nbtcpfrm: Non Blocking TCP/IP Network Control Program

CML00053-01

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com

December 15, 2020

Contents

1 Overview

1.1 Introduction

The Code Magus Limited nbtcpfrm provides standard NCP (Network Control Program) functions to send and receive buffers on a TCP/IP network interface. This NCP is a DLL (or shared program object) that is nominated and loaded by the application program requiring communications on a network interface. The NCP is responsible for the establishment, transmission, control and tear-down of network circuits.

The NCP operates in a non-blocking mode. It is the responsibility of the user to watch the file descriptors on behalf of the NCP. There are two methods provided to facilitate this (see section 2.6 on page 7).

The user must also supply two call back functions to the NCP:

- a function to deal with circuit status changes.
- a function for data received on a circuit.

2 nbtcpfrm User Interface

Function nbncp_init() is the only exported function and must be the first function called once the DLL has been loaded. The other entry points are exposed in the returned data structure. The function prototypes and data structures required are made available to the application program by including the header file nbncp.h (see appendix A on page 11).

2.1 nbncp_init Function

```
typedef int (*nbncp_recv_t)(nbncp_circuit_t *circuit,int length,
      unsigned char *buffer,struct timeval *time_stamp);
typedef void (*nbncp_status_change_t)(nbncp_circuit_t *circuit,
      nbncp_status_t status,char *msg);

nbncp_desc_t *nbncp_init(nbncp_status_change_t status_change,
      nbncp_recv_t recv);
```

This function returns a descriptor of the loaded DLL. It is expected to succeed, but if it does fail then the function will return NULL. This call has two parameters, the first parameter is the user function for receiving notification of circuit changes of an opened circuit. The second parameter is the user function for handling data received on a circuit.

Example:

```
   void *dll;                /* handle of loaded NCP DLL */
   nbncp_init_t ncp_init;    /* function to initialise the NCP interface */
   nbncp_desc_t *ncp;        /* ncp interface structure */

  /* Load the NCP DLL:
   */
   dll = dlopen(ncp_name,RTLD_NOW);
   if (!dll)
      {
      fprintf(stderr,"Open of NCP %s failed: %s",ncp_name,dlerror());
      return -1;
      }

  /* Get the initialisation entry point:
   */
   ncp_init = dlsym(dll,"nbncp_init");
   if (!ncp_init)
      {
      fprintf(stderr,"Open of NCP %s failed: %s",ncp_name,dlerror());
      return -1;
      }

  /* Initialise the NCP interface:
   */
```

```
   ncp = ncp_init(status_change,data_recv);
   if (!ncp)
      {
      fprintf(stderr,"NCP %s failed to inilialise",ncp_name);
      return -1;
      }

  /* If a TCP trace is required, ask the NCP to do the honours.
   */
   if (tcp_trace_fname)
      if (ncp->open_trace(ncp,tcp_trace_fname) < 0)
         {
         fprintf(stderr,"Unable to open trace file: %s",ncp->last_error);
         return -1;
         }
```

2.1.1 nbncp_status_change_t Call Back Function

```
typedef void (*nbncp_status_change_t)(nbncp_circuit_t *circuit,
      nbncp_status_t status,char *msg);
```

This function is the user callback function for receiving notification of circuit status changes. The parameter `status` is the new status of the circuit and `msg` is a description of the status.

`status` has the following values:

- NBNCP_DISCONNECTED
  circuit has disconnected.
- NBNCP_WAITING_CONNECTION
  circuit is waiting for a connection.
- NBNCP_CONNECTED
  circuit has connected.
- NBNCP_NEW_CONNECTION
  a new circuit was created due to a connection to a passive circuit.
- NBNCP_CLOSED
  a passive connection has disconnected and the circuit associated with it, will be freed.

Example:

```
  /* Function status_change()
   * Callback function for NCP circuit status changes.
   */
static void status_change(nbncp_circuit_t *circuit,
      nbncp_status_t status,char *msg)
   {
```

```
   if (verbose)
         fprintf(stdout,"status_change(): Instance %d: %s\n",
               nwk_instance,msg);
   switch (status)
      {
      case NBNCP_CONNECTED:
         host_is_connected = 1;
         ncp->write_trace(ncp,NULL," Instance %d Input "
               "CONNECT",nwk_instance);
         break;
      case NBNCP_DISCONNECTED:
         host_is_connected = 0;
         ncp->write_trace(ncp,NULL," Instance %d Input "
               "DISCONNECT",nwk_instance);
         break;
      }
   }  /* status_change */
```

2.1.2 nbncp_recv_t Call Back Function

```
typedef int (*nbncp_recv_t)(nbncp_circuit_t *circuit,int length,
      unsigned char *buffer,struct timeval *time_stamp);
```

This function is the user call back function for handling a complete buffer received on a circuit. The data is in `buffer`, with `length` set to the length of the data received and `time_stamp` contains the time the data was received. If the function is successful, zero should be returned to the NCP, but if an error occurs -1 should be returned, in which case the NCP will close the circuit.

Example:

```
  /* Function data_recv(()
   * Callback function for data received on the circuit.
   *
   * *circuit is the circuit that the data is received on, len the length of
   * data received and *buf the data.
   * *time_stamp is the time stamp when the data was received.
   *
   * Return zero on success. If an error occurs return -1, which will
   * result in the circuit being closed.
   */
static int data_recv(nbncp_circuit_t *circuit,int length,
      unsigned char *buffer,struct timeval *time_stamp)
   {

  /* Log the message
   */
   bslog_write('r',NULL,length,buffer);
.
.
.
```

```
return 0;
}  /* data_recv */
```

2.2 nbncp_cleanup Function

```
typedef int (*nbncp_cleanup_t)(nbncp_desc_t *dlldesc);
```

This function should be called just before the DLL is unloaded and gives the DLL a chance to clean up any resources it may have acquired. This function returns 0 on success and -1 on error. The last_error message pointer in the nbncp_desc_t structure describes any error.

Example:

```
nbncp_desc_t *ncp;        /* ncp interface structure */

if (ncp->cleanup(ncp) < 0)
   {
   fprintf(stderr,"Unable to terminate NCP: %s",ncp->last_error);
   return -1;
   }
```

2.3 nbncp_open Function

```
typedef nbncp_circuit_t *(*nbncp_open_t)(nbncp_desc_t *dlldesc,
      char *parms[],void *user_data);
```

This function opens a new circuit and associates it with the given descriptor. The parameters to the new circuit are passed as strings to the function. Each string expresses a parameter and its value. Required and optional parameters can be intermixed. These parameters are NULL terminated strings of the following form:

```
parameter-name=parameter-value
```

On successful creation of the new circuit, the address of a circuit structure is returned. This address must be passed back to the DLL on all circuit specific calls. If an error occurs NULL is returned and a descriptive message is placed in last_error in the nbncp_desc_t structure.

Example:

```
nbncp_desc_t *ncp;       /* ncp interface structure */
nbncp_circuit_t *circuit;/* NCP created circuit structure */
char *ncp_parms[20];     /* NCP parameters to open circuit with */
int p = 0;               /* iterator for parms */
char wstring[256];       /* general work string */

ncp_parms[p++] = strdup("name=bsbicisocp");
sprintf(wstring,"hostname=%s",host_name);
ncp_parms[p++] = strdup(wstring);
```

```
sprintf(wstring,"hostport=%d",port);
ncp_parms[p++] = strdup(wstring);
sprintf(wstring,"frametype=%s",frame_type);
ncp_parms[p++] = strdup(wstring);
ncp_parms[p++] = strdup("retry=no");
ncp_parms[p++] = strdup("offset=0");
sprintf(wstring,"bias=%d",bias);
ncp_parms[p++] = strdup(wstring);
if (verbose)
    ncp_parms[p++] = strdup("verbose=yes");
circuit = ncp->open(ncp,ncp_parms,NULL);
if (!circuit)
   {
   fprintf(stderr,"Unable to connect circuit: %s",ncp->last_error);
   return -1;
   }
```

2.3.1 Required parameters

- NAME
  name associated with the circuit.
- HOSTNAME
  IP address or DNS entry of the machine to connect to.
- HOSTPORT
  port to connect to on the specified machine.
- FRAMETYPE
  mnemonic of the frame format to use, valid values:
  `S12, S21, L1234, L4321, B1, L1200, LF, CRLF` or `NONE`.

2.3.2 Optional parameters

- BIAS
  bias of length to total message: Valid values are 0, 2, or 4.
- DIRECTION
  direction indicator to be passed through to user: `S12, S21, L1234` or `L4321`.
- MAX_CONN
  default = 0 (unlimited), value = max connections allowed for a passive circuit.
- OFFSET
  Offset of the length field in a message: MUST BE 0.
- PASSIVE
  default = NO, YES = circuit is passive (listen).

- RECV_EDIT
  buffer edit receive configuration file name.
- RETRY
  default = NO, YES = retry connection when it disconnect.
- SEND_EDIT
  buffer edit send configuration file name.
- VERBOSE
  default = NO, YES = circuit to operate in verbose mode.

2.4 nbncp_close Function

```
typedef int (*nbncp_close_t)(nbncp_circuit_t *circuit);
```

This function closes a circuit and frees any resources used by that circuit. If the close succeeds then 0 is returned, otherwise -1 is returned and a descriptive message is placed in `last_error` in the `nbncp_desc_t` structure.

Example:

```
nbncp_desc_t *ncp;         /* ncp interface structure */
nbncp_circuit_t *circuit; /* NCP created circuit structure */

if (ncp->close(circuit) < 0)
   {
   fprintf(stderr,"Unable to close circuit: %s",ncp->last_error);
   return -1;
   }
```

2.5 nbncp_send Function

```
typedef int (*nbncp_send_t)(nbncp_circuit_t *circuit,int len,
      unsigned char *buf);
```

This function takes a raw buffer and a length and queues that buffer for transmission on the underlying transport layer interface. If the function succeeds then 0 is returned, otherwise -1 is returned and a descriptive message is placed in `last_error` in the `nbncp_desc_t` structure.

Example:

```
nbncp_desc_t *ncp;         /* ncp interface structure */
nbncp_circuit_t *circuit; /* NCP created circuit structure */

if (send(circuit,length,buffer) < 0)
   {
   fprintf(stderr,"Unable to send data on circuit: %s",ncp->last_error);
```

```
      return -1;
      }
```

2.6 nbncp_set_fds and nbncp_check_fds Functions

```
typedef int (*nbncp_set_fds_t)(nbncp_desc_t *dlldesc,int *nfds,
      fd_set *readfds,fd_set *writefds,fd_set *exceptfds);
typedef int (*nbncp_check_fds_t)(nbncp_desc_t *dlldesc,int *fdcnt,
      fd_set *readfds,fd_set *writefd,fd_set *exceptfds);
```

Function `set_fds` sets the `readfds`, `writefds` and `exceptfds` for the NCP circuits that requires attention. `nfds` should be set to the highest-numbered file descriptor in any of the three sets, plus 1.

Function `check_fds` checks the `readfds`, `writefds` and `exceptfds`, and should be called after calling `select()`. `fdcnt` is the return value of the `select()`; the number of file descriptors contained in the three returned descriptor sets that require attention.

Example:

```
int main(int argc,char *argv[],char *envp[])
   {
   fd_set readfds,writefds,exceptfds;  /* Read & write descriptor sets */
   int fdmax,fdcnt;
   nbncp_desc_t *ncp_desc;             /* NCP interface structure */
.
.
.
   struct timeval tv;

  /* For ever loop.
   */
   while (1)
      {
      FD_ZERO(&readfds);
      FD_ZERO(&writefds);
      FD_ZERO(&exceptfds);
      fdmax = 0;

     /* Ask the NCP to set his file descriptors that needs watching.
      */
      ncp_desc->set_fds(ncp_desc,&fdmax,&readfds,&writefds,&exceptfds);

      tv.tv_sec = 1;
      tv.tv_usec = 0;
      fdcnt = select(fdmax+1,&readfds,&writefds,NULL,&tv);

     /* Ask the NCP to check his file descriptors for those that needs
      * attention.
      */
```

```
      if (fdcnt > 0)
         ncp_desc->check_fds(ncp_desc,&fdcnt,&readfds,&writefds,&exceptfds);
      }

   return 0;
   } /* Main */
```

2.7 nbncp_version Function

```
typedef char *(*nbncp_version_t)(void);
```

This function returns the CVS version string of the NCP.

Example:

```
   fprintf(stderr,"NCP Version: %s\n",ncp->version);
```

2.8 nbncp_open_trace Function

```
typedef int (*nbncp_open_trace_t)(nbncp_desc_t *dlldesc,char *name);
```

This function opens a circuit trace file with the name as supplied. If the open succeeds then 0 is returned, otherwise -1 is returned and a descriptive message is placed in `last_error` in the `nbncp_desc_t` structure. Once open, all circuit activity is written to this file.

The following substitution directives, when embedded in the file name will be expanded:

- %D to the current date in the format CCYYMMDD.
- %P to the process id of the caller.
- %T to the current time in the format HHMMSS.

For example `MY_TRACE_%D_%T.TXT` expands to `MY_TRACE_D20080121_T115501.TXT`.

Example:

```
   nbncp_desc_t *ncp;         /* ncp interface structure */

   if (ncp->open_trace(ncp,"MY_TRACE_%D_%T.TXT") < 0)
      {
      fprintf(stderr,"Unable to open trace file: %s",ncp->last_error);
      return -1;
      }
```

2.9 nbncp_trace_isopen Function

```
typedef int (*nbncp_trace_isopen_t)(nbncp_desc_t *dlldesc);
```

This function returns 1 if a circuit trace file is open.

Example:

```
nbncp_desc_t *ncp;        /* ncp interface structure */

if (ncp->trace_isopen(ncp))
   {
   fprintf(stderr,"TCP/IP trace file: %s",ncp->trace_name(ncp));
   return -1;
   }
```

2.10 nbncp_trace_name Function

```
typedef char *(*nbncp_trace_name_t)(nbncp_desc_t *dlldesc);
```

This function returns the name of the current open circuit trace file. If a trace file is not open, NULL is returned.

Example:

```
nbncp_desc_t *ncp;        /* ncp interface structure */

if (ncp->trace_isopen(ncp))
   {
   fprintf(stderr,"TCP/IP trace file: %s",ncp->trace_name(ncp));
   return -1;
   }
```

2.11 nbncp_close_trace Function

```
typedef int (*nbncp_close_trace_t)(nbncp_desc_t *dlldesc);
```

This function closes the current open circuit trace file. If the close succeeds then 0 is returned, otherwise -1 is returned and a descriptive message is placed in last_error in the nbncp_desc_t structure.

Example:

```
nbncp_desc_t *ncp;        /* ncp interface structure */

if (ncp->close_trace(ncp) < 0)
   {
   fprintf(stderr,"Unable to close trace file: %s",ncp->last_error);
   return -1;
   }
```

2.12 nbncp_switch_trace Function

```
typedef int (*nbncp_switch_trace_t)(nbncp_desc_t *dlldesc,char *name);
```

This function closes the current open circuit trace file and opens a new circuit trace file. The file name is optional. If the switch succeeds then 0 is returned, otherwise -1 is returned and a descriptive message is placed in last_error in the nbncp_desc_t structure.

Example:

```
nbncp_desc_t *ncp;        /* ncp interface structure */

if (ncp->switch_trace(ncp,NULL) < 0)
   {
   fprintf(stderr,"Unable to switch trace file: %s",ncp->last_error);
   return -1;
   }
```

2.13 nbncp_write_trace Function

```
typedef int (*nbncp_write_trace_t)(nbncp_desc_t *dlldesc,
      nbncp_circuit_t *circuit,char *format,...);
```

This function writes a message to the trace file. If circuit is supplied, the message is prefixed with the description of the circuit. format is a format string and a variable number of arguments in the style of printf().

If the write succeeds then 0 is returned, otherwise -1 is returned and a descriptive message is placed in last_error in the nbncp_desc_t structure.

Note: if there is no trace file open, this function will ignore the request and return 0.

Example:

```
nbncp_desc_t *ncp;       /* ncp interface structure */
nbncp_circuit_t *circuit;/* NCP created circuit structure */

ncp->write_trace(ncp,circuit,"Instance %d Output %d %s",sm_instance,
      output_number,output_name);

ncp->write_trace(ncp,NULL,"Instance %d Input EXPAND_ERROR",
      nwk_instance);
```

A NCP header file: nbncp.h

```
#ifndef NBNCP_H
#define NBNCP_H
  /* File: nbncp.h
```

```
   *
   * This header file describes the interface between an
   * Network Control Program and a application itself.
   * The Network Control Programs are DLLs that are nominated
   * and loaded by the application program for communications on a
   * network interface. The Network Control Program is responsible for the
   * establishment, transmission, control and tear-down of network circuits.
   *
   * The NCP is operating in a non-blocking mode. It is the responsibility
   * of the user to watch the file descriptors on be halve of the NCP. For
   * this there are two methods provided to facilitating this:
   * Firstly the user must request the DLL to populate his requirements in
   * the supplied file descriptor sets, do the select() and then call the
   * NCP with the resultant file descriptor sets. Further to this, the user
   * must also supply two call back functions to the NCP: a function to deal
   * with circuit status changes and a function for data received on a
   * circuit.
   *
   * Copyright (c) 2008 Code Magus Limited. All rights reserved.
   *
   * Author: Jan Vlok.
   */

  /*
   * $Author: jan $
   * $Date: 2008/12/01 08:05:02 $
   * $Id: nbncp.h,v 1.3 2008/12/01 08:05:02 jan Exp $
   * $Name:  $
   * $Revision: 1.3 $
   * $State: Exp $
   *
   * $Log: nbncp.h,v $
   * Revision 1.3  2008/12/01 08:05:02  jan
   * Added a full sample program
   *
   * Revision 1.2  2008/11/24 08:12:14  jan
   * Spelling fix
   *
   * Revision 1.1  2008/11/19 12:42:40  jan
   * Renamed modules
   *
   */

static char *cvs_nbncp_h =
   "$Id: nbncp.h,v 1.3 2008/12/01 08:05:02 jan Exp $";

  /*
   * Constants and options.
   */

#define NBNCP_MAX_BUF  32760   /* maximum buffer length */
```

```
  /*
   * Data structures and types:
   */

  /* There are two major exposed data structures which are created by a
   * Network Control Program DLL.
   */
typedef struct nbncp_desc nbncp_desc_t;       /* NCP DLL descriptor */
typedef struct nbncp_circuit nbncp_circuit_t; /* circuit state structure */

typedef struct nbncp_parm nbncp_parm_t;/* NCP parameter name and description */

  /* Within an established circuit, there is a notion of named attributes.
   * These named attributes can be set and retrieved using the attrset and
   * attrget methods.
   */
typedef struct nbncp_value nbncp_value_t;

  /* Apart from the initialisation call, the functions are exposed by the
   * DLL by filling in the addresses of the other entry points into a
   * descriptor data structure. Each function is described by a typedef.
   */

  /* Function cleanup is called just before the DLL is unloaded and
   * gives the DLL a chance to clean up any resources it may have acquired.
   * The function returns 0 on success and -1 on error.  The last_error
   * message pointer describes the error.
   */
typedef int (*nbncp_cleanup_t)(nbncp_desc_t *dlldesc);

  /* Function open creates a new circuit and associates it with the
   * given descriptor. The parameters to the new circuit are passed
   * as strings to the function. Each string expresses a parameter and
   * its value. Required and optional parameters can be intermixed.
   * These parameters are null terminated strings of the following
   * form: "<parameter-name>=<parameter-value>".
   * On successful creation of the new circuit, the address of a
   * circuit structure is returned. This circuit structure must be
   * passed back to the DLL on all circuit specific calls. If an
   * error occurs NULL is returned and a descriptive message is
   * placed in last_error in the nbncp_desc_t structure.
   */
typedef nbncp_circuit_t *(*nbncp_open_t)(nbncp_desc_t *dlldesc,
      char *parms[],void *user_data);

  /* Function close cleans up a circuit and frees any resources used by
   * that circuit. If the close succeeds then 0 is returned, otherwise
   * -1 is returned and a descriptive message is placed in last_error in
   * the nbncp_desc_t structure.
   */
typedef int (*nbncp_close_t)(nbncp_circuit_t *circuit);
```

```
  /* Once a circuit is open, the particular NCP may expose certain
   * attributes which can be modified during the life of the circuit.
   * These attributes can be used to control the sending or receiving
   * of messages, but are not a replacement for the parameters whose
   * values must be known at the time that a circuit is established.
   *
   * Function attrset sets a named attribute to the value supplied.
   * Internally, this might result in a specific function being called
   * in order to decode and act on the specific attribute being set.
   * On successfully completion of the method, zero will be returned.
   * If the operation fails, then -1 is returned and an error message
   * is placed in the last_error in the nbncp_desc_t structure.
   */
typedef int (*nbncp_attrset_t)(nbncp_circuit_t *circuit,
      char *attribute, nbncp_value_t *value);

  /* Function attrget gets the value of a named attribute. The name of
   * the value is supplied, and the value is placed in the supplied
   * structure. The value might be more than just a copy and could be
   * as a result of calling certain internal functions. On successfully
   * completion of the method, zero will be returned. If the operation
   * fails, then -1 is returned and an error message is placed in the
   * last_error in the nbncp_desc_t structure.
   */
typedef int (*nbncp_attrget_t)(nbncp_circuit_t *circuit,
      char *attribute, nbncp_value_t *value);

  /* Function version returns the CVS version string of the NCP.
   */
typedef char *(*nbncp_version_t)(void);

   /* Function send takes a raw buffer and a length and queues that
    * buffer for transmission on the underlying transport layer
    * interface. If the send succeeds then 0 is returned, otherwise
    * -1 is returned and a descriptive message is placed in last_error in
    * the nbncp_desc_t structure.
    */
typedef int (*nbncp_send_t)(nbncp_circuit_t *circuit,int len,
      unsigned char *buf);

  /* Function set_fds set the readfds, writefds and exceptfds for the NCP's
   * circuits that needs attention.
   * nfds is set to the highest-numbered file descriptor in any of the three
   * sets, plus 1.
   */
typedef int (*nbncp_set_fds_t)(nbncp_desc_t *dlldesc,int *nfds,
      fd_set *readfds,fd_set *writefds,fd_set *exceptfds);

  /* Function check_fds check readfds, writefds and exceptfds, after the
   * user select().
   * fdcnt is the return value of the select(): the number of file descriptorstors
   * contained in the three returned descriptor sets
```

```
   */
typedef int (*nbncp_check_fds_t)(nbncp_desc_t *dlldesc,int *fdcnt,
      fd_set *readfds,fd_set *writefd,fd_set *exceptfds);

  /* The following functions are for manipulating a circuit trace file.
   * All circuit activity are written to this file, that's to say, if
   * the user opened it.
   */

  /* Function open_trace open a circuit trace file with the name as supplied.
   * If the file name file contains the following literals, it will be expanded
   * to:
   *   %D -> CCYYMMDD
   *   %T -> HHMMSS
   *   %P -> Process id of caller
   * for example "myttrace_%D_%T.txt" expand to "mytrace_D20080121_T115501.txt"
   * If the open succeeds then 0 is returned, otherwise -1 is returned and a
   * descriptive message is placed in last_error in the nbncp_desc_t structure.
   */
typedef int (*nbncp_open_trace_t)(nbncp_desc_t *dlldesc,char *name);

  /* Function trace_isopen returns 1 if a circuit trace file is open.
   */
typedef int (*nbncp_trace_isopen_t)(nbncp_desc_t *dlldesc);

  /* Function _trace_name returns the name of the current open circuit
   * trace file. If a trace file is not open, NULL is returned.
   */
typedef char *(*nbncp_trace_name_t)(nbncp_desc_t *dlldesc);

  /* Function close_trace closes the current open circuit trace file.
   * If the close succeeds then 0 is returned, otherwise -1 is returned and a
   * descriptive message is placed in last_error in the nbncp_desc_t structure.
   */
typedef int (*nbncp_close_trace_t)(nbncp_desc_t *dlldesc);

  /* Function switch_trace closes the current open circuit trace file and
   * open a new circuit trace file. The file name is optional.
   * If the switch succeeds then 0 is returned, otherwise -1 is returned and a
   * descriptive message is placed in last_error in the nbncp_desc_t structure.
   */
typedef int (*nbncp_switch_trace_t)(nbncp_desc_t *dlldesc,char *name);

  /* Function write_trace write a message to the trace file.
   * If circuit is supplied, the message is prefixed with the description
   * of the supplied circuit.
   * If the write succeeds then 0 is returned, otherwise -1 is returned and a
   * descriptive message is placed in last_error in the nbncp_desc_t structure.
   * Note, if there is not a trace file open, this function will ignore the
   * request and return 0.
   */
typedef int (*nbncp_write_trace_t)(nbncp_desc_t *dlldesc,
```

```
      nbncp_circuit_t *circuit,char *format,...);


  /* The first structure is the nbncp_desc_t which is a descriptor of
   * the loaded DLL. The nbncp_desc_t contains the state data of the
   * loaded DLL. Once initialised this descriptor will contain the
   * addresses of the other entry points of the DLL. This descriptor
   * must also be passed back to the interface on the circuit open and
   * DLL unload function calls.
   *
   * All function that return an error condition should also set the
   * last_error string address in the descriptor. This will allow the
   * calling function to display an appropriate message regarding the
   * error.
   *
   * Also returned on the descriptor is a list of required and optional
   * parameters names that must and could, resp., be passed on the
   * open call.
   */

struct nbncp_desc
   {
   nbncp_cleanup_t cleanup;         /* prepare NCP DLL for unloading */
   nbncp_open_t open;               /* NCP circuit open function */
   nbncp_close_t close;             /* NCP circuit close function */
   nbncp_send_t send;               /* NCP circuit send data function */
   nbncp_set_fds_t set_fds;         /* NCP set file descriptor function */
   nbncp_check_fds_t check_fds;     /* NCP check file descriptor function */
   nbncp_attrset_t attrset;         /* set attribute value on circuit */
   nbncp_attrget_t attrget;         /* get attribute value from circuit */
   nbncp_version_t version;         /* NCP CVS version string */
   nbncp_open_trace_t open_trace;   /* open circuit trace file */
   nbncp_close_trace_t close_trace; /* close circuit trace file */
   nbncp_switch_trace_t switch_trace;/* close and open a circuit trace file */
   nbncp_trace_isopen_t trace_isopen;
   nbncp_trace_name_t trace_name;   /* circuit trace file name */
   nbncp_write_trace_t write_trace; /*writes to the trace file */


   char *last_error;                /* text for last error encountered */
   nbncp_parm_t *required;          /* array of required parameters */
   nbncp_parm_t *optional;          /* array of optional parameters */
   char **attributes;               /* array of exposed circuit attr names */

   void *private_data;              /* private DLL instance data */
   };

  /* Name and description of the NCP' parameters.
   */
struct nbncp_parm
   {
   char *name;              /* parameter name */
```

```
   char *desc;              /* parameter description */
   };

  /* The second controls the instances of the circuits created by the
   * DLL.
   */
struct nbncp_circuit
   {
   void *user_data;               /* caller's user data */
   nbncp_desc_t *dlldesc;         /* DLL descriptor owning circuit */
   nbncp_circuit_t *parent;       /* circuit created from a passive (listener)
                                     circuit */
//   nbncp_circuit_t *child;        /* passive (listener) - chain of connected
//                                     circuits */
   nbncp_circuit_t *next;         /* next circuit in the passive (listener)
                                     chain of connected of circuits */
   void *private_data;             /* circuit type specific instance data */
   };

  /* Attribute structure which describes the value of an attribute. The
   * name of the attribute is not part of the structure. The value is
   * supplied separately on the method call.
   */
struct nbncp_value
   {
   int length;                      /* length of attribute value */
   unsigned char *value;            /* value of attribute */
   };

  /* Function recv is the user call back function for when a completed buffer
   * was received on a circuit.
   * The contents is in buffer, with length set to the length of the data
   * received and time_stamp contains the time the buffer was received.
   * Return zero if successful, else -1 - this will cause a close
   * of the circuit.
   */
typedef int (*nbncp_recv_t)(nbncp_circuit_t *circuit,int length,
      unsigned char *buffer,struct timeval *time_stamp);

  /* Function status_change is the user callback function for receiving
   * notification of circuit status changes.
   * The parameter status is the new status of the circuit and msg is
   * a description of the status.
   */
typedef enum
      {
      NBNCP_DISCONNECTED,        /* circuit has disconnected */
      NBNCP_WAITING_CONNECTION,  /* circuit is waiting for a connection */
      NBNCP_CONNECTED,           /* circuit has connected */
      NBNCP_NEW_CONNECTION,      /* a new circuit was created due to a
                                    connection to a passive circuit */
      NBNCP_CLOSED,              /* a passive connection has disconnected,
```

```
                                    the user requested a close
                                    or the circuit had retry set to no:
                                    The circuit associated with it will
                                    be freed */
      } nbncp_status_t;

typedef void (*nbncp_status_change_t)(nbncp_circuit_t *circuit,
      nbncp_status_t status,char *msg);

  /* Function nbncp_init() is the only exported function and must be the
   * first function called once the detail the DLL has been loaded. The
   * other entry points are exposed in the returned data structure. The
   * function is expected to succeed, but if it does fail then the function
   * will return NULL.
   * This call has two parameters, the first parameter is the user function for
   * receiving notification of circuit changes of an opened circuit. The second
   * parameter is the user function for data received on a circuit.
   */
nbncp_desc_t *nbncp_init(nbncp_status_change_t status_change,
      nbncp_recv_t recv);
typedef nbncp_desc_t *(*nbncp_init_t)(nbncp_status_change_t status_change,
      nbncp_recv_t recv);

#endif /* NBNCP_H */
```

B Sample Program: `frmtest.c`

The following worked example illustrates the usage of the library by a program. Entering the following from the command line, executes the program:

```
./frmtest ./nbtcpfrm.so
```

```
  /* File: frmtest.c
   *
   * This program is used to test the Network Control program DLL's.
   *
   * This program loads the DLL and calls the initialisation entry point,
   * It then stays in a for ever loop, waiting for user input and
   * circuit activity.
   *
   * The user has two input options:
   * 1) open a circuit:
   *    o <optional open parameters file>
   * 2) Send a message on the last connected circuit:
   *    s <msg>
   *
   * If any data is received it is displayed.
   *
   * Copyright (c) 2008 Code Magus Limited. All rights reserved.
   *
   * Author: Jan Vlok.
```

```
   */

  /*
   * $Author: hayward $
   * $Date: 2011/01/20 18:45:16 $
   * $Id: frmtest.c,v 1.5 2011/01/20 18:45:16 hayward Exp $
   * $Name:  $
   * $Revision: 1.5 $
   * $State: Exp $
   *
   * $Log: frmtest.c,v $
   * Revision 1.5  2011/01/20 18:45:16  hayward
   * Implement ipmods library for cross platform
   * IP sockets independence.
   *
   * Revision 1.4  2010/02/10 11:59:03  jan
   * Implemented refclock
   *
   * Revision 1.3  2008/12/01 08:05:02  jan
   * Added a full sample program
   *
   * Revision 1.2  2008/11/19 12:42:40  jan
   * Renamed modules
   *
   * Revision 1.1  2008/11/13 08:59:32  jan
   * Take on
   *
   */

static char *cvs =
   "$Id: frmtest.c,v 1.5 2011/01/20 18:45:16 hayward Exp $";
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <dlfcn.h>

#include <osmods.h>
#include <refclock.h>

#include "nbncp.h"
  /*
   * Options and constants:
   */
#define BUF_SIZE  128*1024     /* transmit/receive buffer size in bytes */
#define BUF_COUNT 1            /* number of messages in flight during the test */

  /*
   * Statics:
   */
static nbncp_desc_t *ncp;
static void *dll;               /* handle of loaded DLL */
```

```
static nbncp_init_t ncp_init;    /* function to initialise the interface */
static nbncp_circuit_t *defined_circuit=NULL; /* new circuit descriptor */
static nbncp_circuit_t *connected_circuit=NULL; /* a connected circuit */

  /*
   * Local prototypes:
   */
static void status_change(nbncp_circuit_t *circuit,
      nbncp_status_t status,char *msg);
static int data_recv(nbncp_circuit_t *circuit,int length,
      unsigned char *buffer,struct timeval *time_stamp);
static int read_stdin(void);

int main(int argc, char *argv[])
   {
   struct timeval tv;
   fd_set readfds, writefds, exceptfds;  /* file descriptor sets for select */
   int fdmax, fdcnt;
   int rc, delta;

   delta = refclock_delta_time(&rc);
   if (rc < 0)
      {
      fprintf(stderr,"Reference Clock Server error: %s\n",
            refclock_error());
      return -1;
      }
   printf("Time diff to ref clock = %d\n",delta);

   if (argc != 2)
      {
      fprintf(stderr,"Usage: %s <dll-name>\n",argv[0]);
      exit(1);
      }

   dll = dlopen(argv[1],RTLD_NOW);
   if (!dll)
      {
      fprintf(stderr,"%s: %s\n",argv[1],dlerror());
      exit(1);
      }

   ncp_init = dlsym(dll,"nbncp_init");
   if (!ncp_init)
      {
      fprintf(stderr,"%s: %s\n",argv[1],dlerror());
      exit(1);
      }

  /* Initialise the interface to the NCP:
   */
   ncp = ncp_init(status_change,data_recv);
```

```
   if (!ncp)
      {
      fprintf(stderr,"Interface failed to initialise!\n");
      exit(1);
      }
  fprintf(stderr,"Interface initialised\n");

  /*
   * Main Service loop.
   */
   while (1)

      {
      FD_ZERO(&readfds);
      FD_ZERO(&writefds);
      FD_ZERO(&exceptfds);
      fdmax = 0;

     /* Check stdin.
      */
      FD_SET(0,&readfds);

      ncp->set_fds(ncp,&fdmax,&readfds,&writefds,&exceptfds);

      tv.tv_sec = 1;
      tv.tv_usec = 0;
      fdcnt = select(fdmax+1,&readfds,&writefds,&exceptfds,&tv);

   if (fdcnt > 0 && FD_ISSET(0,&readfds))
      {
      fdcnt--;
      if (read_stdin() == -1) return -1;
      }

      if (fdcnt > 0)
         ncp->check_fds(ncp,&fdcnt,&readfds,&writefds,&exceptfds);
      } /* Main loop */

   } /* main */



  /* Function nc_status_change()
   * Callback function for NC circuit status changes.
   * *circuit is the  circuit this call is applicable to.
   * status is the new status of the circuit and *msg is a description
   * of the status.
   */
static void status_change(nbncp_circuit_t *circuit,
      nbncp_status_t status,char *msg)
   {
```

```
   printf("status_change(): %s\n",msg);
   switch (status)
      {
      case NBNCP_WAITING_CONNECTION:
          printf("status_change(): NBNCP_WAITING_CONNECTION\n");
         break;
      case NBNCP_CONNECTED:
          connected_circuit = circuit;
          printf("status_change(): NBNCP_CONNECTED\n");
         break;
      case NBNCP_NEW_CONNECTION:
          connected_circuit = circuit;
          printf("status_change(): NBNCP_NEW_CONNECTION\n");
         break;
      case NBNCP_DISCONNECTED:
          connected_circuit = NULL;
          printf("status_change(): NBNCP_DISCONNECTED\n");
         break;
      case NBNCP_CLOSED:
          printf("status_change(): CLOSED\n");
          defined_circuit = NULL;
         break;
      }
   }  /* nc_status_change */

  /* Function data_recv()
   * Callback function for data received on the circuit.
   *
   * *circuit is the circuit that the data is received on, len the length of
   * data received and *buf the data.
   * *time_stamp is the time stamp when the data was received.
   *
   * Return zero on success. If an error occurs return -1, which will
   * result in the circuit being closed.
   */
static int data_recv(nbncp_circuit_t *circuit,int length,
      unsigned char *buffer,struct timeval *time_stamp)
   {
   printf("Received length %d:\n",length);
   printf("[%.*s]\n",length,buffer);
   return 0;
   }  /* data_recv */

  /* Function open_circuit()
   */
static void open_circuit(char *fname)
   {
   char *parms[20];          /* NCP parameters */
   char response[200];       /* user supplied value for parameter */
   int i,p,rc;
   FILE *fd;                 /* user open parms */
   char buf[80],*cp;         /* parameter input */
```

```
if (defined_circuit)
   {
   printf("Error: already opened a circuit\n");
   return;
   }
p = 0;

if (*fname)
   {
   if ((fd = fopen(fname, "r")) == NULL)
      {
      printf("Error opening %s: %s\n",fname, strerror(errno));
      return;
      }
   while (fgets(buf,sizeof(buf),fd))
      {
      if (cp = strchr(buf,'\r')) *cp = '\0';
      if (cp = strchr(buf,'\n')) *cp = '\0';
      if (!strlen(buf)) continue;
      if (buf[0] == '#') continue;
      parms[p] = strdup(buf);
      printf("   Using parameter setting: \"%s\"\n",parms[p]);
      p++;
      }
   fclose(fd);
   }
else
   {
  /* Ask the user for required parameters:
   */
   printf("Enter values for required parameters:\n");
   for (i = 0; ncp->required[i].name; i++)
      {
      printf("%s\n   Enter value for %s: ",ncp->required[i].desc,
            ncp->required[i].name);
      printf("\n");
      fgets(response,sizeof(response),stdin);
      response[strlen(response)-1] = 0;
      if (response[0])
         {
         parms[p] = malloc(200);
         sprintf(parms[p],"%s=%s",ncp->required[i].name,response);
         printf("   Using parameter setting: \"%s\"\n",parms[p]);
         p++;
         }
      }

   printf("Enter values for optional parameters:\n");
   for (i = 0; ncp->optional[i].name; i++)
      {
      printf("%s\n   Enter value for %s: ",ncp->optional[i].desc,
```

```
                ncp->optional[i].name);
            printf("\n");
            fgets(response,sizeof(response),stdin);
            response[strlen(response)-1] = 0;
            if (response[0])
               {
               parms[p] = malloc(200);
               sprintf(parms[p],"%s=%s",ncp->optional[i].name,response);
               printf("\n   Using parameter setting: \"%s\"\n",parms[p]);
               p++;
               }
            }
         }

   parms[p] = NULL;
   defined_circuit = ncp->open(ncp,parms,NULL);
   if (!defined_circuit)
      printf("Circuit open error: %s\n",ncp->last_error);
   for (p = 0; parms[p]; p++) free(parms[p]);
   }  /* open_circuit */

  /* Function read_stdin()
   * Reads stdin and process user request.
   * Returns zero on success, otherwise -1.
   */
static int read_stdin(void)
   {
   char *cp, buf [2000];

   memset(buf,0,sizeof(buf));
   if ((cp = fgets(buf, sizeof(buf)-1, stdin)) == NULL)
      {
      fprintf(stderr,"CANNOT READ STDIN - aborted");
      return -1;
      }
   if (cp = strchr(buf,'\n'))
      *cp = '\0';
   if (!strlen(buf)) return 0;

   switch (tolower(buf[0]))
      {
      case 'q':
         exit(0);
         break;
      case 'o':
         open_circuit(buf+2);
         break;
      case 's':
         if (!connected_circuit)
            {
            printf("Error: No connected circuit\n");
            return;
```

```
            }
         if (strlen(buf) < 3) return 0;
         ncp->send(connected_circuit,strlen(buf)-2,buf+2);
         break;
      default:
         printf("Unknown action: %c\n",buf[0]);
         printf("\to <optional parms file> - open a circuit\n");
         printf("\ts <msg> - send the msg on last connected circuit\n");
         printf("\tq - quit\n");
         break;
      }

   return 0;
   } /* read_stdin */
```