CODE MAGUS

NCPTNVT: Network Control Program using Telnet and VT data streams

CML00019-01

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

Business Partner IBM

August 16, 2016

Contents

1 Library Interface

1.1 Introduction

The Code Magus Limited `NCPTNVT` Dynamic Link Library (DLL)[1] provides standard Code Magus Network Control Program (NCP) functions to send and receive a video terminal (VT) data stream encapsulated in a telnet session. It allows the caller to connect, send and close a VT session. The DLL keeps an image of the screen in memory and the caller can retrieve this at any time. The telnet data stream is boundless in that there are no message delimiters, but the NCP offers the ability to recognise when the screen is in a certain state, which allows the caller to know that a reply has or has not been received in response to a message that was sent.

2 `NCPTNVT` NCP Interface reference

The NCP interface is defined in the header `nipncp.h`.

2.1 `tnvt_init()`

2.1.1 Synopsis

```
ncptli_desc_t *ncptli_init(void)
```

2.1.2 Description

The `tnvt_init` function allocates a data structure and returns a pointer to it that is used to describe one instance of this NCP. All other functions to this NCP instance are accessed via function pointers in this structure.

2.1.3 Parameters

There are no parameters to this function.

2.1.4 Return Value

This function must succeed and returns the address of the NCP instance descriptor.

[1]Shared Object on Linux, Unix and z/OS

2.2 Open

2.2.1 Synopsis

```
typedef ncptli_circuit_t *(*ncptli_open_t)(ncptli_desc_t *dlldesc,
      char *parms[]);
(ncptli_desc_t *dlldesc)->open;
```

2.2.2 Description

This function connects to a remote telnet server and readies the caller for interacting with the host application.

2.2.3 Parameters

Parameters are passed to the function via an array of strings comprising `keyword=value` pairs. The following list explains the parameters in more detail.

- `HOSTNAME`
  This mandatory parameter names the host machine to connect to using the telnet protocol. It may be supplied as a domain name (e.g. www.codemagus.com) or as a dotted decimal IPv4 address (e.g. 192.168.17.1).
- `ROWS`
  This optional parameter specifies the size of the telnet and VT screen in rows. It may be 24 or 132. The default is 24.
- `COLUMNS`
  This optional parameter specifies the size of the telnet and VT screen in columns. If specified it must equal the default of 80. Although this may only have one value it is included so that in the future different size screens may be used.
- `HOSTPORT`
  This optional parameter specifies the host port to connect to on the machine defined by `HOSTNAME`. If not specified it defaults to the well known telnet port 23.
- `TWBLOG`
  This optional parameter specifies the name of a log file to which the NCP will log the data sent and received during the conversation with the host. This log is written using the Code Magus TWB log format and can then be analysed using the Code Magus TWB log utilities.

2.2.4 Return Value

On success the address of the circuit descriptor is returned, otherwise NULL is returned and an error message describing the failure is placed in `last_error` of the associated NCP descriptor.

2.3 Send

2.3.1 Synopsis

```
typedef int (*ncptli_send_t)(ncptli_circuit_t *circuit,
      int len, unsigned char *buf);
(ncptli_desc_t *dlldesc)->send;
```

2.3.2 Description

The send function sends the buffer `buf` to the connected remote host and then performs an internal receive to update the in memory screen image. Control is only returned to the caller once the screen image has reached the specified state, an error occurs or one of two time outs occur.

The internal receive function uses the two time-out attributes `Timeout` (section 2.8.1 on page 8) and `MessageCompletionTime` (section 2.8.2 on page 8) to determine when no data or some (but possibly not all) has been received. The `StopMask` (section 2.8.3 on page 9) attribute is used to determine, by matching the screen image to it, whether all the data has been received and the screen image has reached the desired state.

The telnet session uses no framing of messages and usually the only indication that all the data has been received is that the user looking at the terminal recognises that they can now enter more data. Usually the telnet session is in echo mode and the characters typed on the terminal (in this case sent by the NCP) are echoed back immediately. After this there could then be an amount of time to wait while the host processes the request and sends a response. The `Timeout` value specifies the maximum amount of time the NCP will wait for the initial response (the echoed data at least) and the `MessageCompletionTime` time out specifies the maximum amount of time it will wait for the rest of the response to arrive.

The screen image is matched to the `StopMask` after any data is received from the host in order to determine if it is in the required state or not. If after the initial receive (of the echo data) the screen is not in the required state then the NCP goes into a receive loop for the residual data otherwise, if it is, control is returned to the caller. In the loop to receive the residual data the NCP matches the screen image to the `StopMask` each time and returns control to the caller immediately if the screen is in the required state, otherwise the wait time is decremented and the NCP waits for more data. If no data is received within this wait time (i.e. the screen has not reached the required state) the time out is reported to the caller.

2.3.3 Parameters

- `circuit`
  The pointer returned from the open circuit request must be passed back in order to identify the circuit the send is for.
- `len`
  This specifies the length in bytes of the buffer to send.
- `buf`
  This parameter contains the request to be sent; as this is usually the data a terminal operator would type, either a command or an entry on a curses screen, it is in plain text.

2.3.4 Return Value

- Success
  On success the number of bytes sent is returned. This means that the message was sent successfully and a response has altered the in memory screen image and it is in the state described by the `StopMask`.
- Failure
  If a system or non recoverable failure occurs -1 is returned and an error message describing the error is placed in `last_error` of the associated NCP descriptor.
- Data Time Out
  No data has been received from the server within the time specified by the attribute `Timeout`, in which case -2 is returned and a message describing the timeout is placed in `last_error` of the associated NCP descriptor.
- Message Completion Timeout
  Data has been received from the server within the time specified by the `Timeout` attribute but the screen has not reached the state as described by the `StopMask` attribute and no more data has been received within the time specified by the attribute `MessageCompletionTime`, in which case -2 is returned and a message describing the timeout is placed in `last_error` of the associated NCP descriptor.

2.4 Receive

2.4.1 Synopsis

```
typedef int (*ncptli_recv_t)(ncptli_circuit_t *circuit,
      int len, unsigned char *buf);
(ncptli_desc_t *dlldesc)->recv;
```

2.4.2 Description

The function `receive()` returns the current screen image to the caller. It may attempt to receive data from the server, but no error or time out occurs as a result of this attempt.

2.4.3 Parameters

- `circuit`
  The pointer returned from the open circuit request must be passed back in order to identify the circuit the send is for.
- `len`
  This specifies the maximum length in bytes of `buf`.
- `buf`
  This parameter is a pointer to an area where on success the screen image can be copied to.

2.4.4 Return Value

On success this function returns the size of the screen in bytes, otherwise -1 is returned and an error message describing the error is placed in `last_error` of the associated NCP descriptor.

2.5 Progress

2.5.1 Synopsis

```
typedef int (*ncptli_progress_t)(ncptli_desc_t *dlldesc);
(ncptli_desc_t *dlldesc)->progress;
```

2.5.2 Description

The progress function does not do anything in this implementation as the receive and send functions perform their operations directly.

2.5.3 Parameters

- `dlldesc`
  The pointer returned from the `tnvt_init` function must be passed back in order to identify the descriptor that is being cleaned.

2.5.4 Return Value

This function returns zero.

2.6 Close

2.6.1 Synopsis

```
typedef int (*ncptli_close_t)(ncptli_circuit_t *circuit);
(ncptli_desc_t *dlldesc)->close;
```

2.6.2 Description

This function closes an open circuit and frees all related resources. Once complete the circuit may no longer be used.

2.6.3 Parameters

- `circuit`
  The pointer returned from the open circuit request must be passed back in order to identify the circuit the close is for.

2.6.4 Return Value

On success zero is returned. This function is expected to succeed but if it does not, -1 is returned and an error message describing the failure is placed in `last_error` of the associated NCP descriptor.

2.7 Clean Up

2.7.1 Synopsis

```
typedef int (*ncptli_cleanup_t)(ncptli_desc_t *dlldesc);
(ncptli_desc_t *dlldesc)->cleanup;
```

2.7.2 Description

This function releases all resources related to the NCP descriptor `dlldesc`. All circuits associated with this descriptor must be closed before calling this function. Once complete the descriptor may no longer be used.

2.7.3 Parameters

- `dlldesc`
  The pointer returned from the `tnvt_init` function must be passed back in order to identify the descriptor that is being cleaned.

2.7.4 Return Value

On success zero is returned. This function is expected to succeed but if it does not -1 is returned and an error message describing the failure is placed in `last_error` of the associated NCP descriptor.

2.8 `NCPTNVT` NCP Interface Attributes

The following attributes are exposed by the `NCPTNVT` NCP Interface..

2.8.1 `Timeout`

The `Timeout` attribute can be set or retrieved. It determines the amount of time in milliseconds that the NCP will wait to begin receiving data from the host.

2.8.2 `MessageCompletionTime`

The `MessageCompletionTime` attribute can be set or retrieved. It determines the amount of time in milliseconds that the NCP will wait to complete receiving data from the host once it has received some data. This value allows for the fact that some hosts may respond initially to a request (usually the echoed input from the user) but then need time to process the request before completing the message. A human at a terminal running the transaction would wait until the screen has completely updated before reacting and entering more data or commands. This value attempts to model that behaviour in an automated test scenario.

2.8.3 `StopMask`

The `StopMask` attribute can be set or retrieved. It specifies a regular expression that when matched against the current screen can be used to determine if the data received from the server has updated the screen to a point where the client can continue the conversation by sending new data to the server. It therefore describes a state that the caller requires the screen to be in before being able to continue with the conversation.

2.8.4 `StartOfMatch`

The `StartOfMatch` attribute can only be retrieved. It will return the offset into the screen of the start of the first sub string that matched the regular expression as defined in `StopMask`.

2.8.5 `EndOfMatch`

The `EndOfMatch` attribute can only be retrieved. It will return the offset into the screen of the end of the first sub string that matched the regular expression as defined in `StopMask`.

2.8.6 `TerminalTitle`

The `TerminalTitle` attribute can only be retrieved. It will return the VT screen title as set by the operating system command (OSC) string sent from the server. If no such command has been sent by the server an empty string is returned. In other words this attribute will always return a valid NULL terminated string.

A Header file **nipncp.h**

```
#ifndef NIPNCP_H
#define NIPNCP_H
  /* File: nipncp.h
   *
   * This header file describes the interface between an Eresia Network
   * Injection Portal's Network Control Program and the Network Injection
   * Portal itself. The Network Control Programs are DLLs that are nominated
   * and loaded by the Network Injection Portal for executing usecases on a
   * network interface. The Network Control Program is responsible for the
   * establishment, transmission, control and tear-down of network circuits.
   *
   * Copyright (c) 2004 Code Magus Limited. All rights reserved.
   *
   * Author: Stephen Donaldson.
   */

  /*
   * $Author: stephen $
   * $Date: 2005/10/07 18:01:45 $
   * $Id: nipncp.h,v 1.1.1.1 2005/10/07 18:01:45 stephen Exp $
   * $Name:  $
   * $Revision: 1.1.1.1 $
   * $State: Exp $
   *
   * $Log: nipncp.h,v $
   * Revision 1.1.1.1  2005/10/07 18:01:45  stephen
   * Add NIP NCP interface to repository
   *
   * Revision 1.1.1.1  2005/09/21 17:50:49  stephen
   * Import sources in Oxford CVS
   *
   * Revision 1.1.1.1  2005/09/17 18:41:35  stephen
   * Initial add to repository
   *
   * Revision 1.1.1.1  2004/12/29 15:37:43  stephen
   * Initial import of NCPFILE DLL prototype
   *
   * Revision 1.1.1.1  2004/10/31 13:53:19  stephen
   * Add TS Replay NIP NCP DLL
   *
   * Revision 1.1.1.1  2004/10/31 13:36:04  stephen
   * TS Replay feature Standard NIP
   *
   * Revision 1.1.1.1  2004/05/15 00:45:51  stephen
   * Import sources into CVS
   *
   */

static char *cvs_nipncp_h =
   "$Id: nipncp.h,v 1.1.1.1 2005/10/07 18:01:45 stephen Exp $";
  /*
   * Constants and options.
   */

#define NCPTLI_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 ncptli_desc ncptli_desc_t;       /* NCP DLL descriptor */
typedef struct ncptli_circuit ncptli_circuit_t; /* circuit state structure */

  /* 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 ncptli_value ncptli_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 (*ncptli_cleanup_t)(ncptli_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 desciptive message is
   * placed in last_error in the ncptli_desc_t struture.
   */
```

```
typedef ncptli_circuit_t *(*ncptli_open_t)(ncptli_desc_t *dlldesc,
      char *parms[]);

  /* 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 retruned and a desciptive message is placed in last_error in
   * the ncptli_desc_t struture.
   */

typedef int (*ncptli_close_t)(ncptli_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 curcuit 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 ncptli_desc_t structure.
   */

typedef int (*ncptli_attrset_t)(ncptli_circuit_t *circuit,
      char *attribute, ncptli_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 ncptli_desc_t structure.
   */

typedef int (*ncptli_attrget_t)(ncptli_circuit_t *circuit,
      char *attribute, ncptli_value_t *value);

   /* By default the send and recv functions are non-blocking. The can be
    * changed to blocking calls simply by changing the mode in the
    * ncptli_circuit_t structure. Furthermore, if the timeout value in the
    * ncptli_circuit_t structure is set to a non-zero value, then the
    * block will timeout after the given number of microseconds.
    */

   /* Function send takes a raw buffer and a length and queues that
    * buffer for transmission on the underlying transport layer
    * ineraface. If the send succeeds then 0 is returned, otherwise
    * -1 is retruned and a desciptive message is placed in last_error in
    * the ncptli_desc_t struture.
    */

typedef int (*ncptli_send_t)(ncptli_circuit_t *circuit,
      int len, unsigned char *buf);

   /* Function recv dequeues the next full buffer received on the
    * circuit and copies its contents to the caller's raw buffer. The length
    * of data moved to the buffer is returned by the function. If there are
    * no full buffers to consume then the function returns 0. If an error
    * occurs then the function returns -1 and a descriptive message is
    * placed in last_error in the ncptli_desc_t struture.
    */

typedef int (*ncptli_recv_t)(ncptli_circuit_t *circuit,
            int len, unsigned char *buf);

  /* Function progress inspects all the circuits on the passed descriptor and
   * processes any data that might be ready for reading or available for
   * writing. The progress call can also be made into a blocking call
   * if any of the circuits chained off the circuit are operating in a
   * blocking mode. This function returns -1 if an error
   * occurs (in which case last_error will contain an error message);
   * otherwise the function returns the number of circuits that were
   * progressed for reading plus the number of circuits that were
   * progressed for writing.
   */

typedef int (*ncptli_progress_t)(ncptli_desc_t *dlldesc);

  /* The first structure is the ncptli_desc_t which is a descriptor of
   * the loaded DLL. The ncptli_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 ncptli_desc
   {
   ncptli_cleanup_t cleanup;     /* prepare NCP DLL for unloading */
   ncptli_open_t open;           /* NCP cicuit open function */
   ncptli_close_t close;         /* NCP cicuit close function */
   ncptli_send_t send;          /* NCP circuit send data function */
   ncptli_recv_t recv;          /* NCP circuit receive data function */
   ncptli_progress_t progress;  /* advance all open circuits if possible */
   ncptli_attrset_t attrset;    /* set attribute value on circuit */
   ncptli_attrget_t attrget;    /* get attribute value from circuit */

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

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

  /* The second controls the instances of the circuits created by the
   * DLL. There is only one exposed entry point to the DLL and this
   * entry point is called by the using application once the DLL has
   * been loaded. This call has no parameters and returns a structure
   * which contains pointers to all the other functons of
   * the DLL.
   */

struct ncptli_circuit
   {
   ncptli_desc_t *dlldesc;        /* DLL descriptor owning circuit */
   enum {NCP_NONBLOCKING, NCP_BLOCKING} mode;   /* call blocking mode */
   unsigned long timeout;         /* millisecond timeout value */
   void *private_data;            /* circuit type specific instance data */
   };

typedef struct ncp_tli ncp_tli_t;   /* transport layer interface instance */
typedef struct ncp_buf ncp_buf_t;   /* transport message cross the interface */

  /* 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 ncptli_value
   {
   int length;                     /* length of attribute value */
   unsigned char *value;           /* value of attribute */
   };

  /* Function ncptli_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.
   */

ncptli_desc_t *ncptli_init(void);
```

```
typedef ncptli_desc_t *(*ncptli_init_t)(void);

#endif /* NIPNCP_H */
```