Lua スクリプト機能簡易ドキュメント(ver6.3.1+Lua build 1018)

センチュリー・システムズ(株)

1. 序

Lua スクリプト機能とはユーザが開発した Lua プログラムコード(以降、Lua スクリプト)を NXR 上で実行する機能である。
本文書は NXR 上での Lua スクリプトの実行方法や Lua プログラムを記述する上での注意点などを記述する。
Lua についての詳しい説明は省略する。

2. Lua

Lua のバージョンは 5.2.3 をベース。
ただし一部ライブラリについては使用制限または使用できないものがある。

3. 準備

3-1 Lua スクリプトの準備

(1) スクリプトファイルのインポート

ユーザはコマンドラインインタフェース(CLI)から外部のサーバに保存された Lua スクリプトファイルを NXR 本体にインポートすることができる。

また、 外部記録装置(USB フラッシュ、SD カード(以下ディスク))にあらかじめ保存した Lua スクリプトファイルを NXR 本体にインポートすることもできる。

コマンドの詳細については「3.2 (2) Lua スクリプトファイルのインポート」を参照。

(2) ディスク

ユーザは NXR に装着したディスクに保存された Lua スクリプトファイルを実行することができる。
ディスクは NXR で使用できるようにフォーマットしておく必要がある。
ディスクが使用可能になったら、以下のように copy コマンドでディスクに保存する。

```
# copy URL disk0:NAME
```

また、一度フォーマットされたディスクを PC などに接続しファイルをディスクにコピー後 NXR へ再度装着することも可能。

3-2 Lua スクリプト機能コマンドリファレンス

(1) Lua スクリプト機能の ON/OFF

[コマンド](configure mode)

lua enable

no lua enable

[説明]

Lua スクリプト機能の有効／無効を設定する。

機能が無効の場合、run lua コマンドを実行してもスクリプトは起動されない。

(2) Lua スクリプトファイルのインポート

2-1) インポート

[コマンド](configure mode)

lua script-file NAME ssh://USER@ADDRESS/FILENAME (|soucce A.B.C.D|X:X::X:X)

lua script-file NAME ftp://ADDRESS/FILENAME (|soucce A.B.C.D|X:X::X:X)

lua script-file NAME (disk0:FILENAME|disk1:FILENAME)

[説明]

Lua スプリプトファイルをインポートする。

[実行例]

lua script-file test.lua ssh://user@192.168.0.1/lua/test.lua

2-2) コンパイルしてインポート

[コマンド](configure mode)

lua script-file NAME ssh://USER@ADDRESS/FILENAME (|soucce A.B.C.D|X:X::X:X) compile

lua script-file NAME ftp://ADDRESS/FILENAME (|soucce A.B.C.D|X:X::X:X) compile

lua script-file NAME (disk0:FILENAME|disk1:FILENAME) compile

[説明]

Lua スプリプトファイルをバイトコードファイルに変換して保存する。

[実行例]

lua script-file test.lua ssh://user@192.168.0.1/lua/test.lua compile

2-3) 複数のスクリプトファイルをコンパイルしてインポート

[コマンド](configure mode)

lua script-file NAME compile (disk0:FILENAME1|disk1:FILENAME1) (|disk0:FILENAME2|disk1:FILENAME2) ...

[説明]

複数の Lua スプリプトファイルを 1 つのバイトコードファイルにまとめて保存する。

[実行例]

lua script-file test.lua compile disk0:test1.lua disk0:test2.lua

2-4) Lua スクリプトファイルを削除する

[コマンド](configure mode)

no lua script-file NAME

[説明]

Lua スプリプトファイル(バイトコードファイル)を削除する。

(3) Lua スクリプトファイルのコンパイル

[コマンド](exec mode)

lua script-file disk0:FILENAME|disk1:FILENAME compile (disk0:FILENAME1|disk1:FILENAME1)(|disk0:FILENAME2|disk1:FILENAME2) …

[説明]

複数の Lua スプリプトファイルを 1 つのバイトコードファイルにまとめてディスクに保存する。

[実行例]

run lua-compiler disk0:test.bin disk0:test1.lua disk0:test2.lua

(4) 実行

[コマンド](exec mode)

run lua disk0:NAME

run lua NAME

[説明]

NAME で指定された Lua スクリプトを実行する。のコマンドで Lua スクリプトを起動する。

(5) 情報表示

4-1) ファイル情報表示

[コマンド](exec mode)

show lua file

[説明]

NXR 本体に保存したファイルの属性を表示する。

4-2) 状態表示

[コマンド](exec mode)

show lua status

[説明]

実行中の Lua スクリプトの状態を表示する。

一意に付与した ID, 実行したスクリプト名、引数(指定した場合)、起動時刻を表示する。

[実行例]

show lua status

test1.lua: ASCII text

test2.bin: Lua bytecode,

(4) スクリプトの停止

[コマンド](exec mode)

terminate lua <ID>

[説明]

ID で指定した実行中の Lua スクリプトをただちに停止する。

(5) Lua スクリプト機能の再起動

[コマンド](exec mode)

restart lua

[説明]

Lua スクリプト機能を再起動する。

実行中の Lua スクリプト全て停止する。

4. API

4-1. Lua 標準ライブラリ

1. 基本関数
   使用可能。
2. コルーチン操作
   使用可能。
3. モジュール
   使用可能。
4. 文字列操作
   使用可能。
5. テーブル操作
   使用可能。
6. 数学関数
   使用不可。
7. ビット演算
   使用可能。
8. 入出力機能
   以下の関数は使用できない。
   [関数]
   io.popen(), io.pclose(), io.tmpfile()
9. OS 機能
   使用不可。
10. デバッグライブラリ
    使用不可。

4-2. NXR ライブラリ

＊ nxr.cli(command)

CLI コマンドを実行します。

第 1 引数: コマンドラインを文字列で指定。

第 1 戻り値: 実行の成否を示す boolean を返す。

第 2 戻り値: 実行したコマンドラインの出力またはエラー内容を文字列で返す。

実行できるコマンドは下記の通り。

＊ clear コマンド

＊ connect コマンド

＊ dir コマンド

＊ disconnect コマンド

＊ get コマンド

＊ ping コマンド

ただし、"repeat"が省略された場合、"repeat 1"として実行する。

＊ reconnect コマンド

＊ reset コマンド

＊ restart コマンド

ただし、システム再起動をする"restart"は実行できない。

＊ show コマンド

＊ traceroute コマンド

＊ wol コマンド

＊ nxr.mail_send(mail, server, body)

メールを送信する。

第 1 引数: メール設定をテーブルで指定する

```
mail = {
    to = MAIL_TO, -- 宛先アドレスを文字列で指定する。
    form = MAIL_FORM, -- 送信元アドレスを文字列で指定する。
    subject = SUBJECT, -- メールのサブジェクトを文字列で指定する。
}
```

第 2 引数: メールサーバに関する設定を数値またはテーブルで指定する。

数値の場合、CLI の mail server コマンドの番号で指定されたサーバを使用する。

```
server = {
    auth_type = TYPE,
    -- 認証タイプを文字列で指定する。省略した場合は認証なし。
    -- "pop-before-smtp": POP before SMTP
    -- "smtp-auth-login": SMTP authentication(login)
    -- "smtp-auth-plain": SMTP authentication(plain)
    account = {
        username = USERNAME,
        password = PASSWORD,
```

```
        },
        -- 認証に使用するアカウントのユーザ名、パスワードを文字列で指定。
        smtp = {
            address = SMPT_ADDRESS,
            port = SMTP_PORT,
        },
        -- SMPT のアドレスを文字列で、ポート番号を数値で指定。
        -- ポート番号は省略可能。
        pop3 = {
            address = POP3_ADDRESS,
            port = POP3_PORT,
        }
        -- POP3 のアドレスを文字列で、ポート番号を数値で指定。
        -- ポート番号は省略可能。
    }
```

第 3 引数: メールの本文を文字列で指定。

第 1 戻り値: 処理の成否を boolean で戻す。

* nxr.sleep(seconds)

スクリプトの実行を遅延する。

第 1 引数: 遅延する時間(秒)を数値で指定。

第 1 戻り値: 処理の成否を boolean で戻す。

* nxr.syslog(priority, message)

システムログを出力する。

第 1 引数: ログのプライオリティを文字列で指定。

"info", "notice", "debug"が指定可能。

第 2 引数: 出力するメッセージを文字列で指定。

第 1 戻り値: 処理の成否を boolean で戻す。

* serial.open(arguments)

シリアルポート(SERIAL 1)を開く。

第 1 引数: ポートに関する設定をテーブルで指定。

```
    arguments = {
        baudrate = BUADRATE,
        -- ボーレートを数値で指定。
        -- 指定可能なボーレートは
        -- 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800,
        -- 9600, 19200, 38400, 57600, 115200, 230400
        echo = ECHO,
        -- エコーするかどうかを boolean で指定。
    }
```

第 1 戻り値: 新しいハンドラを返す。失敗した場合は nil を返す。
第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

※ 入力はカノニカル入力処理、すわなち行単位で処理される。

＊ serial:close()
シリアルポートを閉じる。
第 1 戻り値: 処理の成否を boolean で返す。
第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

＊ serial:read()
シリアルポートから読み込みを行なう。
第 1 戻り値: 処理の成否を boolean で返す。
第 2 戻り値: 読み出した内容またはエラー内容を文字列で返す。

※ 改行コードは削除される。

4-3 Socket ライブラリ
LuaSocket(http://w3.impa.br/~diego/software/luasocket/home.html)より C 言語で記述されたライブラリのみサポート。
TCP/UDP のトランスポート層および MIME から構成される。

TCP socket では master socket, server socket, client socket の 3 つに分類される。
master socket は socket.tcp により生成されるオブジェクト。
master socket から listen をコールすることで server socket に遷移する。
また master socket から connect メソッドにより client socket に遷移する。
server socket が accept メソッドにより TCP コネクションの接続待ちを行い、接続が確立すると client socket としてオブジェクトが返る。

(1) Socket
＊ gettime()
機器が起動してからの経過時間を秒単位で返す。

＊ socket.newtry(finalizer)
例外を発生させる前にクリーンアップする関数を生成する。
第 1 引数: 例外を発生させる前に呼び出す関数。セーフモードで実行される。

＊ socket.protect(func)
関数を例外を発生する再にセーフモードとなるように変換する。
try または newtry が発生される例外のみを補足するが通常の Lua のエラーは補足されない。
第 1 引数: try, newtry, assert を呼んで例外をスローする関数を指定する。

＊ select(recvt, sendt [, timeout])

ソケットの状態を監視する。
第 1 引数: 読み込み可能な状態のソケットの配列
第 2 引数: 書き出し可能な状態のソケットの配列
第 3 引数: 状態を監視する最大時間を秒単位で指定。nil、負の値または省略時は無期限となる。
第 1 戻り値: 成功した場合、読み取りの準備が出来たオブジェクトのリストを返す。
失敗した場合、nil を返す。
第 2 戻り値: 成功した場合、書き込みの準備が出来たオブジェクトのリストを返す。
失敗した場合、nil を返す。
第 3 戻り値: 失敗した場合、エラー内容を文字列で返す。

＊ socket.skip(D [, ret1, ret2 ... retN])
指定した引数の数を削除し、残りを返す。
ret1, ret2, ... retN の引数のうち、先頭から D 個の引数を削除する。
第 1 戻り値: 削除する引数の数を指定する。
戻り値: 削除後の retD+1 から retN までを返す。
※この関数はダミー変数の作成を回避するために役立つ。

＊ socket.sleep(time)
指定された時間、プログラムの実行を遅延させる。
第 1 引数: 遅延する時間(秒)を数値で指定。

＊ socket._VERSION
LuaSocket のバージョンを文字列で返す。

(2) DNS(in socket)
＊ socket.dns.gethostname()
機器のホスト名を文字列として返す。
＊ socket.dns.tohostname(address)
IP アドレスからホスト名に変換する。
第 1 引数: アドレスを指定する。
第 1 戻り値: ホスト名を文字列で返す。失敗した場合は nil を返す。
第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。
＊ socket.dns.toip(address)
ホスト名から IP アドレスに変換する。
第 1 引数: ホスト名を指定する。
第 1 戻り値: 最初に見つかった IP アドレスを文字列で返す。失敗した場合は nil を返す。
第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

(3) TCP(in socket)
＊ socket.tcp()
TCP のマスターオブジェクトを返す。
第 1 戻り値: 新しい master オブジェクトを返す。失敗した場合 nil を返す。

第 2 戻り値: 失敗した場合、エラー内容を文字列で返す。

＊ server:accept()

リモートからの接続を待機し、TCP コネクションが確立した時 client オブジェクトを返す。

第 1 戻り値: client オブジェクトを返す。タイムアウトまたは失敗時は nil を返す。

第 2 戻り値: タイムアウト時は文字列"timeout"を、その他のエラー時はその内容を文字列で返す。

＊ master:bind(address, port)

master オブジェクトに IP アドレスとポート番号を割り当てる。

第 1 引数: IP アドレスまたはホスト名を文字列で指定。"*"を指定した場合、全てのインタフェースに割り当てる。

第 2 引数: ポート番号を 0-65535 の範囲内で指定。0 を指定した場合、自動的に 1024 - 65535 の範囲内で割り当てる。

第 1 戻り値: 成功した場合は 1 を、失敗した場合は nil を返す。

第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

＊ master:close()

＊ clinet:close()

＊ server:close()

TCP のオブジェクトをクローズする。

オブジェクトによって使用されるソケットをクローズしオブジェクトに bind されたアドレスおよびポートは他のアプリケーションで使用可能となる。

＊ master:connect(address, port)

master オブジェクトをリモートホストへ接続する。

オブジェクトを client オブジェクトへ変換する。

client オブジェクトでは send, receive, getsockname, getpreename, settimeout,

close の各オブジェクトが使用可能。

第 1 引数: IP アドレスまたはホスト名を文字列で指定。

第 2 引数: ポート番号を 1-65535 の範囲内で指定。

第 1 戻り値: 成功した場合は 1 を、失敗した場合は nil を返す。

第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

＊ client:getpeername()

接続したリモート側の client オブジェクトの情報を返す。

第 1 戻り値: 成功した場合、リモートの IP アドレスを返す。

失敗した場合、nil を返す。

第 2 戻り値: 成功した場合、リモートのポート番号を返す。

＊ master:getsockname()

＊ client:getsockname()

＊ server:getsockname()

オブジェクトに関連づけられたローカルアドレス情報を返す。

第 1 戻り値: 成功した場合、ローカルの IP アドレスを返す。
失敗した場合、nil を返す。
第 2 戻り値: 成功した場合、ローカルのポート番号を返す。

＊ master:getstats()
＊ client:getstats()
＊ server:getstats()
ソケットのアカウンティング情報を返す。
第 1 戻り値: オブジェクトで受信したバイト数を返す。
第 2 戻り値: オブジェクトでした送信バイト数を返す。
第 3 戻り値: オブジェクトの生存時間(秒)を返す。

＊ master:listen(backlog)
オブジェクトをコネクションの接続待ちにする。
オブジェクトを server オブジェクトに変換する。
server オブジェクトでは accept, getsockname, setoption, settimeout, close の各オブジェクトが使用可能。
第 1 引数: 保留中の接続のキューの最大長を指定する。
第 1 戻り値: 成功した場合は 1 を、失敗した場合は nil を返す。
第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

＊ client:receive([pattern [, prefix]])
指定された受信パターンに従って client オブジェクトからデータを読み込む。
第 1 引数: 受信パターンの指定は以下の通り。
"*a": 接続が閉じられるまで、ソケットから読み込む。
"*l": 改行コード LF(0x0A)、または CR(0x0D)+LF で終端する 1 行をソケットから読み込む。
読み込んだデータには改行コード(LF, CR+LF)は含まれない。
デフォルトのパターン。
数値: 指定したバイト数だけソケットから読み出す。
第 2 引数: 受信したデータの前に指定した文字列を付加する。
第 1 戻り値: 成功した場合は受信データを返す。失敗した場合は nil を返す。
第 2 戻り値: 受信が完了前に切断された場合は"close"、タイムアウトが発生した場合は"timeout"を返す。その他の場合はエラー内容を文字列で返す。
第 3 戻り値: エラー発生時にエラー発生前までに受信したデータを返す。

＊ client:send(data [, i [, j]])
client オブジェクトを介してデータを送信する。
第 1 引数: 送信するデータを文字列で指定する。
第 2 引数: 送信するデータを開始位置を数値で指定する。
第 3 引数: 送信するデータを終了位置を数値で指定する。
第 1 戻り値: 成功した場合は送信したデータの位置を返す。失敗した場合は nil を返す。
第 2 戻り値: 失敗した場合、エラー内容を文字列で返す。
タイムアウトが発生した場合は"timeout"を返す。

第 3 戻り値: 失敗した場合、送信できた data の最後の位置を返す。

＊ client:setoption(option [, value])

＊ server:setoption(option [, value])

TCP オブジェクトのオプションを設定する。

第 1 引数: オプション名を文字列で指定する。第 2 引数にオプションに対する値を指定する。

'keepalive': Keepalive の有効／無効を true/false で指定する。

'linger': linger の設定についてテーブルで指定する。

テーブルは on(true/false), timeout(数値)のエントリを持つ。

'reuseaddr': ローカルアドレスの再利用を許可するかどうかを tru/false で指定する。

'tcp-nodelay': Nagle のアルゴリズムを有効／無効を true/false で指定する。

＊ master:setstats(received, sent, age)

＊ client:setstats(received, sent, age)

＊ server:setstats(received, sent, age)

ソケットのアカウンティング情報を指定した値にリセットする。

第 1 引数: 受信バイト数を指定する。

第 2 引数: 送信バイト数を指定する。

第 3 引数: 生存時間(秒)を指定する。

第 1 戻り値: 成功した場合は 1 を、失敗した場合は nil を返す。

＊ master:settimeout(value [, mode])

＊ client:settimeout(value [, mode])

＊ server:settimeout(value [, mode])

TCP オブジェクトのタイムアウト値を変更する。

デフォルトでは全ての I/O 操作がブロッキングされおり、メソッドの呼び出しは send, receive および accept 操作が完了するまで無期限にブロックする。

settimeout メソッドは I/O メソッドがブロック出来る時間制限を定義する。

タイムアウトが設定され、指定した時間が経過するとブロックしていたメソッドは処理を中止しエラーを返す。

第 1 引数: タイムアウト時間を秒単位で指定する。

nil または負の値を指定した場合は処理が完了するまで無期限にブロックする。

第 2 引数: タイムアウトモードを以下のうちから指定する。

'B': ブロックタイムアウト。1 つの I/O 操作の完了を待つ間にブロックできる時間を指定する。(デフォルト)

'T': トータルタイムアウト。LuaSocket が呼び出しから戻る前にブロックできる時間を指定する。

＊ client:shutdown(mode)

全二重接続の一部または全部をシャットダウンする。

第 1 引数: シャットダウンするモードを≦のうちから指定する。

'both': 送受信双方をシャットダウンする。

'send': 送信をシャットダウンする。

'receive': 受信をシャットダウンする。

第 1 戻り値: 1 を返す。

(4) UDP(in socket)

＊ socket.udp()

未接続の UDP オブジェクトを返す。

第 1 戻り値: 新しい接続されていない UDP オブジェクトを返す。失敗した場合 nil を返す。

第 2 戻り値: 失敗した場合、エラー内容を文字列で返す。

＊ connected:close()

＊ unconnected:close()

UDP オブジェクトをクローズする。

オブジェクトによって使用されるソケットをクローズしオブジェクトに bind されたアドレスおよびポートは他のアプリケーションで使用可能となる。

＊ connected:getpeername

接続したリモート側の UDP オブジェクトの情報を返す。

第 1 戻り値: 成功した場合、リモートの IP アドレスを返す。

失敗した場合、nil を返す。

第 2 戻り値: 成功した場合、リモートのポート番号を返す。

＊ connected:getsockname()

＊ unconnected:getsockname()

オブジェクトに関連づけられたローカルアドレス情報を返す。

第 1 戻り値: 成功した場合、ローカルの IP アドレスを返す。

失敗した場合、nil を返す。

第 2 戻り値: 成功した場合、ローカルのポート番号を返す。

＊ connected:receive([size])

＊ unconnected:receive([size])

UDP オブジェクトからデータグラムを受信する。

connected オブジェクトの場合はピアからのみデータグラムを受信する。

unconnected オブジェクトの場合は任意のホストからデータグラムを受信できる。

第 1 引数: 受信するデータグラムの最大サイズを指定する。

第 1 戻り値: 成功した場合は受信データグラムを返す。失敗した場合は nil を返す。

第 2 戻り値: タイムアウトが発生した場合は"timeout"を返す。その他の場合はエラー内容を文字列で返す。

＊ unconnected:receivefrom([size])

unconnected UDP オブジェクトからデータグラムを受信する。

第 1 引数: 受信するデータグラムの最大サイズを指定する。

第 1 戻り値: 成功した場合は受信データグラムを返す。失敗した場合は nil を返す。

第 2 戻り値: 成功した場合は送信元 IP アドレスを返す。

タイムアウトが発生した場合は"timeout"を返す。その他の場合はエラー内容を文字列で返す。

第 3 戻り値: 成功した場合は送信元ポート番号を返す。

＊ connected:send(datagram)

connected オブジェクトのピアにデータグラムを送信する。

第 1 引数: 送信するデータを文字列で指定する。

第 1 戻り値: 成功した場合は 1 を、失敗した場合は nil を返す。

第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

＊ unconnected:sendt(datagram, ip, port)

unconnected オブジェクトから指定し IP アドレス、ポートにデータグラムを送信する。

第 1 引数: 送信するデータを文字列で指定する。

第 2 引数: 宛先 IP アドレスを指定する。(ホスト名は禁止)

第 3 引数: 宛先ポート番号を指定する。

第 1 戻り値: 成功した場合は 1 を、失敗した場合は nil を返す。

第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

＊ connected:setpeername('*')

＊ unconnected:setpeername(address, port)

UDP オブジェクトのピアを変更する。

第 1 引数: IP アドレスまたはホスト名を文字列で指定するか"*"(アスタリスク)を指定する。"*"を指定した場合、ピアとの関係付けは解除される(また、第 2 引数は無視される)。

第 2 引数: ピアのポート番号を指定。

第 1 戻り値: 成功した場合は 1 を、失敗した場合は nil を返す。

第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

＊ unconnected:setsockname(address, port)

UDP オブジェクトにローカルの IP アドレスおよびポート番号を割り当てる。

第 1 引数: IP アドレスまたはホスト名を文字列で指定。"*"を指定した場合、全てのインタフェースに割り当てる。

第 2 引数: ポート番号を 0-65535 の範囲内で指定。0 を指定した場合、自動的に 1024 - 65535 の範囲内で割り当てる。

第 1 戻り値: 成功した場合は 1 を、失敗した場合は nil を返す。

第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

＊ connected:setoption(option [, value])

＊ unconnected:setoption(option [, value])

UDP オブジェクトのオプションを設定する。

第 1 引数: オプション名を文字列で指定する。第 2 引数にオプションに対する値を指定する。

'dontroute': ルーティングをバイパスするかどうかを true/false で指定する。

'broadcast': ブロードキャストデータグラムを送信可能とするかを true/false で指定する。

＊ connected:settimeout(value)

＊ unconnected:settimeout(value)

UDP オブジェクトのタイムアウト値を変更する。

第 1 引数: タイムアウト時間を秒単位で指定する。

nil または負の値を指定した場合は処理が完了するまで無期限にブロックする。

第 1 戻り値: 成功した場合は 1 を、失敗した場合は nil を返す。

第 2 戻り値: 失敗した場合にエラー内容を文字列で返す。

(5) MIME

＊ mime.b64(C [, D])

Base64 形式でエンコードする。

第 1 引数: エンコードするデータを指定する。

第 2 引数: エンコードするデータを指定する。nil を指定した場合、第 1 引数は適宜パディングしてからエンコードを行なう。それ以外ではパディングしない。

第 1 戻り値: エンコードされたデータ。(第 1 引数と第 2 引数は連結される。)

第 2 戻り値: エンコードされなかったデータ。

＊ mime.dot(m [, B])

データに"CRLF.CRLF"を付加して SMTP 送信を可能なデータに変換する。

第 1 引数: 変換するデータを指定する。

第 2 引数: 第 2 戻り値と同じ。

メッセージ本文は暗黙 CRLF で始まると定義されているため、正しく変換するには 2 を指定する必要がある。

第 1 戻り値: 変換後のデータを返す。

第 2 戻り値: 変換前のデータ中に含まれる最後の"CRLF"の文字数を返す。

＊ mime.eol(C [, D, marker])

改行コードの変換を行なう。

第 1 引数: 直前前のチャンクの最後の文字の ASCII 値を指定する。

第 2 引数: 変換するデータを指定する。

第 3 引数: 新しい改行コードを指定する。デフォルトは"CRLF"。

第 1 戻り値: 変換後のデータを返す。

第 2 戻り値: チャンクの最後の文字の ASCII 値。

＊ mime.qp(C [, D, marker])

quoted-printable 形式でエンコードする。

第 1 引数: エンコードするデータを指定する。

第 2 引数: エンコードするデータを指定する。nil を指定した場合、第 1 引数は適宜パディングしてからエンコードを行なう。それ以外ではパディングしない。

第 1 戻り値: エンコードされたデータ。(第 1 引数を第 2 引数は連結される。)

第 2 戻り値: エンコードされなかったデータ。

＊ mime.qpwrp(n [, B, length])

quoted-printable 形式でエンコードし、1 行を指定された長さに分割する。

第 1 引数: 直前のチャンクにおいて本関数で変換されなかったデータの長さを指定する。

第 2 引数: 変換するデータを指定する。

第 3 引数: 1 行の長さを指定する。

第 1 戻り値: 変換されたデータを返す。
第 2 戻り値: 変換されなかったデータの長さを返す。

＊ mime.unb64(C [, D])
Base64 形式でデコードする。
第 1 引数: デコードするデータを指定する。
第 2 引数: デコードするデータを指定する。
第 1 戻り値: デコードされたデータ。(第 1 引数と第 2 引数は連結される。) 第 2 引数が nil の場合、空文字を返す。
第 2 戻り値: デコードされなかったデータ。

＊ mime.unqp(C [, D])
quoted-printable 形式でデコードする。
第 1 引数: デコードするデータを指定する。
第 2 引数: デコードするデータを指定する。
第 1 戻り値: デコードされたデータ。(第 1 引数と第 2 引数は連結される。)
第 2 戻り値: デコードされなかったデータ。

＊ mime.wrp(n [, B, length])
1 行が指定した文字数以下になるように改行する。
第 1 引数: 直前のデータに置いて改行されていないデータの長さを指定する。
第 2 引数: 変換するデータを指定する。
第 3 引数: 1 行の長さを指定する。デフォルトは 76。
第 1 戻り値: 変換されたデータを返す。
第 2 戻り値: 改行されなかったデータの長さを返す。

参考
Lua 公式サイト(英文)
http://http://www.lua.org

Lua 非公式リファレンスマニュアル(ver5.2 対応)(日本語)
http://milkpot.sakura.ne.jp/lua/lua52_manual_ja.html

LuaSocket ライブラリ(英文)
http://w3.impa.br/~diego/software/luasocket/home.html

```
-- Sample code 1 --
-- シリアル(SERIAL 1)から入力待ちを行い"exit"が入力されるまでの文字列をメールで送信する。
-- 本文の先頭にはプロダクト情報を付加する

-- サーバの設定
server =
{
```

```
        auth_type = "pop-before-smtp",
        account = {
                username = "any",
                password = "password",
        },
        smtp = {
                address = "smtp.example.com",
        },
        pop3 = {
                address = "po.example.com",
        }
}
-- メールの設定
mail = {
        to = "you@example.com",
        from = "me@example.com",
        subject = "mail from Lua Script on NXR",
}

-- プロダクト情報を CLI で取得
ret,product = nxr.cli("show product")
if ret then
        body = string.format("%s¥n", product)
else
        body = ""
end

-- シリアルポートを開く
tty = {
    baudrate = 115200,
    echo = true,
}
serial = serial.open(tty)

if serial then
        while true do
-- 読み込み
                ret,str = serial:read()
                if ret then
                        if (str == "exit") then
                                break;
                        end
```

```
                body = string.format("%s%s¥n", body, str)
            else
                break;
            end
        end
        serial:close()
-- メールを送信する
        nxr.mail_send(mail, server, body)
else
    -- 失敗時はシステムログに出力
    nxr.syslog("info", "can not open serial port")
end
-- END OF FILE

-- Sample code 2 --
-- UDP で受信してシスログに内容を出力、エコーする
-- 受信がない場合は 5 秒毎にタイムアウトメッセージをシスログに出力する
local socket = require("socket")
-- 受信アドレス／ポートの設定
host = host or "192.168.0.254"
port = port or 5001
if arg then
        host = arg[1] or host
        port = arg[2] or port
end
nxr.syslog("info", "Binding to host '" ..host.. "' and port " .. port .. "...")
-- UDP socket を作成
udp = assert(socket.udp())
-- 待ち受け IP アドレス／ポートを設定
assert(udp:setsockname(host, port))
-- タイムアウト時間を設定
assert(udp:settimeout(5))
ip, port = udp:getsockname()
assert(ip, port)
nxr.syslog("info", "Waiting packets on " .. ip .. ":" .. port .. "...")
while 1 do
        -- データを受信
        dgram, ip, port = udp:receivefrom()
        if dgram then
            -- シスロクに出力
            nxr.syslog("info", "Echoing '" .. dgram .. "' to " .. ip .. ":" .. port)
            -- エコー
```

```
        udp:sendto(dgram, ip, port)
    else
        -- エラーをシスログに出
        nxr.syslog("info", ip)
    end
end
-- END OF FILE
```