Database/compiler API Reference Manual

NetLogic Microsystems

April 2009

Contents

Chapter 1

Main Page

The database/compiler API is an interface to create an image of the database of rules to be loaded into NetL7 HW device.The database of rules consists of one or more groups and each group can consists of one or more rules. The steps to create a loadable binary image of the database are:

```
nlm_database_interface_init
  nlm_database_set_param
  nlm_database_open
    nlm_database_add_group X
      nlm_database_add_rule A
      nlm_database_add_rule B
    nlm_database_add_group Y
      nlm_database_add_rule C
      nlm_database_add_rule D
    nlm_database_compile
  nlm_database_close
nlm_database_interface_fini
```

The key function nlm_database_compile() performs the actual compilation and returns the pointer to the database image and its size. The API is the same in user and kernel mode. Memory allocation and error/warning reporting are done via user specified callback routines.

1.1 INITIALIZATION AND TERMINATION

The control plane defines a model of database initialization and termination via the following functions:

```
err = nlm_database_interface_init (type, database);
err = nlm_database_open (database);
err = nlm_database_close (database);
err = nlm_database_interface_fini (database);
```

such that

```
nlm_status err
nlm_database_type type
struct nlm_database *database
```

The first API that needs to be called by the application is nlm_database_interface_init. The API nlm_database_interface_init initializes the interfaces of the control plane library.

- The first argument to the API is the type of the database. Type specifies the underlying hardware.
- The second argument is a pointer to the handle of the database. A memory allocation needs to be made for the second parameter of the type "struct nlm_database" and the pointer needs to be passed to nlm_database_interface_init. This pointer/handle needs to be given as a parameter to all subsequent API calls involving the database.

The API nlm_database_open initializes the control plane library. After this API call the control plane library is ready to add groups and rules to the database. The argument to the API is the handle to the database. Note that multiple databases can be opened and each database will have a unique handle.

The API nlm_database_close frees and deallocates the rules from the database. The argument to the API is the handle to the database.

The API nlm_database_interface_fini shuts down the interface of the control plane library. This API needs to be called after nlm_database_close. The argument to the API is the handle to the database.

1.2 STATUS RETURN

The control plane API's return pass/fail status information via a value of the nlm_status enum type. Most fundamentally, an nlm_status value gives a boolean error indication: Nonzero/true means that some error has occurred; zero/false means that no error has occurred. The true (error) values are distinguished by the positive values of the type, each of which is an element of the enum.

Each API call returns an nlm_status value as its function value. This return value needs to be checked by an application calling the API and appropriate action needs to be taken in case of an error indication.

Note that the nlm_status enum is also shared with the dataplane API's.

1.3 ADDING GROUPS

Each database can have one or more rule groups. Each rule group in turn can consist of one or more rules. A group is the smallest unit on which a search can be performed. A stream of bytes or a flow can be searched against a group or in other words can be searched for occurrence of any of the rules present in the group.

```
err = nlm_database_add_group (database, group_id, p_group)
```

such that

```
struct nlm_database *database
uint32_t group_id
struct nlm_rule_group **p_group
```

- The first argument to the API is the handle to the database.
- The second argument is the group ID.
- The third parameter which is an output contains the handle to the added group after the call.

1.4 ADDING RULES

As mentioned earlier a group consists of one or more rules. The API nlm_database_add_rule can be used to add a rule to a specific group.

```
err = nlm_database_add_rule (database, group, rule_id, regex, p_rule)
```

such that

```
struct nlm_database *database
struct nlm_rule_group *group
uint32_t rule_id
const char *regex
struct nlm_rule **p_rule
```

- The first argument to the API is the handle to the database.
- The second argument is the handle to the group into which the rule needs to be added
- The third argument specifies the rule ID of the rule.
- The fourth argument is a pointer to the actual rule.
- The fifth argument contains handle to the added rule after the call. This is an output parameter.

1.5 COMPILING THE DATABASE

Compiling a database means that all the groups in the database and the rules belonging to all the groups are converted to a binary representation suitable for hardware and this binary representation can be later written to a file. Subsequently the dataplane application (which utilizes the underlying API's provided by the dataplane library) will load this file into the hardware and later on searches can be performed in the hardware against this loaded file/database.

The API for compiling is as follows

```
err = nlm_status nlm_database_compile (database, p_database, p_size)
```

such that

```
struct nlm_database *database
const void **p_database
uint32_t *p_size
```

- The first argument - database handle.
- The second argument - after the call ∗p_database points to the compiled image
- The third argument - after the call ∗p_size contains the size of the image

1.6 SETTING AND GETTING PARAMETERS FOR A DATABASE

The control plane library provides a number of parameters through which the processing of database can be controlled. For example, the rule format of rules in a particular group may be different from the rule format of rules in other groups. Through the API call of nlm_database_set_param the rule format for the database can be set to the desired format before adding the group to the database. Even if a group consists of rules with different formats this API call can be used before adding a particular rule.

The API call nlm_database_get_param retrieves the currently set value for the parameter.

```
err = nlm_database_set_param (struct nlm_database *database, nlm_database_param param, ...)
err = nlm_database_get_param (struct nlm_database *database, nlm_database_param param, ...)
```

such that

```
struct nlm_database *database
nlm_database_param is an enum (See below)
```

- The first argument to the API is the handle to the database.
- The second argument is an enum which defines the parameter type.

Important parameters of the database are memory allocation and logger callbacks (if none are specified, the compiler library will use standard malloc/free/printf routines)

- NLM_XMALLOC - will be called during compilation and expected to return new memory. The user should pass a function with a prototype: void ∗ (∗xmalloc) (void ∗cookie, uint32_t byte_cnt) The first argument is the cookie specified by NLM_CALLBACK_COOKIE
- NLM_XFREE - will be called to free the memory allocated by xmalloc callback. The user should pass a function with a prototype: void (∗xfree) (void ∗cookie, void ∗mem_ptr); The first argument is the cookie specified by NLM_CALLBACK_COOKIE
- NLM_IMAGE_XMALLOC - similar to NLM_XMALLOC, but will be called only once to allocate memory for final binary image.
- NLM_LOGGER - pointer to the nlm_logger structure that should be defined by the user to intercept all error/warning calls that can happen during compilation of the database. nlm_logger structure is defined as:

```
typedef void (*nlm_callback_type_xprint) (void *cookie, nlm_xprint_type type,
                                          nlm_xprint_module module, const char *str,
                                          unsigned int error_code, void *aux_data);
typedef struct nlm_logger
{
  void *cookie;
  nlm_callback_type_xprint xprint;
  char sprintf_buf[NLM_LOGGER_MAX_SPRINTF_BUF];
} nlm_logger;
```

where 'xprint' is the main callback that may be called during compilation to report errors/warnings.

- The 1st argument is 'cookie' specified in nlm_logger.

- The type of the call is specified by the 2nd argument - nlm_xprint_type enum.
- The module that caused the callback is 3rd argument - nlm_xprint_module enum
- The human readable message and error code comes 4th and 5th.
- Last argument 'aux_data' can always be casted to 'struct nlm_logger_rule_data *' or in some cases to module specific data structures like 'nlm_log_fms_placer_rule_stat'. See header file for details.

1.7 CONDITIONS FOR MATCHING A RULE

With each rule a condition may be specified which can be checked when the rule is detected in the traffic. If the condition is met then a match is declared to the application otherwise nothing is reported. The API below provides a way to specify condition(s) with any rule.

```
err = nlm_database_set_rule_action (database, rule, action)
```

such that

```
struct nlm_database *database
struct nlm_rule *rule
nlm_rule_action *action
```

- The first argument to the API is the handle to the database.
- The second argument is the handle to the rule.
- The third argument is the action specified for the rule. For a list of actions see below.

1.8 NETL7 2GIFA DEVICE (NLS055/NLS205)

The API nlm_database_set_rule_action is NOT supported for "2GIFA" device.

For the API calls nlm_database_set_param and nlm_database_get_param the following parameter values are not supported: NLM_DATABASE_SW_MODEL_TYPE (since there is no builtin sw model of the 2GIFA devices), NLM_DO_AGGRESSIVE_MML (since min-match-length feature is currently not implemented on 2GIFA)

In the NLM_PARSER_PARAM, the only two fully supported values are NLM_SYNTAX_PCRE and NLM_SYNTAX_NETSCREEN. The values NLM_SYNTAX_PCRE_EXTENDED, NLM_SYNTAX_PCRE_IGNORE_CASE, NLM_SYNTAX_PCRE_NON_GREEDY all default to NLM_SYNTAX_PCRE. Similarly NLM_SYNTAX_NETSCREEN_NON_GREEDY defaults to NLM_SYNTAX_NETSCREEN. Note: All NLM_PARSER_PARAM values are fully supported by "3GIFA"

1.9 NETL7 3GIFA DEVICE (NLS2008/NLS2018)

The API nlm_database_set_rule_action is fully supported by "3GIFA" device and has the following interpretation:

- NLM_MIN_MATCH_LENGTH - if this action is specified, its value should be a decimal integer from 1-65,535 (2^16-1). The match must meet the minimum match length criteria to be considered

a match. No result will be returned for a match that does not meet the minimum length. For stateful rule groups, this feature works the same for matches contained entirely within a packet and those that span two or more packets - the match length is a function only of the pattern, not the existence (or lack thereof) of a packet boundary anywhere in the match.

- NLM_MIN_OFFSET | NLM_EXACT_OFFSET | NLM_MAX_OFFSET - the value should be a decimal integer from 1-65,535 (2^16-1). A minimum offset criterion is interpreted as greater than or equal to. A maximum offset criterion is interpreted as less than or equal to. An exact offset criterion is interpreted as equal to. In all cases, the specified offset is relative to start of packet for stateless flows and start of flow for stateful flows. No result will be returned for a match that does not meet the specified criteria.

- NLM_RESULT_PRIORITY - the default priority for the rule is zero. HW allows to set result clamps independently for different priorities. The user can use different result clamps as way to prevent DoS attacks. HW does not sort the results with different priorities, though results for the same flow are guaranteed to come in the same order as packets received.

- NLM_NON_GREEDY - if this action is specified, the rule will be compiled in non-greedy mode: for a series of overlapping matches return the shortest match and start looking for the match again. That is an equivalent way to make rule no

- NLM_MATCH_ONCE - if this action is specified, the rule will be compiled in match-once mode: for a series of overlapping matches return the shortest match and never return the matches again for the given rule until the end of the flow (stateful flows) or until the end of the packet (stateless flows), so the rule will report a match only once.

- NLM_TRIGGER_GROUP - This action takes a valid group-id from 1-1023 (2^10-1) as its value. A group with the specified group-id must be present in the same database. Upon matching the rule, no output is produced and all rules belonging to the specified group-id are activated. Activated rules will start matching the traffic starting from the next byte.

- NLM_MATCH_AND_TRIGGER_GROUP - This action takes a valid group-id from 1-1023 (2^10-1) as its value. A group with the specified group-id must be present in the same database. Upon matching the rule, output is produced and all rules belonging to the specified group-id are activated. Activated rules will start matching the traffic starting from the next byte.

1.10 AN EXAMPLE APPLICATION DEMONSTRATING USE OF CONTROL PLANE API's

The following code is an application written in the C language that demonstrates the use of the above mentioned control plane API's for the purpose of generating a binary representation/image of all the groups in the database. Note that the code below is an example for understanding the API's and a real application may need to do more than what is done below. For example, it would need to check the return value of the calls :)

```
struct nlm_database *db;
const void *image;
uint32_t size, group_id, rule_id;

db = malloc (nlm_database_sizeof (NLM_DATABASE_HW_FMS));
nlm_database_interface_init (NLM_DATABASE_HW_FMS, db);
```

```
nlm_database_set_param (db, NLM_OPTIMIZATION_LEVEL, 1);
nlm_database_set_param (db, NLM_PARSER_PARAM, NLM_SYNTAX_PCRE);

nlm_database_open (db);

for(;;) // every group
  {
    struct nlm_rule_group *grp;
    nlm_database_add_group (db, group_id, &grp);
    for(;;) // every rule
      {
        struct nlm_rule *rule;
        nlm_database_add_rule (db, grp, rule_id, "ab.*cd", &rule);
        nlm_database_set_rule_action (db, rule, NLM_MIN_MATCH_LENGTH, 150);
      }
  }

nlm_database_compile (db, &image, &size);

// write image into a file

nlm_database_close (db);
nlm_database_interface_fini (db);
free (db);
```

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 nlm_log_fms_placer_rule_stat Struct Reference

Auxiliary data for 'xprint' callback from placer.

`#include <nlm_logger.h>`

Data Fields

- struct nlm_logger_rule_data id
  rule and group id of the rule that triggered callback
- unsigned num_cams
  # cams in this rule element
- unsigned num_hubs
  # hubs " " " "
- unsigned num_ctrs
  # ctrs " " "
- unsigned num_sccs
  # SCCs " " "
- uint32_t invalid_graph: 1
  invalid abstract graph
- uint32_t too_big: 1
  resources required exceed those of target device
- uint32_t retries
  number of placement retries
- void * handle
  compiler-internal handle

4.1.1 Detailed Description

Auxiliary data for 'xprint' callback from placer.

This auxiliary data is supplied for error codes:

NLM_LOG_FMS_PLACER_RULE_NOT_PLACED NLM_LOG_FMS_PLACER_RULE_TRACE NLM_LOG_FMS_PLACER_INPUT_GRAPH_INVALID NLM_LOG_FMS_PLACER_INPUT_-GRAPH_TOO_BIG

Note that these logs report the status of exactly one compilation rule element, as opposed to an entire user rule in the sense of nlm_database_add_rule(). In a compiled database, a single user rule may be decomposed into a collection of rule elements. Each of these rule elements either does or does not place independently of all the other rule elements of the user rule.

The documentation for this struct was generated from the following file:

- nlm_logger.h

4.2 nlm_logger Struct Reference

main logger structure.

`#include <nlm_logger.h>`

Data Fields

- void * cookie

  cookie - 1st argument to xprint callback

- nlm_callback_type_xprint xprint

  main callback routine

- char sprintf_buf [NLM_LOGGER_MAX_SPRINTF_BUF]

  internal buffer used to construct a string

4.2.1 Detailed Description

main logger structure.

nlm_logger structure is used to setup an 'xprint' callback routine that may be called during compilation process to report error/warning/debug messages to the user

The documentation for this struct was generated from the following file:

- nlm_logger.h

4.3 nlm_logger_rule_data Struct Reference

Generic auxiliary data for 'xprint' callback.

`#include <nlm_logger.h>`

4.3.1 Detailed Description

Generic auxiliary data for 'xprint' callback.

the last 'aux_data' pointer of the 'xprint' callback can always be casted to 'struct nlm_logger_rule_data *' to retrieve rule and group id of the rule that triggered callback. In some cases 'aux_data' can be casted to 'nlm_log_fms_placer_rule_*'. See below

The documentation for this struct was generated from the following file:

- nlm_logger.h

Chapter 5

File Documentation

5.1 nlm_common_api.h File Reference

header file for shared structures in Dataplane/packet API and Controlplane/database API

```
#include "nlm_stdint.h"
#include "nlm_error_tbl.def"
```

Typedefs

- typedef unsigned long long nlm_phys_addr

  defines physical address for data plane library

Enumerations

- enum nlm_status

  status and error codes returned by all API functions

Functions

- const char * nlm_get_status_string (nlm_status status)

  function to convert status code to string

5.1.1 Detailed Description

header file for shared structures in Dataplane/packet API and Controlplane/database API

5.1.2 Function Documentation

5.1.2.1 const char* nlm_get_status_string (nlm_status *status*)

function to convert status code to string

Parameters:

← ***status*** code to be converted

Returns:

string that describes the code

5.2 nlm_database_api.h File Reference

header file for Controlplane/database API

```
#include "nlm_common_api.h"
#include "nlm_logger.h"
```

Typedefs

- typedef void *(* nlm_callback_type_xmalloc)(void *cookie, uint32_t byte_cnt)

  function prototype for callback accessed by nlm_database_set_param (db, NLM_XMALLOC, malloc_-callback); and nlm_database_set_param (db, NLM_IMAGE_XMALLOC, image_malloc_callback);

- typedef void(* nlm_callback_type_xfree)(void *cookie, void *mem_ptr)

  proto for NLM_XFREE

Enumerations

- enum nlm_database_type {

  NLM_INVALID_DATABASE_TYPE = NLM_FIRST_DATABASE_TYPE, NLM_DATABASE_-HW_MARS1, NLM_DATABASE_HW_MARS2, NLM_DATABASE_HW_FMS,

  NLM_DATABASE_HW_MARS3 }

  type of the database

- enum nlm_parser_param

  parser flags

- enum nlm_mapper_param { ,

  NLM_DO_AGGRESSIVE_MML, NLM_DO_AGGRESSIVE_MATCH_ONCE, NLM_RULE_-SYMBOL_LIMIT, NLM_REMOVE_REDUNDANCY,

  NLM_STITCH_RULES, NLM_USE_COUNTER }

  mapper flags

- enum nlm_database_param {

  NLM_INVALID_DATABASE_PARAM = NLM_FIRST_DATABASE_PARAM, NLM_-OPTIMIZATION_LEVEL, NLM_VERBOSITY_LEVEL, NLM_PARSER_PARAM,

  NLM_MAPPER_PARAM, NLM_DEBUG_FLAGS, NLM_CALLBACK_COOKIE, NLM_-IMAGE_XMALLOC,

  NLM_XMALLOC, NLM_XFREE, NLM_LOGGER, NLM_DATABASE_SW_MODEL_TYPE,

  NLM_DATABASE_STAT }

  configuration parameters that can be set and get by corresponding functions

- enum nlm_rule_action {

  NLM_INVALID_RULE_ACTION = NLM_FIRST_RULE_ACTION, NLM_MIN_MATCH_-LENGTH, NLM_MIN_OFFSET, NLM_EXACT_OFFSET,

  NLM_MAX_OFFSET, NLM_RESULT_PRIORITY, NLM_NON_GREEDY, NLM_MATCH_-ONCE,

NLM_TRIGGER_GROUP, NLM_MATCH_AND_TRIGGER_GROUP }

rule actions to be set

Functions

- uint32_t nlm_database_sizeof (nlm_database_type type)

  returns size of nlm_database (database handle structure)

- nlm_status nlm_database_interface_init (nlm_database_type type, struct nlm_database *database)

  init interfaces of the database compiler

- nlm_status nlm_database_interface_fini (struct nlm_database *database)

  shutdown interfaces of the database compiler

- nlm_status nlm_database_get_param (struct nlm_database *database, nlm_database_param param,...)

  Get specified database parameter.

- nlm_status nlm_database_set_param (struct nlm_database *database, nlm_database_param param,...)

  Set specified database parameter.

- nlm_status nlm_database_open (struct nlm_database *database)

  Open and initialize database compiler.

- nlm_status nlm_database_close (struct nlm_database *database)

  Close and free database of rules.

- nlm_status nlm_database_add_group (struct nlm_database *database, uint32_t group_id, struct nlm_rule_group **p_group)

  Create and add rule group to the database.

- nlm_status nlm_database_add_rule (struct nlm_database *database, struct nlm_rule_group *group, uint32_t rule_id, const char *regex, struct nlm_rule **p_rule)

  Create and add rule *to the rule* group.

- nlm_status nlm_database_set_rule_action (struct nlm_database *database, struct nlm_rule *rule, nlm_rule_action action,...)

  Set rule action.

- nlm_status nlm_database_compile (struct nlm_database *database, const void **p_database, uint32_t *p_size)

  Compile the whole database.

5.2.1 Detailed Description

header file for Controlplane/database API

5.2.2 Enumeration Type Documentation

5.2.2.1 enum nlm_database_type

type of the database

Enumerator:

NLM_INVALID_DATABASE_TYPE marker for invalid database type

NLM_DATABASE_HW_MARS1 NLS055.

NLM_DATABASE_HW_MARS2 NLS205.

NLM_DATABASE_HW_FMS NLS2008.

NLM_DATABASE_HW_MARS3 NLS2018.

5.2.2.2 enum nlm_mapper_param

mapper flags

Enumerator:

NLM_DO_AGGRESSIVE_MML should mml graphs be constructed even for unsafe rules(e.g. rules with overlapping prefix & suffix)

NLM_DO_AGGRESSIVE_MATCH_ONCE compile certain rules in match once mode instead of all-matches to improve HW capacity

NLM_RULE_SYMBOL_LIMIT don't compile rules with symbol count larger than this limit, (default 0 == compile rules of all sizes)

NLM_REMOVE_REDUNDANCY remove the prefix and suffix redundant quantifiers based on this value.
(default 3 == remove both)

NLM_STITCH_RULES enable/disable stitcher based on this value.
(default 1 == enabled)

NLM_USE_COUNTER enable/disable counter usage based on this value.
(default 1 == enabled)

5.2.2.3 enum nlm_database_param

configuration parameters that can be set and get by corresponding functions

Enumerator:

NLM_INVALID_DATABASE_PARAM marker for invalid parameter

NLM_OPTIMIZATION_LEVEL optimization level: 0 - no optimization

NLM_VERBOSITY_LEVEL verbosity level: 0 - no extra messages

NLM_PARSER_PARAM parser flags (one of nlm_parser_param)

NLM_MAPPER_PARAM mapper flags

NLM_DEBUG_FLAGS bitmap to control dumping of internal structures

NLM_CALLBACK_COOKIE 'void ∗cookie' to be sent to xmalloc, xfree, xdump callbacks

NLM_IMAGE_XMALLOC void ∗ (∗xmalloc) (void ∗cookie, uint32_t byte_cnt); callback

NLM_XMALLOC void ∗ (∗xmalloc) (void ∗cookie, uint32_t byte_cnt); callback

NLM_XFREE void (∗xfree) (void ∗cookie, void ∗mem_ptr); callback

NLM_LOGGER pointer to the nlm_logger

NLM_DATABASE_SW_MODEL_TYPE sub-type of software model

NLM_DATABASE_STAT probe function to inquire the status of a compiled database.

It is valid to call the function after any non-fatal return from nlm_database_compile() but before the nlm_database_close()

5.2.2.4 enum nlm_rule_action

rule actions to be set

Enumerator:

NLM_INVALID_RULE_ACTION marker for invalid action

NLM_MIN_MATCH_LENGTH match must meet the minimum match length criterion to be considered a match

NLM_MIN_OFFSET offset for the end of the match must meet the specified offset criterion

NLM_EXACT_OFFSET offset for the end of the match must meet the specified offset criterion

NLM_MAX_OFFSET offset for the end of the match must meet the specified offset criterion

NLM_RESULT_PRIORITY specifies priority of the rule to be clamped differently by dataplane

NLM_NON_GREEDY cause rule to match nongreedy (weak nongreedy)

NLM_MATCH_ONCE cause rule to match just once (strong nongreedy) (only first match reported)

NLM_TRIGGER_GROUP cause rule to activate the linked group without output

NLM_MATCH_AND_TRIGGER_GROUP cause rule to activate the linked group with output

5.2.3 Function Documentation

5.2.3.1 nlm_status nlm_database_interface_init (nlm_database_type *type*, struct nlm_database ∗ *database*)

init interfaces of the database compiler

Parameters:

← ***type*** of database to be initialized

← ***database*** pointer to database handle

Returns:

status

5.2.3.2 nlm_status nlm_database_interface_fini (struct nlm_database * *database*)

shutdown interfaces of the database compiler

Parameters:

← ***database*** pointer to database handle

Returns:

status

5.2.3.3 nlm_status nlm_database_get_param (struct nlm_database * *database*, nlm_database_param *param*, ...)

Get specified database parameter.

Parameters:

← ***database*** pointer to database handle

← ***param*** database parameter

Returns:

status

5.2.3.4 nlm_status nlm_database_set_param (struct nlm_database * *database*, nlm_database_param *param*, ...)

Set specified database parameter.

Parameters:

← ***database*** pointer to database handle

← ***param*** database parameter

Returns:

status

5.2.3.5 nlm_status nlm_database_open (struct nlm_database * *database*)

Open and initialize database compiler.

Parameters:

← ***database*** pointer to database handle

Returns:

status

5.2.3.6 nlm_status nlm_database_close (struct nlm_database * *database*)

Close and free database of rules.

Parameters:

← ***database*** pointer

Returns:

status

5.2.3.7 nlm_status nlm_database_add_group (struct nlm_database * *database*, uint32_t *group_id*, struct nlm_rule_group ** *p_group*)

Create and add rule group to the *database*.

Parameters:

← ***database*** pointer

← ***group_id*** id of the rule group

→ ***p_group*** rule group handle

Returns:

status

5.2.3.8 nlm_status nlm_database_add_rule (struct nlm_database * *database*, struct nlm_rule_group * *group*, uint32_t *rule_id*, const char * *regex*, struct nlm_rule ** *p_rule*)

Create and add *rule* to the rule *group*.

Parameters:

← ***database*** pointer

← ***group*** rule group handle

← ***rule_id*** id of the rule

← ***regex*** rule to be parsed and compiled

→ ***p_rule*** rule handle

Returns:

status

5.2.3.9 nlm_status nlm_database_set_rule_action (struct nlm_database * *database*, struct nlm_rule * *rule*, nlm_rule_action *action*, ...)

Set rule action.

Parameters:

← ***database*** pointer

← ***rule*** handle

← ***action*** rule action type

Returns:

status

5.2.3.10 nlm_status nlm_database_compile (struct nlm_database * *database*, const void ** *p_database*, uint32_t * *p_size*)

Compile the whole database.

Parameters:

← ***database*** pointer

→ ***p_database*** pointer to the compiled image

→ ***p_size*** compiled image size

Returns:

status

5.3 nlm_logger.h File Reference

header file for logger of compiler/database API

```
#include "nlm_stdint.h"
```

Data Structures

- struct nlm_logger

  main logger structure.

- struct nlm_logger_rule_data

  Generic auxiliary data for 'xprint' callback.

- struct nlm_log_fms_placer_rule_stat

  Auxiliary data for 'xprint' callback from placer.

Typedefs

- typedef void(∗ nlm_callback_type_xprint)(void ∗cookie, nlm_xprint_type type, nlm_xprint_module module, const char ∗str, unsigned int error_code, void ∗aux_data)

  proto of 'xprint' callback

Enumerations

- enum nlm_xprint_type

  type of the message reported by 'xprint' callback

- enum nlm_xprint_module

  module that called 'xprint' callback

Functions

- void default_xprint (void ∗cookie, nlm_xprint_type type, nlm_xprint_module module, const char ∗str, unsigned int error_code, void ∗aux_data)

  default 'xprint' callback

- void nop_xprint (void ∗cookie, nlm_xprint_type type, nlm_xprint_module module, const char ∗str, unsigned int error_code, void ∗aux_data)

  nop callback that ignores all arguments

Variables

- nlm_logger default_logger

  logger that prints messages to stdout

- nlm_logger nop_logger

  logger that suppresses all messages

5.3.1 Detailed Description

header file for logger of compiler/database API

Index