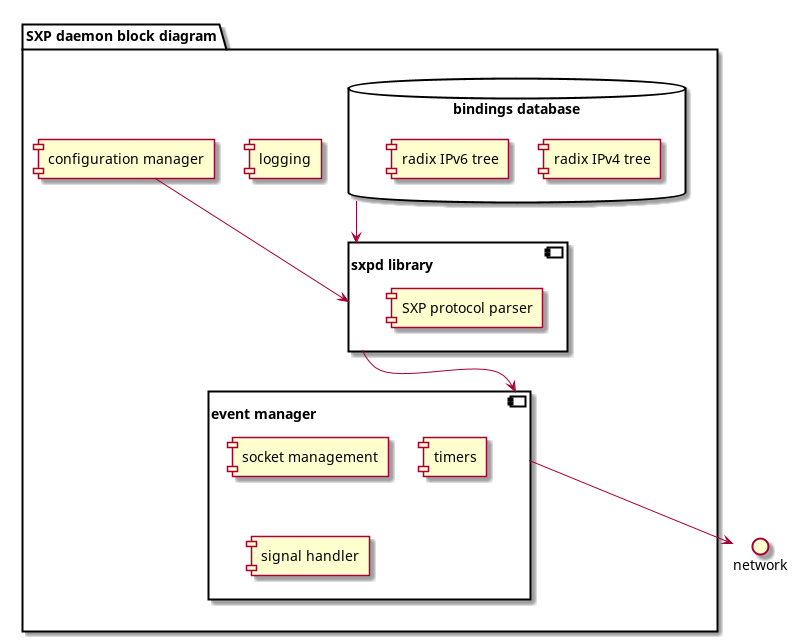
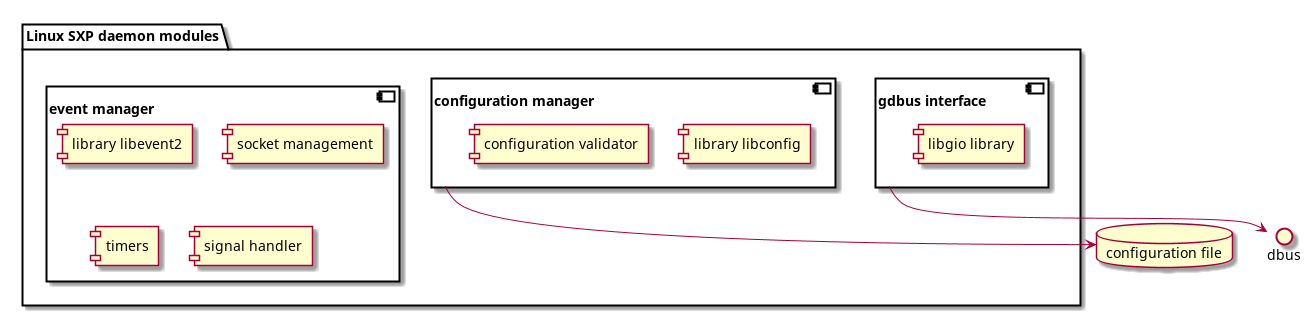
**SXP daemon C code design spec**

updated: 2017-08-15

This guide provides the design specification for implementing SXP daemon in C. Daemon is responsible for keeping track and distribution of IP-SGT bindings. The implementation runs under Linux with easy portability to other operating systems. Due to this, code is separated into platform-dependent and platform-independent code with the goal of having as much functionality as possible in the platform independent code.

The block diagrams for the SXP daemon are below. Platform dependent modules are the event manager, memory management, configuration manager, timestamp generator, random number generator, syslog, DBUS interface, and SXPD main setup code. Platform-independent modules cover the core daemon functionality, SXP protocol parser and radix tree implementations.





**Event Manager**

This module provides APIs for handling the following platform-dependent objects:

* Sockets (create, destroy, connect, bind to address, listen, set tcp-md5 signing, etc.)
* Timers (create, destroy, arm, disarm)
* Signals (set callback when signal is caught)

On Linux, the event manager is basically a wrapper around libevent, which is a common and tried solution for high performance handling of asynchronous events.

Event manger API is:

/\*\*

\* @brief implementation-specific event manager settings structure

\* @ingroup evmgr

\*/

struct evmgr\_settings**;**

/\*\*

\* @brief implementation-specific event manager context

\* @ingroup evmgr

\*/

struct evmgr**;**

/\*\*

\* @brief implementation-specific event manager socket

\* @ingroup evmgr

\*/

struct evmgr\_socket**;**

/\*\*

\* @brief implementation-specific event manager timer

\* @ingroup evmgr

\*/

struct evmgr\_timer**;**

/\*\*

\* @brief implementation-specific event manager listener

\* @ingroup evmgr

\*/

struct evmgr\_listener**;**

/\*\*

\* @brief implementation-specific event manager signal handler

\* @ingroup evmgr

\*/

struct evmgr\_sig\_handler**;**

/\*\*

\* @brief the maximum length of password used for TCP MD5 signing

\* @ingroup evmgr

\*/

#define EVMGR\_TCP\_MD5\_MAX\_PWD\_LEN (80)

/\*\*

\* @brief mapping of address and password used for TCP MD5 signing,

\* if password\_len == 0, it means there is no password for this address

\* @ingroup evmgr

\*/

struct address\_md5\_pwd\_pair **{**

struct sockaddr\_in sin**;**

char password**[**EVMGR\_TCP\_MD5\_MAX\_PWD\_LEN **+** 1**];**

uint16\_t password\_len**;**

**};**

/\*\*

\* @brief test if TCP md5 signing is available

\* @ingroup evmgr

\*

\* @return 0 on success, -1 on error

\*/

int evmgr\_md5sig\_test**(**void**);**

/\*\*

\* @brief create new event manager context

\* @ingroup evmgr

\* @param settings implementation-specific settings affecting timer creation

\*

\* @return event manager context or NULL if no memory

\*/

struct evmgr **\***evmgr\_create**(**struct evmgr\_settings **\***settings**);**

/\*\*

\* @brief free event manager context

\* @ingroup evmgr

\*

\* @param evmgr event manager context to free

\*/

void evmgr\_destroy**(**struct evmgr **\***evmgr**);**

/\*\*

\* @brief enter dispatch loop - process events until evmgr\_dispatch\_break is

\* not called

\* @ingroup evmgr

\*

\* @param evmgr event manager context

\*

\* @return returns 0 if dispatch finished due to evmgr\_dispatch\_break or -1 if

\* error occurs

\*/

int evmgr\_dispatch**(**struct evmgr **\***evmgr**);**

/\*\*

\* @brief break dispatch loop of event manager

\* @ingroup evmgr

\* @param evmgr event manager context

\*

\* @return 0 if success, -1 on error

\*/

int evmgr\_dispatch\_break**(**struct evmgr **\***evmgr**);**

/\*\*

\* @brief callback called when timer fires

\*/

**typedef** void **(\***evmgr\_timer\_callback**)(**struct evmgr\_timer **\***timer**,**

void **\***callback\_ctx**);**

/\*\*

\* @brief create a new timer and associate it with evmgr context

\* @ingroup evmgr

\*

\* @param evmgr event manager context to associate the event with

\* @param settings platform-sepcific settings affecting timer creation

\* @param timeout pointer to timeval structure

\* @param persist if false, then the timer will fire only once (but can be

\* re-armed), otherwise it persists

\* @param callback callback called when event triggers

\* @param callback\_ctx context passed to callback

\*

\* @return pointer to event context or NULL if error (memory, unsupported, etc.)

\*/

struct evmgr\_timer **\***evmgr\_timer\_create**(**struct evmgr **\***evmgr**,**

struct evmgr\_settings **\***settings**,**

struct timeval **\***timeout**,** bool persist**,**

evmgr\_timer\_callback callback**,**

void **\***callback\_ctx**);**

/\*\*

\* @brief arm timer - start countdown

\* @ingroup evmgr

\*

\* @param timer timer to arm

\*

\* @return 0 if success, -1 on error

\*/

int evmgr\_timer\_arm**(**struct evmgr\_timer **\***timer**);**

/\*\*

\* @brief disarm timer - stop countdown

\* @ingroup evmgr

\*

\* @param timer timer to disarm

\*

\* @return 0 if success, -1 on error

\*/

int evmgr\_timer\_disarm**(**struct evmgr\_timer **\***timer**);**

/\*\*

\* @brief disassociate timer from event manager and free memory

\* @ingroup evmgr

\*

\* @param timer timer to destroy

\*/

void evmgr\_timer\_destroy**(**struct evmgr\_timer **\***timer**);**

/\*\*

\* @brief callback called when signal is caught

\* @ingroup evmgr

\*/

**typedef** void **(\***evmgr\_signal\_callback**)(**struct evmgr\_sig\_handler **\***sig\_handler**,**

int signum**,** void **\***callback\_ctx**);**

/\*\*

\* @brief create a signal handler

\* @ingroup evmgr

\*

\* @param evmgr event manager context

\* @param settings platform-sepcific settings affecting handler creation

\* @param signum signal to handle

\* @param callback function to call when signal is caught

\* @param callback\_ctx context passed to callback function

\*

\* @return signal handler context or NULL if error occurred

\*/

struct evmgr\_sig\_handler **\***

evmgr\_sig\_handler\_create**(**struct evmgr **\***evmgr**,** struct evmgr\_settings **\***settings**,**

int signum**,** evmgr\_signal\_callback callback**,**

void **\***callback\_ctx**);**

/\*\*

\* @brief stop handling signal and free memory

\* @ingroup evmgr

\*

\* @param sig\_handler signal handler context

\*/

void evmgr\_sig\_handler\_destroy**(**struct evmgr\_sig\_handler **\***sig\_handler**);**

/\*\*

\* @brief callback called when a new connection on listener has been accepted

\* @ingroup evmgr

\*/

**typedef** void **(\***evmgr\_accept\_callback**)(**struct evmgr\_listener **\***listener**,**

struct evmgr\_socket **\***socket**,**

struct sockaddr\_in **\***address**,**

void **\***callback\_ctx**);**

/\*\*

\* @brief callback called when error occurs on listener

\* @ingroup evmgr

\*/

**typedef** void **(\***evmgr\_error\_callback**)(**struct evmgr\_listener **\***listener**,**

void **\***callback\_ctx**);**

/\*\*

\* @brief create socket and listen for incoming TCP connections, calling

\* specified callback once a connection is accepted

\* @ingroup evmgr

\*

\* @param evmgr event manager context to associate listener with

\* @param settings platform-sepcific settings affecting listener creation

\* @param address address to listen on

\* @param accept\_callback function invoked when connection is accepted

\* @param error\_callback function invoked when error occurs

\* @param callback\_ctx context pointer passed to the callback function

\*

\* @return listener object or NULL if error

\*/

struct evmgr\_listener **\***

evmgr\_listener\_create**(**struct evmgr **\***evmgr**,** struct evmgr\_settings **\***settings**,**

const struct sockaddr\_in **\***address**,**

evmgr\_accept\_callback accept\_callback**,**

evmgr\_error\_callback error\_callback**,** void **\***callback\_ctx**);**

/\*\*

\* @brief set TCP md5 signature for specific client

\* @ingroup evmgr

\*

\* @param listener listener socket

\* @param default\_pwd default connection password

\* @param pwd\_pair password and client socket pair. if password length is 0

\* default password connection password will be used

\*

\* @return 0 on success, -1 on error

\*/

int evmgr\_listener\_md5\_sig\_add**(**struct evmgr\_listener **\***listener**,**

const char **\***default\_pwd**,**

struct address\_md5\_pwd\_pair **\***pwd\_pair**);**

/\*\*

\* @brief remove TCP md5 signature for specific client

\* @ingroup evmgr

\*

\* @param listener listener socket

\* @param pwd\_pair password and client socket pair

\*

\* @return 0 on success, -1 on error

\*/

int evmgr\_listener\_md5\_sig\_del**(**struct evmgr\_listener **\***listener**,**

struct address\_md5\_pwd\_pair **\***pwd\_pair**);**

/\*\* event while reading \*/

#define EVMGR\_SOCK\_EVENT\_READING 0x01

/\*\* event while writing \*/

#define EVMGR\_SOCK\_EVENT\_WRITING 0x02

/\*\* end of file event \*/

#define EVMGR\_SOCK\_EVENT\_EOF 0x10

/\*\* error event \*/

#define EVMGR\_SOCK\_EVENT\_ERROR 0x20

/\*\* connection timeout event \*/

#define EVMGR\_SOCK\_EVENT\_TIMEOUT 0x40

/\*\* connection established event \*/

#define EVMGR\_SOCK\_EVENT\_CONNECTED 0x80

/\*\*

\* @brief destroy listener

\* @ingroup evmgr

\*

\* @param listener listener to destroy

\*/

void evmgr\_listener\_destroy**(**struct evmgr\_listener **\***listener**);**

/\*\*

\* @brief callback called when event occurs on a socket

\* @ingroup evmgr

\*/

**typedef** void **(\***socket\_event\_callback**)(**struct evmgr\_socket **\***socket**,**

int16\_t event\_bits**,** void **\***callback\_ctx**);**

/\*\*

\* @brief callback called when socket contains some data to be read

\* @ingroup evmgr

\*/

**typedef** void **(\***socket\_readable\_callback**)(**struct evmgr\_socket **\***socket**,**

void **\***callback\_ctx**);**

/\*\*

\* @brief callback called when socket is able to accept more data

\* @ingroup evmgr

\*/

**typedef** void **(\***socket\_writeable\_callback**)(**struct evmgr\_socket **\***socket**,**

void **\***callback\_ctx**);**

/\*\*

\* @brief create new socket (for connecting to address)

\* @ingroup evmgr

\*

\* @param evmgr event manager context

\* @param settings platform-sepcific settings affecting socket creation

\*

\* @return socket pointer or NULL if error

\*/

struct evmgr\_socket **\***evmgr\_socket\_create**(**struct evmgr **\***evmgr**,**

struct evmgr\_settings **\***settings**);**

/\*\*

\* @brief start connecting socket to given address

\* @ingroup evmgr

\*

\* @param socket socket to connect

\* @param src\_address address to connect from (may be NULL)

\* @param dst\_address address to connect to (optionally containing MD5 key)

\* @param read\_callback callback invoked when data are ready to be read

\* @param write\_callback callback invoked when socket writable

\* @param event\_callback callback invoked if some event occurs (e.g. EOF)

\* @param callback\_ctx context passed to callback functions

\*

\* @return 0 on success, -1 otherwise

\*/

int evmgr\_socket\_connect**(**struct evmgr\_socket **\***socket**,**

struct sockaddr\_in **\***src\_address**,**

struct address\_md5\_pwd\_pair **\***dst\_address**,**

socket\_readable\_callback read\_callback**,**

socket\_writeable\_callback write\_callback**,**

socket\_event\_callback event\_callback**,**

void **\***callback\_ctx**);**

/\*\*

\* @brief register socket for reading/writing or clear registration (if all

\* callbacks NULL)

\* @ingroup evmgr

\*

\* @param socket socket to register

\* @param read\_callback callback called once data are ready to be read

\* @param write\_callback callback called if data can be written

\* @param event\_callback callback called if an event occurs (such as EOF)

\* @param callback\_ctx context passed to callback functions

\*

\* @return 0 on success, -1 otherwise

\*/

int evmgr\_socket\_cb\_register**(**struct evmgr\_socket **\***socket**,**

socket\_readable\_callback read\_callback**,**

socket\_writeable\_callback write\_callback**,**

socket\_event\_callback event\_callback**,**

void **\***callback\_ctx**);**

/\*\*

\* @brief write data to socket

\* @ingroup evmgr

\*

\* @param socket target socket

\* @param data data to be written

\* @param data\_size size of the data

\*

\* @return 0 on success, -1 otherwise

\*/

int evmgr\_socket\_write**(**struct evmgr\_socket **\***socket**,** const void **\***data**,**

size\_t data\_size**);**

/\*\*

\* @brief read data from socket

\* @ingroup evmgr

\*

\* @param socket source socket

\* @param buffer buffer for storing the data

\* @param buffer\_size size of the buffer

\*

\* @return actual size read from socket

\*/

size\_t evmgr\_socket\_read**(**struct evmgr\_socket **\***socket**,** void **\***buffer**,**

size\_t buffer\_size**);**

/\*\*

\* @brief destroy a socket

\* @ingroup evmgr

\*

\* @param socket socket to destroy

\*/

void evmgr\_socket\_destroy**(**struct evmgr\_socket **\***socket**);**

**Memory Management**

Memory management API abstracts calls to allocate and free memory. On Linux, it’s a simple malloc/calloc/free wrapper.

The API is:

/\* allocate memory the same way as POSIX malloc does \*/

void **\***mem\_malloc**(**size\_t size**);**

/\* free memory the same way as POSIX free does \*/

void mem\_free**(**void **\***ptr**);**

/\* allocate memory the same way as POSIX calloc does \*/

void **\***mem\_calloc**(**size\_t nmemb**,** size\_t size**);**

/\* reallocate memory the same way as POSIX realloc does \*/

void **\***mem\_realloc**(**void **\***ptr**,** size\_t size**);**

**Configuration Management**

Configuration manager is platform dependent module which handles configuration changes. The goal is to have an API, which would allow easy integration with CLI. This is achieved by making the API asynchronous. SXP daemon provides callbacks to the configuration manager at initialization phase and then just listens for callbacks with new/changed/deleted configuration. Configuration manager watches for configuration changes (platform dependend - it might be a configuration file, the configuration could come from a CLI, from network, etc..) and notifies SXPD via callbacks.

The API is:

/\*\*

\* @brief maximum tcp-md5 signing password length

\* @ingroup config

\*/

#define CFG\_PASSWORD\_MAX\_SIZE 81

/\*\*

\* @brief implementation-specific configuration context

\* @ingroup config

\*/

struct cfg\_ctx**;**

struct sxpd\_ctx**;**

/\*\*

\* @brief peer type

\* @ingroup config

\*/

enum peer\_type **{**

PEER\_SPEAKER**,** /\*!< peer which acts like speaker \*/

PEER\_LISTENER**,** /\*!< peer which acts like listener \*/

PEER\_BOTH**,** /\*!< peer which acts like speaker as well as listener \*/

**};**

/\*\*

\* @brief peer configuration info

\* @ingroup config

\*/

struct peer **{**

in\_addr\_t ip\_address**;** /\*!< peer's ip address in network byte order \*/

bool port\_is\_set**;** /\*!< whether the port is set to meaningful value \*/

uint16\_t port**;** /\*!< peer's port in network byte order \*/

char **\***connection\_password**;** /\*!< connection password \*/

enum peer\_type peer\_type**;** /\*!< the type of the peer \*/

**};**

/\*\*

\* @brief type of ip prefix

\* @ingroup config

\*/

enum prefix\_type\_e **{**

PREFIX\_IPV4**,** /\*!< IPV4 \*/

PREFIX\_IPV6**,** /\*!< IPV6 \*/

**};**

/\*\*

\* @brief configured binding

\* @ingroup config

\*/

struct binding **{**

enum prefix\_type\_e type**;**

union **{**

uint32\_t prefix\_v4**;**

uint32\_t prefix\_v6**[**4**];**

**}** prefix**;**

uint8\_t prefix\_length**;**

uint16\_t source\_group\_tag**;**

**};**

#define TUPLE\_STR\_CFG\_DEF(SELECT\_FUNC) \

SELECT\_FUNC(STR\_SETTING\_PASSWORD, "default password")

#define TUPLE\_CFG\_SELECT\_ENUM(enumerator, string) enumerator,

**typedef** enum str\_seting\_type\_e **{**

TUPLE\_STR\_CFG\_DEF**(**TUPLE\_CFG\_SELECT\_ENUM**)**

STR\_SETTING\_LAST**,** /\* place holder \*/

**}** str\_setting\_type\_t**;**

/\*\*

\* @brief callback called when string configuration is added

\* @ingroup config

\*

\* @param ctx sxpd context

\* @param type setting type

\* @param value setting value

\*/

**typedef** int **(\***cfg\_add\_str\_setting\_callback**)(**struct sxpd\_ctx **\***ctx**,**

str\_setting\_type\_t type**,**

const char **\***value**);**

/\*\*

\* @brief callback called when string configuration is deleted

\* @ingroup config

\*

\* @param ctx sxpd context

\* @param type setting type

\*/

**typedef** int **(\***cfg\_del\_str\_setting\_callback**)(**struct sxpd\_ctx **\***ctx**,**

str\_setting\_type\_t type**);**

#define TUPLE\_UINT32\_CFG\_DEF(SELECT\_FUNC) \

SELECT\_FUNC(UINT32\_SETTING\_LOG\_LEVEL, "log\_level") \

SELECT\_FUNC(UINT32\_SETTING\_RETRY\_TIMER, "retry timer") \

SELECT\_FUNC(UINT32\_SETTING\_RECONCILIATION\_TIMER, "reconciliation timer") \

SELECT\_FUNC(UINT32\_SETTING\_LISTENER\_MIN\_HOLD\_TIME, \

"listener min hold time") \

SELECT\_FUNC(UINT32\_SETTING\_LISTENER\_MAX\_HOLD\_TIME, \

"listener max hold time") \

SELECT\_FUNC(UINT32\_SETTING\_SPEAKER\_MIN\_HOLD\_TIME, "speaker min hold time") \

SELECT\_FUNC(UINT32\_SETTING\_KEEPALIVE\_TIMER, "keep alive timer") \

SELECT\_FUNC(UINT32\_SETTING\_SUBNET\_EXPANSION\_LIMIT, \

"subnet expansion limit") \

SELECT\_FUNC(UINT32\_SETTING\_BIND\_ADDRESS, "bind address") \

SELECT\_FUNC(UINT32\_SETTING\_PORT, "port") \

SELECT\_FUNC(UINT32\_SETTING\_NODE\_ID, "node id") \

SELECT\_FUNC(UINT32\_SETTING\_ENABLED, "enabled")

**typedef** enum uin32\_setting\_type\_e **{**

TUPLE\_UINT32\_CFG\_DEF**(**TUPLE\_CFG\_SELECT\_ENUM**)** UINT32\_SETTING\_LAST

**}** uint32\_setting\_type\_t**;**

/\*\*

\* @brief return pretty print of setting name

\* @ingroup config

\*

\* @param type type to print

\*

\* @return string suitable for printing

\*/

const char **\***cfg\_get\_uint32\_setting\_str**(**uint32\_setting\_type\_t type**);**

/\*\*

\* @brief return pretty print of setting name

\* @ingroup config

\*

\* @param type type to print

\*

\* @return string suitable for printing

\*/

const char **\***cfg\_get\_str\_setting\_str**(**str\_setting\_type\_t type**);**

/\*\*

\* @brief callback called when uint32\_t configuration is added

\* @ingroup config

\*

\* @param ctx sxpd context

\* @param type setting type

\* @param value setting value

\*/

**typedef** int **(\***cfg\_add\_uint32\_setting\_callback**)(**struct sxpd\_ctx **\***ctx**,**

uint32\_setting\_type\_t type**,**

uint32\_t value**);**

/\*\*

\* @brief callback called when uint32\_t configuration is deleted

\* @ingroup config

\*

\* @param ctx sxpd context

\* @param type string setting type

\*/

**typedef** int **(\***cfg\_del\_uint32\_setting\_callback**)(**struct sxpd\_ctx **\***ctx**,**

uint32\_setting\_type\_t type**);**

/\*\*

\* @brief callback called when peer configuration is added

\* @ingroup config

\*

\* @param ctx sxpd context

\* @param peer added peer setting

\*/

**typedef** int **(\***cfg\_add\_peer\_callback**)(**struct sxpd\_ctx **\***ctx**,**

const struct peer **\***peer**);**

/\*\*

\* @brief callback called when peer configuration is deleted

\* @ingroup config

\*

\* @param ctx sxpd context

\* @param peer deleted peer setting

\*/

**typedef** int **(\***cfg\_del\_peer\_callback**)(**struct sxpd\_ctx **\***ctx**,**

const struct peer **\***peer**);**

/\*\*

\* @brief callback called when binding configuration is added

\* @ingroup config

\*

\* @param ctx sxpd context

\* @param binding added binding setting

\*/

**typedef** int **(\***cfg\_add\_binding\_callback**)(**struct sxpd\_ctx **\***ctx**,**

const struct binding **\***binding**);**

/\*\*

\* @brief callback called when binding configuration is deleted

\* @ingroup config

\*

\* @param ctx sxpd context

\* @param binding deleted binding setting

\*/

**typedef** int **(\***cfg\_del\_binding\_callback**)(**struct sxpd\_ctx **\***ctx**,**

const struct binding **\***binding**);**

/\*\*

\* @brief create configuration context

\* @ingroup config

\*

\* @param[out] ctx configuration context

\* @param cfg\_string configuration string i.e. file path

\* @param[out] evmgr\_settings event manager settings

\* @return returns 0 on success, -1 on error

\*/

int cfg\_ctx\_create**(**struct cfg\_ctx **\*\***ctx**,** const char **\***cfg\_string**,**

struct evmgr\_settings **\*\***evmgr\_settings**);**

/\*\*

\* @brief register configuration callbacks

\* @ingroup config

\*

\* @param ctx configuration context

\* @param evmgr event manager

\* @param sxpd\_ctx callback context

\* @param add\_uint\_cb callback called when uint32\_t configuration is added

\* @param del\_uint\_cb callback called when uint32\_t configuration is deleted

\* @param add\_str\_cb callback called when string configuration is added

\* @param del\_str\_cb callback called when string configuration is deleted

\* @param add\_peer\_cb callback called when peer configuration is added

\* @param del\_peer\_cb callback called when peer configuration is deleted

\* @param add\_binding\_cb callback called when binding configuration is added

\* @param del\_binding\_cb callback called when binding configuration is deleted

\* @return returns 0 on success, -1 on error

\*/

int cfg\_register\_callbacks**(**struct cfg\_ctx **\***ctx**,** struct evmgr **\***evmgr**,**

struct sxpd\_ctx **\***sxpd\_ctx**,**

cfg\_add\_uint32\_setting\_callback add\_uint\_cb**,**

cfg\_del\_uint32\_setting\_callback del\_uint\_cb**,**

cfg\_add\_str\_setting\_callback add\_str\_cb**,**

cfg\_del\_str\_setting\_callback del\_str\_cb**,**

cfg\_add\_peer\_callback add\_peer\_cb**,**

cfg\_del\_peer\_callback del\_peer\_cb**,**

cfg\_add\_binding\_callback add\_binding\_cb**,**

cfg\_del\_binding\_callback del\_binding\_cb**);**

/\*\*

\* @brief destroy configuration context

\* @ingroup config

\*

\* @param ctx sxpd context

\*/

void cfg\_ctx\_destroy**(**struct cfg\_ctx **\***ctx**);**

**Random numbers generator**

Random number generator uses the random generator provided by the platform to supply random numbers to SXPD. SXPD does not use the provided numbers for cryptography purposes, only for adding random jitter to keep-alive timer.

The API is:

/\*\*

\* @brief implementation-specific maximum random number returned

\* @ingroup rnd

\*/

extern const uint32\_t random\_max**;**

/\*\*

\* @brief initialize the random generator

\* @ingroup rnd

\*/

void random\_init**(**void**);**

/\*\*

\* @brief get a random number in the range [0 .. random\_max]

\* @ingroup rnd

\*

\* @return random number

\*/

uint32\_t random\_get**(**void**);**

**Syslog**

Syslog module provides an abstraction layer for logging messages to syslog and managing the log level.

The API is:

/\*\*

\* @brief log levels

\* @ingroup logging

\*/

enum log\_level **{**

LOG\_LEVEL\_ALERT **=** 1**,** /\*!< immediate user action is required \*/

LOG\_LEVEL\_ERROR**,** /\*!< error occured \*/

LOG\_LEVEL\_TRACE**,** /\*!< informational \*/

LOG\_LEVEL\_DEBUG**,** /\*!< debugging \*/

**};**

/\*\*

\* @brief set the log level to log only messages with level higher or equal to

\*the level set

\* @ingroup logging

\*

\* @param loglevel level to set

\*/

void log\_setloglevel**(**enum log\_level loglevel**);**

/\*\*

\* @brief get log level string by log level enum

\* @ingroup logging

\*

\* @param l log level

\* @return log level string

\*/

const char **\***log\_level\_to\_string**(**enum log\_level l**);**

/\*\*

\* @brief get log level enum by log level string

\* @ingroup logging

\*

\* @param loglevel log level

\* @param parse\_from string to be parsed

\* @return returns 0 on success, -1 on error

\*/

int parse\_log\_level**(**enum log\_level **\***loglevel**,** const char **\***parse\_from**);**

/\*\*

\* @brief write message to system log

\* @ingroup logging

\*

\* @param loglevel importance of the message

\* @param function\_name function where the log message originates from

\* @param file\_name file where the function is located

\* @param file\_line line in the file on which the message is generated

\* @param format printf-like format string

\* @param ... printf-like arguments to formatted

\*/

void log\_syslog**(**enum log\_level loglevel**,** const char **\***function\_name**,**

const char **\***file\_name**,** size\_t file\_line**,** const char **\***format**,**

**...)** \_\_attribute\_\_**((**format**(**printf**,** 5**,** 6**)));**

**DBUS interface**

DBUS interface allows the user to fetch information about the SXPD state such as global settings, peer database and bindings database.

**SXPD main code setup**

This is the module responsible for starting the daemon. It parses command line parameters, creates event manager instance and config manager instance and passes these to the sxpd setup call. Once SXPD instance is created, it enters the event manager loop to run the instance. This separation allows one to implement more than one SXPD instance in one daemon, should it be necessary (for example to support multiple VRFs).

**SXPD daemon core logic**

SXP daemon core logic contains all the processing logic including:

1. Handling peers
   1. Connection management
   2. Peer bindings database
   3. All the necessary timers
2. Handling bindings database
3. Expanding subnets to host entries when configured
4. Inspecting the state of SXPD – peers, bindings and global config

For this the SXP daemon uses the mentioned abstraction layers (event manager, logging, mem, etc), plus SXP protocol parser and radix tree implementation.

The API provided is:

/\*\*

\* @brief create sxpd context

\*

\* @param evmgr event manager context to use

\* @param evmgr\_settings event manager settings passed to event manager when

\* invoking event manager API calls

\* @param default\_loglevel default log level

\*

\* @return context pointer on success, NULL on error

\*/

struct sxpd\_ctx **\***sxpd\_create**(**struct evmgr **\***evmgr**,**

struct evmgr\_settings **\***evmgr\_settings**,**

enum log\_level default\_loglevel**);**

/\*\*

\* @brief return event manager context

\*

\* @param ctx sxpd context

\*

\* @return event manager context stored in sxpd\_context

\*/

struct evmgr **\***sxpd\_get\_evmgr**(**struct sxpd\_ctx **\***ctx**);**

/\*\*

\* @brief return event manager settings

\*

\* @param ctx sxpd context

\*

\* @return event manager settings stored in sxpd\_context

\*/

struct evmgr\_settings **\***sxpd\_get\_evmgr\_settings**(**struct sxpd\_ctx **\***ctx**);**

/\*\*

\* @brief register configuration callbacks

\*

\* @param ctx sxpd context

\* @param cfg\_ctx configuration context

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_register\_config**(**struct sxpd\_ctx **\***ctx**,** struct cfg\_ctx **\***cfg\_ctx**);**

/\*\*

\* @brief destroy sxpd context and free resources

\*

\* @param ctx context to destroy

\*/

void sxpd\_destroy**(**struct sxpd\_ctx **\***ctx**);**

enum ip\_type **{** V4**,** V6 **};**

/\*\*

\* @brief search the master bindings database for best match for given prefix

\*and return the corresponding tag

\*

\* @param[in] ctx sxpd context

\* @param[in] ip\_type type - V4 or V6

\* @param[in] prefix prefix bits

\* @param[in] length length of the prefix

\* @param[out] tag set to corresponding tag, if found

\* @param[out] found set to true if found, otherwise false

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_search\_best**(**struct sxpd\_ctx **\***ctx**,** enum ip\_type ip\_type**,**

uint8\_t **\***prefix**,** uint8\_t length**,** uint16\_t **\***tag**,**

bool **\***found**);**

/\*\*

\* @brief opaque sxpd bindings iterator

\*/

struct sxpd\_bindings\_iterator**;**

/\*\*

\* @brief start or continue iterating the bindings database

\*

\* this function iterates the master bindings database

\* start the iteration by providing pointer to context set to NULL

\* when sxpd\_iterate\_bindings finishes, if \*context is NULL, then there are

\* no more bindings available, if \*context is non-NULL, then a binding was

\* was stored in buffer, length, tag

\*

\* there is NO guarantee on the consistency of the results if there are changes

\* in the bindings set while iterating, some bindings which are present COULD be

\* skipped if the bindings set is changed during the iteration

\*

\* @param ctx sxpd context

\* @param ip\_type V4 for IPv4 bindings, V6 for IPv6

\* @param context pointer to iterator, allocated by sxpd\_iterate\_bindings

\* @param buffer buffer to store the prefix bits

\* @param buffer\_size size of the buffer

\* @param length the number of prefix bits the prefix has

\* @param tag the associated source group tag

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_iterate\_bindings**(**struct sxpd\_ctx **\***ctx**,** enum ip\_type ip\_type**,**

struct sxpd\_bindings\_iterator **\*\***context**,**

uint8\_t **\***buffer**,** size\_t buffer\_size**,** uint8\_t **\***length**,**

uint16\_t **\***tag**);**

/\*\*

\* @brief stop iterating bindings and free iterator

\*

\* @param ctx sxpd context

\* @param iterator iterator returned by sxpd\_iterate\_bindings

\*/

void sxpd\_iterate\_bindings\_finish**(**struct sxpd\_ctx **\***ctx**,**

struct sxpd\_bindings\_iterator **\***iterator**);**

/\*\*

\* @brief opaque sxpd peers iterator

\*/

struct sxpd\_peer\_iterator**;**

/\*\*

\* @brief sxpd peer outgoing connection state

\*/

enum sxpd\_peer\_out\_conn\_state **{**

NONE**,** /\*!< no connection \*/

WAITING\_CONNECT**,** /\*!< waiting for TCP connect to finish \*/

WILL\_SEND\_OPEN**,** /\*!< need to send OPEN message \*/

WAITING\_OPEN**,** /\*!< waiting for OPEN message \*/

WILL\_SEND\_OPEN\_RESP**,** /\*!< need to send OPEN\_RESP message \*/

WAITING\_OPEN\_RESP**,** /\*!< waiting for OPEN\_RESP message \*/

CONNECTED**,** /\*!< connected \*/

ERROR\_CONNECT**,** /\*!< error while connecting \*/

CONNECT\_RETRY\_TIMER**,** /\*!< outgoing connection retry timer running \*/

**};**

#define SXPD\_PEER\_STATE\_ENUMERATOR(F) \

F(NONE) F(WAITING\_CONNECT) F(WILL\_SEND\_OPEN) F(WAITING\_OPEN) \

F(WILL\_SEND\_OPEN\_RESP) F(WAITING\_OPEN\_RESP) F(CONNECTED) \

F(ERROR\_CONNECT) F(CONNECT\_RETRY\_TIMER)

#ifdef SXPD\_PEER\_STATE\_ENUMERATOR

/\* prevent unused macro warning \*/

#endif

/\*\*

\* @brief structure describing peer

\*/

struct sxpd\_peer\_info **{**

uint32\_t nbo\_ip**;** /\*!< IP address \*/

uint16\_t nbo\_port**;** /\*!< port \*/

size\_t connections\_count**;** /\*!< how many connections are active with this

peer \*/

enum sxpd\_peer\_out\_conn\_state

outgoing\_connection\_state**;** /\*!< outgoing connection state \*/

bool retry\_timer\_active**;** /\*!< retry timer armed ? \*/

bool delete\_hold\_down\_timer\_active**;** /\*!< delete hold down timer armed ? \*/

bool reconciliation\_timer\_active**;** /\*!< reconciliation timer armed ? \*/

bool keepalive\_timer\_active**;** /\*!< keepalive timer armed ? \*/

bool hold\_timer\_active**;** /\*!< hold timer armed ? \*/

bool is\_speaker**;** /\*!< does this peer have speaker role ? \*/

bool is\_listener**;** /\*!< does this peer have listener role ? \*/

**};**

/\*\*

\* @brief start or continue iterating the peers

\*

\* this function iterates the peers

\* start the iteration by providing pointer to context set to NULL

\* when sxpd\_iterate\_peers finishes, if \*context is NULL, then there are no more

\* peers available, otherwise peer info is filled in peer

\*

\* there is NO guarantee on the consistency of the results if there are changes

\* in the peer set while iterating peers, some peers MIGHT be skipped and a peer

\* COULD be returned twice if the peer set is reconfigured during iteration,

\* client should use the peer->nbo\_ip/peer->port as the unique identifier

\* of the peer

\*

\* @param ctx sxpd context

\* @param context pointer to iterator, allocated by sxpd\_iterate\_peers

\* @param peer peer info

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_iterate\_peers**(**struct sxpd\_ctx **\***ctx**,**

struct sxpd\_peer\_iterator **\*\***context**,**

struct sxpd\_peer\_info **\***peer**);**

/\*\*

\* @brief stop iterating peers and free iterator

\*

\* @param ctx sxpd context

\* @param iterator iterator returned by sxpd\_iterate\_peers

\*/

void sxpd\_iterate\_peers\_finish**(**struct sxpd\_ctx **\***ctx**,**

struct sxpd\_peer\_iterator **\***iterator**);**

/\*\*

\* @brief sxpd status

\*/

struct sxpd\_info **{**

uint32\_t nbo\_bind\_ip**;** /\*!< IP to which sxpd is bound to \*/

uint16\_t nbo\_port**;** /\*!< port used by sxpd \*/

const char **\***default\_connection\_password**;** /\*!< default connection password \*/

size\_t peer\_count**;** /\*!< number of peers configured \*/

size\_t expanded\_entry\_count**;** /\*!< number of expanded host entries \*/

bool enabled**;** /\*!< daemon enabled ? \*/

**};**

/\*\*

\* @brief get sxpd runtime information

\*

\* @param ctx sxpd context

\* @param info sxpd info

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_get\_info**(**struct sxpd\_ctx **\***ctx**,** struct sxpd\_info **\***info**);**

/\*\*

\* @brief add string type configuration option to sxpd

\*

\* @param ctx sxpd context

\* @param type type of setting

\* @param value value of setting

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_cfg\_add\_str\_setting**(**struct sxpd\_ctx **\***ctx**,** str\_setting\_type\_t type**,**

const char **\***value**);**

/\*\*

\* @brief withdraw string setting from sxpd

\*

\* @param ctx sxpd context

\* @param type type of setting

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_cfg\_del\_str\_setting**(**struct sxpd\_ctx **\***ctx**,** str\_setting\_type\_t type**);**

/\*\*

\* @brief add uint32 type configuration option to sxpd

\*

\* @param ctx sxpd context

\* @param type type of setting

\* @param value value of setting

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_cfg\_add\_uint32\_setting**(**struct sxpd\_ctx **\***ctx**,**

uint32\_setting\_type\_t type**,** uint32\_t value**);**

/\*\*

\* @brief withdraw uint32 setting from sxpd

\*

\* @param ctx sxpd context

\* @param type type of setting

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_cfg\_del\_uint32\_setting**(**struct sxpd\_ctx **\***ctx**,**

uint32\_setting\_type\_t type**);**

/\*\*

\* @brief add new peer to sxpd

\*

\* @param ctx sxpd context

\* @param peer peer info

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_cfg\_add\_peer**(**struct sxpd\_ctx **\***ctx**,** const struct peer **\***peer**);**

/\*\*

\* @brief remove peer from sxpd

\*

\* @param ctx sxpd context

\* @param peer peer info

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_cfg\_del\_peer**(**struct sxpd\_ctx **\***ctx**,** const struct peer **\***peer**);**

/\*\*

\* @brief add binding to sxpd

\*

\* @param ctx sxpd context

\* @param binding binding info

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_cfg\_add\_binding**(**struct sxpd\_ctx **\***ctx**,** const struct binding **\***binding**);**

/\*\*

\* @brief remove binding from sxpd

\*

\* @param ctx sxpd context

\* @param binding binding info

\*

\* @return 0 on success, -1 on error

\*/

int sxpd\_cfg\_del\_binding**(**struct sxpd\_ctx **\***ctx**,** const struct binding **\***binding**);**

**SXP protocol parser**

This module contains functions for creating and parsing SXP messages and attributes contained in the messages in a pre-allocated buffer. It also contains functions for pretty-printing the messages. This module is used by the core daemon logic to create and parse SXP messages.

The API which this module provides is:

/\*\*

\* @brief maximum length of sxp message

\*/

#define SXP\_MAX\_MSG\_LENGTH (4096)

/\*\*

\* @brief sxp message type

\*/

enum sxp\_msg\_type **{**

SXP\_MSG\_OPEN **=** 1**,** /\*!< OPEN message \*/

SXP\_MSG\_OPEN\_RESP **=** 2**,** /\*!< OPEN\_RESP message \*/

SXP\_MSG\_UPDATE **=** 3**,** /\*!< UPDATE message \*/

SXP\_MSG\_ERROR **=** 4**,** /\*!< ERROR message \*/

SXP\_MSG\_PURGE\_ALL **=** 5**,** /\*!< PURGE\_ALL message \*/

SXP\_MSG\_KEEPALIVE **=** 6**,** /\*!< KEEPALIVE message \*/

**};**

#define SXP\_MSG\_TYPE\_ENUMERATOR(F) \

F(SXP\_MSG\_OPEN) F(SXP\_MSG\_OPEN\_RESP) F(SXP\_MSG\_UPDATE) F(SXP\_MSG\_ERROR) \

F(SXP\_MSG\_PURGE\_ALL) F(SXP\_MSG\_KEEPALIVE)

/\*\*

\* @brief return string describing sxp message type

\*

\* @param t message type

\*

\* @return string describing the sxp message type

\*/

const char **\***sxp\_msg\_type\_string**(**enum sxp\_msg\_type t**);**

/\*\*

\* @brief sxp extended error code

\*/

enum sxp\_error\_code **{**

SXP\_ERR\_CODE\_NONE **=** 0**,**

SXP\_ERR\_CODE\_MSG\_HEAD **=** 1**,**

SXP\_ERR\_CODE\_OPEN **=** 2**,**

SXP\_ERR\_CODE\_UPDATE **=** 3**,**

**};**

/\*\*

\* @brief sxp extended error sub-code

\*/

enum sxp\_error\_sub\_code **{**

SXP\_SUB\_ERR\_CODE\_NONE **=** 0**,**

SXP\_SUB\_ERR\_CODE\_MALFORMED\_ATTRIBUTE\_LIST **=** 1**,**

SXP\_SUB\_ERR\_CODE\_UNEXPECTED\_ATTRIBUTE **=** 2**,**

SXP\_SUB\_ERR\_CODE\_MISSING\_WELL\_KNOWN\_ATTRIBUTE **=** 3**,**

SXP\_SUB\_ERR\_CODE\_ATTRIBUTE\_FLAGS\_ERROR **=** 4**,**

SXP\_SUB\_ERR\_CODE\_ATTRIBUTE\_LENGTH\_ERROR **=** 5**,**

SXP\_SUB\_ERR\_CODE\_MALFORMED\_ATTRIBUTE **=** 6**,**

SXP\_SUB\_ERR\_CODE\_OPTIONAL\_ATTRIBUTE\_ERROR **=** 7**,**

SXP\_SUB\_ERR\_CODE\_UNSUPPORTED\_VERSION\_NUMBER **=** 8**,**

SXP\_SUB\_ERR\_CODE\_UNSUPPORTED\_OPTIONAL\_ATTRIBUTE **=** 9**,**

SXP\_SUB\_ERR\_CODE\_UNACCEPTABLE\_HOLD\_TIME **=** 10**,**

**};**

/\*\*

\* @brief sxp non-extended error code

\*/

enum sxp\_error\_non\_extended\_code **{**

SXP\_NON\_EXT\_ERR\_CODE\_NONE **=** 0**,**

SXP\_NON\_EXT\_ERR\_CODE\_VERSION\_MISMATCH **=** 1**,**

SXP\_NON\_EXT\_ERR\_CODE\_MESSAGE\_PARSE\_ERROR **=** 2**,**

**};**

/\*\*

\* @brief return the string representation of error code

\*

\* @param code code to represent

\*

\* @return string representing the code

\*/

const char **\***sxp\_error\_code\_string**(**uint8\_t code**);**

/\*\*

\* @brief return the string representation of error subcode

\*

\* @param subcode subcode to represent

\*

\* @return string representing the subcode

\*/

const char **\***sxp\_error\_subcode\_string**(**uint8\_t subcode**);**

/\*\*

\* @brief return the string representation of non-extended error code

\*

\* @param code code to represent

\*

\* @return string representing the code

\*/

const char **\***sxp\_error\_non\_extended\_code\_string**(**uint32\_t code**);**

/\*\*

\* @brief minimum value for hold-time minimum

\*/

#define HOLD\_TIME\_MINIMUM\_MINIMUM (3)

/\*\*

\* @brief value which indicates that keepalive mechanism is unused

\*/

#define KEEPALIVE\_UNUSED (0xffff)

static inline int sxp\_isok**(**int rc**,** enum sxp\_error\_code code**,**

enum sxp\_error\_sub\_code subcode**)**

**{**

**return** **(**RC\_ISOK**(**rc**)** **&&** code **==** SXP\_ERR\_CODE\_NONE **&&**

subcode **==** SXP\_SUB\_ERR\_CODE\_NONE**);**

**}**

static inline int sxp\_isnotok**(**int rc**,** enum sxp\_error\_code code**,**

enum sxp\_error\_sub\_code subcode**)**

**{**

**return** **!**sxp\_isok**(**rc**,** code**,** subcode**);**

**}**

/\*\*

\* @brief sxp message header

\*/

struct sxp\_msg **{**

uint32\_t length**;**

uint32\_t type**;**

**};**

/\*\*

\* @brief opaque sxp attribute structure

\*/

struct sxp\_attribute**;**

/\*\*

\* @brief create sxp error message in buffer of given size

\*

\* @param buffer buffer to hold the error message, must be large enough

\* @param buffer\_size size of the buffer allocated

\* @param code error code

\* @param sub\_code error sub-code

\* @param err\_attr attribute causing the error - included in error message, may

\*be NULL

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_create\_error\_extended**(**void **\***buffer**,** size\_t buffer\_size**,**

enum sxp\_error\_code code**,**

enum sxp\_error\_sub\_code sub\_code**,**

struct sxp\_attribute **\***err\_attr**);**

/\*\*

\* @brief create sxp error message in buffer of given size

\*

\* @param buffer buffer to hold the error message, must be large enough

\* @param buffer\_size size of the buffer allocated

\* @param code error code

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_create\_error\_basic**(**void **\***buffer**,** size\_t buffer\_size**,**

enum sxp\_error\_non\_extended\_code code**);**

/\*\*

\* @brief parse sxp error message

\*

\* @param msg message to parse

\* @param extended if set to 0, then non\_extended\_code is set, otherwise

\*code/sub\_code are set

\* @param code code stored in error message

\* @param sub\_code sub-code stored in error message

\* @param non\_extended\_code non-extended-code stored in error message

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_parse\_error**(**struct sxp\_msg **\***msg**,** int **\***extended**,**

enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***sub\_code**,**

enum sxp\_error\_non\_extended\_code **\***non\_extended\_code**);**

/\*\*

\* @brief sxp mode

\*/

enum sxp\_mode **{**

SXP\_MODE\_SPEAKER **=** 1**,** /\*!< SPEAKER mode \*/

SXP\_MODE\_LISTENER **=** 2**,** /\*!< LISTENER mode \*/

**};**

/\*\*

\* @brief return string describing sxp mode

\*

\* @param m mode

\*

\* @return string describing sxp mode

\*/

const char **\***sxp\_mode\_string**(**enum sxp\_mode m**);**

/\*\*

\* @brief create open v4 message in buffer of given size

\*

\* @param buffer buffer to hold the message, must be large enough

\* @param size buffer size

\* @param mode sxp mode to set in the message

\* @param node\_id node-id to set in the message

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_create\_open\_v4**(**void **\***buffer**,** size\_t size**,** enum sxp\_mode mode**,**

uint32\_t node\_id**);**

/\*\*

\* @brief create open resp message in buffer of given size

\*

\* @param buffer buffer to hold the message, must be large enough

\* @param version sxp version to declare inside open resp

\* @param size buffer size

\* @param mode sxp mode to set in the message

\* @param node\_id node-id to set in the message(if non-zero), if zero, node-id

\*attribute is not added to the message

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_create\_open\_resp**(**void **\***buffer**,** size\_t size**,** uint32\_t version**,**

enum sxp\_mode mode**,** uint32\_t node\_id**);**

/\*\*

\* @brief create keep-alive message in buffer of given size

\*

\* @param buffer buffer to hold the message, must be large enough

\* @param size buffer size

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_create\_keepalive**(**void **\***buffer**,** size\_t size**);**

/\*\*

\* @brief create purge-all message in buffer of given size

\*

\* @param buffer buffer to hold the message, must be large enough

\* @param size buffer size

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_create\_purge\_all**(**void **\***buffer**,** size\_t size**);**

/\*\*

\* @brief create update message in buffer of given size

\*

\* @param buffer buffer to hold the message, must be large enough

\* @param size buffer size

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_create\_update**(**void **\***buffer**,** size\_t size**);**

/\*\*

\* @brief sxp attribute type

\*/

enum sxp\_attr\_type **{**

SXP\_ATTR\_TYPE\_ADD\_IPV4 **=** 1**,** /\*!< ADD-IPV4 \*/

SXP\_ATTR\_TYPE\_ADD\_IPV6 **=** 2**,** /\*!< ADD-IPV6 \*/

SXP\_ATTR\_TYPE\_DEL\_IPV4 **=** 3**,** /\*!< DEL-IPV4 \*/

SXP\_ATTR\_TYPE\_DEL\_IPV6 **=** 4**,** /\*!< DEL-IPV6 \*/

SXP\_ATTR\_TYPE\_NODE\_ID **=** 5**,** /\*!< NODE-ID \*/

SXP\_ATTR\_TYPE\_CAPABILITIES **=** 6**,** /\*!< CAPABILITIES \*/

SXP\_ATTR\_TYPE\_HOLD\_TIME **=** 7**,** /\*!< HOLD-TIME \*/

SXP\_ATTR\_TYPE\_IPV4\_ADD\_PREFIX **=** 11**,** /\*!< IPV4-ADD-PREFIX \*/

SXP\_ATTR\_TYPE\_IPV6\_ADD\_PREFIX **=** 12**,** /\*!< IPV6-ADD-PREFIX \*/

SXP\_ATTR\_TYPE\_IPV4\_DEL\_PREFIX **=** 13**,** /\*!< IPV4-DEL-PREFIX \*/

SXP\_ATTR\_TYPE\_IPV6\_DEL\_PREFIX **=** 14**,** /\*!< IPV6-DEL\_PREFIX \*/

SXP\_ATTR\_TYPE\_PEER\_SEQUENCE **=** 16**,** /\*!< PEER-SEQUENCE \*/

SXP\_ATTR\_TYPE\_SGT **=** 17**,** /\*!< SOURCE-GROUP-TAG \*/

**};**

/\*\*

\* @brief return string representation of sxp attribute type

\*

\* @param e type to return string for

\*

\* @return string representing the type

\*/

const char **\***sxp\_attr\_type\_string**(**enum sxp\_attr\_type e**);**

/\*\*

\* @brief get the type of sxp attribute

\*

\* @param attr head to get type from

\* @param type pointer to storage for type

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_attr\_get\_type**(**const struct sxp\_attribute **\***attr**,**

enum sxp\_attr\_type **\***type**);**

/\*\*

\* @brief get node id from node id attribute

\*

\* @param attr node id attribute

\* @param node\_id pointer to storage for node id

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_attr\_node\_id\_get\_node\_id**(**const struct sxp\_attribute **\***attr**,**

uint32\_t **\***node\_id**);**

/\*\*

\* @brief get source group tag from source group tag attribute

\*

\* @param attr node id attribute

\* @param sgt pointer to storage for source group tag

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_attr\_sgt\_get\_sgt**(**const struct sxp\_attribute **\***attr**,** uint16\_t **\***sgt**);**

/\*\*

\* @brief returns true if hold time attribute contains maximum hold time value

\*

\* @param attr attribute to inspect

\*

\* @return true if present/false otherwise

\*/

bool sxp\_attr\_hold\_time\_has\_max\_val**(**const struct sxp\_attribute **\***attr**);**

/\*\*

\* @brief get the minimum hold time value from hold time attribute

\*

\* @param attr attribute to parse

\* @param min\_val pointer to storage for minimum hold time

\* @param[out] code error code found while processing attribute

\* @param[out] subcode error sub-code found while processing attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_attr\_hold\_time\_get\_min\_val**(**const struct sxp\_attribute **\***attr**,**

uint16\_t **\***min\_val**,** enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief get the maximum hold time value from hold time attribute

\*

\* @param attr attribute to parse

\* @param max\_val pointer to storage for maximum hold time

\* @param[out] code error code found while processing attribute

\* @param[out] subcode error sub-code found while processing attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_attr\_hold\_time\_get\_max\_val**(**const struct sxp\_attribute **\***attr**,**

uint16\_t **\***max\_val**,** enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief get version from SXP OPEN message

\*

\* @param msg message to process

\* @param version pointer to storage for version

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_open\_get\_version**(**const struct sxp\_msg **\***msg**,** uint32\_t **\***version**);**

/\*\*

\* @brief get mode from SXP OPEN message

\*

\* @param msg message to process

\* @param mode pointer to storage for mode

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_open\_get\_mode**(**const struct sxp\_msg **\***msg**,** enum sxp\_mode **\***mode**);**

/\*\*

\* @brief sxp capabilities

\*/

enum sxp\_capability\_code **{**

SXP\_CAPABILITY\_IPV4\_UNICAST **=** 1**,** /\*!< IPV4-UNICAST capability \*/

SXP\_CAPABILITY\_IPV6\_UNICAST **=** 2**,** /\*!< IPV6-UNICAST capability \*/

SXP\_CAPABILITY\_SUBNET\_BINDINGS **=** 3**,** /\*!< SUBNET-BINDINGS capability \*/

**};**

/\*\*

\* @brief opaque sxp capability structure

\*/

struct sxp\_capability**;**

/\*\*

\* @brief return the capability code of capability

\*

\* @param c capability

\* @param code capability code

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_capability\_get\_code**(**const struct sxp\_capability **\***c**,**

enum sxp\_capability\_code **\***code**);**

/\*\*

\* @brief return length of capability

\*

\* @param c capability

\* @param length capability length

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_capability\_get\_length**(**const struct sxp\_capability **\***c**,** uint8\_t **\***length**);**

/\*\*

\* @brief return pointer to capability value

\*

\* @param c capability

\* @param value value stored in capability

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_capability\_get\_value**(**const struct sxp\_capability **\***c**,**

const void **\*\***value**);**

/\*\*

\* @brief add new capabilities attribute to given message

\*

\* @param[in] msg message to modify

\* @param[in] buffer\_size size of the buffer which holds the message (usable

\*memory)

\* @param[out] capabilities pointer to newly initialized capabilities attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_capabilities**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

struct sxp\_attribute **\*\***capabilities**);**

/\*\*

\* @brief add hold time attribute to sxp message

\*

\* @param msg message to modify

\* @param buffer\_size size of the buffer which holds the message (usable

\*memory)

\* @param min\_val hold time minimum value

\* @param max\_val hold time maximum value, if max-val is set to

\*KEEPALIVE\_UNUSED, then it is not added to attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_hold\_time**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

uint16\_t min\_val**,** uint16\_t max\_val**);**

/\*\*

\* @brief add a new capability in capabilities

\*

\* @param msg message which holds the capabilities

\* @param buffer\_size size of the buffer which holds the message (usable memory)

\* @param capabilities pointer to capabilities within message

\* @param code capability code to add

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_capabilities\_add\_capability**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

struct sxp\_attribute **\***capabilities**,**

enum sxp\_capability\_code code**);**

/\*\*

\* @brief parse capabilities - get first/next capability

\*

\* @param[in] capabilities capabilities attribute to parse

\* @param[in] start if set to NULL, get first capability within capabilities,

\*otherwise get next capability

\* @param[out] next first or next capability depending on value of start or NULL

\*if no (more) capabilities present

\* @param[out] code error code found while processing attribute

\* @param[out] subcode error sub-code found while processing attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_parse\_capabilities**(**struct sxp\_attribute **\***capabilities**,**

struct sxp\_capability **\***start**,**

struct sxp\_capability **\*\***next**,**

enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief add source-group tag attribute to message

\*

\* @param msg message to modify

\* @param buffer\_size size of buffer holding the message

\* @param tag tag to add

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_sgt**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,** uint16\_t tag**);**

/\*\*

\* @brief return the size required for storing sgt attribute

\*

\* @return size in bytes

\*/

uint32\_t sxp\_calc\_sgt\_size**(**void**);**

/\*\*

\* @brief add peer sequence attribute to message

\*

\* @param[in] msg message to modify

\* @param[in] buffer\_size size of buffer holding the message

\* @param[in] sxp\_id\_count the number of sxp id elements which this sequence

\*will store

\* @param[out] sxp\_id\_arr address of array where the sxp id elements can be

\*filled

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_peer\_sequence**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

uint32\_t sxp\_id\_count**,** uint32\_t **\*\***sxp\_id\_arr**);**

/\*\*

\* @brief calculate the size which a peer-sequence takes

\*

\* @param sxp\_id\_count the number of elements in the peer sequence

\*

\* @return size in bytes

\*/

uint32\_t sxp\_calc\_peer\_sequence\_size**(**uint32\_t sxp\_id\_count**);**

/\*\*

\* @brief parse peer sequence attribute

\*

\* @param[in] peer\_sequence peer sequence attribute to parse

\* @param[out] sxp\_id\_count the number of sxp id elements in peer sequence

\* @param[out] sxp\_id\_arr address to array of sxp id elements

\* @param[out] code error code found while processing attribute

\* @param[out] subcode error sub-code found while processing attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_parse\_peer\_sequence**(**struct sxp\_attribute **\***peer\_sequence**,**

size\_t **\***sxp\_id\_count**,** const uint32\_t **\*\***sxp\_id\_arr**,**

enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief peer sequence validation

\*

\* @param nbo\_peer\_node\_id source peer node id

\* @param sxp\_id\_count number of node-id's

\* @param sxp\_id\_arr array of node-id's

\* @param[out] code error code found while processing attribute

\* @param[out] subcode error sub-code found while processing attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_validate\_peer\_sequence**(**uint32\_t nbo\_peer\_node\_id**,** size\_t sxp\_id\_count**,**

const uint32\_t **\***sxp\_id\_arr**,**

enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief add ipv4-add-prefix attribute to message

\*

\* @param[in] msg message to modify

\* @param[in] buffer\_size size of buffer holding the message

\* @param[out] attr address of ipv4-add-prefix attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_ipv4\_add\_prefix**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

struct sxp\_attribute **\*\***attr**);**

/\*\*

\* @brief return the size of ipv4-add-prefix attribute in bytes

\*

\* @return size in bytes

\*/

uint32\_t sxp\_calc\_ipv4\_add\_prefix\_size**(**void**);**

/\*\*

\* @brief add ipv6-add-prefix attribute to message

\*

\* @param[in] msg message to modify

\* @param[in] buffer\_size size of buffer holding the message

\* @param[out] attr address of ipv6-add-prefix attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_ipv6\_add\_prefix**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

struct sxp\_attribute **\*\***attr**);**

/\*\*

\* @brief return the size of ipv6-add-prefix attribute in bytes

\*

\* @return size in bytes

\*/

uint32\_t sxp\_calc\_ipv6\_add\_prefix\_size**(**void**);**

/\*\*

\* @brief add prefix to prefix-list-like attribute (ipv4/6-add/del-prefix)

\*

\* @param msg message whose part is the attribute

\* @param buffer\_size size of buffer holding the message

\* @param attr attribute to modify

\* @param length length of prefix in bits

\* @param prefix memory where prefix is stored

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_prefix\_list\_add\_prefix**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

struct sxp\_attribute **\***attr**,** uint8\_t length**,**

uint8\_t **\***prefix**);**

/\*\*

\* @brief return the size of prefix added to ipv4/6-prefix-list

\*

\* @param prefix\_len length of the prefix in bits

\*

\* @return size in bytes

\*/

uint32\_t sxp\_calc\_prefix\_size**(**uint8\_t prefix\_len**);**

/\*\*

\* @brief opaque sxp prefix structure

\*/

struct sxp\_prefix**;**

/\*\*

\* @brief parse prefix and extract values

\*

\* @param[in] prefix prefix to parse

\* @param[out] length extracted length

\* @param[out] buffer extracted prefix

\* @param[in] buffer\_size size of the buffer

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_parse\_prefix**(**struct sxp\_prefix **\***prefix**,** uint8\_t **\***buffer**,**

size\_t buffer\_size**,** uint8\_t **\***length**);**

/\*\*

\* @brief add ipv4-del-prefix attribute to message

\*

\* @param[in] msg message to modify

\* @param[in] buffer\_size size of buffer holding the message

\* @param[out] attr address of ipv4-del-prefix attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_ipv4\_del\_prefix**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

struct sxp\_attribute **\*\***attr**);**

/\*\*

\* @brief add ipv6-del-prefix attribute to message

\*

\* @param[in] msg message to modify

\* @param[in] buffer\_size size of buffer holding the message

\* @param[out] attr address of ipv6-del-prefix attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_ipv6\_del\_prefix**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

struct sxp\_attribute **\*\***attr**);**

/\*\*

\* @brief parse prefix-list (ipv4/6-add/del-prefix) - get first/next prefix

\*

\* @param[in] prefix\_list attribute to parse

\* @param[in] start if set to NULL, get first prefix, otherwise get next

\* @param[out] next first or next prefix depending on value of start or NULL

\*if no (more) prefixes present

\* @param[out] code error code found while processing attribute

\* @param[out] subcode error sub-code found while processing attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_parse\_prefix\_list**(**struct sxp\_attribute **\***prefix\_list**,**

struct sxp\_prefix **\***start**,** struct sxp\_prefix **\*\***next**,**

enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief parse add-ipv4 attribute

\*

\* @param add\_ipv4 attribute to parse

\* @param buffer storage for parsed ipv4 prefix

\* @param buffer\_size size of storage buffer

\* @param sgt storage for parsed source group tag

\* @param have\_prefix\_length flag set to true if prefix length is present

\* @param prefix\_length prefix length, if present

\* @param code error code found while processing attribute

\* @param subcode error sub-code found while processing attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_parse\_add\_ipv4**(**struct sxp\_attribute **\***add\_ipv4**,** uint8\_t **\***buffer**,**

size\_t buffer\_size**,** uint16\_t **\***sgt**,**

bool **\***have\_prefix\_length**,** uint8\_t **\***prefix\_length**,**

enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief parse add-ipv6 attribute

\*

\* @param add\_ipv6 attribute to parse

\* @param buffer storage for parsed ipv6 prefix

\* @param buffer\_size size of storage buffer

\* @param sgt storage for parsed source group tag

\* @param have\_prefix\_length flag set to true if prefix length is present

\* @param prefix\_length prefix length, if present

\* @param code error code found while processing attribute

\* @param subcode error sub-code found while processing attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_parse\_add\_ipv6**(**struct sxp\_attribute **\***add\_ipv6**,** uint8\_t **\***buffer**,**

size\_t buffer\_size**,** uint16\_t **\***sgt**,**

bool **\***have\_prefix\_length**,** uint8\_t **\***prefix\_length**,**

enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief parse del-ipv4 attribute

\*

\* @param del\_ipv4 attribute to parse

\* @param buffer storage for parsed ipv4 prefix

\* @param buffer\_size size of storage buffer

\* @param have\_prefix\_length flag set to true if prefix length is present

\* @param prefix\_length prefix length, if present

\* @param code error code found while processing attribute

\* @param subcode error sub-code found while processing attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_parse\_del\_ipv4**(**struct sxp\_attribute **\***del\_ipv4**,** uint8\_t **\***buffer**,**

size\_t buffer\_size**,** bool **\***have\_prefix\_length**,**

uint8\_t **\***prefix\_length**,** enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief parse del-ipv6 attribute

\*

\* @param del\_ipv6 attribute to parse

\* @param buffer storage for parsed ipv6 prefix

\* @param buffer\_size size of storage buffer

\* @param have\_prefix\_length flag set to true if prefix length is present

\* @param prefix\_length prefix length, if present

\* @param code error code found while processing attribute

\* @param subcode error sub-code found while processing attribute

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_parse\_del\_ipv6**(**struct sxp\_attribute **\***del\_ipv6**,** uint8\_t **\***buffer**,**

size\_t buffer\_size**,** bool **\***have\_prefix\_length**,**

uint8\_t **\***prefix\_length**,** enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief calculate the length of add-ipv4 attribute containing prefix

\*

\* @param prefix\_len length of the prefix

\*

\* @return size of the element

\*/

uint32\_t sxp\_calc\_add\_ipv4\_size**(**uint8\_t prefix\_len**);**

/\*\*

\* @brief calculate the length of del-ipv4 attribute containing prefix

\*

\* @param prefix\_len length of the prefix

\*

\* @return size of the element

\*/

uint32\_t sxp\_calc\_del\_ipv4\_size**(**uint8\_t prefix\_len**);**

/\*\*

\* @brief calculate the length of add-ipv6 attribute containing prefix

\*

\* @param prefix\_len length of the prefix

\*

\* @return size of the element

\*/

uint32\_t sxp\_calc\_add\_ipv6\_size**(**uint8\_t prefix\_len**);**

/\*\*

\* @brief calculate the length of del-ipv6 attribute containing prefix

\*

\* @param prefix\_len length of the prefix

\*

\* @return size of the element

\*/

uint32\_t sxp\_calc\_del\_ipv6\_size**(**uint8\_t prefix\_len**);**

/\*\*

\* @brief add add-ipv4 attribute to message

\*

\* @param msg message to modify

\* @param buffer\_size size of buffer holding the message

\* @param tag source group tag associated with the prefix

\* @param prefix\_len length of the prefix

\* @param prefix prefix

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_add\_ipv4**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,** uint16\_t tag**,**

uint8\_t prefix\_len**,** uint8\_t **\***prefix**);**

/\*\*

\* @brief add add-ipv6 attribute to message

\*

\* @param msg message to modify

\* @param buffer\_size size of buffer holding the message

\* @param tag source group tag associated with the prefix

\* @param prefix\_len length of the prefix

\* @param prefix prefix

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_add\_ipv6**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,** uint16\_t tag**,**

uint8\_t prefix\_len**,** uint8\_t **\***prefix**);**

/\*\*

\* @brief add del-ipv4 attribute to message

\*

\* @param msg message to modify

\* @param buffer\_size size of buffer holding the message

\* @param prefix\_len length of the prefix

\* @param prefix prefix

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_del\_ipv4**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

uint8\_t prefix\_len**,** uint8\_t **\***prefix**);**

/\*\*

\* @brief add del-ipv6 attribute to message

\*

\* @param msg message to modify

\* @param buffer\_size size of buffer holding the message

\* @param prefix\_len length of the prefix

\* @param prefix prefix

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_add\_del\_ipv6**(**struct sxp\_msg **\***msg**,** size\_t buffer\_size**,**

uint8\_t prefix\_len**,** uint8\_t **\***prefix**);**

/\*\*

\* @brief return string representation of capability code

\*

\* @param c capability code

\*

\* @return string representation

\*/

const char **\***sxp\_capability\_code\_string**(**enum sxp\_capability\_code c**);**

/\*\*

\* @brief swap the relevant fields of given SXP message from host to network

\* byte order

\*

\* @param msg message to swap

\* @param code error code found while processing message

\* @param subcode error sub-code found while processing message

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_hton\_swap**(**struct sxp\_msg **\***msg**,** enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief swap the relevant fields of given SXP message from network to host

\* byte order

\*

\* @param msg message to swap

\* @param code error code found while processing message

\* @param subcode error sub-code found while processing message

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_msg\_ntoh\_swap**(**struct sxp\_msg **\***msg**,** enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief pretty-print the message in host byte-order at trace level to log

\*

\* @param msg message to pretty-print

\*

\* @return 0 on success, -1 on error

\* @param code error code found while processing message

\* @param subcode error sub-code found while processing message

\*/

int sxp\_hbo\_pretty\_print\_msg**(**struct sxp\_msg **\***msg**,** enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

/\*\*

\* @brief parse message - get first/next attribute

\*

\* @param[in] msg message to parse

\* @param[in] start if set to NULL, get first attribute present in msg,

\*otherwise get next attribute following attribute at located at start

\* @param[out] next first or next attribute found in msg or NULL if no (more)

\*attributes present

\* @param[out] code error code found during message parsing

\* @param[out] subcode error sub code found during message parsing

\*

\* @return 0 on success, -1 on error

\*/

int sxp\_parse\_msg**(**struct sxp\_msg **\***msg**,** struct sxp\_attribute **\***start**,**

struct sxp\_attribute **\*\***next**,** enum sxp\_error\_code **\***code**,**

enum sxp\_error\_sub\_code **\***subcode**);**

### Radix tree

Radix tree is used in the SXP daemon for storing ipv4 and ipv6 prefixes with efficient insertion, deletion and lookup. This module is used by the core daemon logic to create per-peer and global bindings databases and also expanded entries database.

The API which this module provides is:

/\*\*

\* @brief opaque radix node structure

\*/

struct radix\_node**;**

/\*\*

\* @brief opaque radix tree structure

\*/

struct radix\_tree**;**

/\*\*

\* @brief allocate a new radix tree

\*

\* @return non-NULL tree if success, NULL on error

\*

\*/

struct radix\_tree **\***radix\_create**(**uint8\_t maxbits**);**

/\*\*

\* @brief callback called by radix\_destroy for each node's data before

\*destroying the node

\*

\* @param data pointer stored in radix node

\*/

**typedef** void **(\***radix\_data\_free\_cb**)(**void **\***data**);**

/\*\*

\* @brief free a radix tree

\*

\* @param tree tree to be freed

\* @param cb callback called for each node of the radix before the node is freed

\*/

void radix\_destroy**(**struct radix\_tree **\***tree**,** radix\_data\_free\_cb cb**);**

/\*\*

\* @brief store prefix with prefix length in the tree associated with value

\*overwriting any existing stored value

\*

\* @param[in] tree the tree to operate on

\* @param[in] prefix memory containing bits to be stored in the tree

\* @param[in] prefix\_len number of bits (at most prefix\_len / 8 + 1 bytes will

\*be

\* read from prefix)

\* @param[in] value value to associate with stored prefix (must not be NULL)

\* @param[out] node the node created/updated by the store command

\*

\* @return returns 0 on success, -1 if error occurs

\*/

int radix\_store**(**struct radix\_tree **\***tree**,** const uint8\_t **\***prefix**,**

uint8\_t prefix\_len**,** void **\***value**,** struct radix\_node **\*\***node**);**

/\*\*

\* @brief delete a node from radix tree and return next node

\*

\* @param tree tree to delete node from

\* @param node node to delete

\*

\* @return 0 on success, -1 on error

\*/

int radix\_delete\_node**(**struct radix\_tree **\***tree**,** struct radix\_node **\***node**);**

**typedef** int **(\***radix\_match\_cb**)(**struct radix\_node **\***node**,** void **\***ctx**);**

/\*\*

\* @brief remove all entries for which the callback function returns non-zero

\*

\* @param tree tree to balk

\* @param callback function which gets value and ctx and must return zero if the

\*node should be kept or non-zero otherwise

\* @param ctx context passed to callback function

\*

\* @return 0 on success, -1 on error

\*/

int radix\_delete\_matching**(**struct radix\_tree **\***tree**,** radix\_match\_cb callback**,**

void **\***ctx**);**

/\*\*

\* @brief search the tree for given prefix/length and return the associated node

\*

\* @param tree the tree to operate on

\* @param prefix memory containing bits to search for

\* @param prefix\_len number of bits (at most prefix\_len / 8 + 1 bytes will be

\* read from prefix)

\* @param[out] node radix node matching the search criteria

\*

\* @return 0 on success, -1 on error

\*/

int radix\_search**(**const struct radix\_tree **\***tree**,** const uint8\_t **\***prefix**,**

uint8\_t prefix\_len**,** struct radix\_node **\*\***node**);**

/\*\*

\* @brief search the tree for given prefix/length and return the node which has

\* the longest matching prefix

\*

\* @param tree the tree to operate on

\* @param prefix memory containing bits to search for

\* @param prefix\_len number of bits (at most prefix\_len / 8 + 1 bytes will be

\* read from prefix)

\* @param[out] result radix node matching the search criteria

\*

\* @return 0 on success, -1 on error

\*/

int radix\_search\_best**(**struct radix\_tree **\***tree**,** const uint8\_t **\***prefix**,**

uint8\_t prefix\_len**,** struct radix\_node **\*\***result**);**

/\*\*

\* @brief iterate to first or next node in the radix tree

\*

\* NOTE: modifying the tree invalidates any pointers returned from radix\_iterate

\*

\* @param[in] tree the tree to operate on

\* @param[in] node node returned by previous call of radix\_iterate or NULL if

\*starting iteration

\* @param[out] next first node if NULL == node or next node otherwise

\*

\* @return 0 on success, -1 on error

\*/

int radix\_iterate**(**const struct radix\_tree **\***tree**,** struct radix\_node **\***node**,**

struct radix\_node **\*\***next**);**

/\*\*

\* @brief value returned from radix\_next\_search\_cb indicating radix node should

\* be returned

\*/

#define RADIX\_MATCH (1)

/\*\*

\* @brief value returned from radix\_next\_search\_cb indicating radix node should

\* be skipped

\*/

#define RADIX\_SKIP (-1)

/\*\*

\* @brief callback callback used to find next node that matches callback

\*criteria

\*

\* @param node node to be validate

\* @param ctx context passed to radix\_iterate\_matching

\*

\* @return RADIX\_MATCH if node should be matched, RADIX\_SKIP, if skipped

\*/

**typedef** int **(\***radix\_next\_search\_cb**)(**const struct radix\_node **\***node**,** void **\***ctx**);**

/\*\*

\* @brief iterate to first or next node in the radix tree for which the callback

\*function returns non-zero

\*

\* NOTE: modifying the tree invalidates any pointers returned from radix\_iterate

\*

\* @param[in] tree the tree to operate on

\* @param[in] node node returned by previous call of radix\_iterate or NULL if

\*starting iteration

\* @param[out] next first node if NULL == node or next node otherwise

\* @param cb function which gets node, next node and ctx and must return zero if

\*the

\* next node match

\* @param ctx context passed to callback function

\*

\* @return 0 on success, -1 on error

\*/

int radix\_iterate\_matching**(**const struct radix\_tree **\***tree**,**

struct radix\_node **\***node**,** struct radix\_node **\*\***next**,**

radix\_next\_search\_cb cb**,** void **\***ctx**);**

/\*\*

\* @brief parse radix node

\*

\* @param[in] node node to parse, must not be NULL

\* @param[out] prefix\_buffer buffer to store prefix bits - if NULL then unused

\* @param[out] prefix\_buffer\_size size of the buffer in bytes - if prefix\_buffer

\*is NULL, then this value is not filled

\* @param[out] prefix\_length actual length of prefix stored in bits

\* @param[out] value value associated with node

\*

\* @return 0 on success, -1 on error

\*/

int radix\_parse\_node**(**const struct radix\_node **\***node**,** uint8\_t **\***prefix\_buffer**,**

size\_t prefix\_buffer\_size**,** uint8\_t **\***prefix\_length**,**

void **\*\***value**);**

/\*\*

\* @brief return the parent node to the given radix node

\*

\* @param node child node

\* @param parent parent node or NULL if no such node exists

\*

\* @return 0 on success, -1 on error

\*/

int radix\_get\_parent\_node**(**const struct radix\_node **\***node**,**

struct radix\_node **\*\***parent**);**

/\*\*

\* @brief set new value for given radix node

\*

\* @param node node to modify

\* @param value value to set

\*

\* @return 0 on success, -1 on error

\*/

int radix\_node\_set\_value**(**struct radix\_node **\***node**,** void **\***value**);**

**System Flow**

The SXP daemon is event based – events include timer events, configuration events, socket events, and signals.

Initialization code is responsible for setting up the daemon. It creates an event manager instance, registers configuration management and then enters event loop.

There are 5 types of timer events:

* Connection retry timer. This timer is setup for each peer whenever a failure occurs while doing any operation regarding that peer which results in tearing down the connection. Timer is canceled if a connection if brought up which results in having enough connections for that peer.
* Keep-alive timer. Timer is setup/active in case the keep-alive mechanism is negotiated for peers which are listeners. When it expires, a keep-alive message is sent to the peer.
* Hold timer. This timer is setup if keep-alive mechanism is negotiated for peers which are speaker. If hold timer expires, sxp daemon assumes the connection failed, tears down the appropriate socket and starts delete hold-down timer.
* Reconciliation timer. This timer is started when a speaker peer reconnects. At this time, a reconciliation timestamp is saved and when the timer fires, all bindings from this peer are traversed. Any binding, whose timestamp is older than the reconciliation timestamp is removed. (That means all bindings which were not advertised during the reconciliation timer).
* Delete hold-down timer. This timer is started when a connection to speaker peer is lost. When the timer expires, all bindings from this peer are deleted. The timer is cancelled when the peer reconnects.

Configuration events include configuration parameter change, peer change, and binding change.

* Parameter changes

The parameters are:

1. Default connection password (string)
2. Retry timer value in seconds
3. Reconciliation timer value in seconds
4. Hold timer minimum in seconds
5. Hold timer maximum in seconds
6. Keep alive timer timeout in seconds
7. Expansion counter limit
8. Bind address (address where the daemon binds to for listening)
9. Listening port
10. Node id for sxp daemon instance
11. Enable – global setting which says whether the daemon is enabled or disable

* Peer changes. Peers can be added, modified and removed. When a peer is added, an empty bindings database is created for it and connection attempt is started. When a peer is removed, all connections are torn down and all bindings from this peer are removed from the master bindings database.
* Binding change. Bindings can be added or deleted. When a binding is added, it is added to the local bindings database and propagated to the master bindings database. When its removed, it is removed from both these databases.

Socket events are:

* New connection. When a new connection is received, peer database is searched for the source ip to see if the peer is recognized. If not, then connection is closed, otherwise if the connection is needed (or the peers ip address is higher) then the connection is kept. This might cause existing connection (outgoing connection) to be torn down.
* Write. This event is utilized for error cases to detect whether an error message was written to the socket, if yes, then the connection is closed.
* Read. When a read event occurs, data are read into local buffer and if there is enough data to contain SXP message header, then this is parsed and if enough data are received so that a complete message is in the buffer, then that message is parsed. This is repeated multiple times if more messages are in the buffer.
* Error. This indicates that the other side closed the socket or some error occurred. The affected socket is destroyed and retry timer is scheduled if needed.
* Signals. Signal events are only used for debugging and/or platform specific handling. On Linux, SIGHUP causes the configuration management to reload the configuration file and call the appropriate callbacks for any changes detected. Also SIGQUIT causes the daemon to print debugging info to the standard output.

**SXP data structures**

The key SXP data structures are in sxpd\_internal.h. The daemon context, sxpd\_ctx, holds the run-time state of the SXP daemon. The SXP peer structure, sxpd\_peer, holds the run-time state of sxp peer. The SXP daemon mask, sxpd\_mask, is an array of bits which are allocated when set with unallocated bits considered to be set to zero. The SXPD binding represents bindings contributed by a peer. The SXPD binding list, sxpd\_binding\_list, represents contributing bindings in the master binding database. The Expansion track entry, sxpd\_expansion\_track\_entry, is the expanded host entry and contains the binding list, mask, and radix node.

The size of the data structures are taken using gdb with sxpd binary compiled on x86\_64 linux.

Reading symbols from linux/src/sxpd...done.

(gdb) p sizeof(struct sxpd\_ctx)

$1 = 368

(gdb) p sizeof(struct sxpd\_peer)

$2 = 280

(gdb) p sizeof(struct sxpd\_mask)

$3 = 24

(gdb) p sizeof(struct sxpd\_peer\_sequence)

$4 = 24

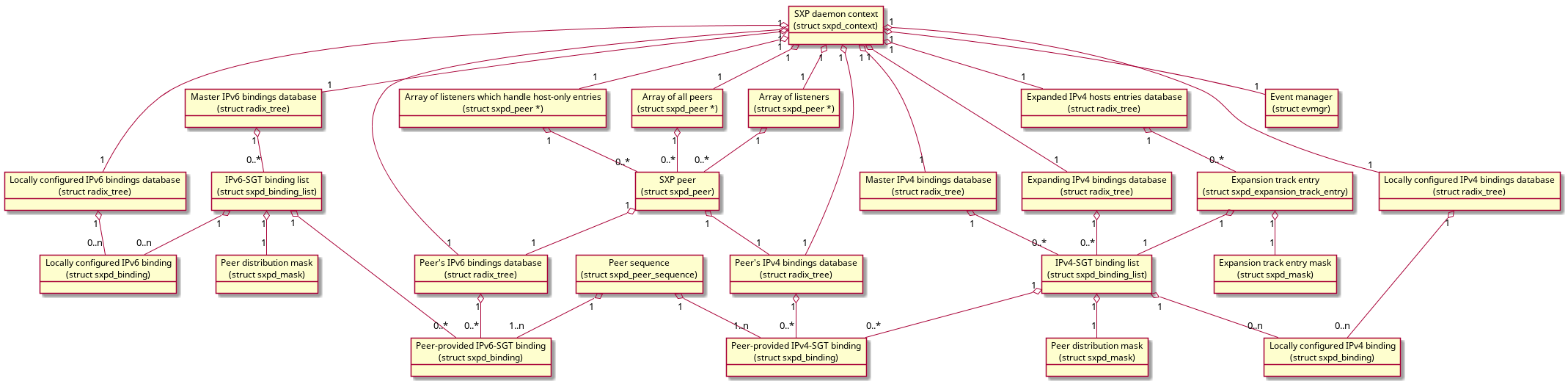
(gdb) p sizeof(struct sxpd\_binding\_list)

$5 = 72

(gdb) p sizeof(struct sxpd\_expansion\_track\_entry)

$6 = 56

**Data dependency diagram**



**Peer connection handling**

### The peer handling algorithms are concerned with:

### SXP daemon enabled/disabled. When daemon is disabled, all peer connections are torn down, listening socket is destroyed and no outgoing connections are attempted. When the state changes from disabled to enabled, listening socket is created and an outgoing connection attempt is made for every peer. SXP daemon then waits for event manager notifications about successful connection(s).

### SXP peer configured. When a peer is configured, outgoing connection attempt is made. SXP daemon then waits for event manager notifications about successful connection(s).

### SXP peer removed from configuration. When a peer is removed from configuration, all of its sockets are closed before destroying the peer structure and freeing memory. If the peer has listener role, then a PURGE-ALL message is sent to the peer on the corresponding socket before closing the connection(s).

**Bindings database**

There are several databases:

* IPv4 and IPv6 databases. SXP daemon stores bindings in separate databases (radix trees) according to IP type. There is one database for IPv4 bindings and one database for IPv6 bindings. This allows simplification when generating UPDATE messages to peers and efficient use of attributes due to ip-prefix aggregation. Separate databases are used in all cases where bindings are stored and the code flow for handling the bindings is the same for both cases with one exception – subnet expansion is done (when configured) only for IPv4 bindings.
* Local bindings database. SXP daemon stores locally configured IP-SGT bindings in two databases sxpd\_ctx.bindings\_v4 (for IPv4 bindings) and sxpd\_ctx.bindings\_v6 (for IPv6 bindings).
* Per-peer bindings database. SXP daemon stores bindings learned from speaker peers in two data databases sxpd\_peer.bindings\_v4 (for IPv4 bindings) and sxpd\_peer.bindings\_v6 (for IPv6 bindings).
* Master bindings database. SXP daemon contains two databases sxpd\_ctx.master\_bindings\_v4 (for IPv4 bindings) and sxpd\_ctx.master\_bindings\_v6 (for IPv6 bindings) for storing aggregated binding information. While local and per-peer bindings databases contain pointers to sxpd\_binding structures, which represent bindings, master bindings database contains pointers to sxpd\_binding\_list structures.
  + Binding list (sxpd\_binding\_list) is a structure which contains a list of contributing bindings for a given network prefix. When two different peers (or a peer and local configuration) advertise the same network prefix, two different bindings are created - one per each peer. These are then sorted according to either peer-sequence length rules or timestamps in the binding list. This binding list is stored in the master bindings database with the same network prefix as the bindings as key. The first binding in the binding list is the binding which is selected for export when sending UPDATE message to peers.
* Expanded bindings & expansion track entries. If subnet expansion is configured, then whenever an IPv4 binding is received, it’s number of host entries is calculated and if further expansion ‘slots’ are available, then this binding is expanded, otherwise an error message is logged. Expanded binding list is stored in the expanded bindings database (sxpd\_ctx.expand\_bindings\_v4). The host entries which are generated from the binding are stored as expansion track entries in the expansion track entry database (sxpd\_ctx.expand\_entries\_v4). Expansion track entries correspond to host entries and contain the peer distribution masks themselves.

When a binding is updated (with new SGT or so), the corresponding expansion track entries’ masks’ bits are reset.

When an already expanded binding list is updated, the expansion track entries are updated accordingly.

When an expansion track entry is created, the binding list to which it points is taken from the longest matching prefix. So, if there are two subnets which both contain the same host entry, the host entry is taken from the subnet with longer prefix.

**Bindings export**

Creation of new binding list, removal of all elements from binding list or a change on the first position in the binding list all trigger bindings export (sxpd\_export\_bindings).

Each binding list contains a distribution mask (sxpd\_mask). The mask is on-demand allocated structure, which holds bits corresponding to peers. Each listener peer has an index allocated (sxpd\_peer.listener\_bit\_pos), which specifies the corresponding bit in distribution mask. When a bit in distribution mask is set to “1”, it means that the corresponding peer has the current version of IP-SGT binding for the network prefix under which the binding list is stored. Bits in the mask are set to “1” when an UPDATE message containing the IP-SGT binding is successfully sent to peer. Bits in the mask are set to “0”, when the binding list gets empty or when the first position in the binding list changes.

**Exporting bindings to a listener peer**

#### Common logic when exporting items to peer

When exporting bindings or host entries to peer, whether is add or delete, the common code flow is like this:

1. First an empty UPDATE message is created.
2. Next item for export is found and the size needed to store this item in the UPDATE message is calculated.
3. If there is enough room to store the item, then the item is added to the message and to a linked list of elements in message. This linked list is not allocated; each item holds an internal “next” pointer which is reserved for this use. Only the pointer to the first element is thus needed.
4. If adding this item would exceed the maximum message size, then the existing UPDATE message is sent to the client and a new empty UPDATE message is created before proceeding further.
5. When sending the UPDATE message to client, once the message was successfully written to the socket, the linked list containing elements in message is iterated and bit mask of each item in the linked list is updated to “1” at position corresponding to the peer for which the UPDATE message is being generated. This ensures that the masks are always correctly update even in the event if the UPDATE message is dropped because it cannot be written to the socket (e.g. if the buffer is full or so).

When bindings export is triggered, the array of listener peers is traversed and a write callback is registered for corresponding listener socket for each peer. Once the socket becomes writable, an export for that peer is performend. This export is done in these steps:

#### Exporting delete updates for IPv4 bindings

The master bindings database is iterated and each binding list which is not empty is skipped. When an empty binding list is found, the listener bit is checked in the binding list’s mask. If this bit is not set, then the corresponding IP-SGT prefix is to be exported as deleted to the peer.

Bindings which are non-host (networks) are not exported to peer which does not support subnet bindings.

#### Exporting delete updates for IPv6 bindings

The master bindings database is iterated and each binding list which is not empty is skipped. When an empty binding list is found, the listener bit is checked in the binding list’s mask. If this bit is not set, then the corresponding IP-SGT prefix is to be exported as deleted to the peer.

IPv6 bindings are not exported to peers which don’t support IPv6 bindings.

#### Exporting add updates for IPv4 bindings

The master bindings database is iterated and each binding list which is empty is skipped. When a non-empty binding list is found, the listener bit is checked in the binding list’s mask. If this bit is not set, then the corresponding IP-SGT prefix is to be exported as added to the peer.

Bindings which are non-host (networks) are not exported to peer which does not support subnet bindings.

#### Exporting add updates for IPv6 bindings

The master bindings database is iterated and each binding list which is empty is skipped. When a non-empty binding list is found, the listener bit is checked in the binding list’s mask. If this bit is not set, then the corresponding IP-SGT prefix is to be exported as added to the peer.

IPv6 bindings are not exported to peers which don’t support IPv6 bindings.

#### Exporting delete IPv4 expanded host entries

This step is performed only for peers which do not advertise subnet handling capability.

Expansion track entries database is iterated and each expansion track entry which points to non-empty binding list is skipped. When a empty expansion track entry/binding list is found, the expansion bit is checked in the expansion track entry’s mask. If this bit is not set, then the corresponding expanded IP-SGT host entry is to be exported as deleted to the peer.

#### Exporting add IPv4 expanded host entries

This step is performed only for peers which do not advertise subnet handling capability.

Expansion track entries database is iterated and each expansion track entry which points to empty binding list is skipped. When a non-empty expansion track entry/binding list is found, the expansion bit is checked in the expansion track entry’s mask. If this bit is not set, then the corresponding expanded IP-SGT host entry is to be exported as added to the peer.

**Interface design**

Read-only DBUS interface via G-DBUS library is provided. SXPD has no IPC interface to export any information to other processes. SXPD can be optionally built with d-bus gdbus interface. Gdbus interface exports basic sxpd information, IPv4 and IPv6 binding lists and peer list. Gdbus interface is using "d-bus system bus" which requires to update d-bus configuration before using SXPD.

Example of debian d-bus configuration, which allows users of group nobody to connect system bus and own d-bus name "com.xoxo.sxpd":

configuration file path:

/etc/dbus-1/system.d/com.xoxo.sxpd.conf

configuration file content:

<busconfig>

<policy at\_console="true">

<allow own="com.xoxo.sxpd"/>

</policy>

<policy group="nobody">

<allow own="com.xoxo.sxpd"/>

</policy>

<policy group="nobody">

<allow send\_destination="com.xoxo.sxpd"

send\_interface="com.xoxo.sxpd"/>

</policy>

<policy context="default">

<allow send\_destination="com.xoxo.sxpd"

send\_interface="org.freedesktop.DBus.Introspectable" />

<allow send\_destination="com.xoxo.sxpd"

send\_interface="org.freedesktop.DBus.Properties" />

</policy>

</busconfig>

Detailed documentation about d-bus configuration can be found at:

http://dbus.freedesktop.org/doc/dbus-daemon.1.html

**Basic compilation**

Build system CMake is used for SXPD compilation.

Building SXPD:

Simply create directory which will be used by cmake to generate make rules and also used as destination for compiled binary files. Change to this newly created directory and call cmake command with path to SXPD source directory as one of of the parameters. Once cmake is finished, invoke make.

Command line compilation example (assuming the sxpd source code is in ~/sxpd and

building in ~/build):

# create and change directory to build files directory

mkdir ~/build && cd ~/build

# use cmake to generate make rules used to build linux SXPD binary

cmake -DTARGET\_BUILD\_PLATFORM=linux ~/sxpd

# build linux SXPD binary

make all

**Advanced compilation options**

Sxpd project has these cmake configuration options:

(1) (mandatory) target operating system cmake option:

-DTARGET\_BUILD\_PLATFORM=<linux>

(2) (optional) option to build SXPD binary with or without debug symbols:

-DCMAKE\_BUILD\_TYPE=<Debug|Release>

(3) (optional) option to build SXPD binary with or without d-bus interface.

By default, d-bus interface is disabled.

To know more about d-bus interface see section "SXPD GDBUS INTERFACE":

-DENABLE\_GDBUS\_INTERFACE=false to disable d-bus

-DENABLE\_GDBUS\_INTERFACE=<any other value> to enable d-bus

(4) (optional) option to print all logging information to console:

By default, log printing is disabled.

-DENABLE\_LOG\_PRINTING=false to disable log printing

-DENABLE\_LOG\_PRINTING=<any other value> to enable log printing

(5) (optional) option to strictly check binding configuration:

By default, binding configuration strict checker is enabled.

-DENABLE\_STRICT\_BINDING\_CFG\_CHECK=false to disable option

-DENABLE\_STRICT\_BINDING\_CFG\_CHECK=<any other value> to enable option

Detailed documentaion about CMake build-system can be found at:

<http://www.cmake.org/documentation/>

**Linux configuration file**

SXP daemon is using libconfig-like format configuration file, not all options, which libconfig allows/supports are used/allowed though. See "default.cfg" file in the root SXPD source directory, which contains a commented example with all options explained.

SXP daemon can be run from the command-line with the default or user specified parameters.

Usage:

<sxpd\_binary\_path> [config\_file\_path] [log\_level] [pid\_file\_path]

sxpd\_binary\_path: path to sxpd binary

config\_file\_path: path to configuration file used by this SXPD instance

log\_level: default log-level used by the SXPD instance,

must be one of the following:

alert: critical messages which require user interaction

error: run-time error messages

debug: debugging messages

trace: detailed debugging messages

NOTE: the log-level can be overriden in the configuration file, in which

case, the log-level specified on command-line is used only until

config-file log-level is read and applied

pid\_file\_path: path to file where SXPD will store its process ID

the default parameters are:

config\_file\_path: /etc/sxpd.cfg

log\_level: error

pid\_file\_path: /tmp/sxpd.pid

**Modifying runtime configuration on Linux**

Modifying run-time configuration is possible by changing the configuration file,

which SXPD instance uses and then sending the HANGUP signal to the SXPD.

A convenience shell script, called sxpd-reload.sh is provided in the source code

root directory. The script reads the PID from the SXPD pid-file and sendts the

appropriate signal to the SXPD instance.

Example:

1. SXPD run with default parameters:

$ ./sxpd-reload.sh

No parameter specified, use default pid file location /tmp/sxpd.pid?y

PID TTY TIME CMD

30155 pts/14 00:00:00 sxpd

Success: sent configuration reload trigger to process 30155

2. SXPD with custom pid-file path /tmp/my\_sxpd.pid

$ ./sxpd-reload.sh /tmp/my\_sxpd.pid

PID TTY TIME CMD

30155 pts/14 00:00:00 sxpd

Success: sent configuration reload trigger to process 30155

**Development Unit Testing**

Test plan has been designed to cover unit testing, topology simulation testing, code coverage testing, static analysis testing and memory testing.

All tests were built on top of CMake CTest automated testing tool.

Other tools:

* CTest – unit testing, topology simulation testing
* GNU coverage - code coverage testing
* Valgrind – memory testing
* Scan-build – static analysis testing

Unit test plan and test results are part of separate document OS\_SXP\_C\_test\_plan.doc

**SXP directory structure**

<SXPD source root>/README.txt - readme file.

<SXPD source root>/default.cfg – default configuration file with documentation

<SXPD source root>/license - SXPD source code license

<SXPD source root>/inc - header files directory

<SXPD source root>/src - source files directory

<SXPD source root>/test - test files directory

<SXPD source root>/linux - linux specific source and test files directory

END OF DOCUMENT