NAME

libscamperfile — scamper warts file library

LIBRARY

scamper warts file library (libscamperfile -lscamperfile)

SYNOPSIS

```
#include <scamper_file.h>
#include <scamper_addr.h>
#include <scamper_list.h>
#include <scamper_icmpext.h>
#include <scamper_trace.h>
#include <scamper_ping.h>
#include <scamper_tracelb.h>
#include <scamper_dealias.h>
#include <scamper_neighbourdisc.h>
#include <scamper_tbit.h>
#include <scamper_sting.h>
#include <scamper_sniff.h>
#include <scamper_host.h>
#include <scamper_http.h>
#include <scamper_udpprobe.h>
```

DESCRIPTION

The **libscamperfile** library provides the ability to read and write warts files produced by scamper, read arts files produced by CAIDA's Skitter, and write simple text representations of these data. A program that uses libscamperfile to read a warts file (1) allocates a filter defining the types of data contained in the file the program is interested in, (2) opens the file, (3) reads each object from the file until end of file is reached, (4) closes the file and frees the filter. A program that uses libscamperfile is responsible for freeing the data returned.

ROUTINES

scamper_file_t * **scamper_file_filter_alloc**(*uint16_t *types*, *uint16_t num*)
Allocate a filter to use with **scamper_file_read**() that specifies the types of data the program is interested in with the types parameter, and the number of types in the num parameter. When the filter is no longer required, the caller should use **scamper_file_filter_free**() to free the filter. For each data type specified the caller is subsequently responsible for freeing the data object when it is no longer required. Valid data type values to specify in the types array are:
- **SCAMPER_FILE_OBJ_LIST**
- **SCAMPER_FILE_OBJ_CYCLE_START**
- **SCAMPER_FILE_OBJ_CYCLE_DEF**
- **SCAMPER_FILE_OBJ_CYCLE_STOP**
- **SCAMPER_FILE_OBJ_ADDR**
- **SCAMPER_FILE_OBJ_TRACE**
- **SCAMPER_FILE_OBJ_PING**
- **SCAMPER_FILE_OBJ_TRACELB**
- **SCAMPER_FILE_OBJ_DEALIAS**
- **SCAMPER_FILE_OBJ_NEIGHBOURDISC**
- **SCAMPER_FILE_OBJ_TBIT**

– **SCAMPER_FILE_OBJ_STING**
– **SCAMPER_FILE_OBJ_SNIFF**
– **SCAMPER_FILE_OBJ_HOST**
– **SCAMPER_FILE_OBJ_HTTP**
– **SCAMPER_FILE_OBJ_UDPPROBE**

void **scamper_file_filter_free**(*scamper_file_filter_t *filter*)
Free a file filter structure.

int **scamper_file_filter_isset**(*scamper_file_filter_t *filter*, *uint16_t type*)
Determine if a particular data type will be passed by the filter. It returns zero if the type is not set in the filter, and one if it is.

*scamper_file_t ** **scamper_file_open**(*char *name*, *char mode*, *char *type*)
Open the file specified by the name parameter. The mode parameter specifies how the file should be opened. If the mode character ‘r’ is specified the file is opened read-only. If the mode character ‘w’ is specified the file is truncated and opened for writing. If the mode character ‘a’ is specified the file is open for writing, but without truncating any existing file. When opening a file for reading, the type parameter is optional as the type of file will be automatically determined. When writing a file, the type parameter allows the caller to define whether the file should be written in "warts" or "text". Note that only "warts" and "arts" can be read by **libscamperfile** so use of "warts" is highly recommended.

*scamper_file_t ** **scamper_file_openfd**(*int fd*, *char *name*, *char mode*, *char *type*)
Allocate a new scamper_file_t structure with the file descriptor passed. The name parameter is an optional parameter giving the scamper_file_t a name. The mode parameter has the same meaning as in the **scamper_file_open**() function. The type parameter is used to define the type of file to be read. If the file descriptor is a socket and the file is to be read, this parameter is useful as **libscamperfile** is currently unable to automatically determine the type of file in this scenario. Valid types are "warts", "arts", and "text".

*scamper_file_t ** **scamper_file_opennull**(*void*)
Allocate a new scamper_file_t that does not write to a file. Use in conjunction with **scamper_file_setreadfunc**() and **scamper_file_setwritefunc**() to do your own I/O.

void **scamper_file_close**(*scamper_file_t *sf*)
Close the file and free the scamper_file_t.

void **scamper_file_free**(*scamper_file_t *sf*)
Free the scamper_file_t but do not close the underlying file descriptor. This function is useful if the scamper_file_t was created using a file descriptor and **scamper_file_openfd**().

int **scamper_file_read**(*scamper_file_t *sf*, *scamper_file_filter_t *filter*, *uint16_t *obj_type*, *void **obj_data*)
Read the next data object from the file according to the filter passed in. Returns zero on success, -1 if an error occurred. When the end of file is reached, zero is returned and obj_data is set to NULL. The next data object is returned in obj_data, and the type of that object is returned in obj_type. The caller is responsible for freeing the data after it is no longer required; for example, call **scamper_trace_free**() for trace objects, and **scamper_ping_free**() for ping objects.

void **scamper_file_setreadfunc**(*scamper_file_t *sf*, *void *param*, *scamper_file_readfunc_t readfunc*)
Override the function used to read bytes from an input stream. The readfunc takes three parameters: the first is the param variable passed to **scamper_file_setreadfunc**(), the second is a pointer to a buffer with data read, and the third is the length of the data to read, in bytes. The pointer to a buffer with data read must be dynamically allocated with malloc by the readfunc, as it will be freed when **libscamperfile** no

longer requires the data. This function is used in conjunction with **scamper_file_opennull**().

void **scamper_file_setwritefunc**(*scamper_file_t *sf*, *void *param*, *scamper_file_writefunc_t writefunc*)
Override the function used to write warts data. The writefunc takes three parameters: the first is the param variable passed to **scamper_file_setwritefunc**(), the second is a pointer to the data to write, and the third is the length of the data, in bytes. This function is used in conjunction with **scamper_file_opennull**().

EXAMPLE

The following opens the file specified by name, reads all traceroute and ping data until end of file, processes the data, calls the appropriate methods to free the data, and then closes the file.

```
uint16_t types[] = {
  SCAMPER_FILE_OBJ_TRACE,
  SCAMPER_FILE_OBJ_PING,
};
scamper_file_t *in;
scamper_file_filter_t *filter;
scamper_trace_t *trace;
scamper_ping_t *ping;
uint16_t type;
void *data;

if((filter = scamper_file_filter_alloc(types, 2)) == NULL) {
  fprintf(stderr, "could not allocate filter\n");
  return -1;
}

if((in = scamper_file_open(name, 'r', NULL)) == NULL) {
  fprintf(stderr, "could not open %s: %s\n", name, strerror(errno));
  return -1;
}

while(scamper_file_read(in, filter, &type, (void *)&data) == 0) {

  if(data == NULL)
    break; /* EOF */

  switch(type) {
    case SCAMPER_FILE_OBJ_TRACE:
       trace = data;
       process_trace(trace);
       scamper_trace_free(trace);
       break;

    case SCAMPER_FILE_OBJ_PING:
       ping = data;
       process_ping(ping);
       scamper_ping_free(ping);
       break;
  }
}
```

```
        scamper_file_close(in);
        scamper_file_filter_free(filter);
```

SEE ALSO

`scamper`(1), `sc_wartsdump`(1), `sc_warts2text`(1),

M. Luckie, *Scamper: a Scalable and Extensible Packet Prober for Active Measurement of the Internet*, Proc. ACM/SIGCOMM Internet Measurement Conference 2010.

AUTHORS

`libscamperfile` was written by Matthew Luckie <mjl@luckie.org.nz>.