint we are sending through. It must be bound to a virtual resource (resource\_id is in the endpoint).
- ← *msg* The message we want to send
- ← *size* The size in bytes of the message

#### Returns:

FNA\_NO\_ERROR: in this case it also means contract accepted  
 FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors  
 FNA\_ERR\_NOT\_BOUND: if endpoint is not bound to a valid vres  
 FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized  
 FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id  
 FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL  
 FNA\_ERR\_TOO\_LARGE: if the message is too large for the network protocol  
 FNA\_ERR\_BUFFER\_FULL: if the message has been discarded because the queue is full (and does not have the policy FNA\_QP\_OLDEST)

## 3.6 FNA Network Configuration

### Functions

- `int fna_message_get_tx_time` (const frsh\_resource\_id\_t resource\_id, const size\_t nbytes, struct timespec \*tx\_time)
- `int fna_message_get_max_size` (const frsh\_resource\_id\_t resource\_id, size\_t \*max\_size)

### 3.6.1 Detailed Description

These functions are needed to set/get some network dependent values

### 3.6.2 Function Documentation

#### 3.6.2.1 `int fna_message_get_max_size` (const frsh\_resource\_id\_t resource\_id, size\_t \* max\_size)

`fna_message_get_max_size()`

This operation gives the maximum number of bytes that can be sent in a single message through the network designated by resource\_id. If the application needs to send bigger messages it will have to split them. Some protocols like TCP/IP are capable of sending large messages (and use fragmentation internally) but other protocols doesn't provide fragmentation features so the maximum size will be the MTU itself.

It is be used by the application to calculate the minimum and maximum budgets used in the preparation of network contracts.

#### Parameters:

- ← *resource\_id* The network we want the tx time from.
- *max\_size* The maximum number of bytes for each message

#### Returns:

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL or nbytes is bigger than the maximum message size

#### 3.6.2.2 `int fna_message_get_tx_time` (const frsh\_resource\_id\_t resource\_id, const size\_t nbytes, struct timespec \* tx\_time)

`fna_message_get_tx_time()`

This operation gives the worst case transmission time that it takes to send a message of the nbytes size through the network designated by resource\_id, including any network overheads (fragmentation in packets, headers, retransmissions,...).

It is be used by the application to calculate the minimum and maximum budgets used in the preparation of network contracts.

#### Parameters:

- ← *resource\_id* The network we want the tx time from.

← *nbytes* Number of bytes of the message

→ *tx\_time* The transmission time for nbytes (including overheads)

**Returns:**

FNA\_NO\_ERROR: in this case it also means contract accepted

FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors

FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized

FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id

FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL or nbytes is bigger than the maximum message size



## 3.7 FNA Public Interface

### Modules

- [FNA Mapping Functions](#)
- [FNA Negotiation Service Parameters](#)

### 3.7.1 Detailed Description

FNA is a Network adaption layer that allows to plugin new network protocols to the distributed module.

It is divided in two parts:

- FRSH\_FNA: public types and functions for the FRSH API
- FNA: private functions only used within FRSH.

## 3.8 FNA Mapping Functions

### Functions

- `int frsh_rtep_map_network_address` (`const frsh_resource_id_t resource_id, const rtep_network_address_t *in_address, frsh_network_address_t *out_address`)
- `int frsh_rtep_map_stream_id` (`const frsh_resource_id_t resource_id, const rtep_channel_t *in_stream, frsh_stream_id_t *out_stream`)

### 3.8.1 Detailed Description

These functions are needed to map network specific types to FRSH types. Instead of providing this mapping functions a static hardwired configuration could be used.

### 3.8.2 Function Documentation

#### 3.8.2.1 `int frsh_rtep_map_network_address` (`const frsh_resource_id_t resource_id, const rtep_network_address_t * in_address, frsh_network_address_t * out_address`)

`frsh_XXXX_map_network_address()`

To map a XXXX protocol network address into a FRSH address. The protocol must keep this mapping consistent. Instead of using a function a hardwired mapping could be used.

#### Parameters:

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *in\_address* The network address we want to map to a frsh address
- *out\_address* The FRSH abstract network address

#### Returns:

- `FNA_NO_ERROR`: in this case it also means contract accepted
- `FNA_ERR_INTERNAL_ERROR`: protocol dependent internal errors
- `FNA_ERR_NOT_INITIALIZED`: if the protocol is not initialized
- `FNA_ERR_RESOURCE_ID_INVALID`: if we are not in charge of `resource_id`
- `FNA_ERR_BAD_ARGUMENT`: if pointers are NULL

#### 3.8.2.2 `int frsh_rtep_map_stream_id` (`const frsh_resource_id_t resource_id, const rtep_channel_t * in_stream, frsh_stream_id_t * out_stream`)

`frsh_XXXX_map_stream_id()`

To map a XXXX protocol network stream, port, channel... into a FRSH stream. The protocol must keep this mapping consistent. Instead of using a function a hardwired mapping could be used.

#### Parameters:

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *in\_stream* The network stream we want to map to a FRSH stream
- *out\_stream* The FRSH abstract network stream

**Returns:**

FNA\_NO\_ERROR: in this case it also means contract accepted  
FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors  
FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized  
FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id  
FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

## 3.9 FNA Negotiation Service Parameters

### Functions

- `int frsh_rtep_negotiation_messages_vres_renegotiate` (const frsh\_resource\_id\_t resource\_id, const struct timespec \*period, bool \*accepted)
- `int frsh_rtep_negotiation_messages_vres_get_period` (const frsh\_resource\_id\_t resource\_id, struct timespec \*period)
- `int frsh_rtep_service_thread_vres_renegotiate` (const frsh\_resource\_id\_t resource\_id, const struct timespec \*budget, const struct timespec \*period, bool \*accepted)
- `int frsh_rtep_service_thread_vres_get_budget_and_period` (const frsh\_resource\_id\_t resource\_id, struct timespec \*budget, struct timespec \*period)

### 3.9.1 Detailed Description

These functions are needed to set/get the negotiation service parameters.

### 3.9.2 Function Documentation

#### 3.9.2.1 `int frsh_rtep_negotiation_messages_vres_get_period` (const frsh\_resource\_id\_t resource\_id, struct timespec \* period)

`frsh_XXXX_negotiation_messages_vres_get_period()`

This function gets the minimum period of the negotiation messages sent through the network.

#### Parameters:

← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)

→ *period* The period for negotiation messages

#### Returns:

FNA\_NO\_ERROR: in this case it also means contract accepted

FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors

FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized

FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id

FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

#### 3.9.2.2 `int frsh_rtep_negotiation_messages_vres_renegotiate` (const frsh\_resource\_id\_t resource\_id, const struct timespec \* period, bool \* accepted)

`frsh_XXXX_negotiation_messages__vres_renegotiate()`

This function allows the application to change the minimum period of the negotiation messages sent through the network. It is similar to the service thread but for the network messages. We do not provide budget here because the size of the negotiation messages is fixed.

This change is similar to a renegotiation so a schedulability test must be done to see if the change can be accepted or not.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *period* The new period for negotiation messages
- *accepted* If the change has been accepted or not

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

### 3.9.2.3 int frsh\_rtep\_service\_thread\_vres\_get\_budget\_and\_period (const frsh\_resource\_id\_t resource\_id, struct timespec \* budget, struct timespec \* period)

frsh\_XXXX\_service\_thread\_vres\_get\_budget\_and\_period()

This function gets the budget and period of a service thread.

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- *budget* The budget of the service thread
- *period* The period of the service thread

**Returns:**

- FNA\_NO\_ERROR: in this case it also means contract accepted
- FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors
- FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized
- FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id
- FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

### 3.9.2.4 int frsh\_rtep\_service\_thread\_vres\_renegotiate (const frsh\_resource\_id\_t resource\_id, const struct timespec \* budget, const struct timespec \* period, bool \* accepted)

frsh\_XXXX\_service\_thread\_vres\_renegotiate()

This function allows the application to change the period and budget of the service thread that makes the negotiations and schedulability tests in a network.

The service thread starts with a default budget and period that should be configurable

**Parameters:**

- ← *resource\_id* The network we are referring to (a protocol could be able to handle several networks at the same time)
- ← *budget* The new budget for the service thread
- ← *period* The new period for the service thread
- *accepted* If the negotiation has been accepted or not

**Returns:**

FNA\_NO\_ERROR: in this case it also means contract accepted  
FNA\_ERR\_INTERNAL\_ERROR: protocol dependent internal errors  
FNA\_ERR\_NOT\_INITIALIZED: if the protocol is not initialized  
FNA\_ERR\_RESOURCE\_ID\_INVALID: if we are not in charge of resource\_id  
FNA\_ERR\_BAD\_ARGUMENT: if pointers are NULL

# Chapter 4

## FNA File Documentation

### 4.1 fna.h File Reference

```
#include "frsh_core_types.h"
#include "fna_error.h"
#include <stdint.h>
```

#### Typedefs

- typedef void \* [fna\\_vres\\_id\\_t](#)

#### Functions

- int [fna\\_init](#) (const frsh\_resource\_id\_t resource\_id)
- int [fna\\_vres\\_create](#) (const frsh\_resource\_id\_t resource\_id, const frsh\_contract\_t \*contract, [fna\\_vres\\_id\\_t](#) \*vres)
- int [fna\\_vres\\_renegotiate\\_sync](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, const frsh\_contract\_t \*new\_contract)
- int [fna\\_vres\\_renegotiate\\_async](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, const frsh\_contract\_t \*new\_contract, frsh\_signal\_t signal\_to\_notify, frsh\_signal\_info\_t signal\_info)
- int [fna\\_vres\\_get\\_renegotiation\\_status](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, frsh\_renegotiation\_status\_t \*renegotiation\_status)
- int [fna\\_vres\\_destroy](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres)
- int [fna\\_vres\\_get\\_contract](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, frsh\_contract\_t \*contract)
- int [fna\\_vres\\_get\\_usage](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, struct timespec \*usage)
- int [fna\\_vres\\_get\\_remaining\\_budget](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, struct timespec \*remaining\_budget)
- int [fna\\_vres\\_get\\_budget\\_and\\_period](#) (const frsh\_resource\_id\_t resource\_id, const [fna\\_vres\\_id\\_t](#) vres, struct timespec \*budget, struct timespec \*period)
- int [fna\\_spare\\_get\\_capacity](#) (const frsh\_resource\_id\_t resource\_id, const int importance, uint32\_t \*capacity)
- int [fna\\_spare\\_get\\_total\\_weight](#) (const frsh\_resource\_id\_t resource\_id, const int importance, int \*total\_weight)

- `int fna_spare_release_capacity` (const `frsh_resource_id_t` resource\_id, const `fna_vres_id_t` vres, const struct timespec new\_budget, const struct timespec new\_period)
- `int fna_send_sync` (const `frsh_send_endpoint_t` \*endpoint, const void \*msg, const size\_t size)
- `int fna_send_async` (const `frsh_send_endpoint_t` \*endpoint, const void \*msg, const size\_t size)
- `int fna_receive_sync` (const `frsh_receive_endpoint_t` \*endpoint, void \*buffer, const size\_t buffer\_size, size\_t \*received\_bytes)
- `int fna_receive_async` (const `frsh_receive_endpoint_t` \*endpoint, void \*buffer, const size\_t buffer\_size, size\_t \*received\_bytes)
- `int fna_send_endpoint_get_status` (const `frsh_send_endpoint_t` \*endpoint, int \*number\_of\_pending\_messages, `frsh_endpoint_network_status` \*network\_status, `frsh_protocol_status_t` \*protocol\_status)
- `int fna_receive_endpoint_create` (const `frsh_resource_id_t` resource\_id, const `frsh_stream_id_t` stream\_id, const `frsh_endpoint_queueing_info_t` queueing\_info, const `frsh_protocol_info_t` protocol\_info, `frsh_receive_endpoint_t` \*endpoint)
- `int fna_receive_endpoint_get_status` (const `frsh_receive_endpoint_t` \*endpoint, int \*number\_of\_pending\_messages, `frsh_endpoint_network_status` \*network\_status, `frsh_protocol_status_t` \*protocol\_status)
- `int fna_message_get_tx_time` (const `frsh_resource_id_t` resource\_id, const size\_t nbytes, struct timespec \*tx\_time)
- `int fna_message_get_max_size` (const `frsh_resource_id_t` resource\_id, size\_t \*max\_size)



## 4.2 frsh\_fna.h File Reference

### Defines

- #define [RTEP](#)

### Functions

- int [frsh\\_rtep\\_map\\_network\\_address](#) (const frsh\_resource\_id\_t resource\_id, const rtep\_network\_address\_t \*in\_address, frsh\_network\_address\_t \*out\_address)
- int [frsh\\_rtep\\_map\\_stream\\_id](#) (const frsh\_resource\_id\_t resource\_id, const rtep\_channel\_t \*in\_stream, frsh\_stream\_id\_t \*out\_stream)
- int [frsh\\_rtep\\_negotiation\\_messages\\_vres\\_renegotiate](#) (const frsh\_resource\_id\_t resource\_id, const struct timespec \*period, bool \*accepted)
- int [frsh\\_rtep\\_negotiation\\_messages\\_vres\\_get\\_period](#) (const frsh\_resource\_id\_t resource\_id, struct timespec \*period)
- int [frsh\\_rtep\\_service\\_thread\\_vres\\_renegotiate](#) (const frsh\_resource\_id\_t resource\_id, const struct timespec \*budget, const struct timespec \*period, bool \*accepted)
- int [frsh\\_rtep\\_service\\_thread\\_vres\\_get\\_budget\\_and\\_period](#) (const frsh\_resource\_id\_t resource\_id, struct timespec \*budget, struct timespec \*period)

### 4.2.1 Define Documentation

#### 4.2.1.1 #define RTEP

Definition at line 75 of file frsh\_fna.h.

# Index

- FNA Initialization, 6
- FNA Mapping Functions, 22
- FNA Negotiation Service Parameters, 24
- FNA Network Configuration, 19
- FNA Private Interface, 5
- FNA Public Interface, 21
- FNA Send and Receive, 15
- FNA Spare Capacity, 13
- FNA Virtual Resources, 7
- fna.h, 27
- fna\_init
  - fnainit, 6
- fna\_message\_get\_max\_size
  - frshfnaconfig, 19
- fna\_message\_get\_tx\_time
  - frshfnaconfig, 19
- fna\_receive\_async
  - fnasendrecv, 15
- fna\_receive\_endpoint\_create
  - fnasendrecv, 15
- fna\_receive\_endpoint\_get\_status
  - fnasendrecv, 16
- fna\_receive\_sync
  - fnasendrecv, 16
- fna\_send\_async
  - fnasendrecv, 17
- fna\_send\_endpoint\_get\_status
  - fnasendrecv, 18
- fna\_send\_sync
  - fnasendrecv, 18
- fna\_spare\_get\_capacity
  - fnaspares, 13
- fna\_spare\_get\_total\_weight
  - fnaspares, 13
- fna\_spare\_release\_capacity
  - fnaspares, 14
- fna\_vres\_create
  - fnavres, 7
- fna\_vres\_destroy
  - fnavres, 8
- fna\_vres\_get\_budget\_and\_period
  - fnavres, 8
- fna\_vres\_get\_contract
  - fnavres, 9
- fna\_vres\_get\_remaining\_budget
  - fnavres, 9
- fna\_vres\_get\_renegotiation\_status
  - fnavres, 9
- fna\_vres\_get\_usage
  - fnavres, 10
- fna\_vres\_id\_t
  - fnavres, 7
- fna\_vres\_renegotiate\_async
  - fnavres, 11
- fna\_vres\_renegotiate\_sync
  - fnavres, 11
- fnainit
  - fna\_init, 6
- fnasendrecv
  - fna\_receive\_async, 15
  - fna\_receive\_endpoint\_create, 15
  - fna\_receive\_endpoint\_get\_status, 16
  - fna\_receive\_sync, 16
  - fna\_send\_async, 17
  - fna\_send\_endpoint\_get\_status, 18
  - fna\_send\_sync, 18
- fnaspares
  - fna\_spare\_get\_capacity, 13
  - fna\_spare\_get\_total\_weight, 13
  - fna\_spare\_release\_capacity, 14
- fnavres
  - fna\_vres\_create, 7
  - fna\_vres\_destroy, 8
  - fna\_vres\_get\_budget\_and\_period, 8
  - fna\_vres\_get\_contract, 9
  - fna\_vres\_get\_remaining\_budget, 9
  - fna\_vres\_get\_renegotiation\_status, 10
  - fna\_vres\_get\_usage, 10
  - fna\_vres\_id\_t, 7
  - fna\_vres\_renegotiate\_async, 11
  - fna\_vres\_renegotiate\_sync, 11
- frsh\_fna.h, 29
  - RTEP, 29
- frsh\_rtep\_map\_network\_address
  - frshfnamap, 22
- frsh\_rtep\_map\_stream\_id
  - frshfnamap, 22
- frsh\_rtep\_negotiation\_messages\_vres\_get\_period
  - frshfnaserv, 24
- frsh\_rtep\_negotiation\_messages\_vres\_renegotiate

---

- frshfnaserv, [24](#)
- frsh\_rtep\_service\_thread\_vres\_get\_budget\_and\_period
  - frshfnaserv, [25](#)
- frsh\_rtep\_service\_thread\_vres\_renegotiate
  - frshfnaserv, [25](#)
- frshfnaconfig
  - fna\_message\_get\_max\_size, [19](#)
  - fna\_message\_get\_tx\_time, [19](#)
- frshfnamap
  - frsh\_rtep\_map\_network\_address, [22](#)
  - frsh\_rtep\_map\_stream\_id, [22](#)
- frshfnaserv
  - frsh\_rtep\_negotiation\_messages\_vres\_get\_period, [24](#)
  - frsh\_rtep\_negotiation\_messages\_vres\_renegotiate, [24](#)
  - frsh\_rtep\_service\_thread\_vres\_get\_budget\_and\_period, [25](#)
  - frsh\_rtep\_service\_thread\_vres\_renegotiate, [25](#)
- RTEP
  - frsh\_fna.h, [29](#)