



μ Net3 ネットワークアプリケーションガイド

第 12 版 イー・フォース株式会社

はじめに

μNet3 Series は組み込み機器用の TCP/IP プロトコルスタックの機能を提供するソフトウェアです。また、FTP、HTTPd、DHCP などの各種プロトコルの機能を提供するソフトウェアも付属しています。本書はその付属プロトコルの API の関数を説明します。

制限事項

μNet3 Series に付属の各プロトコルの制限事項は本文の各プロトコルの制限事項の節をお読みください。

本説明書に記述した社名と製品名はそれぞれ各社の商標または登録商標です。
本書で記載されている内容は予告無く変更する場合があります。

改版履歴

第 2 版で訂正された項目

ページ／章	内容
	コンフィグレータ対応による変更
第 4～19 章	<ul style="list-style-type: none"> ・コンフィグレーション画面を追加（n.4 コンフィグレーション画面） ・上記追加により n.4 以降の項番をインクリメント

第 3 版で訂正された項目

ページ／章	内容
	FTP サーバーアプリを Compact/Standard 版、Pro 版と同じにしたための記述変更
第 1 章	<ul style="list-style-type: none"> ・FTP サーバーアプリに関する記述を修正 ・簡易ファイルシステムが Compact/Standard 版で収録する記述に変更

第 4 版で訂正された項目

ページ／章	内容
	HTTP サーバーアプリの API 追加による記述変更
第 9 章	<ul style="list-style-type: none"> ・HTTP サーバーアプリの停止 API に関する記述を追記 ・HTTP サーバー停止 API 追加に伴う HTTP サーバー制御情報構造体のメンバーの記述を変更

第 5 版で訂正された項目

ページ／章	内容
	DHCP サーバーアプリの拡張に伴う記述変更
第 6 章	<ul style="list-style-type: none"> ・リース期間追加に伴う記述追記 ・DHCP オプション機能追加に伴う記述追記 ・DHCP サーバーアプリの停止 API に関する記述を追記

第 6 版で訂正された項目

ページ／章	内容
	HTTP サーバーアプリの停止 API のパラメータ変更に伴う記述変更
第 9 章	<ul style="list-style-type: none"> ・HTTP サーバーアプリの停止 API の説明を変更 ・T_HTTP_SERVER 構造体のメンバーを変更

第 7 版で訂正された項目

ページ／章	内容
	FTP サーバーアプリの停止 API のパラメータ変更に伴う記述変更
第 13 章	<ul style="list-style-type: none"> ・FTP サーバーアプリの停止 API の説明を追加
第 20 章	POP3 クライアントの説明を追加

第 8 版で訂正された項目

ページ／章	内容
	DHCP サーバアプリの誤記修正
第 6 章	<ul style="list-style-type: none"> ・制限事項の DHCPRELEASE の表記を削除 ・制限事項のリースデータの「最大 10 件」の表記を削除 ・制限事項の IP Address Lease Time オプションは常に 0xFFFFFFFF(infinity)を削除

第 9 版で訂正された項目

ページ／章	内容
	HTTP サーバーアプリの API 廃止・追加による記述変更
第 9 章	<ul style="list-style-type: none"> ・ HttpSendImage() を廃止（廃止の記述追加） ・ HttpSendResponse() を追加
第 3 章	net_strncpy() 関数の追加

第 10 版で訂正された項目

ページ／章	内容
第 5 章	・ 任意オプション指定のための記述を追加（T_DHCP_UOPT 構造体）
第 8 章	・ HTTPS の設定をコンフィグレータ画面に追加
第 9 章	<ul style="list-style-type: none"> ・ HTTP サーバアプリの HTTP Keep-Alive 機能対応による記述変更 ・ http_server() の戻り値に関する説明を修正
第 12 章	<ul style="list-style-type: none"> ・ タイムアウトに関する記述変更 ・ 定義値 FTPC_MODE_ACTIVE, FTPC_MODE_PASSIVE の値を修正 ・ FTPS 対応、いくつか FTP コマンド追加による記述追加
第 13 章	FTP サーバーアプリのセキュリティ機能に関する記述追加

第 11 版で訂正された項目

ページ／章	内容
第 9 章	HTTP サーバアプリの機能追加による記述変更 <ul style="list-style-type: none"> ・ http_server_extcbk() コールバックの説明追加 ・ API に HttpSetContent(), HttpSetContentKpa(), HttpSendBuffer() を追加 ・ コンフィグレータ画面変更に関する記述変更

第 12 版で訂正された項目

ページ／章	内容
第 9 章	HTTP サーバアプリの Cookie 対応による記述変更 <ul style="list-style-type: none"> ・ API に CookieGetItem(), HttpSetContentCookie() を追加

目次

はじめに	2
制限事項	3
目次	6
1 プロトコル一覧	13
2 ファイル構成	14
2. 1 コンフィグレータ未使用時の注意点	14
3 STRING ライブラリ (CMP/STD/PRO)	15
3. 1 機能仕様	15
3. 2 制限事項	15
3. 3 OS/ネットワーク資源	15
3. 4 API	16
net_atoi (文字列を int 型の数値変換)	16
net_atol (文字列を long 型の数値変換)	16
net_itoa (int 型の数値を文字列変換)	17
net_strncasecmp (文字列の比較 英大文字・小文字同一視)	17
net_strcmp (文字列の比較)	18
net_strncmp (文字列の比較)	18
net_strcpy (文字列のコピー)	19
net_strlen (文字列長の取得)	19
net_strncat (文字列の連結)	20
net_strcat (文字列の連結)	20
net_strchr (文字の検索)	21
net_strstr (文字列の検索)	21
net_strcasestr (文字列の検索 英大文字・小文字同一視)	22
net_strncpy (文字列の n 文字コピー)	22
4 DHCP クライアント (CMP/STD/PRO)	23
4. 1 機能仕様	23
4. 2 制限事項	23
4. 3 OS/ネットワーク資源	23
4. 4 コンフィグレーション画面	24
4. 5 API	25
dhcp_client (開始)	25

4. 6	設定用マクロ/構造体/定数.....	27
	T_HOST_ADDR (ホストアドレス情報)	27
5	DHCP クライアント拡張版 (CMP/STD/PRO).....	28
5. 1	機能仕様	28
5. 2	制限事項	28
5. 3	OS/ネットワーク資源	28
5. 4	コンフィグレーション画面	29
5. 5	API.....	30
	dhcpcd_bind (リース情報の取得)	30
	dhcpcd_renew (リース情報の有効期間延長)	31
	dhcpcd_reboot (再起動)	32
	dhcpcd_release (リース情報の解放)	33
	dhcpcd_inform (オプションの取得)	34
5. 6	設定用マクロ/構造体/定数.....	35
	T_DHCP_CLIENT (DHCP クライアント情報)	35
	DHCP クライアントステータス定義値	36
	T_DHCP_UOPT (DHCP 取得オプション情報)	36
	T_DHCP_UOPT 要素判別・状態フラグ定義値.....	37
	T_DHCP_UOPT 設定用マクロ	38
6	DHCP サーバー (CMP/STD/PRO).....	39
6. 1	機能仕様	39
6. 2	制限事項	39
6. 3	OS/ネットワーク資源	40
6. 4	コンフィグレーション画面	41
6. 5	API.....	46
	dhcpcd_server (開始)	46
	dhcpcd_server_stop (停止)	47
6. 6	設定用マクロ/構造体/定数.....	48
7	DNS クライアント (CMP/STD/PRO)	49
7. 1	機能仕様	49
7. 2	制限事項	49
7. 3	OS/ネットワーク資源	49
7. 4	コンフィグレーション画面	50
7. 5	API.....	51
	dns_get_ipaddr (IP アドレスの取得)	51
	dns_get_name (ホスト名の取得)	52

7. 6	設定用マクロ/構造体/定数.....	52
8	HTTP クライアント (PRO).....	53
8. 1	機能仕様	53
8. 2	制限事項	53
8. 3	OS/ネットワーク資源	53
8. 4	コンフィグレーション画面	54
8. 5	API.....	55
	API 一覧	55
8. 6	設定用マクロ/構造体/定数.....	55
9	HTTP サーバー (CMP/STD/PRO)	56
9. 1	機能仕様	56
9. 2	制限事項	56
9. 3	OS/ネットワーク資源	56
9. 4	コンフィグレーション画面	56
9. 5	API.....	60
	http_server (開始)	60
	http_server_stop (停止)	61
	CgiGetParam (CGI 引数の解析)	62
	CookieGetItem (Cookie ヘッダの解析)	63
	HttpSendText (テキストコンテンツの送信)	64
	HttpSendFile (ファイルの添付送信)	65
	HttpSendImage (画像コンテンツの送信)	66
	HttpSendResponse (指定コンテンツの送信)	67
	http_server_extcbk (拡張動作コールバック関数)	68
	HttpSetContent (HTTP サーバ制御情報の送信バッファ追記)	71
	HttpSetContentKpa (送信バッファ に HTTP Keep-Alive ヘッダ付与)	71
	HttpSetContentCookie (送信バッファ に Set-Cookie ヘッダ付与)	72
	HttpSendBuffer (指定バッファ内容の送信)	73
9. 6	設定用マクロ/構造体/定数.....	74
	HTTP サーバーサンプル	74
	T_HTTP_SERVER (HTTP サーバ制御情報)	75
	T_HTTP_FILE (HTTP コンテンツ情報)	76
	コンテンツ拡張動作フラグの定義値	77
	T_HTTP_HEADER (HTTP ヘッダ情報)	77
	コンフィグレーション定義 (http_server_cfg.h)	78
10	SMTP クライアント (PRO).....	79

1 0. 1	機能仕様	79
1 0. 2	制限事項	79
1 0. 3	OS/ネットワーク資源.....	79
1 0. 4	コンフィグレーション画面	80
1 0. 5	API	81
	smtp_login (接続)	81
	smtp_quit (切断)	82
	smtp_snd_mail (メール送信)	83
	smtp_snd_cmd (コマンド発行)	84
1 0. 6	設定用マクロ/構造体/定数.....	85
	T_SMTP_CLIENT (SMTP クライアント制御情報)	85
	T_SMTP_MAIL (SMTP メール情報)	86
	T_SMTP_FILE (SMTP ファイル情報)	87
	コンフィグレーション定義 (smtp_client_cfg.h)	88
	SMTP 認証方式の定義値.....	88
	添付ファイル形式の定義値	88
1 1	TELNET サーバー (PRO)	89
1 1. 1	機能仕様	89
1 1. 2	制限事項	89
1 1. 3	OS/ネットワーク資源.....	90
1 1. 4	コンフィグレーション画面	92
	【シェル基本設定】	93
	【シェルプロンプト表示設定】	94
	【ログインユーザ】	95
	【コマンド設定】	96
	●デフォルトコマンド.....	97
	●ユーザーコマンドの形式について	97
	●ユーザーコマンド内処理でのシェルとのやりとり	97
1 1. 5	API	98
	telnet_server (サーバーの開始)	98
1 1. 6	設定用マクロ/構造体/定数.....	99
	T_TELNET_SERVER (TELNET サーバ制御情報)	99
	T_SHELL_BLK (SHELL データ転送構造体)	100
	コンフィグレーション定義 (telnet_server_cfg.h)	100
	Telnet サーバー通知用イベントフラグ定義値	100
1 2	FTP クライアント (PRO)	101

1 2. 1	機能仕様	101
1 2. 2	制限事項	101
1 2. 3	OS/ネットワーク資源.....	101
1 2. 4	コンフィグレーション画面	102
	【File system 画面】	103
1 2. 5	API	104
	ftp_login (接続)	104
	ftp_quit (切断)	105
	ftp_get_file (ファイル取得)	106
	ftp_put_file (ファイル設置)	106
	ftp_del_file (ファイル削除)	107
	ftp_ren_file (ファイルのリネーム)	107
	ftp_cmd_cd (作業ディレクトリの移動)	108
	ftp_cmd_pwd (作業ディレクトリの取得)	108
	ftp_cmd_mkd (新規ディレクトリの作成)	109
	ftp_cmd_rmd (既存ディレクトリの削除)	109
	ftp_cmd_noop (何もしない)	110
	ftp_cmd_list (ファイル名の一覧取得)	110
	ftp_cmd_list (ファイル名の一覧取得続き)	111
1 2. 6	設定用マクロ/構造体/定数.....	112
	T_FTP_CLIENT (FTP クライアント制御情報)	112
	T_FTP_LIST (LIST/NLST コマンド結果格納構造体)	113
	FTP 転送モードの定義値	114
	FTP 転送タイプの定義値	114
	コンフィグレーション定義 (ftp_client_cfg.h)	114
1 3	FTP サーバー (CMP/STD/PRO)	115
1 3. 1	機能仕様	115
1 3. 2	制限事項	115
1 3. 3	OS/ネットワーク資源.....	115
1 3. 4	コンフィグレーション画面	116
1 3. 5	API	118
	ftp_server (開始)	118
	ftp_server_stop (停止)	119
1 3. 6	設定用マクロ/構造体/定数.....	120
	T_FTP_SERVER (FTP サーバー制御情報)	120
	コンフィグレーション定義 (ftp_server_cfg.h)	121
	アカウント設定 (ftp_server_cfg.c)	122

1 4	TFTP クライアント (PRO).....	123
1 4. 1	機能仕様	123
1 4. 2	制限事項	123
1 4. 3	OS/ネットワーク資源.....	123
1 4. 4	コンフィグレーション画面	124
1 4. 5	API	125
	tftp_get_file (ファイル取得)	125
	tftp_put_file (ファイル設置)	126
1 4. 6	設定用マクロ/構造体/定数.....	127
	T_TFTP_CLIENT (TFTP クライアント制御情報)	127
	コンフィグレーション定義 (tftp_client_cfg.h)	127
1 5	TFTP サーバー (PRO).....	128
1 5. 1	機能仕様	128
1 5. 2	制限事項	128
1 5. 3	OS/ネットワーク資源.....	128
1 5. 4	コンフィグレーション画面	129
1 5. 5	API	130
	tftp_server (サーバーの開始)	130
1 5. 6	設定用マクロ/構造体/定数.....	131
	T_TFTP_SERVER (TFTP サーバー制御情報)	131
	コンフィグレーション定義 (tftp_server_cfg.h)	131
1 6	SNTP クライアント (CMP/STD/PRO)	132
1 6. 1	機能仕様	132
1 6. 2	制限事項	132
1 6. 3	OS/ネットワーク資源.....	132
1 6. 4	コンフィグレーション画面	133
1 6. 5	API	134
	sntp_client (NTP 時刻の取得)	134
1 6. 6	設定用マクロ/構造体/定数.....	135
	T_SNTP_CLIENT (SNTP クライアント制御情報)	135
1 7	SNTP サーバー (PRO).....	136
1 7. 1	機能仕様	136
1 7. 2	制限事項	136
1 7. 3	OS/ネットワーク資源.....	136
1 7. 4	コンフィグレーション画面	137

17.5	API	138
	API 一覧	138
17.6	設定用マクロ/構造体/定数	139
18	PING クライアント (CMP/STD/PRO)	140
18.1	機能仕様	140
18.2	制限事項	140
18.3	OS/ネットワーク資源	140
18.4	コンフィグレーション画面	141
18.5	API	142
	ping_client (送信)	142
18.6	設定用マクロ/構造体/定数	143
	T_PING_CLIENT (PING クライアント制御情報)	143
19	SNMP エージェント (PRO)	144
19.1	機能仕様	144
19.2	制限事項	144
19.3	OS/ネットワーク資源	144
19.4	コンフィグレーション画面	145
	【SNMP 基本設定】	146
	【SNMP 標準トラップの選択】	148
	【MIB2 グループの選択】	149
	【マネージャ設定】	150
	【コミュニティ設定】	151
	【標準トラップ送信先】	152
	【MIB システム設定】	154
19.5	API	155
	API 一覧	155
19.6	設定用マクロ/構造体/定数	155
20	POP3 クライアント (PRO)	156
20.1	機能仕様	156
20.2	制限事項	156
20.3	OS/ネットワーク資源	156
20.4	コンフィグレーション画面	157
20.5	API	158
	API 一覧	158
20.6	設定用マクロ/構造体/定数	159

1 プロトコル一覧

μNet3 Series には Compact/Standard 版と Professional 版があります。各版で収録している付属プロトコルは異なります。付属プロトコルの一覧は次の表ようになります。

表 1 付属プロトコル

章	プロトコル	Compact /Standard	Professional	備考
4、5	DHCP クライアント	○	○	
6	DHCP サーバー	×	○	*1 専用ドキュメントあり
7	DNS クライアント	○	○	
8	HTTP クライアント	×	○	*2 専用ドキュメントあり
9	HTTP サーバー	○	○	
0	SMTP クライアント	×	○	
1 1	Telnet サーバー	×	○	
1 1	FTP クライアント	×	○	
1 2	FTP サーバー	○	○	
1 4	TFTP クライアント	×	○	
1 5	TFTP サーバー	×	○	
1 6	SNTP クライアント	○	○	
1 7	SNTP サーバー	×	○	*3 専用ドキュメントあり
1 8	Ping クライアント	○	○	
1 9	SNMP エージェント	×	○	*4 専用ドキュメントあり
2 0	POP3 クライアント	×	○	*5 専用ドキュメントあり

*1 ... μ Net3/DHCPs ユーザーズガイド

*2 ... μ Net3/HTTPC ユーザーズガイド

*3 ... μ Net3/SNTPS ユーザーズガイド

*4 ... μ Net3-SNMP ユーザーズガイド

*5 ... μ Net3/POP3C ユーザーズガイド

また、次の表のように上記のプロトコルをサポートする補助システムも収録しています。これらのソフトウェアはサンプルです。

表 2 付属システム

補助システム	Compact /Standard	Professional	備考
簡易ファイルシステム	○	○	以下プロトコルで使用 ・ FTP サーバー・クライアント ・ TFTP サーバー・クライアント ・ TFTP クライアント ・ SMTP クライアント ・ POP3 クライアント
簡易シェル	×	○	Telnet サーバーで使用

2 ファイル構成

各プロトコルのファイル構成は次のようになります。プロトコルの API を使用する際には対象プロトコルファイルのヘッダをインクルードします。

表 3 付属プロトコル — ファイル構成

章	プロトコル	ソース/ヘッダ	コンフィグ
3	String ライブラリ	net_strlib.c/h	—
4、5	DHCP クライアント	dhcp_client.c/h	—
6	DHCP サーバー	dhcp_server.c/h	—
7	DNS クライアント	dns_client.c/h	—
8	HTTP クライアント	http_client.c/h	http_client_cfg.h
9	HTTP サーバー	http_server.c/h	http_server_cfg.h
0	SMTP クライアント	smtp_client.c/h	smtp_client_cfg.h
1 1	Telnet サーバー	telnet_server.c/h	telnet_server_cfg.h
1 2	FTP クライアント	ftp_client.c/h	ftp_client_cfg.h
1 3	FTP サーバー	ftp_server.c/h	ftp_server_cfg.c/h
1 4	TFTP クライアント	tftp_client.c/h	tftp_client_cfg.h
1 5	TFTP サーバー	tftp_server.c/h	tftp_server_cfg.h
1 6	SNTP クライアント	sntp_client.c/h	—
1 7	SNTP サーバー	sntp_server.c/h	sntp_server_cfg.h
1 8	Ping クライアント	ping_client.c/h	—
1 9	SNMP エージェント	専用ドキュメント参照	
2 0	POP3 クライアント	pop3_client.c/h	pop3_client_cfg.h

2. 1 コンフィグレータ未使用時の注意点

本書はμ C3/Compact、μ C3/Standard 付属のコンフィグレータを使用することを前提として、説明の記載が記述されています。コンフィグレータを使用せずに各プロトコルアプリケーションを使用するには予めそれぞれが使用するカーネルオブジェクト、ネットワークのソケットを作成・設定しておく必要があります。（「3.3 OS/ネットワーク資源」を参照）

コンフィグレータ未使用の場合、記述が必要なソースコードについては、サンプルを参照して頂きますようお願いいたします。

3 String ライブラリ (Cmp/Std/Pro)

3. 1 機能仕様

μ Net3 ではコンパイラに依存しないよう String 系の標準ライブラリのいくつかを提供します。ネットワークアプリケーションではこれらの提供関数を使用できます。

3. 2 制限事項

- 全ての String 系の標準ライブラリはサポートしていません。

3. 3 OS/ネットワーク資源

OS およびネットワークの資源は使用しません。

3. 4 API

net_atoi（文字列を int 型の数値変換）

【書式】

```
int net_atoi(const char *str);
```

【パラメータ】

const char	*str	対象文字列
------------	------	-------

【戻り値】

int	変換結果
-----	------

【解説】

str によって指示される文字列のはじめの部分を int 型整数に変換します。変換不能な場合は 0 を返します。

net_atol（文字列を long 型の数値変換）

【書式】

```
long net_atol(const char *str);
```

【パラメータ】

const char	*str	対象文字列
------------	------	-------

【戻り値】

Long	変換結果
------	------

【解説】

str によって指示される文字列のはじめの部分を long 型整数に変換します。変換不能な場合は 0 を返します。

net_itoa (int 型の数値を文字列変換)**【書式】**

```
char* net_itoa(int num, char *str, int base);
```

【パラメータ】

int	num	対象数値
char	*str	変換結果文字列
int	base	変換基数

【戻り値】

char *	変換結果文字列
--------	---------

【解説】

非標準 C ライブラリです。num によって指示される数値を base の基数で文字列に変換し、結果を str および戻り値に返します。

net_strncasecmp (文字列の比較 英大文字・小文字同一視)**【書式】**

```
int net_strncasecmp(const char *str1, const char *str2, SIZE len);
```

【パラメータ】

const char	*str1	比較文字列 1
const char	*str2	比較文字列 2
SIZE	len	比較文字数

【戻り値】

int	比較結果
-----	------

【解説】

文字コードで比較した結果、str1==str2 なら 0 を返します。str1>str2 なら正の値、str1<str2 なら負の値を返します。比較文字数分またはいずれかの文字列の終端に辿り着くまでが比較対象になります。

本関数では英字の大文字・小文字を同一視します。

net_strcmp (文字列の比較)

【書式】

```
int net_strcmp(const char *str1, const char *str2);
```

【パラメータ】

const char	*str1	比較文字列 1
const char	*str2	比較文字列 2

【戻り値】

int	比較結果
-----	------

【解説】

文字コードで比較した結果、`str1==str2` なら 0 を返します。`str1>str2` なら正の値、`str1<str2` なら負の値を返します。いずれかの文字列の終端に辿り着くまでが比較対象になります。

net_strncmp (文字列の比較)

【書式】

```
int net_strncmp(const char *str1, const char *str2, SIZE len);
```

【パラメータ】

const char	*str1	比較文字列 1
const char	*str2	比較文字列 2
SIZE	len	比較文字数

【戻り値】

int	比較結果
-----	------

【解説】

文字コードで比較した結果、`str1==str2` なら 0 を返します。`str1>str2` なら正の値、`str1<str2` なら負の値を返します。比較文字数分またはいずれかの文字列の終端に辿り着くまでが比較対象になります。

net_strcpy（文字列のコピー）

【書式】

```
char* net_strcpy(char *str1, const char *str2);
```

【パラメータ】

char	*str1	コピー先文字列のアドレス
const char	*str2	コピー元文字列のアドレス

【戻り値】

char *	コピー先文字列のアドレス
--------	--------------

【解説】

コピー元文字列 `str2` の終端（NULL）までを `str1` にコピーします。

net_strlen（文字列長の取得）

【書式】

```
SIZE net_strlen(const char *str);
```

【パラメータ】

char	*str	文字列
------	------	-----

【戻り値】

SIZE	文字列長
------	------

【解説】

`str` の終端（NULL）までの文字列を取得します。（NULL は含まない）

net_strncat（文字列の連結）

【書式】

```
char* net_strncat(char *str1, const char *str2, SIZE len);
```

【パラメータ】

char	*str1	連結先文字列のアドレス
const char	*str2	連結元文字列のアドレス
SIZE	len	連結文字数

【戻り値】

char *	連結先文字列のアドレス
--------	-------------

【解説】

連結先文字列 **str1** の終端（NULL）を開始位置として **str2** の連結文字数分または終端までをコピーします。

net_strcat（文字列の連結）

【書式】

```
char* net_strcat(char *str1, const char *str2);
```

【パラメータ】

char	*str1	連結先文字列のアドレス
const char	*str2	連結元文字列のアドレス

【戻り値】

char *	連結先文字列のアドレス
--------	-------------

【解説】

連結先文字列 **str1** の終端（NULL）を開始位置として **str2** の終端までをコピーします。

net_strchr（文字の検索）

【書式】

```
char* net_strchr(const char *str, int ch);
```

【パラメータ】

const char	*str	検索対象文字列
int	ch	検索文字

【戻り値】

char *	検索対象文字列の検索文字が現れるアドレス
--------	----------------------

【解説】

検索対象文字列 **str** の先頭から終端（NULL）に検索文字 **ch** が存在するか検索します。検索文字が存在する場合はその先頭アドレスを、存在しない場合は NULL を戻り値に返します。

net_strstr（文字列の検索）

【書式】

```
char* net_strstr(const char *str1, const char *str2);
```

【パラメータ】

const char	*str1	検索対象文字列
const char	*str2	検索文字列

【戻り値】

char *	検索対象文字列の検索文字列が現れるアドレス
--------	-----------------------

【解説】

検索対象文字列 **str1** の先頭から終端（NULL）に検索文字列 **str2** が存在するか検索します。検索文字列が存在する場合はその先頭アドレスを、存在しない場合は NULL を戻り値に返します。

net_strcasestr（文字列の検索 英大文字・小文字同一視）

【書式】

```
char* net_strcasestr(const char *str1, const char *str2);
```

【パラメータ】

const char	*str1	検索対象文字列
const char	*str2	検索文字列

【戻り値】

char *	検索対象文字列の検索文字列が現れるアドレス
--------	-----------------------

【解説】

検索対象文字列 **str1** の先頭から終端（NULL）に検索文字列 **str2** が存在するか検索します。検索文字列が存在する場合はその先頭アドレスを、存在しない場合は NULL を戻り値に返します。

本関数では英字の大文字・小文字を同一視します。

net_strncpy（文字列の n 文字コピー）

【書式】

```
char* net_strncpy(char *str1, const char *str2, SIZE len);
```

【パラメータ】

char	*str1	コピー先文字列のアドレス
const char	*str2	コピー元文字列のアドレス
SIZE	len	コピー文字数

【戻り値】

char *	コピー先文字列のアドレス
--------	--------------

【解説】

コピー元文字列 **str2** の終端（NULL）もしくは **len**(コピー文字数)まで **str1** にコピーします。

len が **str2** の長さより小さい場合、**len** 文字しかコピーされないため **str1** の **len+1** 文字目に終端文字は付加されません。

len が **str2** の長さより大きい場合、**str2** の終端以降の文字列は **len** の長さまで NULL で埋められます。

4 DHCP クライアント (Cmp/Std/Pro)

4. 1 機能仕様

DHCP (Dynamic Host Configuration Protocol) はインターネットや LAN などのネットワークに一時的に接続するホストの IP アドレスを動的に設定します。DHCP クライアントは DHCP サーバーからネットワークで利用できる IP アドレス情報を取得します。

4. 2 制限事項

- IPv4 ソケットのみサポート (IPv6 は未サポート)
- RENEW, RELEASE, DECLINE, INFORM 機能が使えません。この機能の使用が必要な場合は DHCP クライアント拡張版をご使用ください。

4. 3 OS/ネットワーク資源

DHCP クライアントでは 1 つの UDP ソケットが必要になります。ソケットのパラメータは以下の通りです。(通常コンフィグレータが自動生成します)

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_DHCP	0	v4	UDP	68	3 秒	3 秒	-	-	-	-

4. 4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→DHCP Client Ext の画面で DHCP クライアント通常版／拡張版のコンフィグレーションを行います。

拡張機能を使用する

チェックなしの場合、DHCP クライアント通常版がソース生成対象となります。チェックが有りの場合、DHCP クライアント拡張版がソース生成対象となります。

リトライ回数

DHCP クライアントのアドレス取得試行回数を指定します。(IP アドレスを自動的に取得する場合に有効)

4. 5 API

dhcp_client (開始)

【書式】

```
ER ercd = dhcp_client(T_HOST_ADDR *addr);
```

【パラメータ】

T_HOST_ADDR	*addr	ホストアドレス情報
-------------	-------	-----------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	*addr が NULL または socid が指定されていない。
E_OBJ	ソケットの状態が不正 (ソケットが未作成)
E_TMOUT	DHCP サーバーからの応答が遅延。または DHCP サーバーが存在しない。

【解説】

この API は DHCP サーバーから IP アドレス、サブネットマスク、ゲートウェイアドレスを取得してホストに割り当てます。使用しているネットワークの構成によっては E_TMOUT エラーが発生することがあります。その時は正常終了するまでリトライしてみることが推奨します。

またこの API は新規に DHCP セッションを開始します。つまり API を呼び出すと、必ず DISCOVER の送信を開始して、OFFER の受信、REQUEST の送信、ACK の受信を期待する動作になります。

DHCP サーバーから取得した IP アドレスの有効期限は 'lease(リース期間)' で指定されています。DHCP クライアントは有効期限が切れる前に新たにリースを行うには次のようにします。

使用例

```
void dhcp_tsk(VP_INT exinf)
{
    ER ercd;
    T_HOST_ADDR dhcp_addr = {0};
    UB status = DHCP_STS_INIT;

    dhcp_addr.socid = ID_SOC_DHCP;
    dhcp_addr.dev_num = ID_DEVNUM_ETHER;

    for (;;) {
        ercd = dhcp_client(&dhcp_addr);
        if (ercd == E_OK) {
            /* BOUND 期間 */
            dly_tsk(dhcp_addr.t1*1000);
            /* RENEWING 期間 */
        }
    }
}
```

```
        status = DHCP_STS_RENEWING;
        continue;
    }
    if (status == DHCP_STS_RENEWING) {
        /* REBINDING 期間 */
        dly_tsk((dhcp_addr.t2-dhcp_addr.t1)*1000);
        status = DHCP_STS_INIT;
        continue;
    }
    /* INIT 期間 */
    dly_tsk(1000);
}
}
```

尚 REQUEST メッセージでリース期間を延長する場合、DHCP クライアント拡張版をご使用下さい。

4. 6 設定用マクロ/構造体/定数

T_HOST_ADDR (ホストアドレス情報)

```
typedef struct t_host_addr {  
    UW    ipaddr;           /* IP アドレス */  
    UW    subnet;           /* サブネットマスク */  
    UW    gateway;          /* ゲートウェイ */  
    UW    dhcp;             /* DHCP サーバーアドレス */  
    UW    dns[2];           /* DNS アドレス */  
    UW    lease;            /* DHCP アドレスのリース期間 */  
    UW    t1;               /* DHCP アドレスのリニューアル期間 */  
    UW    t2;               /* DHCP アドレスのリバインド期間 */  
    UB    mac[6];           /* MAC アドレス */  
    UH    dev_num;          /* デバイス番号 */  
    UB    state;            /* DHCP クライアント状態 */  
    UH    socid;            /* UDP ソケット ID */  
} T_HOST_ADDR;
```

この構造体は DHCP クライアント API の引数として使用されます。デバイス番号と UDP ソケット ID はユーザアプリケーションでセットする必要があります。残りのパラメータは DHCP サーバーからの応答によりセットされます。

5 DHCP クライアント拡張版 (Cmp/Std/Pro)

5. 1 機能仕様

「4 DHCP クライアント (Cmp/Std/Pro)」の拡張機能版です。

5. 2 制限事項

- 通常版と異なり、RENEW, RELEASE, DECLINE, INFORM 機能が使えます。
- 通常版と異なり、任意のオプションの値をサーバから取得できます。

5. 3 OS/ネットワーク資源

DHCP クライアントでは1つのUDP ソケットが必要になります。ソケットのパラメータは以下の通りです。(通常コンフィグレータが自動生成します)

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_DHCP	0	v4	UDP	68	3 秒	3 秒	-	-	-	-

5. 4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→DHCP Client Ext の画面で DHCP クライアント通常版／拡張版のコンフィグレーションを行います。

拡張機能を使用する

チェックなしの場合、DHCP クライアント通常版がソース生成対象となります。チェックが有りの場合、DHCP クライアント拡張版がソース生成対象となります。

リトライ回数

DHCP クライアントのアドレス取得試行回数を指定します。(IP アドレスを自動的に取得する場合に有効)

5. 5 API

dhcp_bind（リース情報の取得）

【書式】

```
ER ercd = dhcp_bind(T_DHCP_CLIENT *dhcp);
```

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCP クライアント情報
---------------	-------	---------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	* dhcp が NULL または socid が指定されていない
E_OBJ	ソケットの状態が不正（ソケットが未作成）
E_SYS	割り当てられた IP アドレスが他のホストと競合。
E_TMOUT	DHCP サーバーからの応答が遅延。または DHCP サーバーが存在しない。

【解説】

この API は従来の dhcp_client() API と同等の機能を提供します。

取得した IP アドレスが他のホストと重複していないかを検証するには、引数の DHCP クライアント情報の IP 重複チェック有無に ARP_CHECK_ON を設定します。このとき IP アドレスの重複が検出された場合、DHCP サーバーに DHCP_DECLINE メッセージを送信し、API は E_SYS を返却します。

dhcp_renew（リース情報の有効期間延長）**【書式】**

```
ER ercd = dhcp_renew(T_DHCP_CLIENT *dhcp);
```

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCP クライアント情報
---------------	-------	---------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	* dhcp が NULL または socid が指定されていない
E_OBJ	不正な DHCP クライアント情報。または DHCP サーバーが要求を拒否した。
E_SYS	割り当てられた IP アドレスが他のホストと競合。
E_TMOUT	DHCP サーバーからの応答が遅延。または DHCP サーバーが存在しない。

【解説】

この API は DHCP サーバーから取得した IP アドレスの有効期間を延長します。引数には dhcp_bind() で取得した DHCP クライアント情報を指定します。

この API は有効期間内(t1)に呼び出して下さい。有効期間はタイマやタスク制御を使用してアプリケーションで計測します。

この RENEW 機能は REBIND 機能も含みます。両者の違いは REQUEST メッセージをユニキャスト送信するかブロードキャスト送信するかのみです。はじめに REQUEST メッセージを DHCP サーバーに送信後 ACK を受信できなければ、即座にブロードキャスト送信を行います。

延長した IP アドレスが他のホストと重複していないかを検証するには、引数の DHCP クライアント情報の IP 重複チェック有無に ARP_CHECK_ON を設定します。このとき IP アドレスの重複が検出された場合、DHCP サーバーに DHCP_DECLINE メッセージを送信し、API は E_SYS を返却します。

dhcp_reboot (再起動)**【書式】**

```
ER ercd = dhcp_reboot(T_DHCP_CLIENT *dhcp);
```

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCP クライアント情報
---------------	-------	---------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	* dhcp が NULL または socid が指定されていない または再利用するアドレスが無い。
E_OBJ	不正な DHCP クライアント情報。または DHCP サーバーが要求を拒否した。
E_SYS	割り当てられた IP アドレスが他のホストと競合。
E_TMOUT	DHCP サーバーからの応答が遅延。または DHCP サーバーが存在しない。

【解説】

この API はクライアントが以前使用していた IP リソースを再び使用する場合、その正当性を DHCP サーバーに確認するために使用します。たとえば休止中の LAN インタフェースが再び活性化した場合や LAN ケーブルを抜き差しした時など、その前後で同じネットワークに参加している保障が無い場合に、それまで使用していた IP リソースを DHCP サーバーに告知します。

引数には dhcp_bind() で取得した DHCP クライアント情報を指定します。

この API は REQUEST メッセージ送信後 ACK を受信できなかった場合、もしくは DHCPNAK を受信した場合はエラーとします。

告知した IP アドレスが他のホストと重複していないかを検証するには、引数の DHCP クライアント情報の IP 重複チェック有無に ARP_CHECK_ON を設定します。このとき IP アドレスの重複が検出された場合、DHCP サーバーに DHCP_DECLINE メッセージを送信し、API は E_SYS を返却します。

dhcp_release（リース情報の解放）

【書式】

```
ER ercd = dhcp_release(T_DHCP_CLIENT *dhcp);
```

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCP クライアント情報
---------------	-------	---------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	* dhcp が NULL または socid が指定されていない
E_OBJ	不正な DHCP クライアント情報。
E_TMOUT	DHCPRELEASE メッセージ送信タイムアウト。

【解説】

この API は DHCP サーバーから取得した IP アドレスを使用しなくなった場合に、DHCP サーバーにリソースの解放を通知します。

引数には dhcp_bind() で取得した DHCP 情報を指定します。

dhcp_inform（オプションの取得）

【書式】

```
ER ercd = dhcp_inform(T_DHCP_CLIENT *dhcp);
```

【パラメータ】

T_DHCP_CLIENT	*dhcp	DHCP クライアント情報
---------------	-------	---------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	* dhcp が NULL または socid が指定されていない または再利用するアドレスが無い。
E_OBJ	ホストに IP アドレスが設定されていない。
E_TMOUT	DHCP サーバーからの応答が遅延。または DHCP サーバーが存在しない。

【解説】

この API は DHCP サーバーから IP アドレス以外の情報を取得します。たとえば静的に IP アドレスを設定し、DNS サーバーのアドレスは DHCP サーバーから取得したい場合などに使用します。

引数の DHCP クライアント情報にはインタフェースのデバイス番号とソケット ID のみを設定します。

5. 6 設定用マクロ/構造体/定数

T_DHCP_CLIENT (DHCP クライアント情報)

```
typedef struct t_dhcp_client {
    T_DHCP_CTL    ctl        /* 制御情報 */
    UW            ipaddr;    /* IP アドレス */
    UW            subnet;    /* サブネットマスク */
    UW            gateway;   /* ゲートウェイ */
    UW            dhcp;      /* DHCP サーバーアドレス */
    UW            dns[2];    /* DNS アドレス */
    UW            lease;     /* DHCP アドレスのリース期間 */
    UW            t1;        /* DHCP アドレスのリニューアル期間 */
    UW            t2;        /* DHCP アドレスのリバインド期間 */
    UB            mac[6];    /* MAC アドレス */
    UH            dev_num;   /* デバイス番号 */
    UB            state     /* DHCP クライアント状態 */
    UH            socid;     /* UDP ソケット ID */
    UB            arpchk;    /* IP 重複チェック有無 */
    T_DHCP_UOPT   *uopt;    /* DHCP オプション取得パラメータ */
    UB            uopt_len;  /* DHCP オプション取得パラメータの数 */
} T_DHCP_CLIENT;
```

この構造体は DHCP クライアント API の引数として使用するもので、ホストアドレス情報構造体を拡張したものです。通常版同様、**デバイス番号**と**UDP ソケット ID**はユーザアプリケーションでセットする必要があります。設定する値は DHCP クライアントを参照して下さい。

IP 重複チェック有無に「ARP_CHECK_ON」を設定した場合、DHCP サーバーからリースされた IP に対し ACD 機能を使用した重複チェックを行います。

DHCP サーバーから任意のオプション情報(※)を取得したい場合、**DHCP オプション取得パラメータ**を指定します。値を指定すると、DHCP DISCOVER/REQUEST/INFORM メッセージの Parameter Request List に指定オプションコードが追加され、応答メッセージから該当オプションの取得を試みます。特に不要な場合は 0 を指定してください。

※取得可能な値形式は バイナリ(1,2,4 バイト)、文字列、アドレス のいずれかです。

この構造体は IP アドレス取得時に設定したものを IP アドレス更新時でも使用します。そのためアプリケーションで制御情報や DHCP クライアント状態を変更することはできません。

DHCP クライアントステータス定義値

```

#define DHCP_STS_INIT          0
#define DHCP_STS_INITREBOOT    1
#define DHCP_STS_REBOOTING     2
#define DHCP_STS_REQUESTING    3
#define DHCP_STS_BOUND         4
#define DHCP_STS_SELECTING     5
#define DHCP_STS_REBINDING     6
#define DHCP_STS_RENEWING      7

```

T_DHCP_CLIENT::state に入る定義値

T_DHCP_UOPT (DHCP 取得オプション情報)

```

typedef struct t_dhcp_uopt {
    UB      code;          /* DHCP オプションコード */
    UB      len;           /* オプション要素のサイズ／個 */
    UB      ary;           /* オプション要素の数 */
    UB      flag;          /* 要素判別・状態フラグ */
    VP      val;           /* 要素格納先ポインタ */
} T_DHCP_UOPT;

```

この構造体は DHCP サーバーから任意のオプション情報を取得するために使用します。**code** に取得したいオプションのコード番号、**len** に対象オプションの値 1 つあたりのサイズ、**ary** に対象オプションの要素数、**flag** に後述するフラグ値の設定、**val** に値を実際に格納する先の変数・配列をそれぞれ指定します。以下にオプションの設定例を示します。

	Time Offset(2)	Host Name(12)	Log Server(7)
code;	2	12	7
len;	4	1	4
ary;	1	バッファのサイズ	1 ... n (複数取得時)
flag;	DHCP_UOPT_BIN	DHCP_UOPT_STR	DHCP_UOPT_IPA
val;	INT 型変数のポインタ	バッファのポインタ	UW 型変数・配列のポインタ

T_DHCP_CLIENT 構造体の **DHCP オプション取得パラメータ**にこの構造体の値を指定した後、dhcp_bind(), dhcp_inform() のいずれかをコールします。応答メッセージに対象オプションが含まれている場合に **val** のポインタ先に取得値が入ります。値が取得できなかった場合は **flag** に DHCP_UOPT_STS_SET ビット値が立ちます。

次に T_DHCP_UOPT 構造体の実際の使用方法について記述する。

使用例

```

ER ercd;
T_DHCP_CLIENT dhcp;
T_DHCP_UOPT dhcp_uopt[3];
INT   uo_i4;
UINT  uo_u4;
VB    uo_buf[64];

/* T_DHCP_UOPT構造体の設定 */
net_memset(&dhcp_uopt[0], 0, sizeof(dhcp_uopt));

/* Time offset(2), Host Name(12), Log Server(7) */
SET_DHCP_UOPT_BIN4(dhcp_uopt[0], 2, &uo_i4);
SET_DHCP_UOPT_STR (dhcp_uopt[1], 12, uo_buf, sizeof(uo_buf));
SET_DHCP_UOPTS_IPA(dhcp_uopt[2], 7, uo_u4, sizeof(uo_u4));
/* {T_DHCP_CLIENT構造体の各種設定} */
dhcp.uopt = dhcp_uopt;
dhcp.uopt_len = 3;

/* {... T_DHCP_CLIENT構造体の設定 ...} */
ercd = dhcp_bind(&dhcp);
if (E_OK == ercd) {
    for (ercd = 0; ercd < sizeof(dhcp_uopt); ++ercd) {
        if (dhcp_uopt[ercd].flag & DHCP_UOPT_STS_SET) {
            /* オプション取得時処理を行う */
            /* dhcp_uopt[ercd].valのポインタに値が入る */
        }
    }
}

```

ソース解説

上記例では TimeOffset(2)、HostName(12)、LogServer(7)のオプションを取得するように設定を行っています。処理の簡単化のためマクロを使用しています。(後述) T_DHCP_UOPT 構造体の変数とは別に、対象オプションの値を格納する変数が必要になることに注意してください。情報取得用の DHCP クライアント API(ここでは dhcp_bind()) を実行後に取得したオプションの値が上記変数に格納されます。

T_DHCP_UOPT 要素判別・状態フラグ定義値

/* ユーザー設定値 */		
#define DHCP_UOPT_STR	0x80	オプションは文字列形式
#define DHCP_UOPT_IPA	0x40	オプションはアドレス形式
#define DHCP_UOPT_BIN	0x20	オプションはバイナリ形式
/* ユーザー参照値 */		
#define DHCP_UOPT_STS_SET	0x01	オプションの値が設定された

T_DHCP_UOPT::flag に入る定義値

T_DHCP_UOPT 設定用マクロ

/* オプションの取得値が単数の場合用 */

SET_DHCP_UOPT_BIN1(_uopt_,_code_,_pval_)	1 バイトのバイナリ用
SET_DHCP_UOPT_BIN2(_uopt_,_code_,_pval_)	2 バイトのバイナリ用
SET_DHCP_UOPT_BIN4(_uopt_,_code_,_pval_)	4 バイトのバイナリ用
SET_DHCP_UOPT_IPA(_uopt_,_code_,_pval_)	アドレス用
SET_DHCP_UOPT_STR(_uopt_,_code_,_pval_,_len_)	文字列用

/* オプションの取得値が複数の場合用 */

SET_DHCP_UOPTS_BIN1(_uopt_,_code_,_pval_,_len_)	1 バイトのバイナリ用
SET_DHCP_UOPTS_BIN2(_uopt_,_code_,_pval_,_len_)	2 バイトのバイナリ用
SET_DHCP_UOPTS_BIN4(_uopt_,_code_,_pval_,_len_)	4 バイトのバイナリ用
SET_DHCP_UOPTS_IPA(_uopt_,_code_,_pval_,_len_)	アドレス用

T_DHCP_UOPT 構造体の値を設定するための補助マクロ。マクロの引数にはそれぞれ
uopt : T_DHCP_UOPT 構造体の実体、_code_ : オプションコード、
pval : オプション値の格納先ポインタ、_len_ : 取得可能な要素数 (_pval_ の要素数)
 を指定します。

6 DHCP サーバー (Cmp/Std/Pro)

6. 1 機能仕様

DHCP (Dynamic Host Configuration Protocol) はインターネットや LAN などのネットワークに一時的に接続するホストの IP アドレスを動的に設定します。DHCP サーバーは DHCP クライアントからの要求に応じて利用できる IP アドレス情報を発行します。

6. 2 制限事項

- 対応するリースデータ
 - リースデータは以下の 3 つです。DHCPDISCOVER メッセージの Parameter request list オプションには対応していません。
 - IP アドレス
 - サブネットマスク
 - デフォルトゲートウェイ
- DHCP サーバーで保持するデータ (静的データ)
 - 以下のデータは DHCP サーバープログラムにおいて、あらかじめ固定で保持します。
 - DHCP サーバーの MAC アドレス
 - DHCP サーバーの IP アドレス
 - DHCP サーバーの待ち受けポート番号 (67 番)
- DHCP サーバーで保持するデータ (起動パラメータによる設定データ)
 - 以下のデータは DHCP サーバープリを起動する時にパラメータにより設定します。
 - リースデータ (IP アドレス) 先頭アドレス
 - リースデータ (IP アドレス) 最大件数
 - リースデータ (サブネットマスク、デフォルトゲートウェイ)
 - 登録 MAC アドレス

その他、機能制限には以下のものがあります。

- **DHCPDECLINE, DHCPINFORM** メッセージを受信した場合は何も応答せずに破棄します。
- **Bootp flags** フィールドの **Broadcast** ビットによらず、メッセージはすべてブロードキャスト送信します。
- リースデータ(IP アドレス)の定義方法は指定された先頭のアドレスから連続した番号をリースデータの最大件数分だけ割り振ります。
- 登録 **MAC** アドレスについては、何も登録しないことも可能です。
- 同一ネットワーク内に、リースデータと重複する **IP** アドレスが **STATIC** 設定されて使用されていても、リースデータとして使用します。

詳細は「μ Net3/DHCPs ユーザーズガイド」を参照してください。

6. 3 OS/ネットワーク資源

詳細は「μ Net3/DHCPs ユーザーズガイド」を参照してください。

6. 4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→DHCPv4 Server の画面で DHCP サーバーのコンフィグレーションを行います。

ネットワークインタフェース	リースデータ数	割付け先頭アドレス	サブネットマスク	デフォルトゲートウェイ
ID_NETIF_DEV1	10	192.168.2.100	255.255.255.0	192.168.2.1

有効にする

DHCP サーバーアプリがソース生成対象となります。

登録サーバー一覧

ネットワークインタフェース毎に割り当てられたサーバーの設定情報を表示します。

追加

登録サーバー一覧リストにサーバー情報を追加します。後述する設定画面から情報の設定を行います。

削除

登録サーバー一覧リストからサーバー情報を削除します。

DHCP サーバー情報設定画面を次に示します。

DHCPv4 Server

ネットワークインターフェース: ID_NETIF_DEV1

リース期間(秒): 300

リースデータ最大数: 10

リース情報

割付け先頭アドレス: 192 . 168 . 2 . 10

サブネットマスク: 255 . 255 . 255 . 0

デフォルトゲートウェイ: 192 . 168 . 2 . 1

IPアドレスを割付けるMACアドレス

XX-XX-XX-XX-XX-XX形式で登録してください。

12-34-56-78-9A-BC

通知用のコールバック関数

DHCPオプションを設定する

Domain Server

登録オプションの要素一覧

192.16.2.2

192.16.2.3

登録済オプション一覧

フォーマット	オプション値	データ長	変換形式	データ項目数
アドレス	6	8	-	2
バイナリ	19	1	リトルエンディアン	1
バイナリ	24	4	リトルエンディアン	1
バイナリ	26	2	リトルエンディアン	1
文字列	18	12	-	1

OK キャンセル

ネットワークインタフェース

DHCP サーバーアプリが対応する NIC を指定します。

リース期間 (秒)

DHCP サーバーアプリが DHCP クライアントへ IP をリースする期限を設定します。リース期限を超える前に DHCP クライアントがリース更新要求を行った場合、リース期限はリセット (延長) されます。

リースデータ最大数

IP アドレスのリース最大数を指定します。

リース情報

DHCP クライアントから要求された際にリースする情報を指定します。

IP アドレスを割付ける MAC アドレス

特定の MAC アドレスに IP アドレスをリースする場合、こちらを設定します。

通知用のコールバック関数

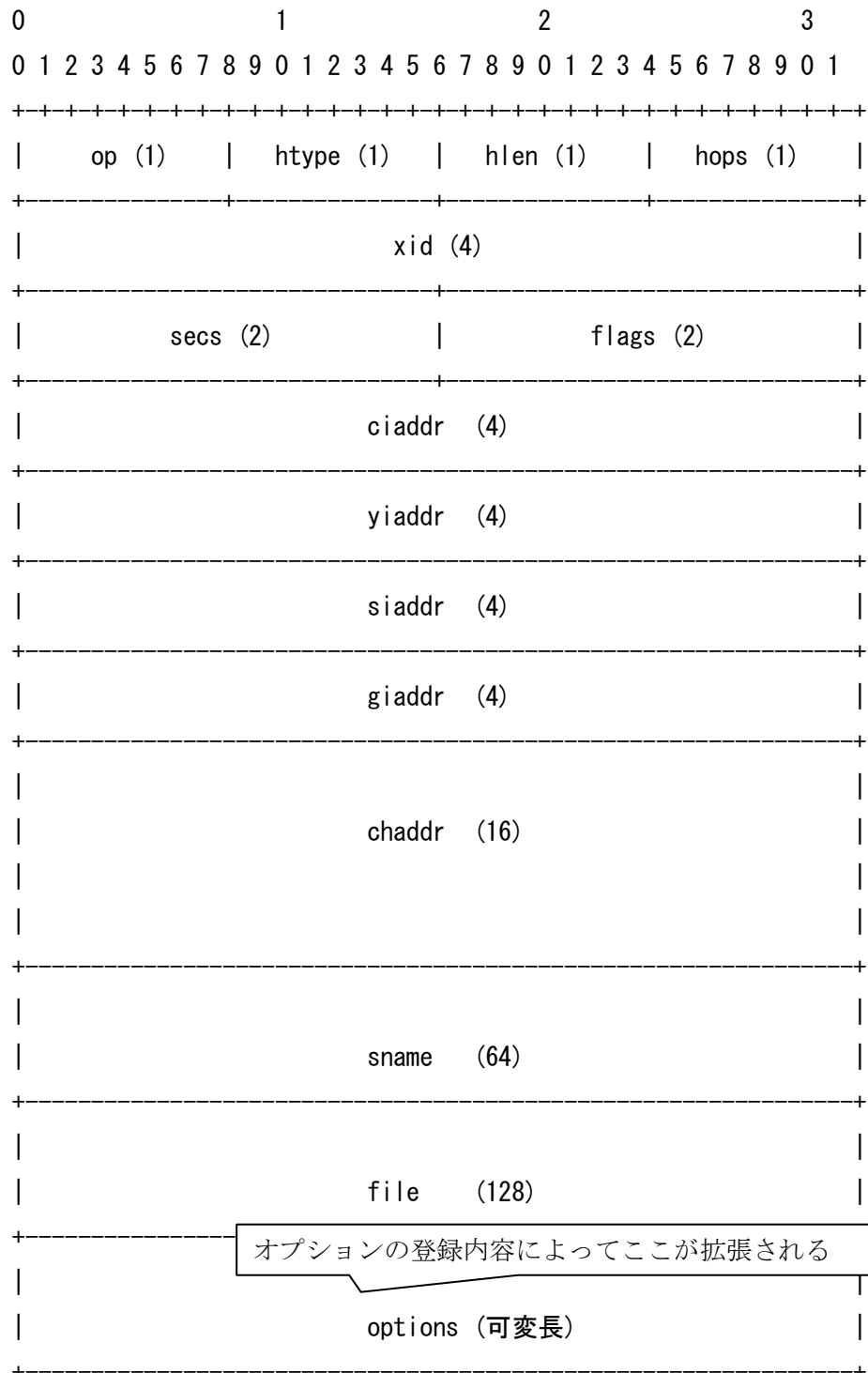
通知用のコールバック関数名を指定します。詳細は「μ Net3/DHCPs ユーザーズガイド」の「2.2 設定用マクロ/構造体 - T_DHCP_SERVER」を参照してください。

DHCP オプションを設定する

DHCP サーバーが DHCP クライアントへ送信する DHCP メッセージに任意のオプション情報を付与します。DHCP メッセージの電文フォーマットは下記参照。

なお、各オプションの形式（バイナリ、アドレス、文字列）や単位（オクテット）の詳細については RFC 2132（DHCP Options and BOOTP Vendor）を参照ください。

・ DHCP メッセージフォーマット



バイナリ形式、アドレス形式、文字列形式

バイナリ形式を選択した場合、10進入力、16進入力および「コード生成時にリトルエンディアンで出力」のコントロールが活性化されます。

アドレス形式および文字列形式を選択した場合は上記のコントロールは非活性となります。

10進入力、16進入力

10進入力を選択した場合、データ単位のコントロール (8byte, 4byte, 2byte, 1byte) が活性化されます。入力フォーマットは10進で入力してください。

16進入力を選択した場合、入力フォーマットは必ず2文字単位で入力してください。

入力例：000000FF (4byte 出力)、00FF (2byte 出力)、FF (1byte 出力)

コード生成時にリトルエンディアンで出力

この項目を有効にした場合、DHCP サーバーがオプションを設定する際にリトルエンディアン形式に変換してデータを設定します。

例：10進入力でデータ単位が4byte、値が1000のリトルエンディアン出力の場合。

DHCP メッセージの該当オプションのデータ部には「0xE8030000」が設定される。

リトルエンディアン出力が有効ではない場合は「0x000003E8」が設定される。

データ単位 (8byte, 4byte, 2byte, 1byte)

この項目は選択したデータ単位に応じた数値データをコンフィグレータが出力します。

※8byteのデータ単位は既存のDHCPオプションで該当する項目が無いため使用する必要はありません。

登録オプションの要素一覧

登録するオプションのデータ部を入力します。

バイナリ形式の場合は数値のみを入力してください。

アドレス形式の場合は「XXX.XXX.XXX.XXX」の形式で入力してください。

文字列形式の場合はフリーフォーマットで入力できます。

なお、データ要素は複数登録可能です。(オプション詳細はRFC2132 参照)

登録済オプション一覧

登録済オプションの一覧が表示されます。

追加／更新ボタン

入力したオプション内容を登録済オプション一覧へ登録します。また、登録済オプションの更新 (変更) なども行います。

削除ボタン

登録済オプション一覧の中から選択した登録オプションを削除します。

その他（オプションフォーマット）

Code : オプション番号

Len : オプション長

Option Value : オプションデータ

Code	Len	Option Value (Len)
1	4	1 2 3 4

6. 5 API

dhcp_server (開始)

【書式】

```
ER ercd = dhcp_server(T_DHCP_SERVER *dhcp);
```

【パラメータ】

T_DHCP_SERVER	*dhcp	DHCP サーバー制御情報
---------------	-------	---------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。 (*dhcp が NULL。starting_addr が指定されていない。 lease_num が指定されていない。)
-------	--

【解説】

この API は DHCP サーバーの初期化および DHCP サーバー受信タスク (dhcp_rcv_tsk) を起動し、DHCP クライアントからの要求受け付け処理を行います。この API の処理で起動する DHCP サーバー受信タスクのタスク ID は dhcp->tsk_id に設定されます。また、この API を実行したタスクのタスク ID は dhcp->server_tsk_id に設定されます。

なお、この API が実行されている時に外部タスクが DHCP サーバー停止 (dhcp_server_stop) API を実行した場合、この API は戻り値として E_RLWAI を返却して制御を戻します。

この API はブロッキング呼び出しになっていますので、専用タスクを用意しそこから呼び出すようにして下さい。

dhcp_server_stop (停止)**【書式】**

```
ER ercd = dhcp_server_stop(UW dev_num, UW retry);
```

【パラメータ】

UW	dev_num	停止する DHCP サーバーのデバイス番号
UW	retry	サーバタスク停止処理のリトライ回数

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。 (デバイス番号が不正。*dhcp が NULL。dhcp->tsk_id または dhcp->server_tsk_id に TSK_SELF が設定されている。)
E_TMOUT	サーバタスク停止処理のリトライオーバーが発生した。

【解説】

この API は起動している DHCP サーバーを停止します。この API が実行された場合、以下のタスクに対する停止処理を実行します。

- DHCP サーバ受信タスク (dhcp_rcv_tsk)
- DHCP サーバタスク (dhcp_server)

なお、パラメータチェックされる dhcp->tsk_id と dhcp->server_tsk_id は DHCP サーバー起動 API (dhcp_server) 実行時に自動でタスク ID が設定される為、基本的に意識する必要はありません。

この API は実行時に指定されたリトライ回数 (retry) のリトライオーバーが発生した場合、戻り値として E_TMOUT が返却されます。

【推奨】

この API で指定するリトライ回数 (retry) の値は「5」以上である事を推奨します。例えばリトライ回数に「0」を指定して実行した場合、この API が失敗する確率がより高くなります。

6. 6 設定用マクロ/構造体/定数

詳細は「μ Net3/DHCPs ユーザーズガイド」を参照してください。

7 DNS クライアント (Cmp/Std/Pro)

7. 1 機能仕様

DNS (Domain Name System) はインターネット上のホスト名を名前解決して IP アドレスと対応、もしくはその逆を行うプロトコルです。DNS クライアントはサーバーに対して名前解決の要求を発行し、その応答で対象のホスト名・IP アドレスを参照します。

7. 2 制限事項

- UDP ソケットのみサポート (TCP は未サポート)

7. 3 OS/ネットワーク資源

DNS クライアントでは1つの UDP ソケットが必要になります。ソケットのパラメータは以下の通りです。(通常コンフィグレータが自動生成します)

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_DNS	0	v4	UDP	0	5 秒	5 秒	-	-	-	-

7. 4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→DNS Client の画面で DNS クライアントのコンフィグレーションを行います。



有効にする

DNS クライアントアプリがソース生成対象となります。

7. 5 API

dns_get_ipaddr (IP アドレスの取得)

【書式】

```
ER ercd = dns_get_ipaddr(ID socid, UW dns_server, char *name, UW *ipaddr);
```

【パラメータ】

ID	socid	UDP ソケット ID
UW	dns_server	DNS サーバーの IP アドレス
char	*name	ホスト名
UW	*ipaddr	取得する IP アドレス

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。
E_TMOUT	DNS サーバーから応答なし
E_NOMEM	メモリエラー
E_OBJ	ホスト名から IP アドレス解決できない

使用例

```
UW ip;
ER ercd;
UW dns_server = ip_aton("192.168.11.1");

dns_get_ipaddr(ID_SOC_DNS, dns_server, "www.eforce.co.jp", &ip);
```

dns_get_name (ホスト名の取得)**【書式】**

```
ER ercd = dns_get_name(ID socid, UW dns_server, char *name, UW *ipaddr);
```

【パラメータ】

ID	socid	UDP ソケット ID
UW	dns_server	DNS サーバーの IP アドレス
char	*name	取得するホスト名
UW	*ipaddr	IP アドレス

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。
E_TMOUT	DNS サーバーから応答なし
E_NOMEM	メモリエラー
E_OBJ	IP アドレスからホスト名を取得できない

使用例

```
UW ip = ip_aton("192.168.11.30");
ER ercd;
char host_name[256];
UW dns_server = ip_aton("192.168.11.1");

dns_get_name(ID_SOC_DNS, dns_server, host_name, &ip);
```

7. 6 設定用マクロ/構造体/定数

設定用のマクロと構造体はありません。

8 HTTP クライアント (Pro)

8. 1 機能仕様

HTTP クライアントはアプリケーションから指定された URL から、HTTP サーバの IP アドレスおよびポート番号を解決し、GET リクエストか、POST リクエストを送信します。このアプリの主な機能は以下となります。

- GET/POST リクエスト送信
- コンテンツ受信
- セッション解放
- HTTP の認証に対応 (Basic 認証、Digest 認証)
- HTTPS に対応 (別途 μ Net3/SSL が必要となります)

8. 2 制限事項

- IPv4 TCP ソケットのみサポート
- 設定可能な HTTP ヘッダ
 - Host
 - User-Agent
 - Content-Length
 - Content-Type
 - Authorization
- 取得可能な HTTP ヘッダ
 - Content-Type
 - Content-Length
 - Transfer-Encoding
 - WWW-Authenticate
- Digest 認証の制限
 - 認証アルゴリズム (algorithm) は MD5 のみサポート
 - 保護品質 (qop) は auth のみ、または qop 指定なしをサポート

詳細は「μ Net3/HTTPc ユーザーズガイド」を参照してください。

8. 3 OS/ネットワーク資源

詳細は「μ Net3/HTTPc ユーザーズガイド」を参照してください。

8. 4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→HTTP Client の画面で HTTP クライアントのコンフィグレーションを行います。

DHCP Client Ext DNS Client HTTP Server HTTP Client FTP Server FTP Client SNTP Client Ping File system

☒ 有効にする

☒ HTTPベーシック認証機能を使用する

☒ HTTPダイジェスト認証機能を使用する

☒ HTTP over SSL/TLS を有効にする (uNet3/SSLが必要)

有効にする

HTTP クライアントアプリがソース生成対象となります。

HTTP ベーシック認証機能を使用する

HTTP クライアントアプリがベーシック認証機能をサポートします。

HTTP ダイジェスト認証機能を使用する

HTTP クライアントアプリがダイジェスト認証機能をサポートします。

HTTP over SSL/TLS を有効にする (μNet3/SSL が必要)

HTTPS クライアントとして動作可能とするかを選択できます。

8. 5 API

API 一覧

詳細は「μ Net3/HTTPc ユーザーズガイド」を参照してください。

No	API	概要説明	備考
	書式		
1	http_get_ipaddr	URL から IP アドレス取得	
	ER ercd = http_get_ipaddr(const char *urlstr, T_NODE *svr, ID dns_id, UW dns_ip);		
2	http_cmd_get	GET リクエスト送信	
	ER http_cmd_get(T_HTTP_CLIENT *http, const char *url);		
3	http_cmd_post	POST リクエスト送信	
	ER http_cmd_post(T_HTTP_CLIENT *http, const char *url, T_HTTP_POSTDATA *data);		
4	http_rcv_res	コンテンツ受信	
	ER http_rcv_res(T_HTTP_CLIENT *http, UB *buf, UW len);		
5	http_cls	HTTP クライアント終了	
	ER http_cls(T_HTTP_CLIENT *http);		

8. 6 設定用マクロ/構造体/定数

別途「μ Net3/HTTPc ユーザーズガイド」を参照してください。

9 HTTP サーバー (Cmp/Std/Pro)

9. 1 機能仕様

HTTP サーバーは、HTTP クライアント（インターネットブラウザ）に静的または動的なコンテンツを送信します。

9. 2 制限事項

- ファイルシステム未対応（コンテンツメモリ割り当て）
- HTTP メソッドは”GET”、”HEAD”、”POST”をサポート
- HTTP バージョンは”HTTP/1.1”、”HTTP/1.0”をサポート

9. 3 OS/ネットワーク資源

HTTP サーバーでは 1 つの TCP ソケットが必要になります。ソケットのパラメータは以下の通りです。（通常コンフィグレータが自動生成します）

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_HTTP	0	v4	TCP	80	25 秒	25 秒	25 秒	25 秒	1024	1024

※HTTP Keep-Alive 機能のタイムアウト時間は上記の受信タイムアウトを使用します。

9. 4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→HTTP Server の画面で HTTP サーバーのコンフィグレーションを行います。

☒ 有効にする
 セッション最大数: 1

コンテンツ **機能設定**

URL	Resource	Type	拡張フラグ値
/	#index.html	text/html	0
/sample.cgi	sample_fnc	cgi	0
/auth/	#auth.html	text/html	1

追加

削除

有効にする

HTTP サーバーアプリがソース生成対象となります。

セッション最大数

HTTP サーバーへ接続するセッションの上限を指定します。

【コンテンツ】タブ画面

コンテンツ一覧

登録されているコンテンツが表示されます。登録可能コンテンツ数は 50 個です。

追加

コンテンツ一覧リストに新しいコンテンツを追加します。後述する設定画面から情報の設定を行います。

削除

コンテンツ一覧リストから選択されているコンテンツを削除します。

DHCP Client Ext DNS Client **HTTP Server** FTP Server SNTp Client Ping File system

☒ 有効にする

セッション最大数 1

コンテンツ 機能設定

☒ HTTP Keep-Alive機能を有効にする

接続持続要求数 100

☐ ユーザ拡張動作機能を使用する

拡張動作コールバック関数

【機器設定】タブ画面

HTTP Keep-Alive 機能を有効にする

HTTP サーバーアプリの通信で HTTP Keep-Alive 機能が有効になります。

接続持続要求数

HTTP Keep-Alive 機能の 1 回の接続で有効なクライアントからの要求数を指定します。
この設定数のクライアント要求に達するまで HTTP サーバーアプリはクライアントとの接続を維持します。(タイムアウト発生時には接続を切断します)
設定数に達すると、HTTP サーバーアプリはクライアントとの接続を切断します。

ユーザ拡張動作機能を使用する

HTTP サーバーアプリ処理中に任意のユーザ処理を行える機能を有効にします。
詳細は後続の `http_server_extcbk()` の説明を参照願います。

拡張動作コールバック関数

ユーザ拡張動作機能が有効時にユーザ処理を行うコールバック関数名を指定します。

コンテンツ情報設定画面を次に示します。

コンテンツ

Content-Type: text/html

URL: URLの入力は'/'から開始してください。
/auth/

Resource: HTMLファイル名を入力してください。
#auth.html

拡張動作フラグ

拡張フラグ	値
<input checked="" type="checkbox"/> HTTPD_EXT_AUTH	0x01
<input type="checkbox"/> HTTPD_EXT_UHDR	0x02

OK キャンセル

Content-Type

登録するコンテンツタイプ（インターネットメディアタイプ）を指定します。コンテンツタイプのいくつかは予めリストに用意されています。入力指定も可能です。

URL

コンテンツの URL を指定します。URL は"/"から開始してください。

入力例)

text/html の場合、/index.html

cgi の場合、/function.cgi (CGI のスクリプト名)

Resource

コンテンツのリソースを指定します。

➤ コンテンツタイプが CGI 以外の場合

指定した Content-Type に従い実際のファイルを指定します。[...]ボタンを押下すると"ファイル選択画面"が表示され、そこからファイルも指定できます。

➤ コンテンツタイプが CGI の場合

CGI スクリプトを実行する関数名を指定します。指定した関数名は main.c に出力されます。関数名に次の文字を含めることはできません。

禁則文字: " ` { } * @ ; + : * , . # \$ % & ' ¥ " ! ? ~ ^ = | / ¥ < > () "

拡張動作フラグ

ユーザ拡張動作機能が有効時にコンテンツに対してフラグ値を付与します。付与されたコンテンツにアクセスした際に拡張動作コールバック関数が呼ばれます。詳細は後続の http_server_extcbk() の説明を参照願います。

9. 5 API

http_server (開始)

【書式】

```
ER ercd = http_server(T_HTTP_SERVER *http);
```

【パラメータ】

T_HTTP_SERV	* http	HTTP サーバー制御情報
-------------	--------	---------------

【戻り値】

ER	ercd	次のエラーコード
----	------	----------

【エラーコード】

E_RLWAI	http_server_stop()がコールされ停止した。
E_PAR	不正なパラメータが指定された。 (*http が NULL。SocketID が指定されていない。)
その他	ソケットの con_soc のエラーコード

【解説】

この API は HTTP セッションを初期化し、HTTP クライアントからの要求を受け付け処理します。クライアントから要求された URL がコンテンツテーブル(T_HTTP_FILE)に存在する時は、そのコンテンツをクライアントに送信し、存在しない時は HTTP エラーメッセージ”404 File not found”を送信します。コンテンツが動的の場合(cbk が NULL でない)は CGI 用として、そのコールバック関数を呼び出します。

この API はブロッキング呼出しになっていますので、専用タスクを用意しそこから呼び出すようにしてください。

引数の制御情報の受信バッファが NULL の場合、HTTP サーバーはネットワークバッファを使用します。

引数の制御情報の送信バッファが NULL の場合、HTTP サーバーはネットワークバッファを使用します。

http_server_stop (停止)**【書式】**

```
ER ercd = http_server_stop( UW retry )
```

【パラメータ】

UW	retry	サーバータスク停止処理のリトライ回数
----	-------	--------------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_TMOUT	サーバータスク停止処理のリトライオーバーが発生した。
---------	----------------------------

【解説】

この API は起動している HTTP サーバーを停止します。この API が実行された場合、HTTP サーバーの全タスクの停止処理を実行します。

HTTP サーバーのタスク情報は HTTP サーバー起動 API (http_server) 実行時に自動で内部メモリに保持されます。

なお、この API は実行時に指定されたリトライ回数 (retry) のリトライオーバーが発生した場合、戻り値として E_TMOUT が返却されます。

【推奨】

この API で指定するリトライ回数 (retry) の値は「5」以上である事を推奨します。例えばリトライ回数に「0」を指定して実行した場合、この API が失敗する確率がより高くなります。

CgiGetParam (CGI 引数の解析)

【書式】

```
void CgiGetParam(char *msg, int clen, char *cgi_var[], char *cgi_val[], int *cgi_cnt);
```

【パラメータ】

char	*msg	CGI 引数
int	clen	CGI 引数サイズ
char	*cgi_var[]	解析した CGI 引数
char	*cgi_val[]	解析した CGI 引数の値
int	*cgi_cnt	解析した CGI 引数の個数

【戻り値】

なし

【エラーコード】

なし

【解説】

この API は ‘フィールドー値’ の組で構成されるクエリ文字列を解析します。例えばクエリ文字列が “name1=value1&name2=value2” と与えられ場合の解析結果は以下のようになります。

```
cgi_cnt = 2;
```

```
cgi_var[0] = “name1”;
```

```
cgi_var[1] = “name2”;
```

```
cgi_val[0] = “value1”;
```

```
cgi_val[1] = “value2”;
```

CookieGetItem (Cookie ヘッダの解析)

【書式】

UB CookieGetItem(char **cookie, char **name, char **value)

【パラメータ】

char	**cookie	Cookie ヘッダの値
char	**name	取得した Cookie 名称
char	**value	取得した Cookie 名称の値

【戻り値】

UB Cookie ペアが取得出来た場合 1、それ以外は 0

【エラーコード】

なし

【解説】

この API は Cookie ヘッダの値 ‘name1=value1; name2=value2; ..’ 形式から Cookie 名称と値を取得します。Cookie ペアが取得できると名称は name、値は value に格納され戻り値に 1 が返ります。それ以外の場合、戻り値に 0 が返ります。

この API をコールすると、**cookie の指すポインタ位置が次の Cookie ペアの位置になり、且つバッファ内容が変更されます。**cookie の指すバッファ内容が必要な場合はこの API をコール前に予め別のバッファなどに退避させてください。

使用例

```
void Http_Callback(T_HTTP_SERVER *http)
{
    VB *cname, *cval;
    VB buf[128];

    net_strcpy(buf, "<html><body>");

    /* Cookieヘッダの内容をHTMLに出力*/
    net_strcat(buf, "%r%r<pre>%r%r");
    while (CookieGetItem(&http->hdr.cookie, &cname, &cval)) {
        net_strcat(buf, cname); /* Cookie名称*/
        net_strcat(buf, "=");
        net_strcat(buf, cval); /* Cookie名称の値*/
        net_strcat(buf, "%r%r");
    }
    net_strcat(buf, "%r%r</pre>%r%r");
    net_strcat(buf, "</body></html>");
    HttpSendText(http, buf, net_strlen(buf));
}
```

HttpSendText（テキストコンテンツの送信）

【書式】

```
ER ercd = HttpSendText(T_HTTP_SERVER *http, char *str, int len);
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
char	*str	送信する文字列
int	len	送信する文字列長

【戻り値】

ER	ercd	正常終了 (E_OK) のみ
----	------	----------------

【エラーコード】

なし

【解説】

この API は動的コンテンツを送信します。この API は CGI 用コールバック関数からのみ呼び出してください。

使用例

```
char page1[] = "<html><body> Welcome to this web server </body></html>";

void Http_Callback(T_HTTP_SERVER *http)
{
    HttpSendText(http, page1, sizeof(page1));
}
```


HttpSendFile（ファイルの添付送信）

【書式】

```
ER ercd = HttpSendFile(T_HTTP_SERVER*http, char*str,
                      int len, char*name, char *type);
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
char	*str	送信するファイルの実体
int	len	送信するファイルのバイトサイズ
char	*name	送信するファイル名
char	*type	HTTP ヘッダーの Content-Type の値(文字列)

【戻り値】

ER	ercd	正常終了 (E_OK) のみ
----	------	----------------

【エラーコード】

なし

【解説】

この API は動的コンテンツを送信します。この API は CGI 用コールバック関数からのみ呼び出してください。

この API ではファイルは添付送信(Content-Disposition: attachment)で送信されます。

使用例

```
char file[1024];

void Http_Callback(T_HTTP_SERVER *http)
{
    int len;
    /*          :                               /
    /   ファイルの内容をfileに、サイズをlenに出力する処理 /
    /          :                               */
    HttpSendFile(http, file, len, "FILE NAME", "text/plain");
}
```

HttpSendImage（画像コンテンツの送信）

【書式】

```
ER ercd = HttpSendImage(T_HTTP_SERVER *http, char *str, int len);
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
char	*str	送信する画像バイト列
int	len	送信する画像バイト列長

【戻り値】

ER	ercd	正常終了 (E_OK) のみ
----	------	----------------

【エラーコード】

なし

【解説】

この API は動的コンテンツを送信します。この API は CGI 用コールバック関数からのみ呼び出してください。

※ 2015/3/26 更新版で当 API は廃止されました。代替関数 [HttpSendResponse](#) をお使いください。

HttpSendResponse（指定コンテンツの送信）

【書式】

```
ER ercd = HttpSendResponse(T_HTTP_SERVER *http, char *str, int len, char *type)
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
char	*str	送信するコンテンツバイト列
int	len	送信するコンテンツバイト列長
char	*type	コンテンツタイプ名称

【戻り値】

ER	ercd	正常終了 (E_OK) のみ
----	------	----------------

【エラーコード】

なし

【解説】

この API は動的コンテンツを送信します。この API は CGI 用コールバック関数からのみ呼び出してください。

http_server_extcbk (拡張動作コールバック関数)

【書式】

```
ER ercd = http_server_extcbk(T_HTTP_SERVER *http, const T_HTTP_FILE *fp,
UB evt)
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
T_HTTP_FILE	*fp	アクセスされたコンテンツ情報
UB	Evt	発生イベントの判定フラグ

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【解説】

このコールバック関数 (拡張動作コールバック) は、HTTP サーバタスクから呼び出されます。拡張動作フラグが設定されているコンテンツに対して、Web ブラウザからアクセスを行うと拡張動作コールバック関数が呼ばれます。アクセスされた該当コンテンツ情報は fp に入ります。また、判定フラグ evt には以下の値が入ります。

HTTPD_EXT_AUTH

(HTTP 認証の使用)

該当コンテンツに HTTP 認証が適用されます。コールバック関数はクライアント要求にある Authorization ヘッダの値が http->hdr.auth に入った状態、もしくは http->hdr.auth が NULL の状態で呼ばれます。

http->hdr.auth に値が入っている場合は、関数内で値のユーザ名・パスワード (or ダイジェスト値) が正しいかのチェックを行います。結果、成功すれば E_OK を、失敗では E_OK 以外を返すように設計してください。

http->hdr. が NULL の場合は、関数内で認証失敗の応答に表記する WWW-Authenticate ヘッダ情報の作成 (realm や nonce の指定など) を行ってください。

HTTPD_EXT_UHDR

(カスタムヘッダの指定)

Web ブラウザにコンテンツ情報を応答する前にコールバック関数が呼ばれ、関数内で任意の HTTP ヘッダを追記できます。CGI には適用されません。

(このフラグを CGI に適用してもカスタムヘッダは付与されません)

コールバック関数内で HttpSetContent() 以外の HTTP サーバ API、待ち要因がある API の呼び出しは禁止です。コールバック関数は HTTP サーバタスクのスタックを使用します。必要に応じて HTTP サーバタスクのスタックサイズを増やしてください。サーバタスクが複数ある場合はコールバック関数内の処理をリエントラントな作りにする必要があります。

以下に使用例、および解説を記します。

使用例

【net_cfg.c - コンテンツ登録部】

```

/*****
HTTP Content List
*****/
T_HTTP_FILE const content_list[] =
{
    {"/", "text/html", index_html1, sizeof(index_html1), NULL, 0},
    {"/manage/", "text/html", manage_html2, sizeof(manage_html2), NULL,
        HTTPD_EXT_AUTH | HTTPD_EXT_UHDR},
    {"/sample.cgi", "", NULL, 0, sample_fnc, HTTPD_EXT_AUTH},
    {"", NULL, NULL, 0, NULL}
};

```

【net_cfg.c - net_setup() HTTPサーバタスク起動部】

{前略}

```

gHTTP_EXT_CBK = httpd_evt_callback;    /* 拡張動作コールバック登録 */
gHTTP_FILE = (T_HTTP_FILE*)content_list;

/* Start HTTPd Task */
sta_tsk(ID_HTTPD_TSK1, 0);

{後略}

```

【ユーザソース】

```

ER httpd_evt_callback(T_HTTP_SERVER *http, const T_HTTP_FILE *fp, UB evt)
{
    T_HTTP_HEADER *hdr = &http->hdr;
    ER ercd;
    ercd = E_OK;

    switch (evt) {
    case HTTPD_EXT_AUTH:    /* HTTP認証(ベーシック) */
        if (hdr->auth) {    /* 認証情報あり */
            {省略}    /* 値はBase64のため、デコード処理を行う */
            {省略}    /* デコード後、ユーザ名・パスワードを取得する */
            ercd = (OK == 妥当性チェック) ? E_OK : E_OBJ;
        }
        else {              /* 認証失敗 */
            /* 応答メッセージ用のWWW-Authenticateヘッダ作成 */
            HttpSetContent(http, "WWW-Authenticate: Basic ");
            HttpSetContent(http, "realm=¥\"Secret Zone¥\"¥r¥n");
        }
        break;

    case HTTPD_EXT_UHDR:    /* カスタムヘッダ指定 */
        HttpSetContent(http, "X-Frame-Options: DENY¥r¥n");
        break;
    }
    return ercd;
}

```

【解説】

拡張動作コールバック関数は次の条件を満たした場合に呼ばれます。

- ・ `http_server_set_extcbk()` で処理コールバック関数を登録している。
- ・ コンテンツリストの `ext` メンバに値を設定している。
- ・ 上記該当コンテンツに Web ブラウザからアクセスする。

例では、<http://xxx.xxx.xxx.xxx/> にアクセスした場合にはコールバック関数は呼ばれず、それ以外の URL にアクセスした場合はコールバック関数が呼ばれます。

次にコールバック関数内の処理を見ていきます。はじめに、コールバック関数ではフラグ `evt` の値を見てどの拡張動作（認証処理、カスタムヘッダ処理）なのかを判定します。

フラグ `evt` が HTTP 認証処理 (`HTTPD_EXT_AUTH`) の場合、`hdr->auth` の値が `NULL` 以外で認証チェックの処理を行います。`hdr->auth` にブラウザから取得した `Authorization` ヘッダ値が入ります。値をデコード・解析してユーザ名・パスワード、ダイジェスト値の妥当性チェックを行います。（`BASE64`、`MD5` 処理はユーザで対応が必要です）成功時は `E_OK`、それ以外は `E_OK` 以外を戻り値に指定するようにします。

認証が失敗時は `hdr->auth` の値が `NULL` でコールバックが呼ばれます。この場合は、`WWW-Authenticate` ヘッダ情報を指定するようにします。（～改行コードまで）

フラグ `evt` がカスタムヘッダ処理 (`HTTPD_EXT_UHDR`) の場合、HTTP サーバタスクが用意する HTTP ヘッダ末尾に任意のヘッダ情報を付与できます。改行コードで終了するヘッダ情報を記述してください。

ファイル名やコンテンツタイプの判定処理が必要な場合、`fp` を参照してください。

HttpSetContent（HTTP サーバ制御情報の送信バッファ 追記）

【書式】

```
ER ercd = HttpSetContent(T_HTTP_SERVER *http, char *str);
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバ制御情報
char	*str	追記する文字列、NULL 指定でバッファクリア

【戻り値】

ER	ercd	追記バイト数（≥0）またはエラーコード
----	------	---------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。 (*http が NULL)
-------	-----------------------------------

【解説】

この API は HTTP サーバ制御情報の送信バッファに文字列を追記します。この API は 拡張動作コールバック関数・CGI 用コールバック関数からのみ呼び出してください。

拡張動作コールバック関数内で使用する場合、str の引数に NULL は指定しないでください。HTTP サーバアプリが途中作成した HTTP ヘッダの内容がクリアされ、動作が不定になります。

HttpSetContentKpa（送信バッファ に HTTP Keep-Alive ヘッダ付与）

【書式】

```
ER HttpSetContentKpa(T_HTTP_SERVER *http);
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバ制御情報
---------------	-------	--------------

【戻り値】

ER	ercd	追記バイト数（≥0）またはエラーコード
----	------	---------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。 (*http が NULL)
-------	-----------------------------------

【解説】

この API は HTTP サーバ制御情報の送信バッファに HTTP Keep-Alive のヘッダ情報を付与します。この API は CGI 用コールバック関数からのみ呼び出してください。

HttpSetContentCookie（送信バッファ に Set-Cookie ヘッダ付与）

【書式】

```
ER HttpSetContentCookie(T_HTTP_SERVER *http,
                        char *name, char *val, char *opt)
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
char	*name	Cookie 名称
char	*value	Cookie 名称の値
char	*opt	属性指定用バッファ（オプション）

【戻り値】

ER	ercd	追記バイト数（ ≥ 0 ）またはエラーコード
----	------	-----------------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。 (*http が NULL)
-------	-----------------------------------

【解説】

この API は HTTP サーバ制御情報の送信バッファに Set-Cookie ヘッダ情報を付与します。この API は拡張動作コールバック関数・CGI 用コールバック関数からのみ呼び出してください。

Cookie の属性指定が必要な場合、引数の opt に属性を文字列形式で指定してください。opt に指定する文字列は {属性名 1}={値 1} の形式で指定してください、複数の属性を指定する場合はセミコロン（;）で属性ペアを結合した文字列にします。

HttpSendBuffer（指定バッファ内容の送信）

【書式】

```
ER ercd = HttpSendBuffer(T_HTTP_SERVER *http, char *str, int len);
```

【パラメータ】

T_HTTP_SERVER	*http	HTTP サーバー制御情報
char	*str	送信するバッファ
int	len	送信するバッファ長

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。 (*http が NULL。*str が NULL)
-------	---

【解説】

この API は HttpSetContent() で作成した HTTP サーバ制御情報の送信バッファの送信、および指定バッファの送信をします。この API は CGI 用コールバック関数からのみ呼び出してください。以下に使用例を記します。

使用例

```
ER HttpSendResponse(T_HTTP_SERVER *http, char *str, int len, char *type)
{
    /* メッセージヘッダ部の作成 */
    HttpSetContent(http, 0);
    HttpSetContent(http, "HTTP/1.1 200 OK\r\n");
    HttpSetContent(http, "Content-Type: ");
    HttpSetContent(http, type);
    HttpSetContent(http, "\r\n");
    HttpSetContentLen(http, "Content-Length: ", len);
    HttpSetContentKpa(http);
    HttpSetContent(http, "\r\n");

    /* メッセージヘッダ部と、指定バッファ（ボディ部）の送信 */
    HttpSendBuffer(http, str, len);

    return E_OK;
}

/* HTTPサーバ CGI用コールバック関数（サンプル） */
void user_cgi_callback (T_HTTP_SERVER *http)
{
    VB contents[128];
    net_strcpy(contents, "<html><body>\r\n");
    net_strcat(contents, "<center>uNet CGI demo</center>\r\n");
    net_strcat(contents, "</body></html>");
    HttpSendResponse(http, contents, net_strlen(contents), "text/html");
}
```

9. 6 設定用マクロ/構造体/定数

HTTP サーバーサンプル

/ コンテンツの定義 */*

```
const char index_html[] =
    "<html>¥
    <title> uNet3 HTTP Server </title>¥
    <body>¥
    <h1>Hello World!</h1>¥
    </body>¥
    </html>";
```

/ コンテンツリストの初期化 */*

```
T_HTTP_FILE const content_list[] =
{
    {"/", "text/html", index_html, sizeof(index_html), NULL},
    {"", NULL, NULL, 0, NULL} /* 終端 */
};
```

/ HTTP セッションの開始 */*

```
static T_HTTP_SERVER http_server1;
void httpd_tsk1(VP_INT exinf)
{
    /* Initialize the content list global pointer */
    gHTTP_FILE = (T_HTTP_FILE*)content_list;

    memset((char*)&http_server1, 0, sizeof(http_server1));
    http_server1.SocketID = ID_SOC_HTTP1;

    http_server(&http_server1);
}
```

T_HTTP_SERVER (HTTP サーバー制御情報)

```

typedef struct t_http_server {
    UW          sbufsz;          /* 送信バッファサイズ */
    UW          rbufsz;          /* 受信バッファサイズ */
    UW          txlen;           /* 内部データ */
    UW          rxlen;           /* 内部データ */
    UW          rdlen;           /* 内部データ */
    UW          len;             /* 内部データ */
    UB          *rbuf;           /* 送信バッファ */
    UB          *sbuf;           /* 受信バッファ */
    UB          *req;            /* 内部データ */
    UH          Port;           /* リスニングポート番号 */
    UH          SocketID;        /* ソケット ID */
    T_HTTP_HEADER hdr;          /* HTTP クライアントリクエスト */
    UB          NetChannel;      /* デバイス番号 */
    UB          ver;             /* IP バージョン */
    UB          server_tsk_stat /* HTTP サーバータスク起動状態 */
    ID          server_tsk_id   /* HTTP サーバータスク ID */
    struct t_http_server *next /* HTTP サーバーオブジェクト */
    UH          kpa_max          /* HTTP KeepAlive max 値 (制御用) */
} T_HTTP_SERVER;

```

この構造体は HTTP サーバーAPI の引数として使用されます。ソケット ID はユーザアプリケーションでセットする必要があります。

デバイス番号

デバイス番号には HTTP サーバーで使用するネットワークデバイスを指定します。'0' を指定した場合はデフォルトのネットワークデバイスが使用されます。(通常は 0 をセットしてください。)

ソケット ID

HTTP サーバー用に作成したソケット ID を指定します。

受信バッファ・送信バッファ

HTTP サーバーではパケットの送受信ごとにプロトコルスタックのネットワークバッファを使用します。

コンテンツサイズなどの理由 (例えばネットワークバッファより大きなコンテンツを送受信したいなど) で、これにアプリケーション独自のバッファを利用したい場合は、受信 (送信) バッファおよび、受信 (送信) バッファサイズに独自バッファの値を設定します。その場合 HTTP サーバーでネットワークバッファを取得することはありません。

また独自設定した領域は他の HTTP サーバープロセスと共有することはできません。

T_HTTP_FILE (HTTP コンテンツ情報)

```
typedef struct t_http_file {
    const char    *path;           /* URL */
    const char    *ctype;          /* コンテンツタイプ */
    const char    *file;           /* コンテンツ */
    Int           len;             /* コンテンツサイズ */
    void(*cbk)(T_HTTP_SERVER *http); /* HTTP コールバック関数
                                     or CGI ハンドラ */
    UB            ext;            /* 拡張動作フラグ */
} T_HTTP_FILE;
```

この構造体に HTTP サーバーで使用するコンテンツを登録します。

URL

コンテンツの URL を表しています。例えば、クライアントからその URL に対して要求があった場合、対応するコンテンツがクライアントに送信されます。

URL に NULL を指定することはできません。また、URL は常に '/' から開始します。

コンテンツタイプ

text/html 等の Content-Type を指定します。動的コンテンツの場合は NULL を指定します。

コンテンツ

実際のコンテンツを指定します。動的コンテンツの場合は NULL を指定します。

コンテンツサイズ

コンテンツのサイズを指定します。動的コンテンツの場合は 0 を指定します。

コールバック関数または CGI ハンドラ

動的コンテンツの時に HTTP サーバーから呼び出される関数のポインタを指定します。静的コンテンツの場合には NULL を指定します。

拡張動作フラグ

コンテンツに対して拡張動作を指定します。(HTTP 認証を使用する、任意の HTTP ヘッダを付与するなど) 拡張動作を使用しない場合 0 を指定します。

コンテンツ拡張動作フラグの定義値

```
#define HTTPD_EXT_AUTH      0x01  /* HTTP 認証使用 */
#define HTTPD_EXT_UHDR      0x02  /* カスタムヘッダ使用 */
```

T_HTTP_FILE::ext に入る定義値

T_HTTP_HEADER (HTTP ヘッダ情報)

```
typedef struct t_http_header {
    char      *method;    /* メソッド */
    char      *url;       /* パス名 */
    char      *url_q;     /* URL クエリ */
    char      *ver;       /* バージョン */
    char      *host;      /* ホスト名 */
    char      *ctype;     /* コンテンツタイプ */
    char      *Content;   /* コンテンツ */
    char      ContentLen; /* コンテンツ長 */
    char      kpa;        /* HTTP Keep Alive 用フラグ (制御用) */
    char      *auth;      /* Authorization ヘッダ */
    char      *cookie;    /* Cookie ヘッダ */
}T_HTTP_HEADER;
```

この構造体は T_HTTP_SERVER::rbuf のバッファを HTTP のリクエストメッセージの各要素として表します。この構造体は HTTP コールバック関数からのみ参照してください。

メソッド [method]

メソッドを表します。"GET"、"HEAD"、"POST"のいずれかが入ります。

パス名 [URL]

URL のパス名を表します。

URL クエリ

URL のクエリパラメータを表します。

バージョン [version]

HTTP バージョンを表します。"HTTP/1.1"、"HTTP/1.0"のいずれかが入ります。

ホスト名 [Host]

リクエスト先ホスト名を表します。

コンテンツタイプ [Content-Type]

コンテンツタイプ名を表します。

コンテンツ

コンテンツ本体のバッファ先頭アドレスを示します。

コンテンツ長 [Content-Length]

コンテンツのバッファ長を示します。

Authorization ヘッダ [auth]

HTTP 認証で使われる Authorization ヘッダの内容を表します。

Cookie ヘッダ [cookie]

リクエストラインの Cookie ヘッダの内容を表します。

コンフィグレーション定義 (http_server_cfg.h)

設定例

```
#define ENA_KEEP_ALIVE          /* Enable HTTP Keep-Alive */
#define HTTP_KPA_MAX    100    /* Keep-Alive max value */
#define ENA_USER_EXT           /* Enable User Extension */
```

ENA_KEEP_ALIVE

定義ありで HTTP Keep-Alive 機能を有効にします。定義なしで無効にします。

HTTP_KPA_MAX

HTTP Keep-Alive ヘッダの max 項目の初期値を指定します。

ENA_USER_EXT

定義ありで HTTP サーバアプリのユーザ拡張動作機能を有効にします。定義なしで無効にします。

10 SMTP クライアント (Pro)

10.1 機能仕様

SMTP サーバーに対してメールの送信を行います。SMTP は TCP を使用したメール転送プロトコルです。

10.2 制限事項

- IPv6、SSL 上での動作は対応していません。(IPv4 のみ)
- ファイルを扱うにはファイルシステムが必要です。
- quoted-printable エンコードには対応していません。文字列エンコード処理は BASE64 エンコードが使用されます。
- このアプリケーションで使用しているコマンドの一覧を次に示します。

コマンド	説明	対応 API
HELO n	指定ホスト名 n で通信を開始する。	smtp_login
EHLO n	指定ホスト名 n で通信を開始する。 認証など、SMTP サービス拡張を使用した通信を行う。	smtp_login
RSET	メールの送信状態をクリアする。	smtp_snd_mail
MAIL FROM: n	指定メールアドレス n を差出人とする。	smtp_snd_mail
RCPT TO: n	指定メールアドレス n を宛て先とする。	smtp_snd_mail
DATA	メールのデータ送信を行うことを通知する。	smtp_snd_mail
QUIT	現在の通信を終了する。	smtp_quit

10.3 OS/ネットワーク資源

SMTP クライアントでは 1 つの TCP ソケットが必要になります。ソケットのパラメータは以下の通りです。(通常コンフィグレータが自動生成します)

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_SMTP	0	v4	TCP	0	0	0	25 秒	25 秒	1024	1024

※ SMTP クライアントの送信・受信タイムアウトの設定は「10.6 設定用マクロ/構造体/定数 コンフィグレーション定義 (smtp_client_cfg.h)」で行います。

10.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→SMTP User Agent の画面で SMTP クライアントのコンフィグレーションを行います。

※当アプリにはファイルシステムが必要です。「12.4 コンフィグレーション画面」最下部参照。

Category	Item	Value
タイムアウト設定	コマンド送受信タイムアウト(ミリ秒)	30000
	コネクション確立タイムアウト(ミリ秒)	30000
	受信タイムアウト(ミリ秒)	
	MAILコマンド送信後	30000
	RCPTコマンド送信後	30000
	DATAコマンド送信後	
	ヘッダ待ち	12000
	データ待ち	18000
	フッタ待ち	60000
	MIME Boundary	---- uC3 MIME Boundary

有効にする

SMTP クライアントアプリがソース生成対象となります。

MIME Boundary

MIME のパートを区別するための境界文字列を指定します。

タイムアウト設定

SMTP クライアントの各処理におけるタイムアウトの値を指定します。通常デフォルト設定で問題ありません。

10.5 API

smtp_login（接続）

【書式】

```
ER smtp_login(T_SMTP_CLIENT *smtp);
```

【パラメータ】

T_SMTP_CLIENT	*smtp	SMTP クライアント制御情報
---------------	-------	-----------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	smtp が NULL
E_PAR	接続先の IP アドレスが未指定
E_SYS	応答コード異常 (smtp->buf の内容を確認する)
E_NOSPT	認証方式が未サポート

【解説】

この API は引数で設定した SMTP サーバーに接続します。SMTP クライアント制御情報には、IPv4 アドレス、ポート番号、認証方式を予め指定して下さい。

SMTP クライアントでは TCP ソケットを使用します。使用可能なソケット ID を引数に設定して下さい。内部的には HELO,EHLO コマンドを発行します。

SMTP サーバーの接続には EHLO コマンドを使用します。EHLO コマンドが未サポートのサーバーの場合、もしくは認証方式なしの場合に限り HELO コマンドを使用して接続を試みます。

smtp_quit (切断)**【書式】**

```
ER smtp_quit(T_SMTP_CLIENT *smtp);
```

【パラメータ】

T_SMTP_CLIENT	*smtp	SMTP クライアント制御情報
---------------	-------	-----------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	smtp が NULL
-------	-------------

【解説】

この API は引数で設定した SMTP サーバーから切断します。上記エラーコードを除き、この API は常に E_OK を返却します。内部的には QUIT コマンドを発行します。

smtp_snd_mail（メール送信）

【書式】

```
ER smtp_snd_mail(T_SMTP_CLIENT *smtp, T_SMTP_MAIL *mail);
```

【パラメータ】

T_SMTP_CLIENT	*smtp	SMTP クライアント制御情報
const VB	*cmd	SMTP コマンド文字列
const VB	*res	正常と認識する応答コード

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	smtp が NULL
E_SYS	応答コード異常 (smtp->buf の内容を確認する)
E_PAR	引数の値が異常 (コマンド文字列が NULL)
E_CLS	SMTP の通信中に異常が生じた

【解説】

この API は引数で設定した SMTP サーバーに対してメールを送信します。メールの内容は T_SMTP_MAIL 構造体に値をセットすることで設定します。

SMTP サーバーにログインが完了した状態で、この API を発行してください。

smtp_snd_cmd (コマンド発行)

【書式】

```
ER smtp_snd_cmd(T_SMTP_CLIENT *smtp, const VB *cmd, const VB *res);
```

【パラメータ】

T_SMTP_CLIENT	*smtp	SMTP クライアント制御情報
const VB	*cmd	SMTP コマンド文字列
const VB	*res	正常と認識する応答コード

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	smtp が NULL
E_SYS	応答コード異常 (smtp->buf の内容を確認する)
E_PAR	引数の値が異常 (コマンド文字列が NULL)
E_CLS	SMTP の通信中に異常が生じた

【解説】

この API は引数で設定した SMTP サーバーに対して任意のコマンドを発行します。

10.6 設定用マクロ/構造体/定数

T_SMTP_CLIENT (SMTP クライアント制御情報)

```
typedef struct t_smtp_client {
    UW    ipa;           /* 接続先 SMTP サーバーの IP アドレス */
    UH    dev_num;       /* デバイス番号 */
    UH    sid;           /* ソケット ID */
    UH    port;          /* ポート番号 (接続先) */
    UB    flag;          /* ステータスフラグ (制御用) */
    VB    *buf;          /* メール送信用バッファ */
    UH    len;           /* バッファ長 */
    UB    auth_type;     /* 認証方式 */
    VB    *usr;          /* SMTP 認証 ユーザーID */
    VB    *pw;           /* SMTP 認証 パスワード */
} T_SMTP_CLIENT;
```

この構造体に必要な情報をセットして SMTP クライアント API の引数として渡します。

デバイス番号

デバイス番号には SMTP クライアントで使用するネットワークデバイスを指定します。'0'を指定した場合はデフォルトのネットワークデバイスが使用されます。(通常は0をセットしてください。)

ソケット ID

SMTP クライアント用に作成した TCP ソケットの ID を指定してください。

ポート番号 (接続先)

SMTP サーバーのポート番号(25)を任意の番号に変更する場合はこちらの値を変更して下さい。(通常は0をセットしてください。)

メール送信用バッファ、バッファ長

SMTP クライアントアプリで使用するバッファを指定します。バッファのサイズは1024以上を指定して下さい。サイズが大きいほどパケット分割が少なくなります。

認証方式、SMTP 認証ユーザーID、SMTP 認証パスワード

認証方式には SMTP_NO_AUTH (認証なし)、SMTP_AUTH_PLAIN (平文認証)、SMTP_AUTH_CMD5 (CRAM-MD5 認証) のいずれかを指定して下さい。認証方式に認証なし以外を選択した場合はユーザーID・パスワードを指定します。

T SMTP_MAIL (SMTP メール情報)

```
typedef struct t_smtp_mail {
    VB          *from;          /* 差出人メールアドレス */
    VB          *to;            /* 宛先メールアドレス */
    VB          *cc;            /* CC */
    VB          *bcc;           /* BCC */
    VB          *date;          /* 送信日時 */
    VB          *subject;       /* メール件名 */
    VB          *body;          /* メール本文 */
    VB          *charset;       /* 使用文字コード */
    T SMTP_FILE *file;          /* 添付ファイル情報 */
    UB          cs8bit;         /* 8bit 文字コード判定フラグ */
} T SMTP_MAIL;
```

この構造体に必要な情報をセットして smtp_snd_mail API の引数として渡します。

差出人メールアドレス、宛先メールアドレス、CC、BCC

各種メールアドレスを文字列で指定します。”メールアドレス”,”表示名 <メールアドレス>”,”<メールアドレス>”のいずれかで指定します。(ex: “<test@example.com>”, “Tester <test@example.com>”, “<test@example.com>”) 複数指定はカンマ,”で区切ります。差出人メールアドレスに複数指定した場合、初回のものを採用します。

送信日時

メールの送信日時を指定する際に指定します。(値チェックは行いません。0 を指定すると DATE ヘッダは省略された状態でメールが作成されます。)

メール件名、メール本文

件名、本文を指定します。使用文字コードに対応した文字列を指定します。

使用文字コード、8bit 文字使用コード判定フラグ

メール件名、メール本文で使用する文字コードを指定します。未指定時の場合、US-ASCII として認識されます。また、8bit 目を使用する文字コードの場合は、判定フラグの値を 0 以外に設定します。判定フラグが 0 以外の場合、メール本文は base64 エンコードされた形式でメール送信されます。

添付ファイル情報

ファイルを添付する場合に指定します。

T SMTP_FILE (SMTP ファイル情報)

```
typedef struct t_smtp_file {
    struct t_smtp_file *next;    /* 次の添付ファイル情報 */
    VB *name;                   /* 添付ファイル名 */
    VB *buf;                    /* パス or ファイルバッファ */
    UW len;                     /* バッファのサイズ */
    UB type;                    /* ファイルタイプ */
} T SMTP_FILE;
```

次の添付ファイル情報

添付ファイルが複数ある場合に指定します。終端のファイルでは 0 に設定します。

添付ファイル名

メールに添付されるファイルの名称を指定します。

ファイルタイプ、パス or ファイルバッファ

ファイルシステムからのファイル読込 (SMTP_FTYPE_FS) か、メモリ領域読込 (SMTP_FTYPE_MEM) かを指定します。buf には、ファイル読込の場合、パスを指定します。それ以外の場合、メモリ領域の先頭バッファを指定します。

バッファのサイズ

メモリ領域のバッファサイズを示します。ファイル読込の場合は無視されます。

コンフィグレーション定義 (smtp_client_cfg.h)

設定例

```
#include "ffsys.h"          /* File system */

/* SMTP Mail settings */
#define SMTP_MIME_BND_HEAD  "---- uC3 MIME Boundary "

/* SMTP Timeout settings */
#define SMTP_TMO_CMD      300000    /* CMD Send Timeout */
#define SMTP_TMO_INIT    300000    /* 220 Recv Timeout */
#define SMTP_TMO_MAIL    300000
#define SMTP_TMO_RCPT    300000
#define SMTP_TMO_DATA1   120000    /* DATA start input */
#define SMTP_TMO_DATA2   180000    /* DATA TCP send */
#define SMTP_TMO_DATA3   600000    /* DATA Terminate */
```

SMTP_MIME_BND_HEAD

boundary パラメータ (MIME の境界文字列) を指定します。

SMTP_TMO_xxx

SMTP の各プロセスでのタイムアウト値を指定します。

SMTP 認証方式の定義値

```
/* SMTP Authentication type (T_SMTP_CLIENT::auth_type) */
#define SMTP_NO_AUTH      0          /* SMTP 認証を使用しない */
#define SMTP_AUTH_PLAIN   1          /* PLAIN */
#define SMTP_AUTH_CMD5    2          /* CRAM-MD5 */
```

T_SMTP_CLIENT::auth_type に入る定義値

添付ファイル形式の定義値

```
/* Attach File type (T_SMTP_FILE::type) */
#define SMTP_FTYPE_FS     0x01      /* ファイル指定 (FileSystem 使用) */
#define SMTP_FTYPE_MEM    0x02      /* メモリ指定 */
```

T_SMTP_FILE::type に入る定義値

1 1 Telnet サーバー (Pro)

1 1. 1 機能仕様

Telnet サーバーは、Telnet クライアントからの接続を待ち受け、ホスト間でのプレーンテキストのやりとりを行います。

1 1. 2 制限事項

- IPv6、SSL 上での動作は対応していません。(IPv4 のみ)
- 動作にはシェルが必要となります。Telnet サーバーアプリ単体では動作しません。
- Telnet コマンドの対応一覧は以下となります。(○：対応)

対応	名前	コード	HEX	説明
	EOF	236	EC	EOF
	SUSP	237	ED	現行プロセスの中断(ジョブ制御)
	ABORT	238	EE	プロセス打ち切り
	EOR	239	EF	レコード終端
○	SE	240	F0	サブオプション終了
○	NOP	241	F1	オペレーションなし
	DM	242	F2	データ・マーク
	BRK	243	F3	ブレーク
	IP	244	F4	プロセスの割り込み
	AO	245	F5	出力打ち切り
○	AYT	246	F6	そこにいますか？
○	EC	247	F7	エスケープ・キャラクタ
	EL	248	F8	行消去
	GA	249	F9	先へ進め
○	SB	250	FA	サブオプション開始
○	WILL	251	FB	オプション交渉
○	WONT	252	FC	オプション交渉
○	DO	253	FD	オプション交渉
○	DONT	254	FE	オプション交渉
○	IAC	255	FF	データ・バイト 255

- Telnet オプションの対応一覧は以下となります。(○：対応)

対応	RFC	コード	HEX	説明
○	857	1	1	echo (エコー)
○	858	3	3	suppress go ahead (進行抑止)
○	859	5	5	status (状態)
	860	6	6	timing mark (タイミング・マーク)
	1091	24	18	terminal type (ターミナル・タイプ)
	1073	31	1F	window size (ウィンドウ・サイズ)
	1079	32	20	terminal speed (ターミナル速度)
	1372	33	21	remote flow control (リモート・フロー制御)
	1184	34	22	linemode (ラインモード)
	1408	36	24	environment variables (環境変数)

1 1. 3 OS/ネットワーク資源

Telnet サーバーでは 1 つの TCP ソケット、2 つのタスク (Telnet サーバー用、シェル用)、タスク間の通信用に 1 つのメールボックスと 1 つの固定長メモリプール、1 つのイベントフラグを使用します。(通常コンフィグレータが自動生成します)

各種リソース ID のパラメータは以下の通りです。

ソケット

ID	デバイス番号	IP ver	プロトコル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_TELNET	n	v4	TCP	23	25 秒	25 秒	25 秒	25 秒	1024	1024

タスク

推奨 ID 名	推奨関数名	優先度	属性	スタック	拡張情報
ID_TELNET_TSK n	telnet_svr_tsk n	4	なし	1024	なし
ID_SHELL_TSK n	shell_tsk n	4	なし	—※	なし

※スタックサイズはシェル上で動作させるユーザーアプリにより変わるため適宜設定してください。

メールボックス

推奨 ID 名	属性
ID_TELNET_MBX n	TA_TFIFO TA_MFIFO

固定長メモリプール

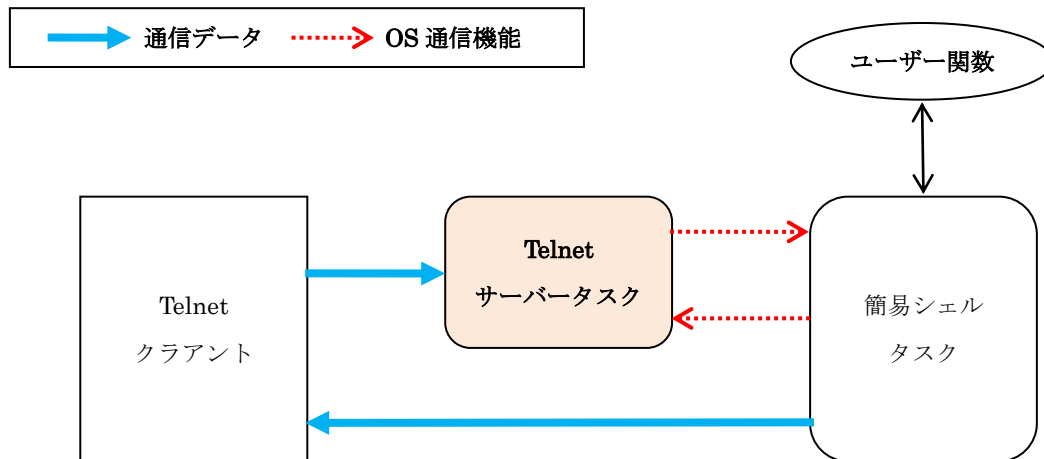
推奨 ID 名	ブロック数	ブロックサイズ	属性
ID_TELNET_MPF n	4	144	TA_TFIFO

イベントフラグ

推奨 ID 名	初期値	複数タスク待ち	解除時のクリア
ID_TELNET_FLG n	0	許可しない (TA_WSGL)	チェックなし (しない)

シェルタスクの用意

Telnet サーバーの動作にはシェルが必要となります。簡易シェルをサンプルとして提供しています。ここでは Telnet サーバーと簡易シェル間でのデータのやりとりを説明します。



- ① Telnet クライアントからデータを受信します。
- ② 制御データ (Telnet コマンド、オプション) 以外を受信した場合、固定長メモリプール・メールボックスを使用して簡易シェルにデータを転送します。
- ③ Telnet サーバーはシェルからデータを受信する口を用意していません。そのため、シェルから Telnet クライアントに対して直接データを送信します。
- ④ 簡易シェルから Telnet サーバーに対してイベントフラグを使用して、命令を通知します。(シェルの終了、echo オプションの有無要求)
- ⑤ 簡易シェルではユーザー関数をコマンドとして実行する仕組みを提供しています。

ユーザーでシェルを用意する場合は上記①～④を考慮して作成して下さい。

11.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→Telnet Server の画面で Telnet サーバー、および簡易シェルのコンフィグレーションを行います。（上部：Telnet サーバーの設定、下部：簡易シェルの設定）

有効にする

Telnet サーバーアプリがソース生成対象となります。

セッション最大数

Telnet サーバーへ接続するセッションの上限を指定します。

AYT コマンド応答文字列

Telnet 制御コマンド”AYT”をサーバーが受信した際に返信する文字列を指定します。

【シェル基本設定】

簡易シェルの動作に関する設定を行います。

シェル基本設定	シェルプロンプト表示設定	ログインユーザ	コマンド設定
ユーザ名最大文字数	16		
パスワード最大文字数	16		
ログイン再試行回数	3		
ヒストリバッファ(2の階乗で指定)	3		
改行コード	CR		
<input checked="" type="checkbox"/> コマンド結果のエラーコードを表示する <input checked="" type="checkbox"/> Welcomeメッセージを表示する <input checked="" type="checkbox"/> Loginプロンプトを表示する			

ユーザー名最大文字数、パスワード最大文字数

ログイン時のユーザー名・パスワードそれぞれのバッファサイズを指定します。

ログイン再試行回数

ログインプロンプトで試行可能な回数を指定します。

ヒストリバッファ (2 の累乗で指定)

シェルのヒストリバッファサイズを 2 の累乗で指定します。(2ⁿ の n を指定)

改行コード

シェルの改行コードを指定します。

コマンド結果のエラーコードを表示する

シェルのコマンド実行後、戻り値のエラーコードをコンソール上に表示します。

Welcome メッセージを表示する

Telnet サーバーに接続後、ログインプロンプト前にメッセージを表示します。表示メッセージは【シェルプロンプト表示設定】で設定します。

Login プロンプトを表示する

ログインプロンプトを表示します。チェックなしの場合、ログインプロンプトは表示されず認証なしでシェルの入力待ちに遷移します。

【シェルプロンプト表示設定】

簡易シェルに表示するメッセージに関する設定を行います。

シェル基本設定	シェルプロンプト表示設定	ログインユーザ	コマンド設定
Welcomeメッセージ	<input type="text" value="uC3 Shell 1.0"/>		
Userプロンプト	<input type="text" value="Login:"/>		
Passwordプロンプト	<input type="text" value="Password:"/>		
Login成功メッセージ	<input type="text" value="Login correct."/>		
Login失敗メッセージ	<input type="text" value="Login incorrect."/>		
コマンドプロンプト	<input type="text" value="Shell>"/>		
入力バッファ超過	<input type="text" value="Too long"/>		

Welcome メッセージ

Telnet サーバー接続直後に表示されるメッセージを指定します。

User プロンプト、Password プロンプト

ログインプロンプトのユーザー名・パスワードそれぞれの表示メッセージを指定します。

Login 成功メッセージ、Login 失敗メッセージ

ログインが成功・失敗した時の表示メッセージを指定します。

コマンドプロンプト

シェルのコマンドプロンプト表示メッセージを指定します。

入力バッファ超過

シェルのコマンドプロンプト入力バッファ長を入力文字数が超えた際に表示する警告メッセージを指定します。

【ログインユーザ】

簡易シェルのログインプロンプトでログイン可能なユーザー名の設定を行います。

シェル基本設定 シェルプロンプト表示設定 ログインユーザ コマンド設定

User Name	Password
User	Password

追加... 削除

ログインユーザー名一覧

ログイン可能なユーザー名を表示します。ダブルクリックで編集が行えます。

追加

リストに新規ユーザー名を追加します。後述する設定画面が表示します。

削除

リストから選択ユーザー名を削除します。

ログインユーザー設定画面

Login User

User Name

Password

OK キャンセル

User Name

ログインユーザー名を指定します。同名ユーザーの指定は出来ません。

Password

ログインユーザー名に対応するパスワードを指定します。

【コマンド設定】

簡易シェルのコマンド設定を行います。

コマンド	実行関数名	引数	説明	使用例
ip	shell_cmd_ip	0	Display IP Address	
quit	shell_cmd_quit	0	Disconnect Telnet server	
help	shell_cmd_help	0	Help	
?	shell_cmd_help	0	Help	

追加... 削除

コマンド一覧

使用可能なコマンドを表示します。ダブルクリックで編集が行えます。

追加

リストに新規コマンドを追加します。後述する設定画面が表示します。

削除

リストから選択コマンドを削除します。

コマンド設定画面

Command

コマンド

ping

実行関数名

shell_usr_cmd_ping

引数

1

説明

Ping Request

使用例

destination

OK

キャンセル

コマンド

コマンドを指定します。同名コマンドの指定は出来ません。

実行関数名

コマンド実行時にコールされる関数名を指定します。

引数

コマンド実行に最低必要な引数の数を指定します。引数の数がこの値以下の場合、簡易シェルはコマンドを実行せず、代わりに「使用例」メッセージを表示します。

説明

Help コマンド実行時に表示されるコマンドの説明文を指定します。

使用例

コマンドの使用例を指定します。

●デフォルトコマンド

簡易シェルのソース内にあるデフォルトコマンドを以下に示します。コンフィグレータでは初期コマンドとして登録済みです。コマンドが不要な場合は削除可能です。

コマンド	実行関数名	引数	説明	処理内容
ip	shell_cmd_ip	0	Display IP Address	IP アドレス情報を表示する
quit	shell_cmd_quit	0	Disconnect Telnet server	Telnet サーバから切断する
help	shell_cmd_help	0	Help	ヘルプ（コマンド一覧）を表示する
?	shell_cmd_help	0	Help	同上

●ユーザーコマンドの形式について

ユーザーコマンドは次の書式の関数を用意します。（*cmd_xxx* は任意の関数名）

書式) *ER cmd_xxx(VP ctrl, INT argc, VB *argv[])*

argc は引数の数、*argv* は引数の値、*ctrl* は制御用の変数となります。

●ユーザーコマンド内処理でのシェルとのやりとり

ユーザーコマンド関数内でシェルに対してやりとりを行う場合、shell_puts, shell_gets 関数を使用します。その際、第 1 引数のパラメータには上記の *ctrl* を指定します。

関数定義	説明
ER shell_puts(T_SHELL_CTL *sh, const VB *str);	シェルに str を表示する。
ER shell_gets(T_SHELL_CTL *sh, VB *str, UH len);	シェルから str を取得する。（バッファサイズ len）

1 1. 5 API

telnet_server（サーバーの開始）

【書式】

```
ER ercd = telnet_server(T_TELNET_SERVER *telnet);
```

【パラメータ】

T_TELNET_SERV	*telnet	Telnet サーバー制御情報
ER		

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_ID	デバイス番号が不正です。
E_PAR	不正なパラメータが指定された。 (*telnet が NULL。各種 id が指定されていない。)

【解説】

この API は Telnet クライアントの要求を受け付け処理します。Telnet サーバーでは TCP ソケットを使用します。使用可能なソケット ID を制御情報に設定して下さい。

この API はブロッキング呼出しになっていますので、専用タスクを用意しそこから呼び出すようにしてください。

1 1. 6 設定用マクロ/構造体/定数

T_TELNET_SERVER (TELNET サーバ制御情報)

```
typedef struct t_telnet_server {
    UH          dev_num;    /* デバイス番号 */
    UH          port;       /* ポート番号 */
    UH          sid;        /* ソケット ID */
    ID          shell_tid;  /* 起動シェルタスク ID */
    ID          mpf_id;     /* 固定長メモリプール ID */
    ID          flg_id;     /* イベントフラグ ID */
    ID          mbx_id;     /* メールボックス ID */
    T_TELNET_OPTION opt;    /* (制御用) オプション交渉ステータス */
    T_TELNET_OPTION snd_opt; /* (制御用) オプション交渉送信フラグ */
} T_TELNET_SERVER;
```

この構造体に必要な情報をセットして Telnet サーバ API の引数として渡します。

デバイス番号

デバイス番号には Telnet サーバで使用するネットワークデバイスを指定します。

ポート番号

ポート番号には Telnet サーバで使用するリスンポートを指定します。'0'を指定した場合はソケット ID 指定のリスンポートが使用されます。

ソケット ID

Telnet サーバ用に作成した TCP ソケットの ID を指定してください。

固定長メモリプール ID、イベントフラグ ID、メールボックス ID、シェルタスク ID

Telnet サーバ用に作成した各種 OS リソースの ID を指定してください。

T_SHELL_BLK (SHELL データ転送構造体)

```
typedef struct t_shell_blk {
    T_MSG      *msg;
    VB         buf[128];    /* データバッファ */
    UW         len;         /* バッファサイズ */
}; T_SHELL_BLK
```

Telnet サーバーが、Telnet クライアントから受信したデータをシェル側にメールボックスで転送する際の固定長メモリプールの構造です。

コンフィグレーション定義 (telnet_server_cfg.h)

設定例

```
/* Return string of AYT command reception */
#define MSG_TC_AYT      "%r%n[uC3 Telnet server: yes]%r%n"
```

MSG_TC_AYT

クライアントから AYT コマンドが発行された際に返答するメッセージです。

Telnet サーバー通知用イベントフラグ定義値

```
/* シェル→Telnet サーバーに通知可能なイベント */
#define SFC_ECHO_ON      0x0100    /* echo 機能の有効を要求 */
#define SFC_ECHO_OFF     0x0200    /* echo 機能の無効を要求 */
#define SFC_FECHO_ON     0x0400    /* echo 機能を有効化 (強制) */
#define SFC_FECHO_OFF    0x0800    /* echo 機能は無効化 (強制) */
#define SFC_SHELL_QUIT   0x4000    /* シェルの終了(セッション切断) */
```

シェルから Telnet サーバーに対してイベントフラグを使用して上記命令を発行できます。ユーザーがシェルを用意する場合、Telnet サーバーでセッション切断の判定を行う必要があるためシェルの終了 (SFC_SHELL_QUIT) は必ず実行して下さい。

1 2 FTP クライアント (Pro)

1 2. 1 機能仕様

FTP クライアントは、FTP サーバーに対してファイルのアップロードとダウンロードを可能にします。FTP は TCP を使用したファイル転送プロトコルです。

1 2. 2 制限事項

- FTPS クライアントとしての動作が可能です。(別途 μ Net3/SSL が必要となります)
- IPv6 上での動作は対応していません。(IPv4 のみ)
- ファイルを扱うにはファイルシステムが必要です。
- FTP クライアントで対応しているコマンドの一覧を次に示します。

コマンド	説明	備考、対応 API
USER n	指定したユーザー名 n でログインする。	ftp_login
PASS n	ユーザーのパスワード n を指定する、	ftp_login
AUTH n PBSZ 0 PROT P	SSL/TLS 認証が対応しているか問い合わせる。 データ接続はストリーム転送とする。 データ接続をプライベートとする。	ftp_login
RETR n	指定したファイル n をサーバから取得する。	ftp_get_file
STOR n	指定したファイル n をサーバに設置する。	ftp_put_file
DELE n	指定したファイル n をサーバから削除する。	ftp_del_file
RNFR n RNTO i	指定したファイル n を i にリネームする。	ftp_rename_file
PWD	現在のワーキングディレクトリを表示する。	ftp_cmd_pwd
CWD n	現在のワーキングディレクトリを n に移動する。	ftp_cmd_cd
MKD n	新規ディレクトリ n を作成する。	ftp_cmd_mkd
RMD n	既存ディレクトリ n を削除する。	ftp_cmd_rmd
LIST n NLST n	ディレクトリ n のファイル一覧を表示する。 (詳細、簡易どちらも可)	ftp_cmd_list ftp_cmd_list_next
NOOP	何もしない。(疎通確認)	ftp_cmd_noop
QUIT	ログアウトする。	ftp_quit

1 2. 3 OS/ネットワーク資源

FTP クライアントでは制御用ポート、データ用ポート用の 2 つの TCP ソケットが必要になります。ソケットのパラメータは以下の通りです。(通常コンフィグレータが自動生成します)

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_FTP_CMD	0	v4	TCP	0	—	—	—	—	1024	1024
ID_SOC_FTP_DATA	0	v4	TCP	0	—	—	—	—	1024	1024

※ FTP クライアントのタイムアウト設定は「1 2. 6 設定用マクロ/構造体/定数」の T FTP CLIENT の cmd_tmo メンバで行います。

12.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→FTP Client の画面で FTP クライアントのコンフィグレーションを行います。



有効にする

FTP クライアントアプリがソース生成対象となります。

FTP over SSL/TLS を有効にする（μNet3/SSL が必要）

FTPS クライアントとして動作可能とするかを選択できます。

【File system 画面】

いくつかのプロトコルではファイルシステムが必要となります。TCP/IP メニュー画面のネットアプリケーション→FileSystem の画面でファイルシステムのコンフィグレーションを行います。 ※ μ Net3 パッケージには簡易ファイルシステムが付属しています。

有効にする

ファイルシステムがソース生成対象となります。

他ファイルシステムを使用する、インクルードファイルの指定

簡易ファイルシステム以外を使用する場合に選択します。エディットには当該ファイルシステムのヘッダファイルを指定します。

eForce 簡易ファイルシステムを使用する

簡易ファイルシステムを使用する場合に選択します。

最大ファイル数、最大ファイルサイズ

簡易ファイルシステムで扱えるファイル数、ファイルサイズを指定します。

1 2 . 5 API

ftp_login（接続）

【書式】

```
ER ercd = ftp_login(T_FTP_CLIENT *ftp, const VB *user, const VB *pw);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
const VB	*user	FTP サーバーの認証ユーザ名
const VB	*pw	FTP サーバーの認証パスワード

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
E_PAR	接続先の IP アドレスが未指定
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は引数で設定した FTP サーバーに接続します。FTP サーバーの設定には、IPv4 アドレス、ポート番号、ファイル転送モードを指定して下さい。また、引数の `user` もしくは `pw` に NULL を指定した場合は "anonymous" が代わりに使用されます。

FTP クライアントでは TCP ソケットを使用します。使用可能なソケット ID を制御情報に設定して下さい。内部的には USER,PASS コマンドを発行します。

FTP サーバーに接続できた場合は E_OK を返却します。この API 以外の FTP クライアント API は、FTP サーバー接続時に有効です。

μ Net3/SSL をお使いの場合、FTPS サーバへの接続が可能です。詳細は「FTP 転送モードの定義値」を参照してください。

ftp_quit (切断)

【書式】

```
ER ercd = ftp_quit(T_FTP_CLIENT *ftp);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
--------------	------	----------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
-------	------------

【解説】

この API は引数で設定した FTP サーバーから切断します。上記エラーコードを除き、この API は常に E_OK を返却します。内部的には QUIT コマンドを発行します。

ftp_get_file（ファイル取得）**【書式】**

```
ER ercd = ftp_get_file(T_FTP_CLIENT *ftp, const VB *lo_file, const VB *rmt_file);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
const VB	*lo_file	ローカルファイルパス
const VB	*rmt_file	FTP サーバー側ファイルパス

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
E_PAR	引数の値が異常（ファイル指定が不正など）
E_CLS	FTP の通信中に異常が生じた。
E_NOMEM	ファイルの書込みに失敗した可能性

【解説】

この API は引数で設定した FTP サーバーからファイルを取得します。FTP サーバーの設定には、IPv4 アドレス、ポート番号、ファイル転送モードを指定して下さい。内部的には RETR コマンドを発行します。コマンドが正常終了時に E_OK を返却します。

ftp_put_file（ファイル設置）**【書式】**

```
ER ercd = ftp_put_file(T_FTP_CLIENT *ftp, const VB *lo_file, const VB *rmt_file);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
const VB	*lo_file	ローカルファイルパス
const VB	*rmt_file	FTP サーバー側ファイルパス

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
E_PAR	引数の値が異常（ファイル指定が不正など）
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は引数で設定した FTP サーバーへファイルを設置します。FTP サーバーの設定には、IPv4 アドレス、ポート番号、ファイル転送モードを指定して下さい。内部的には STOR コマンドを発行します。コマンドが正常終了時に E_OK を返却します。

ftp_del_file（ファイル削除）**【書式】**

```
ER ercd = ftp_del_file(T_FTP_CLIENT *ftp, const, const VB *rmt_file);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
const VB	*rmt_file	FTP サーバー側ファイルパス

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
E_PAR	引数の値が異常（ファイル指定が不正など）
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は引数で設定した FTP サーバーからファイルを削除します。内部的には DELE コマンドを発行します。コマンドが正常終了時に E_OK を返却します。

ftp_ren_file（ファイルのリネーム）**【書式】**

```
ER ercd = ftp_ren_file(T_FTP_CLIENT *ftp, const VB *org_name, const VB *ren_name);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
const VB	*org_name	変更前 FTP サーバー側ファイルパス
const VB	*ren_name	変更後 FTP サーバー側ファイルパス

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
E_PAR	引数の値が異常（ファイル指定が不正など）
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は引数で設定した FTP サーバーのファイルをリネームします。内部的には RNFR, RNTD コマンドを発行します。コマンドが正常終了時に E_OK を返却します。

ftp_cmd_cd（作業ディレクトリの移動）**【書式】**

```
ER ercd = ftp_cmd_cd(T_FTP_CLIENT *ftp, const VB *dir);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
const VB	*dir	移動先ディレクトリ

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
E_PAR	引数の値が異常（ファイル指定が不正など）
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は引数で設定した FTP サーバーの作業ディレクトリを **dir** に移動します。内部的には CWD コマンドを発行します。コマンドが正常終了時に **E_OK** を返却します。

ftp_cmd_pwd（作業ディレクトリの取得）**【書式】**

```
ER ercd = ftp_cmd_pwd(T_FTP_CLIENT *ftp, VB *dir);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
VB	*dir	作業ディレクトリ

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
E_PAR	引数の値が異常（ファイル指定が不正など）
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は引数で設定した FTP サーバーの作業ディレクトリを取得します。内部的には PWD コマンドを発行します。コマンドが正常終了時に **E_OK** を返却します。

ftp_cmd_mkd（新規ディレクトリの作成）**【書式】**

```
ER ercd = ftp_cmd_mkd(T_FTP_CLIENT *ftp, const VB *dir);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
const VB	*dir	新規ディレクトリ名

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
E_PAR	引数の値が異常（ファイル指定が不正など）
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は引数で設定した FTP サーバーに新規ディレクトリ **dir** を作成します。内部的には MKD コマンドを発行します。コマンドが正常終了時に E_OK を返却します。

ftp_cmd_rmd（既存ディレクトリの削除）**【書式】**

```
ER ercd = ftp_cmd_mkd(T_FTP_CLIENT *ftp, const VB *dir);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
const VB	*dir	新規ディレクトリ名

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
E_PAR	引数の値が異常（ファイル指定が不正など）
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は引数で設定した FTP サーバーの既存ディレクトリ **dir** を削除します。内部的には RMD コマンドを発行します。コマンドが正常終了時に E_OK を返却します。

ftp_cmd_noop（何もしない）**【書式】**

```
ER ercd = ftp_cmd_noop(T_FTP_CLIENT *ftp);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
--------------	------	----------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OBJ	ftp が NULL
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は何もしません。FTP サーバーとの疎通確認用に使用できます。内部的には NOOP コマンドを発行します。コマンドが正常終了時に E_OK を返却します。

ftp_cmd_list（ファイル名の一覧取得）**【書式】**

```
ER ercd = ftp_cmd_list(T_FTP_CLIENT *ftp, const VB *dir, T_FTP_LIST *list);
```

【パラメータ】

T_FTP_CLIENT	*ftp	FTP クライアント制御情報
const VB	*dir	作業ディレクトリパス（未指定は NULL）
T_FTP_LIST	*list	LIST/NLST コマンド結果格納構造体

【戻り値】

ER	ercd	バッファに格納したデータサイズ（≥0）、またはエラーコード
----	------	-------------------------------

【エラーコード】

E_OBJ	ftp が NULL
E_PAR	引数の値が異常（list, list->buf が NULL など）
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は引数で設定した FTP サーバーに対してファイル名一覧を取得、結果を list に格納します。内部的には LIST、NLST コマンドを発行します。

戻り値のデータサイズ < 構造体メンバ len の場合、LIST 結果データの終端を表します。それ以外の場合、ftp_cmd_list_next を呼び出すことで残りの結果データを取得出来ます。

ftp_cmd_list (ファイル名の一覧取得続き)**【書式】**

```
ER ercd = ftp_cmd_list_next(T_FTP_LIST *list);
```

【パラメータ】

T_FTP_LIST	*list	LIST/NLST コマンド結果格納構造体
------------	-------	-----------------------

【戻り値】

ER	ercd	バッファに格納したデータサイズ (≥0)、またはエラーコード
----	------	--------------------------------

【エラーコード】

E_OBJ	ftp が NULL ※ftp_list 実行時パラメータ
E_PAR	引数の値が異常 ※ftp_list 実行時パラメータ
E_CLS	FTP の通信中に異常が生じた。

【解説】

この API は ftp_cmd_list 実行時に引数で設定した FTP サーバーに対して LIST、NLST コマンドを発行してその結果を list に取得します。LIST 結果データの終端までバッファに読み込む場合は、戻り値のデータサイズ < 構造体メンバ len になるまで API を実行し続けて下さい。

使用例

```
ER sample_list(T_FTP_CLIENT *ftp, const VB *dir, UB nameonly)
{
    T_FTP_LIST list;
    ER ercd;
    VB ls_buf[1024];

    net_memset(&list, 0, sizeof(list));
    list.buf = ls_buf;
    list.len = sizeof(ls_buf) - 1;
    list.nameonly = nameonly;

    ercd = ftp_cmd_list(ftp, dir, &list);
    while (list.len == ercd) {
        ls_buf[ercd] = '\0';
        printf(ls_buf);
        ercd = ftp_cmd_list_next(&list);
    }

    if (0 > ercd) {
        /* Error LIST/NLST Command */
    }
    else {
        ls_buf[ercd] = '\0';
        printf(ls_buf);
        ercd = E_OK;
    }
}
```

1 2 . 6 設定用マクロ/構造体/定数

T_FTP_CLIENT (FTP クライアント制御情報)

```
typedef struct t_ftp_client {
    UW    ipa;                /* 接続先 FTP サーバーの IP アドレス */
    UH    dev_num;            /* デバイス番号 */
    UH    ctl_sid;            /* 制御用ポートのソケット ID */
    UH    dat_sid;            /* データ用ポートのソケット ID */
    UH    ctl_port;           /* 制御用ポート番号 (接続先) */
    UH    dat_port;           /* データ用ポート番号 (接続先) */
    UB    mode;               /* 転送モード */
    UB    type;               /* 転送タイプ */

    UW    cmd_tmo;            /* コマンドタイムアウト */
    UH    flag;               /* (制御用) 接続状態フラグ */
    UB    Cmd[];              /* (制御用) 直近コマンド */
    UB    Res[];              /* (制御用) 直近コマンド応答コード */
    UB    Data[];             /* (制御用) 直近データ */
    UB    RxResBuf[];         /* (制御用) 作業用受信バッファ */
    UH    RxResLen;           /* (制御用) 受信バッファ長 */
    UH    RxResIndex;         /* (制御用) 受信バッファ位置 */
    ID    ssl_sid;            /* (制御用) SSL セッション ID */
} T_FTP_CLIENT;
```

この構造体に必要な情報をセットして FTP クライアント API の引数として渡します。

デバイス番号

デバイス番号には FTP クライアントで使用するネットワークデバイスを指定します。'0' を指定した場合はデフォルトのネットワークデバイスが使用されます。(通常は 0 をセットしてください。)

制御用ポートのソケット ID、データ用ポートのソケット ID

FTP クライアント用に作成した TCP ソケットの ID を指定してください。

制御用ポート番号、データ用ポート番号

接続先 FTP サーバーの制御用ポート番号(20)、データ用ポート番号(21)を任意の番号に変更する場合はこちらの値を変更して下さい。(通常は 0 をセットしてください。)

転送モード

FTP の転送モードを指定します。パッシブモード (FTPC_MODE_PASSIVE)、アクティブモード (FTPC_MODE_ACTIVE) のいずれかを指定します。また、μ Net3/SSL をお使いの場合は、SSL の使用 (FTPC_MODE_SSL_USE) を指定で FTPS が使えます。併せてサーバ証明書の CA の署名を検証する (FTPC_MODE_SSL_VFY)、FTPS Implicit モードで通信を行う (FTPC_MODE_SSL_IMP) の指定が可能です。

転送タイプ

ファイルの転送タイプを指定します。アスキー転送 (FTPC_TYPE_ASCII)、バイナリ転送 (FTPC_TYPE_BINARY) が指定出来ます。アスキー転送が機能するかは使用するファイルシステムに依存します。

コマンドタイムアウト

FTP クライアントのタイムアウトを設定します。未設定(0)の場合は、制御用ポートのソケット ID に設定された接続タイムアウト値が適用されます。

T_FTP_LIST (LIST/NLST コマンド結果格納構造体)

```
typedef struct t_ftp_list {
    VB          *buf;          /* コマンド結果格納バッファ */
    UW          len;           /* バッファサイズ */
    UB          nameonly;      /* ファイル名のみ取得フラグ */
    UW          next_pos;      /* (制御用) 次処理のバッファ位置 */
    VP          dat[];         /* (制御用) 制御用データ */
} T_FTP_LIST;
```

この構造体に必要な情報をセットして ftp_cmd_list API の引数として渡します。

コマンド結果格納バッファ、バッファサイズ

LIST、NLST コマンドの結果が入るバッファを指定します。ftp_cmd_list、ftp_cmd_list_next は関数コール毎に LIST、NLST コマンドを発行します。無駄な通信を抑えるために、バッファサイズは適宜調整して下さい。

ファイル名のみ取得フラグ

0 で LIST コマンドを発行、それ以外で NLST コマンドを発行します。

FTP 転送モードの定義値

```
#define FTPC_MODE_ACTIVE    1        /* アクティブモード */
#define FTPC_MODE_PASSIVE   0        /* パッシブモード */

#define FTPC_MODE_SSL_USE    0x80     /* FTPS 使用 */
#define FTPC_MODE_SSL_VFY    0x40     /* SSL 証明書の検証有効 */
#define FTPC_MODE_SSL_IMP    0x20     /* Implicit モードを使用 */
```

T_FTP_CLIENT::mode に入る定義値

FTP 転送タイプの定義値

```
#define FTPC_TYPE_BINARY    'T'       /* バイナリ転送 */
#define FTPC_TYPE_ASCII     'A'       /* アスキー転送 */
```

T_FTP_CLIENT::type に入る定義値

コンフィグレーション定義 (ftp_client_cfg.h)

設定例

```
#include "ffsys.h"        /* File system */

#define FTPC_SSL_SUP      /* Enable FTPS */
```

FTPC_SSL_SUP

定義有で FTPS 対応となります。(別途 μ Net3/SSL が必要となります)

1 3 FTP サーバー (Cmp/Std/Pro)

1 3. 1 機能仕様

FTP サーバーはリモートホストに対してファイルのアップロードとダウンロードを可能にします。モードはアクティブモードとパッシブモードが使用できます。

1 3. 2 制限事項

- IPv6、SSL 上での動作は対応していません。(IPv4 のみ)
- ファイルを扱うにはファイルシステムが必要です。

1 3. 3 OS/ネットワーク資源

FTP サーバーでは制御用とデータ用の 2 つの TCP ソケットが必要になります。ソケットのパラメータは以下の通りです。(通常コンフィグレータが自動生成します)

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_FTP_CTL	<u><i>n</i></u>	v4	TCP	21	-	-	-	-	-	-
ID_SOC_FTP_DATA	<u><i>n</i></u>	v4	TCP	20	-	-	-	-	-	-

※ *n*にはネットワークデバイス番号が入ります。デバイスが 1 つしか存在しない場合、*n* = 1 となります。

13.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→FTP Server の画面で FTP サーバーのコンフィグレーションを行います。

※当アプリにはファイルシステムが必要です。「12.4 コンフィグレーション画面」最下部参照。

DHCPv4 Server	DHCP Client Ext	DNS Client	HTTP Server	HTTP Client	SMTP User Agent	POP3 Client
FTP Server	FTP Client	TFTP Server	TFTP Client	SNTP Server	SNTP Client	Telnet Server
Ping	File system					

☒ 有効にする

セッション最大数

ルートディレクトリ

ログインユーザ

User Name	Password	
User	Password	

タイムアウト設定

制御用ソケットの送信,切断(ミリ秒)

制御用ソケットの受信(ミリ秒)

データ用ソケットの接続,送信,受信,切断(秒)

セキュリティポリシー

☐ PORTコマンドを拒否

☐ well-knownポート接続拒否

有効にする

FTP サーバーアプリがソース生成対象となります。

セッション最大数

FTP サーバーへ接続するセッションの上限を指定します。

ルートディレクトリ

FTP サーバーが参照するドライブレターを指定します。

ログインユーザー一覧

FTP サーバーにログイン可能なユーザー一覧が表示されます。

追加

ログイン可能なユーザーを新規追加します。

削除

ログインユーザー一覧リストから選択されているユーザーを削除します。

タイムアウト設定

FTP サーバーの各処理におけるタイムアウトの値を指定します。通常デフォルト設定で問題ありません。

PORT コマンドを拒否

PORT コマンドを拒否します。バウンス攻撃 (Bounce Attack) などのセキュリティ対策を行う場合に使用します。

Well-known ポート接続拒否

Well-known ポート (～1023) からの接続を拒否します。セキュリティ対策を行う場合に使用します。

1 3. 5 API

ftp_server (開始)

【書式】

```
ER ercd = ftp_server(T_TFTP_SERVER *tftp);
```

【パラメータ】

T_TFTP_SERVER	*tftp	FTP サーバー制御情報
---------------	-------	--------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。(*tftp が NULL。 ctl_sid か dat_sid が指定されていない。)
E_RLWAI	FTP サーバーの停止要求が実行された。

【解説】

この API は FTP サーバーを初期化し、FTP クライアントからの要求を受け付けて処理します。この API はブロッキング呼出しになっていますので、専用タスクを用意しそのタスクから呼びだすようしてください。

ftp_server_stop (停止)**【書式】**

```
ER ercd = ftp_server_stop( UW retry );
```

【パラメータ】

UW	retry	サーバータスク停止処理のリトライ回数
----	-------	--------------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_TMOUT	サーバータスク停止処理のリトライオーバーが発生した。
---------	----------------------------

【解説】

この API は起動している FTP サーバーを停止します。この API が実行された場合、FTP サーバーの全タスクの停止処理を実行します。

FTP サーバーのタスク情報は FTP サーバー起動 API (ftp_server) 実行時に自動で内部メモリに保持されます。

なお、この API は実行時に指定されたリトライ回数 (retry) のリトライオーバーが発生した場合、戻り値として E_TMOUT が返却されます。

【推奨】

この API で指定するリトライ回数 (retry) の値は「5」以上である事を推奨します。例えばリトライ回数に「0」を指定して実行した場合、この API が失敗する確率がより高くなります。

1 3 . 6 設定用マクロ/構造体/定数

T_FTP_SERVER (FTP サーバー制御情報)

```
typedef struct t_ftp_server {
    .
    UW      sec;          /* セキュリティポリシー */
    .
    .
    .
    UH      dev_num;      /* デバイス番号*/
    UH      ctl_sid;      /* 制御用ソケット ID */
    UH      dat_sid;      /* データ用ソケット ID */
    .
    .
    .
} T_FTP_SERVER;
```

この構造体に必要な情報をセットして FTP サーバーAPI の引数として渡します。

セキュリティポリシー

セキュリティポリシーにはセキュリティ設定を行います。「0」を設定した場合はセキュリティ機能を使用しません。「ENA_DENY_PORTCOMMAND」を設定した場合は PORT コマンドの拒否機能が適用されます。「ENA_NOTCON_WELL_KNOWNPORT」を設定した場合は Well-known ポートからの接続拒否が適用されます。

なお、PORT コマンドの拒否機能を適用した場合、必然的に Well-known ポートでの接続は行われません。従って、「ENA_DENY_PORTCOMMAND」を設定した場合は「ENA_NOTCON_WELL_KNOWNPORT」の機能も自動的に有効となります。

ENA_DENY_PORTCOMMAND

PORT コマンド機能を拒否

ENA_NOTCON_WELL_KNOWNPORT

Well-known ポート接続を拒否

デバイス番号

デバイス番号には FTP サーバーで使用するネットワークデバイスを指定します。'0' (DEV_ANY) を指定した場合はデフォルトのネットワークデバイスが使用されます。(通常は 0 をセットしてください。)

制御用ソケット ID、データ用ソケット ID

FTP サーバー用に作成した TCP ソケットの ID を指定してください。

コンフィグレーション定義 (ftp_server_cfg.h)

設定例

```
/* FTP Server Configurables */
#include "net_cfg.h"      /* Network */
#include "ffsys.h"       /* File system */

/* Configuration */
#define CFG_FTPS_NET_SOC_MAX    CFG_NET_SOC_MAX /* Number of sockets */
#define CFG_FTPS_CMD_TMO       5000           /* 5 sec */
#define CFG_FTPS_IDLE_TMO      (1 * 60 * 1000) /* 1 minute */
#define CFG_FTPS_DAT_TMO       5000           /* 5 sec */

#define CFG_FTPS_DRV_NAME      'C'             /* Drive name */
#define CFG_FTPS_PATH_MAX      PATH_MAX        /* Max. length of file path */
```

CFG_FTPS_NET_SOC_MAX

マクロの CFG_NET_SOC_MAX を指定します。CFG_NET_SOC_MAX (net_cfg.h) は使用するソケットの最大個数です。

CFG_FTPS_CMD_TMO

制御用ソケットの送信と切断のタイムアウト (msec) の値を指定します。

CFG_FTPS_IDLE_TMO

制御用ソケットの受信のタイムアウト (msec) の値を指定します。

CFG_FTPS_DAT_TMO

データ用ソケットの接続、送信、受信、切断のタイムアウト (msec) の値を指定します。

CFG_FTPS_DRV_NAME

ファイルシステムで使用するドライブ名 ('A'や'C'など) を指定します。FTP サーバーはファイルを開く場合、本マクロのドライブ名を使用します。

CFG_FTPS_PATH_MAX

ファイルシステムのファイルパスの最大長を指定します。ここでのパス長はルートのディレクトリ名+ファイル名の文字列の長さを表します。

アカウント設定 (ftp_server_cfg.c)

設定例

```
/* FTP Server Account */
#include "kernel.h"
#include "net_hdr.h"
#include "ftp_server.h"

/* Login user table (Max. 256 users) (DEV_ANY: All device is allowed) */
const T_FTP_USR_TBL ftp_usr_tbl[] = {
    {DEV_ANY, "", ""}, /* Anyone can login (No user name,password) */
    {DEV_ANY, "User", "Password"},

    {0x00, 0x00, 0x00} /* Terminate mark (Do not change) */
};
```

ファイルの ftp_server_cfg.c にはアカウントを設定する構造体の配列変数を宣言してください。アカウントを設定する構造体は次のようになります。

```
typedef struct t_ftp_usr_tbl {
    UH      dev_num; /* デバイス番号*/
    VB*     usr;      /* ユーザー名 */
    VB*     pwd;      /* パスワード */
} T_FTP_USR_TBL;
```

構造体の dev_num はアカウントの認証時に使用するネットワークデバイスの番号を指定します。0 (DEV_ANY) を指定した場合は全てのネットワークデバイスで認証を行います。一般に dev_num には 0 を代入してください。usr はユーザー名を文字列で指定します。pwd はパスワードを文字列で指定します。

この構造体の配列変数を ftp_usr_tbl の変数名で宣言してください。配列の最後の要素は終端用のデータです。終端用のデータはすべての変数に 0 を代入してください。ユーザー名とパスワードは最大で 256 個まで登録が可能です。

なお、上記の設定例の「{DEV_ANY, "", ""}」はユーザー名とパスワードの入力を省いても FTP サーバーにログインできる設定です。よって、テストの目的以外では登録しないでください。

1 4 TFTP クライアント (Pro)

1 4. 1 機能仕様

TFTP クライアントは、TFTP サーバーに対してファイルのアップロードとダウンロードを可能にします。FTP に対して、TFTP は UDP を使用した簡易プロトコルです。

1 4. 2 制限事項

- IPv6、SSL 上での動作は対応していません。(IPv4 のみ)
- ファイルを扱うにはファイルシステムが必要です。
- ファイル転送モード”mail”はサポートしていません。

1 4. 3 OS/ネットワーク資源

TFTP クライアントではリクエスト・データ共用の 1 つの UDP ソケットが必要になります。ソケットのパラメータは以下の通りです。(通常コンフィグレータが自動生成します)

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_TFTP	0	v4	UDP	0	5 秒	5 秒	-	-	-	-

14.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→TFTP Client の画面で TFTP クライアントのコンフィグレーションを行います。

※当アプリにはファイルシステムが必要です。「12.4 コンフィグレーション画面」最下部参照。

DHCPv4 Server	DHCP Client Ext	DNS Client	HTTP Server	HTTP Client	SMTP User Agent	FTP Server
FTP Client	TFTP Server	TFTP Client	SNTP Server	SNTP Client	Telnet Server	Ping
File system	SNMP					

☒ 有効にする

通信リトライ回数

有効にする

TFTP クライアントアプリがソース生成対象となります。

通信リトライ回数

TFTP データ通信中に異常が発生した際に再試行する回数を指定します。

1 4. 5 API

tftp_get_file（ファイル取得）

【書式】

```
ER ercd = tftp_get_file(T_TFTP_CLIENT *tftp,
                        const VB *lo_file, const VB *rmt_file);
```

【パラメータ】

T_TFTP_CLIENT	* tftp	TFTP クライアント制御情報
const VB	*lo_file	ローカルファイルパス
const VB	*rmt_file	TFTP サーバー側ファイルパス

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OACV	ファイルのオープンに失敗した。
E_NOMEM	ファイルの書込みに失敗した。
E_NOSPT	TFTP の通信手順に異常が生じた。(ローカル)
E_SYS	TFTP の通信手順に異常が生じた。(リモート)
その他	上記エラーの他、rcv_soc, snd_soc のエラーが生じる。

【解説】

この API は引数で設定した TFTP サーバーからファイルを取得します。TFTP クライアント制御情報にサーバーの接続情報 (IPv4 アドレス、ポート番号)、ファイル転送モードを指定して下さい。TFTP クライアントでは UDP ソケットを使用します。使用可能なソケット ID を引数に設定して下さい。

ファイル取得ができた場合は E_OK を返却します。

tftp_put_file（ファイル設置）**【書式】**

```
ER ercd = tftp_put_file(T_TFTP_CLIENT *tftp,
                        const VB *lo_file, const VB *rmt_file);
```

【パラメータ】

T_TFTP_CLIENT	* tftp	TFTP クライアント制御情報
const VB	*lo_file	ローカルファイルパス
const VB	*rmt_file	TFTP サーバー側ファイルパス

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_OACV	ファイルのオープンに失敗した。
E_NOMEM	ファイルの書込みに失敗した。
E_NOSPT	TFTP の通信手順に異常が生じた。(ローカル)
E_SYS	TFTP の通信手順に異常が生じた。(リモート)
その他	上記エラーの他、rcv_soc, snd_soc のエラーが生じる。

【解説】

この API は引数で設定した TFTP サーバーへファイルを設置します。TFTP クライアント制御情報にサーバーの接続情報 (IPv4 アドレス、ポート番号)、ファイル転送モードを指定して下さい。TFTP クライアントでは UDP ソケットを使用します。使用可能なソケット ID を引数に設定して下さい。

ファイル設置ができた場合は E_OK を返却します。

1 4 . 6 設定用マクロ/構造体/定数

T_TFTP_CLIENT (TFTP クライアント制御情報)

```
typedef struct t_tftp_client {
    UH      dev_num;      /* デバイス番号 */
    UH      ver;
    UH      data_sid;     /* ソケット ID */
    UB      ascii;        /* 転送モード(0:バイナリ、0 以外:テキスト) */
    T_NODE   rmt;          /* リモート接続情報 (制御用) */
} T_TFTP_CLIENT;
```

この構造体に必要な情報をセットして TFTP クライアント API の引数として渡します。

デバイス番号

デバイス番号には TFTP クライアントで使用するネットワークデバイスを指定します。'0'を指定した場合はデフォルトのネットワークデバイスが使用されます。(通常は0をセットしてください。)

ソケット ID

TFTP クライアント用に作成した UDP ソケットの ID を指定してください。

転送モード

ファイルの転送モードを指定します。内部的には `fopen` でファイルを開く際の第 2 引数の `'b'` 付きの有無です。

コンフィグレーション定義 (tftp_client_cfg.h)

設定例

```
#include "ffsys.h"          /* File system */

/* TFTP Configurables */
#define TFTP_RETRY_CNT      3      /* TFTP Communication retries */
```

TFTP_RETRY_CNT

ファイル転送時の通信リトライ回数を指定します。

1 5 TFTP サーバー (Pro)

1 5. 1 機能仕様

TFTP サーバーは、リモートホストに対してファイルのアップロードとダウンロードを可能にします。FTP に対して、TFTP は UDP を使用した簡易プロトコルです。

1 5. 2 制限事項

- IPv6、SSL 上での動作は対応していません。(IPv4 のみ)
- ファイルを扱うにはファイルシステムが必要です。
- 複数クライアントからの同時要求は対応していません。
- ファイル転送モード”mail”はサポートしていません。

1 5. 3 OS/ネットワーク資源

TFTP サーバーではリクエスト用とデータ用の 2 つの UDP ソケットが必要になります。ソケットのパラメータは以下の通りです。(通常コンフィグレータが自動生成します)

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_TFTP_REQ <i>n</i>	<i>n</i>	v4	UDP	69	-1	-1	-	-	-	-
ID_SOC_TFTP_DATA <i>n</i>	<i>n</i>	v4	UDP	0	5 秒	5 秒	-	-	-	-

※ ***n***にはネットワークデバイス番号が入ります。デバイスが 1 つしか存在しない場合、***n*** = 1 となります。

15.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→TFTP Server の画面で TFTP サーバーのコンフィグレーションを行います。

※当アプリにはファイルシステムが必要です。「12.4 コンフィグレーション画面」最下部参照。

有効にする

TFTP サーバーアプリがソース生成対象となります。

ルートディレクトリ

TFTP サーバーが参照するドライブレターを指定します。

最大パスサイズ

ファイルパスの最大長を指定します。

通信リトライ回数

TFTP データ通信中に異常が発生した際に再試行する回数を指定します。

追加

ネットワークインタフェースに対応したサーバを追加します。

削除

登録サーバー一覧リストから選択されているサーバー対応を削除します。

15.5 API

tftp_server（サーバーの開始）

【書式】

```
ER ercd = tftp_server(T_TFTP_SERVER *tftp);
```

【パラメータ】

T_TFTP_SERVER	* tftp	TFTP サーバー制御情報
---------------	--------	---------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。(*tftp が NULL。 req_sid か dat_sid が指定されていない。)
-------	--

【解説】

この API は TFTP サーバーを初期化し TFTP クライアントからの要求を受け付け処理します。この API はブロッキング呼出しになっていますので、専用タスクを用意しそこから呼び出すようにしてください。

15.6 設定用マクロ/構造体/定数

T_TFTP_SERVER (TFTP サーバー制御情報)

```
typedef struct t_tftp_server {
    UH      dev_num;      /* デバイス番号 */
    UH      req_sid;      /* リクエスト用ソケット ID*/
    UH      dat_sid;      /* データ用ソケット ID */
    T_NODE  rmt;          /* リモート接続情報 (制御用) */
} T_TFTP_SERVER;
```

この構造体に必要な情報をセットして TFTP サーバーAPI の引数として渡します。

デバイス番号

デバイス番号には TFTP サーバーで使用するネットワークデバイスを指定します。'0' を指定した場合はデフォルトのネットワークデバイスが使用されます。(通常は0をセットしてください。)

リクエスト用ソケット ID、データ用ソケット ID

TFTP サーバー用に作成した UDP ソケットの ID を指定してください。

コンフィグレーション定義 (tftp_server_cfg.h)

```
設定例
#include "ffsys.h"      /* File system */

/* TFTP Configurables */
#define TFTP_RETRY_CNT  3      /* TFTP Communication retries */

#define TFTP_ROOT_DIR   "A:¥¥" /* TFTP Root Directory */
#define TFTP_FILEPATH_MAX 64   /* File path max size */
```

TFTP_RETRY_CNT

ファイル転送時の通信リトライ回数を指定します。

TFTP_ROOT_DIR

TFTP サーバーのルートディレクトリを指定します。

TFTP_FILEPATH_MAX

ファイルパスの最大長を指定します。ここでのパス長は、ルートのディレクトリ名+ファイル名の文字列の長さを表します。

16 SNTP クライアント (Cmp/Std/Pro)

16.1 機能仕様

SNTP クライアントは NTP パケットを利用してネットワーク上の時刻サーバー (NTP サーバー) から NTP 時刻 (1900/1/1 を起点とした秒数) を取得します。

16.2 制限事項

- IPv6、SSL 上での動作は対応していません。(IPv4 のみ)

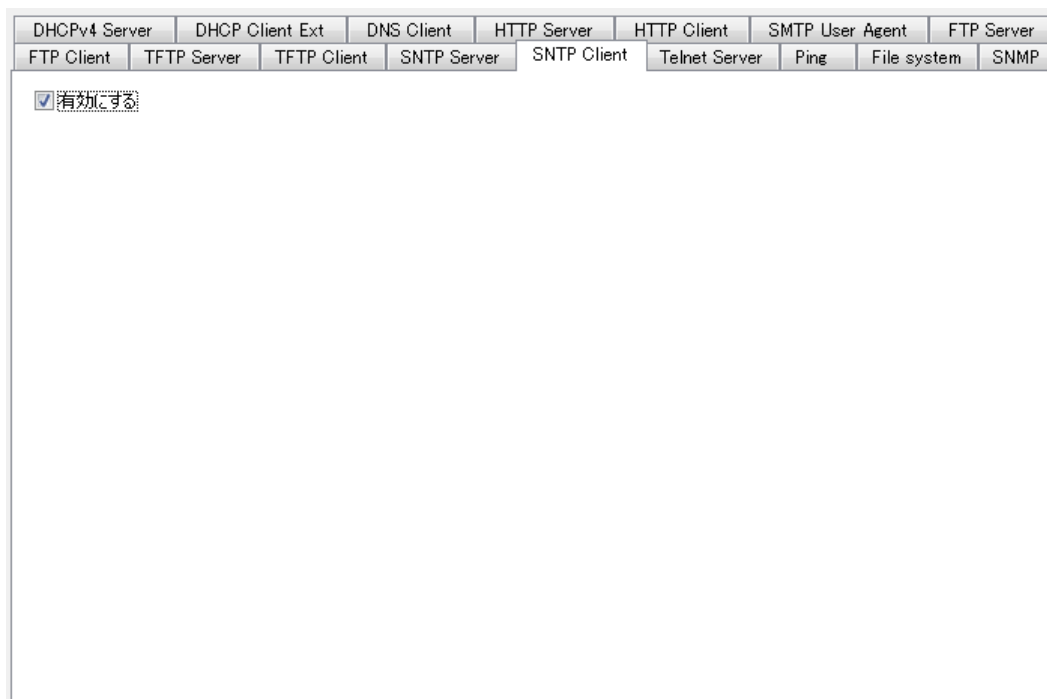
16.3 OS/ネットワーク資源

SNTP クライアントでは 1 つの UDP ソケットが必要になります。ソケットのパラメータは以下の通りです。(通常コンフィグレータが自動生成します)

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_SOC_SNTPC	0	v4	UDP	0	0	0	-	-	-	-

16.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→SNTP Client の画面で SNTP クライアントのコンフィグレーションを行います。



有効にする

SNTP クライアントアプリがソース生成対象となります。

16.5 API

sntp_client (NTP 時刻の取得)

【書式】

```
ER ercd = sntp_client(T_SNTP_CLIENT *sntp_client, UW *sec, UW *fra);
```

【パラメータ】

T_SNTP_CLIENT	*sntp_client	SNTP クライアント情報
UW	*sec	NTP 時刻 (秒数)
UW	*fra	NTP 時刻 (秒端数)

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	不正なパラメータが指定された。
E_TMOUT	アドレス解決失敗、宛先からの応答なし。
E_NOMEM	メモリエラー。
E_OBJ	不正な SNTP クライアント情報。

【解説】

このAPIは引数で設定したSNTPサーバーからNTP時刻を取得します。SNTPサーバーの設定には、IPv4 アドレスとポート番号を指定して下さい。SNTP クライアントではUDPソケットを使用します。使用可能なソケットIDを引数に指定して下さい。

正常にNTP時刻を取得できた場合はE_OKを返却します。このとき引数のsecとfraにはNTP時刻が入ります。NTP時刻は1900/1/1を起点としているため、UTC(JST)やUnix時刻への変換は呼出し元で計算する必要があります。

使用例

```
T_SNTP_CLIENT sc = {0};
UW sec, fra;
ER ercd;

sc.sid = ID_SOC_SNTPC;
ercd = sntp_client(&sc, &sec, &fra);
if (ercd == E_OK) {
    /* UnixTime 変換 */
    sec -= 2208988800;

    /* ミリ秒精度の整数表現 */
    fra = ((fra >> 16) * 1000) >> 16;
}
```

(Pro) SNTP サーバー用の時刻変換 API を使用して各種フォーマットに変換できます。

16.6 設定用マクロ/構造体/定数

T_SNTP_CLIENT (SNTP クライアント制御情報)

```
typedef struct t_sntp_client {
    ID      sid;          /* ソケット ID */
    UW      ipa;          /* SNTP サーバーIP アドレス */
    TMO     tmo;          /* タイムアウト設定 */
    UH      devnum;       /* デバイス番号 */
    UH      port;         /* ポート番号 */
    UB      ipv;          /* IP バージョン */
} T_SNTP_CLIENT;
```

この構造体に必要な情報をセットして SNTP クライアント API の引数として渡します。

デバイス番号

デバイス番号には TFTP クライアントで使用するネットワークデバイスを指定します。'0'を指定した場合はデフォルトのネットワークデバイスが使用されます。(通常は0をセットしてください。)

SNTP サーバーIP アドレス

接続先 SNTP サーバーの IP アドレスを指定してください。

タイムアウト設定

SNTP サーバーからの応答パケットを受信するタイムアウトの時間を指定してください。

ポート番号

SNTP サーバーの接続先ポート番号を変更する際に指定します。通常は0を入れてください。

IP バージョン

IP バージョンを指定します。

1 7 SNTP サーバー (Pro)

1 7. 1 機能仕様

SNTP (Simple Network Time Protocol) は、TCP/IP ネットワークを介してシステム内の時計の同期を取るプロトコルの 1 つです。

- SNTPv4 準拠の SNTP サーバー (RFC2030、一部機能制限あり)
- SNTP 応答機能 (ユニキャスト、エニーキャストモード)
- 複数タイムフォーマットによる時刻補正機能
- SNTP サーバー使用ポート番号変更機能
- SNTP マルチキャスト送信機能

詳細は「μ Net3/SNTPS ユーザーズガイド」を参照してください。

1 7. 2 制限事項

- IPv4 UDP ソケットのみサポート (IPv4/v6 TCP、IPv6 UDP は未サポート)
- NTP サーバー機能である、認証や IP フィルタは未サポート
- SNTP クライアントからのマルチキャスト要求の受信は未サポート
- SNTP サーバー時刻の精度は OS のチック時間 (最少 1msec)

1 7. 3 OS/ネットワーク資源

詳細は「μ Net3/SNTPS ユーザーズガイド」を参照してください。

17.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→SNTP Server の画面で SNTP サーバーのコンフィグレーションを行います。

☒ 有効にする

NTPタイムスタンプの時刻範囲の設定

☐ 1900/01/01 00:00:00 - 2036/02/07 06:28:15

☒ 1968/01/20 03:14:08 - 2104/02/26 09:42:23

☒ SNTP/パケットのマルチキャスト送信機能を有効にする

☐ ブロードキャスト宛て(255.255.255.255)

☒ マルチキャスト宛て(224.0.1.1)

登録サーバー一覧

ネットワークインターフェース	
ID_NETIF_DEV1	

追加

削除

有効にする

SNTP サーバーアプリがソース生成対象となります。

NTP タイムスタンプの時刻範囲の設定

SNTP サーバーの時刻範囲を指定します。

SNTP パケットのマルチキャスト送信機能を有効にする

マルチキャスト送信機能が使用可能になります。

ブロードキャスト宛て/マルチキャスト宛て

マルチキャスト送信の宛て先を指定します。(ブロードキャスト or マルチキャスト)

追加

ネットワークインターフェースに対応したサーバを追加します。

削除

登録サーバー一覧リストから選択されているサーバー対応を削除します。

17.5 API

API 一覧

詳細は「μ Net3/SNTPS ユーザーズガイド」を参照してください。

No	API	概要説明	備考
	書式		
1	sntp_server	開始	
	ER sntp_server(T_SNTP_SERVER *sntp);		
2	sntp_set_tim	サーバー時刻の設定	
	ER sntp_set_tim(const T_NTP_TIM *ntm); /* NTP */		
	ER sntp_set_tim_UnixTime(const W *ut); /* UnixTime */		
	ER sntp_set_tim_TM(const T_TM_SIMPLE *tm); /* Date(Original Type) */		
	ER sntp_set_tim_Str(VB const *str, UB type); /* String */		
3	sntp_get_tim	サーバー時刻の参照	
	ER sntp_get_tim(T_NTP_TIM *ntm); /* NTP */		
	ER sntp_get_tim_UnixTime(W *ut); /* UnixTime */		
	ER sntp_get_tim_TM(T_TM_SIMPLE *tm); /* Date */		
	ER sntp_get_tim_Str(VB *str, UB type); /* String */		
4	sntp_set_port	ポート番号設定	
	ER sntp_set_port(UB dev_num, UH port);		
5	sntp_get_port	ポート番号参照	
	ER sntp_get_port(UB dev_num, UH *port);		
6	sntp_get_err	エラー情報参照	
	ER sntp_get_err(UB dev_num, UB *err);		
7	sntp_clr_err	エラー情報クリア	
	ER sntp_clr_err(UB dev_num);		
8	sntp_mc_send	マルチキャスト送信の開始	
	ER sntp_mc_send(UB dev_num, UB interval, UW send_cnt);		
9	sntp_mc_stop	マルチキャスト送信の停止	
	ER sntp_mc_stop(UB dev_num);		
10	cnv_xxx_to_xxx	時刻変換 (変換元→変換先)	
	ER cnv_ut_to_ntp(T_NTP_TIM *ntp, const W *ut); /* UnixTime → NTP */		
	ER cnv_ntp_to_ut(W *ut, const T_NTP_TIM *ntp); /* NTP → UnixTime */		
	ER cnv_tm_to_ntp(T_NTP_TIM *ntp, const T_TM_SIMPLE *tm); /* Date → NTP */		
	ER cnv_ntp_to_tm(T_TM_SIMPLE *tm, const T_NTP_TIM *ntp); /* NTP → Date */		
	ER cnv_str_to_tm(T_TM_SIMPLE *tm, VB const *str, UB type); /* String → Date */		
	ER cnv_tm_to_str(VB *str, const T_TM_SIMPLE *tm, UB type); /* Date → String */		
11	chk_tm_val	時刻有効性チェック (Date)	
	ER chk_tm_val(const T_TM_SIMPLE *tm);		

17.6 設定用マクロ/構造体/定数

詳細は「μ Net3/SNTPS ユーザーズガイド」を参照してください。

1 8 Ping クライアント (Cmp/Std/Pro)

1 8. 1 機能仕様

Ping クライアントは任意の宛先に対して、ICMP エコー要求を送信します。相手先からエコー応答があれば IP アドレスでの通信が可能であることが分かります。

1 8. 2 制限事項

1 8. 3 OS/ネットワーク資源

Ping クライアントでは 1 つの ICMP ソケットが必要になります。ソケットのパラメータは以下の通りです。(通常コンフィグレータが自動生成します)

ID	デバイス 番号	IP ver	プロト コル	ポート	タイムアウト				バッファ	
					送信	受信	接続	切断	送信	受信
ID_ICMP	0	v4	ICMP	0	-	-	-	-	-	-

ネットワークの ICMP 用のソケットを 1 個作成してください。

本機能を使用する場合はコンフィグレータの TCP/IP タグのネットアプリケーションの Ping で「ICMP Echo 要求を使用する」を有効にしてください。有効にすると関数の ping_client で使用するソケット (ID : “ID_ICMP”) が作成されます。

Standard 版の OS 上で本機能を使用する場合は、次のように cre_soc でソケットを作成してください。ポート番号は 0 にします。作成したソケットの ID を構造体の T_PING_CLIENT の sid に代入してください。

```
T_NODE node;

node.num = 1;          /* ネットワークのデバイス番号 */
node.ipa = INADDR_ANY;
node.port = 0;         /* Port should 0 for ICMP */
node.ver = IP_VER4;
sid = soc_cre(IP_PROTO_ICMP, & node);
if (sid <= 0) {
    return E_NOMEM;
}
```

18.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→Ping の画面で Ping クライアントのコンフィグレーションを行います。

DHCPv4 Server	DHCP Client Ext	DNS Client	HTTP Server	HTTP Client	SMTP User Agent	FTP Server
FTP Client	TFTP Server	TFTP Client	SNTP Server	SNTP Client	Telnet Server	Ping
File system						
SNMP						

☒ ICMP Echo 要求を使用する

ICMP Echo 要求を使用する

Ping クライアントアプリがソース生成対象となります。

18.5 API

ping_client (送信)

【書式】

```
ER ping_client(T_PING_CLIENT *ping_client);
```

【パラメータ】

T_PING_CLIENT	*ping_client	構造体の T_PING_CLIENT のポインタ
---------------	--------------	--------------------------

【戻り値】

ER	ercd	正常終了 (E_OK) またはエラーコード
----	------	-----------------------

【エラーコード】

E_PAR	引数の誤り
E_TMOUT	アドレス解決の失敗、または、応答無し
E_NOMEM	メモリ不足
E_OBJ	その他のエラー

【解説】

宛先の IP アドレスへ ping を送信します。本関数は送信先から応答が返るまで待ち続けます。また、送信先から応答が無い場合はタイムアウト (E_TMOUT) のエラーを返します。

18.6 設定用マクロ/構造体/定数

T_PING_CLIENT (PING クライアント制御情報)

```
typedef struct t_ping_client {  
    ID        sid;          /* ICMP ソケット ID */  
    UW        ipa;          /* 宛先 IP アドレス */  
    TMO       tmo;          /* 応答待ちタイムアウト (ms) */  
    UH        devnum;       /* デバイス番号 */  
    UH        len;          /* パケットサイズ (バイト) */  
} T_PING_CLIENT;
```

この構造体に必要な情報をセットして PING クライアント API の引数として渡します。

1 9 SNMP エージェント (Pro)

1 9. 1 機能仕様

SNMP (Simple Network Management Protocol) は、TCP/IP ネットワークを介してルータやコンピュータなどのノードをネットワーク経由で監視・制御するためのプロトコルです。μNet3 - SNMP は μNet3 (TCP/IP プロトコルスタック) 上に SNMP のエージェントの機能を提供するソフトウェアです。

1 9. 2 制限事項

- IPv4 のみサポート (IPv6 は未サポート)
- SNMP の MIB-II のオブジェクトに対応していますが、すべてのオブジェクトには対応していません。
- ベンダー独自のプライベート MIB で使用可能な MIB のオブジェクトのデータ型は Integer/Counter(32)/Gauge(32)/Time Ticks/IP Address/Octet String (文字列) のみです。それ以外のデータ型には対応していません。
- ベンダー独自のプライベート MIB のオブジェクトのツリーは動作中に変更することはできません。つまり、機器の動作中に MIB のツリーにオブジェクト (ノード) を追加/削除することはできません。
- PPP のインターフェースには対応していません。イーサネットのインターフェースに対応しています。
- イーサネットの複数ポートには対応していません。

詳細は「μ Net3 - SNMP ユーザーズガイド」を参照してください。

1 9. 3 OS/ネットワーク資源

詳細は「μ Net3 - SNMP ユーザーズガイド」を参照してください。

19.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→SNMP の画面で SNMP のコンフィグレーションを行います。

DHCPv4 Server	DHCP Client Ext	DNS Client	HTTP Server	HTTP Client	SMTP User Agent	FTP Server
FTP Client	TFTP Server	TFTP Client	SNTP Server	SNTP Client	Telnet Server	Ping
File system	SNMP					

☒ 有効にする

コミュニティ設定		標準トラップ送信先		MIBシステム設定	
SNMP基本設定		標準トラップの選択	MIB2グループの選択	マネージャー設定	
ネットワークインタフェースID	ID_NETIF_DEV1				
同時に送信するトラップ/インフォームの最大個数 (0の場合はトラップを使用しない)	12				
Variable-bindingsのデータの最大個数	32				
MIBのツリーのノードの最大個数	150				
MIBのツリーのノードの最大の深さ (オブジェクトIDのドット区切りの数字の最大個数)	32				
MIBのオブジェクトのデータの最大サイズ(byte)	64				
SNMPのメッセージ最大サイズ(byte)	2048				
コールバック関数	apl_snmp_cbk_0				

有効にする

SNMP アプリがソース生成対象となります。

【SNMP 基本設定】

コミュニティ設定		標準トラップ送信先		MIBシステム設定	
SNMP基本設定		標準トラップの選択	MIB2グループの選択	マネージャー設定	
ネットワークインタフェースID	ID_NETIF_DEV1				
同時に送信するトラップ/インフォームの最大個数 (0の場合はトラップを使用しない)	12				
Variable-bindingsのデータの最大個数	32				
MIBのツリーのノードの最大個数	150				
MIBのツリーのノードの最大の深さ (オブジェクトIDのドット区切りの数字の最大個数)	32				
MIBのオブジェクトのデータの最大サイズ(byte)	64				
SNMPのメッセージ最大サイズ(byte)	2048				
コールバック関数	apl_snmp_cbk_0				

ネットワークインタフェース ID

SNMP アプリと対応付けるネットワークインタフェースを指定します。μ Net3 システム上のネットワークインタフェース 1 つのみ割り当てることが可能です。

同時に送信するトラップ/インフォームの最大個数

同時に使用するトラップ用資源の最大個数を指定します。トラップを使用しない場合はこの値を 0 にしてください。0 の場合はトラップ用のタスクを起床しません。

Variable-bindings のデータの最大個数

受信/送信する SNMP のパケット内の Variable-binding の個数の最大値を指定します。4 以上の整数を設定してください。受信した SNMP の v1 のパケットに最大値を超えた個数の Variable-binding が含まれていた場合、本システムはパケットの送り先にエラー (tooBig) を返します。

MIB のツリーの最大個数

MIB のツリーのノードの数を設定します。

MIB のツリーの最大の深さ

MIB-II とベンダー固有 MIB のツリーの最大の深さを設定します。例えばユーザーがベンダー固有 MIB の OID を “1.3.6.1.4.1.1234.1.2.3.4.5.6.7” と設定した場合は、数字の数が 14 個あるので、値は 14 になります。つまり、OID のドット(.)で区切られた数字の個数を設定してください。

MIB のオブジェクトのデータの最大サイズ (Byte)

MIB のオブジェクトのデータの最大サイズ (バイト) を指定します。SNMP の仕様ではデータの最大サイズはデータの型が Integer の場合は 4 バイト、データ型が Octet String (文字列) の場合は 65535 文字 (バイト) です。

SNMP のメッセージ最大サイズ

SNMP のメッセージの受信と送信の最大サイズをバイト単位で設定します。本システムは rcv_soc (UDP の受信) で SNMP のメッセージを受け取りますが、その rcv_soc の引数に指定するバッファのサイズとなります。

コールバック関数

ベンダーMIB 用の標準コールバック関数を設定します。

【SNMP 標準トラップの選択】

初期化時に有効にする標準トラップを指定します。

コミュニティ設定	標準トラップ送信先	MIBシステム設定
SNMP基本設定	標準トラップの選択	MIB2グループの選択
<div>マネージャー設定</div> <div> <input checked="" type="checkbox"/> coldStart <input checked="" type="checkbox"/> warmStart <input type="checkbox"/> linkDown <input checked="" type="checkbox"/> linkUp <input checked="" type="checkbox"/> authenticationFailure <input type="checkbox"/> egpNeighborLoss </div>		

coldStart

coldStart トラップを有効にします。

warmStart

warmStart トラップを有効にします。

linkUp

linkUp は非対応なので選択できません。

authenticationFailure

authenticationFailure トラップを有効にします。

egpNeighborLoss

egpNeighborLoss は非対応なので選択できません。

【MIB2 グループの選択】

MIB-II の各グループの有効/無効を設定します。

コミュニティ設定		標準トラップ送信先		MIBシステム設定	
SNMP基本設定		標準トラップの選択		MIB2グループの選択	
<input checked="" type="checkbox"/>	Interfaces (1.3.6.1.2.1.2)				
<input type="checkbox"/>	Address trans (1.3.6.1.2.1.3)				
<input type="checkbox"/>	IP (1.3.6.1.2.1.4)				
<input type="checkbox"/>	ICMP (1.3.6.1.2.1.5)				
<input type="checkbox"/>	TCP (1.3.6.1.2.1.6)				
<input type="checkbox"/>	UDP (1.3.6.1.2.1.7)				
<input type="checkbox"/>	SNMP (1.3.6.1.2.1.11)				

Interfaces (1.3.6.1.2.1.2)

Interfaces (1.3.6.1.2.1.2)を有効にします。

Address trans (1.3.6.1.2.1.3)

Address trans (1.3.6.1.2.1.3)を有効にします。

IP (1.3.6.1.2.1.4)

IP (1.3.6.1.2.1.4)を有効にします。

ICMP (1.3.6.1.2.1.5)

ICMP(1.3.6.1.2.1.5)を有効にします。

TCP(1.3.6.1.2.1.6)

TCP(1.3.6.1.2.1.6)を有効にします。

UDP(1.3.6.1.2.1.7)

UDP(1.3.6.1.2.1.7)を有効にします。

SNMP (1.3.6.1.2.1.11)

SNMP(1.3.6.1.2.1.11)を有効にします。

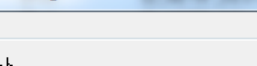
【マネージャー設定】

受信を許可する **SNMP** のメッセージの送信元（マネージャー）を設定します。本設定以外のマネージャーの **SNMP** のパケットは受信しても破棄します。また、マネージャーを制限しないで、すべての **SNMP** のパケットを受信する設定も可能です。

[illegible]

追加ボタン

マネージャーを追加します。ボタンを押下すると下記マネージャー登録画面が表示されます。



The image shows a dialog box titled "SNMP manager". It contains four configuration fields:

- ポート (Port):** A numeric input field with the value "0".
- IPバージョン (IP Version):** A dropdown menu set to "IPv4".
- ネットワークインタフェースID (Network Interface ID):** A dropdown menu set to "ID_NETIF_DEV1".
- IPアドレス (IP Address):** A text field containing "0 . 0 . 0 . 0".

At the bottom of the dialog are two buttons: "OK" and "キャンセル" (Cancel).

受信を許可するマネージャーの IP アドレスを設定します。ネットワークデバイスと port は 0、ver は IP_VER4 を固定されます。

削除ボタン

選択したマネージャーを削除します。

【コミュニティ設定】

SNMP基本設定		標準トラップの選択	MIB2グループの選択	マネージャー設定
コミュニティ設定		標準トラップ送信先		MIBシステム設定
コミュニティ名	アクセスモード			
public	STS_RO			
private	STS_RW			
unet3_mng	STS_RO			

追加... 削除

追加ボタン

コミュニティ追加します。ボタンを押下すると下記コミュニティ登録画面が表示されます。

Community

コミュニティ名:

アクセスモード:
 STS_RO
 STS_RO
 STS_RW

OK キャンセル

コミュニティ名とアクセスモードを設定します。

アクセスモード

STS_RO: 読み込みのみ

STS_RW: 読み書き可能

削除ボタン

選択したコミュニティを削除します。

【標準トラップ送信先】

[illegible]

追加ボタン

標準トラップの送信先を追加します。ボタンを押下すると下記登録画面が表示されます。

標準トラップの送信先

コミュニティ名 unet3_mng

ポート 0

IPバージョン IPv4

ネットワークインタフェースID ID_NETIF_DEV1

IPアドレス 192 . 168 . 10 . 250

トラップバージョン

- SNMP_VER_V1
- SNMP_VER_V1
- SNMP_VER_V2C

OK キャンセル

標準トラップの送信先を設定します。ネットワークデバイスと **port** は0、**ver** はIP_VER4を固定されます。

コミュニティ名

送信先のコミュニティ名を設定します。

IP アドレス

送信先の IP アドレスを設定します。

トラップバージョン

バージョン 1 : SNMP_VER_V1

バージョン 2c : SNMP_VER_V2C

削除ボタン

選択した標準トラップの送信先を削除します。

【MIB システム設定】

ベンダーに依存する MIB-II の System グループを設定します。

SNMP基本設定	標準トラップの選択	MIB2グループの選択	マネージャー設定
コミュニティ設定		標準トラップ送信先	
MIBシステム設定			
sysDescr (1.3.6.1.2.1.1.1) 機器のハードウェア/ソフトウェアの名前やバージョン	<input type="text" value="HwVer.1.0.0 SWVer.1.0.0"/>		
sysObjectID (1.3.6.1.2.1.1.2) ベンダーのオブジェクトID	<input type="text" value="1.3.6.1.4.1.1234"/>		
sysContact (1.3.6.1.2.1.1.4) 機器の管理者の連絡先(メールアドレス)	<input type="text" value="Email address"/>		
sysName (1.3.6.1.2.1.1.5) 機器のドメインネーム	<input type="text" value="Evaluation board"/>		
sysLocation (1.3.6.1.2.1.1.6) 機器の物理的な位置	<input type="text" value="First floor"/>		
sysServices (1.3.6.1.2.1.1.7) 機器が提供するサービスの値	<input type="text" value="64"/>		

sysDescr (1.3.6.1.2.1.1.1)

機器のハードウェア、ソフトウェアの名前やバージョンを設定します。

sysObjectID (1.3.6.1.2.1.1.2)

ベンダーのオブジェクト ID を設定します。

sysContact (1.3.6.1.2.1.1.4)

機器の管理者の連絡先（メールアドレス）を設定します。

sysName (1.3.6.1.2.1.1.5)

機器のドメインネームを設定します。

sysLocation (1.3.6.1.2.1.1.6)

機器の物理的な位置を設定します。

sysServices (1.3.6.1.2.1.1.7)

機器が提供するサービスの値を設定します。

19.5 API

API 一覧

詳細は「μ Net3 - SNMP ユーザーズガイド」を参照してください。

No	API	概要説明	備考
	書式		
1	snmp_ini	初期化	
	ER snmp_ini(UH* mib_nod_cnt);		
2	snmp_ext	終了	
	ER snmp_ext(void);		
3	snmp_ena	有効化	
	ER snmp_ena(void);		
4	snmp_dis	無効化	
	ER snmp_dis(void);		
5	get_mib_obj	オブジェクトのデータの読み込み	
	ER get_mib_obj(VP buf, UH* len, UH mib_id, UH obj_id);		
6	set_mib_obj	オブジェクトのデータの書き出し	
	ER set_mib_obj(VP buf, UH len, UH mib_id, UH obj_id);		
7	ena_trp	標準トラップの有効化	
	ER ena_trp(UH trp_bit);		
8	dis_trp	標準トラップの無効化	
	ER dis_trp(UH trp_bit);		
9	snd_trp	ベンダーのトラップの送信	
	ER snd_trp(T_NODE* nod, T_SNMP_TRP* trp, TMO tmo);		

19.6 設定用マクロ/構造体/定数

詳細は「μ Net3 - SNMP ユーザーズガイド」を参照してください。

2 0 POP3 クライアント (Pro)

2 0. 1 機能仕様

POP3 サーバーからメールの回収を行います。POP3 は TCP を使用したメール受信用プロトコルです。

2 0. 2 制限事項

- IPv6、SSL 上での動作は対応していません。(IPv4 のみ)
- ファイルを扱うにはファイルシステムが必要です。
- 受信したメッセージのサポート API を用意、以下制限事項があります。
 - 文字列デコード処理は BASE64 デコードのみ対応しています。
(quoted-printable デコード処理は未対応、メール受信は可能)
 - ヘッダ情報取得は一般的な項目のみ対応しています。
 - MIME ヘッダ情報取得は Content-Type (複数パラメータは未対応)、Content-Transfer-Encoding のみ対応しています。

※ 詳細は 2.5 サポート API、および 2.6 T POP3 MAIL (POP3 メール情報) を参照

2 0. 3 OS/ネットワーク資源

詳細は「μ Net3/POP3C ユーザーズガイド」を参照してください。

20.4 コンフィグレーション画面

TCP/IP メニュー画面のネットアプリケーション→POP3 Client の画面で POP3 クライアントのコンフィグレーションを行います。

※当アプリにはファイルシステムが必要です。μNet3 付属の簡易ファイルシステムも利用可能です。

FTP Client	TFTP Server	TFTP Client	SNTP Server	SNTP Client	Telnet Server	Ping	File system	SNMP
DHCPv4 Server	DHCP Client Ext	DNS Client	HTTP Server	HTTP Client	SMTP User Agent	POP3 Client	FTP Server	

☒ 有効にする
☒ POP before SMTP 認証を有効にする (SMTP User Agent)

有効にする

POP3 クライアントアプリがソース生成対象となります。

POP before SMTP 認証を有効にする (SMTP User Agent)

SMTP ユーザーエージェントアプリ使用時の認証で「POP before SMTP」認証が選択可能になります。

20.5 API

API 一覧

詳細は「μ Net3/POP3C ユーザーズガイド」を参照してください。

No	API	概要説明	備考
	書式		
1	pop3_login	サーバー接続	
	ER pop3_login(T_POP3_CLIENT *pop3);		
2	pop3_quit	サーバー切断	
	ER pop3_quit(T_POP3_CLIENT *pop3);		
3	pop3_rcv_msg	メッセージ受信	
	ER pop3_rcv_msg(const T_POP3_CLIENT *pop3, UW mid, const T_POP3_FILE *file)		
4	pop3_snd_cmd	コマンド発行	
	ER pop3_snd_cmd(const T_POP3_CLIENT *pop3, const VB *str); ER pop3_snd_cmd_0(const T_POP3_CLIENT *pop3, const VB *cmd); ER pop3_snd_cmd_i(const T_POP3_CLIENT *pop3, const VB *cmd, INT p1); ER pop3_snd_cmd_a(const T_POP3_CLIENT *pop3, const VB *cmd, const VB *p1); ER pop3_snd_cmd_i2(const T_POP3_CLIENT *pop3, const VB *cmd, INT p1, INT p2); ER pop3_snd_cmd_a2(const T_POP3_CLIENT *pop3, const VB *cmd, const VB *p1, const VB *p2);		
5	pop3_mline_next	複数行形式応答の受信	
	ER pop3_mline_next(const T_POP3_CLIENT *pop3);		
6	pop3_mline_eod	複数行形式応答の終了判定	
	ER pop3_mline_eod(const T_POP3_CLIENT *pop3);		
7	pop3_parse_res	応答メッセージの簡易パース	
	H pop3_parse_res_a(const VB *buf, UW *msg, VB *str) H pop3_parse_res_i(const VB *buf, UW *msg, UW *num)		
8	pop3_cnv_mail	メッセージを各メール項目に変換	
	ER pop3_cnv_mail(T_POP3_MAIL *mail, VB *buf)		
9	pop3_cnv_part	ボディをパート情報に変換	
	ER pop3_cnv_part(T_POP3_PART *part, VB *buf, const VB *boundary)		
10	pop3_b64dec_hdr	メールヘッダ BASE64 デコード	
	ER pop3_b64dec_hdr(VB *dst, const VB *src, VB *ctprm);		
11	pop3_b64dec_body	メール本文 BASE64 デコード	
	ER pop3_b64dec_body(VB *dst, const VB *src, UB chk);		

20.6 設定用マクロ/構造体/定数

詳細は「μ Net3/POP3C ユーザーズガイド」を参照してください。

μ Net3 ネットワークアプリケーションガイド

2015 年 6 月 第 12 版

イー・フォース株式会社 <http://www.eforce.co.jp/>

〒103-0014 東京都中央区日本橋富沢町 5-4 ゲンベエビル 7F

TEL 03-5614-6918 FAX 03-5614-6919

お問い合わせ info@eforce.co.jp

Copyright (C) 2015 eForce Co., Ltd. All Rights Reserved.
