
# µNet3 ユーザーズガイド

第１０版　イー・フォース株式会社

# はじめに

**μNet3（マイクロ・ネット・キューブ）**は弊社　リアルタイム. オペレーティング. システム **μC3（マイクロ・シー・キューブ）**向けに実装された TCP/IP プロトコルスタックです。

μNet3 は省メモリでわかりやすく、そして柔軟なインタフェースを実現すべく設計されています。

## 本書の位置づけ

本書は、μNet3/Compact および μNet3/Standard のマニュアルとしており、リアルタイム OS の使用方法などについては「μC3/Compact ユーザーズガイド」および「μC3/Standard ユーザーズガイド」を参照してください。

---



---

## 改訂記録

### 第２版で訂正された項目

| 章 | 内容 |
|---|---|
| 2.1.24, 4.1.3 | MTU ついて更新 |
| 2.1.26, 3.1.1 | IP リアセンブリついて更新 |
| 2.1.21, 5.4 | コールバック関数ついて更新 |
| 4.3 | コンフィグレータで行うネットワーク初期化タスク、MTU、ネットワークバッファ数などの設定ついて更新 |

### 第３版で訂正された項目

| 章 | 内容 |
|---|---|
| 1.3, 4.2, 5.4 | Standard 版の開発手順・コンフィグレーション・API を追加 |
| 2.2.1, 4.1.5, 4.1.6, 5.4 | ソケットに対して任意のネットワークデバイスの関連付けが可能となる対応による更新 |
| 2.3, 4.1.3 | Template ネットワークデバイス追加による更新 |

### 第４版で訂正された項目

| 章 | 内容 |
|---|---|
| 4.2 | コンフィグレーション設定項目の追加に伴う更新 |

### 第５版で訂正された項目

| 章 | 内容 |
|---|---|
| 4.2 | 新コンフィグレータ対応 |
| 6.3 | HTTP サーバー機能追加に伴う更新 |

### 第６版で訂正された項目

| 章 | 内容 |
|---|---|
| 2.3, 6.5, 6.6., 6.7 | ネットワークアプリ追加による修正 |
| 3.2 | ループバックインタフェース追加による修正 |
| 3.1.4 | TCP Keep-Alive 機能追加による修正 |
| 3.1.2, 5.2 | ACD 機能追加による修正 |
| 3.4, 6.8 | String 系標準ライブラリの非使用化による修正 |

### 第７版で訂正された項目

| 章 | 内容 |
|---|---|
| 4.3.1 | uNet3/Standard のコンフィグレーション項目の追加 |

### 第８版で訂正された項目

| 章 | 内容 |
|---|---|
| 5.4 | cre_soc()のソケット最大数オーバー時のエラーコードを E_ID から E_NOMEM に変更 |

### 第９版で訂正された項目

| 章 | 内容 |
|---|---|
| 5.4, 7.3 | エラーコード EV_ADDR についての記載を追加 |
| 6.3 | HTTP サーバー制御情報構造体の誤りを修正 |

### 第１０版で訂正された項目

| 章 | 内容 |
|---|---|
| 6(削除) | μNet3 ネットワークアプリケーションガイドの作成に伴い記載を削除 |
| 2.3.2, 3.4, 4.3.6 | μNet3 ver3, μNet3PRO のリリースに伴う修正 |
| 5.4, 6.3 | エラーコード EV_ADDR についての記載を追加 |
| | 通信パラメータの説明に DEF_XXX_XXX を使用せずに CFG_XXX_XXX を使うように修正 |

# 目次

**はじめに** ..... 2
**目次** ..... 4
**第１章　μNet3 とは** ..... 7
１．１　特長 ..... 7
１．２　主な機能 ..... 7
１．３　開発手順 ..... 7
１．４　サンプルを使ったチュートリアル ..... 11
１．４．１　コンフィグレーションファイルの読み込み ..... 12
１．４．２　コンフィグレータを使った設定 ..... 13
１．４．３　コンフィグレータ設定の保存 ..... 15
１．４．４　ソースコードの生成 ..... 17
１．４．４　プログラムの作成 ..... 20
１．４．５　WEB サーバーの実行 ..... 21
**第２章　μNet3 の基本概念** ..... 23
２．１　用語の意味 ..... 23
２．１．１　プロトコル ..... 23
２．１．２　プロトコルスタック ..... 23
２．１．３　IP(Internet Protocol) アドレス ..... 23
２．１．４　MAC(Media Access Control) アドレス ..... 24
２．１．５　ポート番号 ..... 24
２．１．６　ビックエンディアンとリトルエンディアン ..... 24
２．１．７　パケット ..... 24
２．１．８　ホストとノード ..... 24
２．１．９　Address Resolution Protocol（ARP） ..... 25
２．１．１０　Internet　Protocol（IP） ..... 25
２．１．１１　Internet Control Message Protocol（ICMP） ..... 25
２．１．１２　Internet Group Management Protocol（IGMP） ..... 25
２．１．１３　User Datagram Protocol（UDP） ..... 25
２．１．１４　Transmission Control Protocol（TCP） ..... 25
２．１．１５　Dynamic Host Configuration Protocol（DHCP） ..... 25
２．１．１６　Hyper Text Transfer Protocol（HTTP） ..... 26
２．１．１７　File Transfer Protocol（FTP） ..... 26
２．１．１８　Domain Name System（DNS） ..... 26
２．１．１９　ソケット ..... 26
２．１．２０　ブロッキングとノンブロッキング ..... 26
２．１．２１　コールバック関数 ..... 26

２．１．２２　タスクコンテキスト ..... 27
２．１．２３　リソース ..... 27
２．１．２４　MTU ..... 27
２．１．２５　MSS ..... 27
２．１．２６　IP リアセンブリ・フラグメント ..... 27
２．２　ネットワークシステムのアーキテクチャ ..... 28
２．２．１　ネットワークシステム構成図 ..... 28
２．３　ディレクトリとファイル構成 ..... 30
２．３．１　μNet3 ver.1.xx/μNet3 ver.2.xx のファイル構成 ..... 30
２．３．２　μNet3 ver.3.xx/μNet3 PRO のファイル構成 ..... 32
**第３章　μNet3 の機能概要** ..... 36
３．１　プロトコルスタック ..... 36
３．１．１　IP モジュール ..... 36
３．１．２　ARP モジュール ..... 38
３．１．３　UDP モジュール ..... 38
３．１．４　TCP モジュール ..... 41
３．２　ネットワーク・デバイスドライバ ..... 46
３．２．１　デバイス構造体 ..... 46
３．２．２　インタフェース ..... 48
３．２．３　パケットのルーティング ..... 55
３．２．４　ループバックインタフェース ..... 57
３．３　メモリ管理 ..... 58
３．３．１　ネットワークバッファ ..... 59
３．３．２　ネットワークバッファ API ..... 60
３．４　メモリＩ/O 処理 ..... 62
３．４．１　メモリＩ/O API (uNet3 ver.2.xx) ..... 62
３．４．２　メモリＩ/O API (uNet3 ver.3.xx) ..... 64
**第４章　コンフィグレーション** ..... 66
４．１　μNet3/Compact(μNet3 ver.1.xx)のコンフィグレーション ..... 66
４．１．１　コンフィグレータの起動 ..... 66
Ａ．新規でプロジェクトを生成する場合 ..... 66
Ｂ．既存のプロジェクトを開く場合 ..... 68
Ｃ．メイン画面 ..... 69
４．１．２　TCP/IP プロトコルスタックの設定 ..... 70
４．１．３　全般のコンフィグレーション ..... 71
４．１．４　通信テスト ..... 74
４．１．５　TCP ソケットのコンフィグレーション ..... 75
４．１．６　UDP ソケットのコンフィグレーション ..... 77

４．１．７　アプリケーションのコンフィグレーション ..... 79
４．１．８　プロジェクトファイルの保存 ..... 82
４．１．９　ソース生成 ..... 84
４．２　μNet3/Compact(μNet3 ver.2)のコンフィグレーション ..... 86
４．２．１　コンフィグレータの起動 ..... 86
A．新規でプロジェクトを生成する場合 ..... 86
B．既存のプロジェクトを開く場合 ..... 88
C．メイン画面 ..... 89
４．２．２　TCP/IP プロトコルスタックの設定 ..... 90
４．２．３　μNet3 全般の設定 ..... 90
４．２．４　インタフェースの設定 ..... 97
４．２．５　ソケットの設定 ..... 100
４．１．６　ネットアプリケーションの設定 ..... 103
４．２．７　IP アドレス取得の実施 ..... 109
４．２．８　プロジェクトファイルの保存 ..... 111
４．２．９　ソース生成 ..... 113
４．３　μNet3/Standard のコンフィグレーション ..... 115
４．３．１　コンフィグレーション一覧 ..... 115
４．３．２　IP アドレス ..... 117
４．３．３　デバイスドライバ ..... 117
４．３．４　プロトコルスタック情報テーブル ..... 117
４．３．５　μC3 リソース ..... 118
４．３．６　ネットワーク情報管理リソース(μNet3 ver.3 以降) ..... 118
**第５章　アプリケーションプログラミングインタフェースの説明** ..... 119
５．１　プロトコルスタックの初期化 ..... 119
５．２　ネットワーク・インタフェース API ..... 120
５．３　ネットワークデバイス制御 API ..... 127
５．４　ソケット API ..... 131
５．５　その他 API ..... 147
**第６章　付録** ..... 151
６．１　パケット形式 ..... 151
６．２　定数とマクロ ..... 154
６．３　エラーコード一覧 ..... 156
６．４　API 一覧 ..... 157
索引 ..... 158

# 第１章　μNet3 とは

## １．１　　特長

μNet3 は、ワンチップマイコン向けに最適化されたコンパクトな TCP/IP プロトコルスタックです。また、導入を容易にするために、わかりやすい独自 API を採用しています。

## １．２　　主な機能

- IPv4、ARP、ICMP、IGMPv2、UDP、TCP　プロトコルをサポート
- DHCP クライアント、DNS クライアント、FTP サーバー、HTTP サーバー機能が利用可能
- コンフィグレータによる TCP/IP の設定が可能(Compact 版)
- TCP 高速再送/高速復帰アルゴリズムサポート
- IP 再構築とフラグメンテーションサポート
- 複数のネットワーク・インタフェースをサポート

## １．３　　開発手順

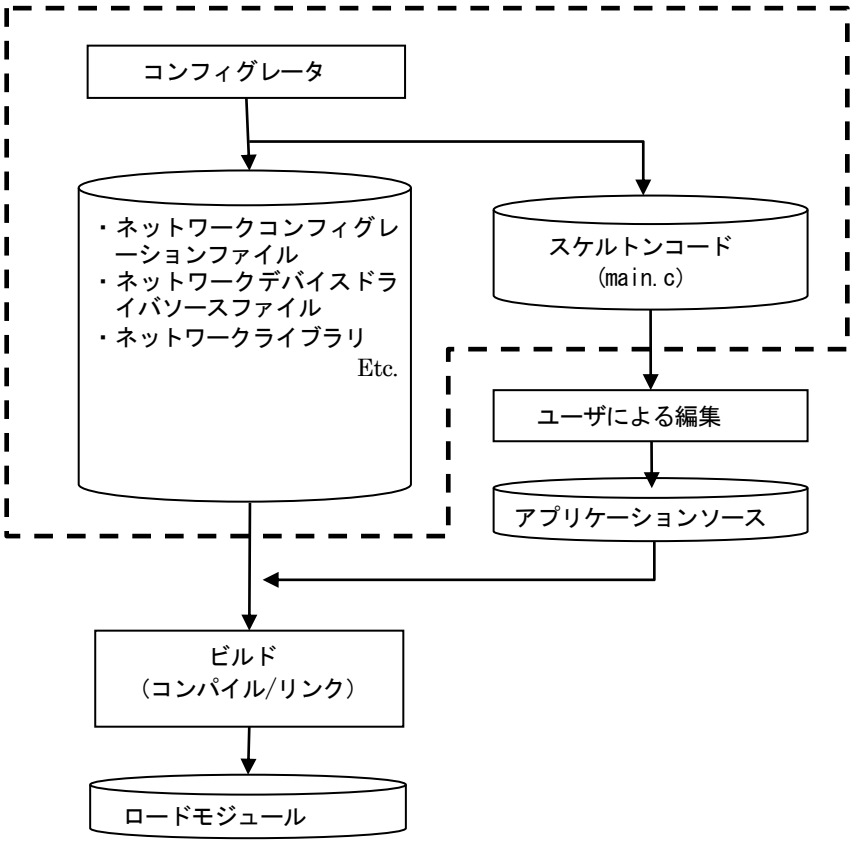
μNet3 を使用した開発手順を図に示します。

Compact 版を使用する場合、最初にコンフィグレータに TCP/IP に関する情報を入力します。コンフィグレータは入力された情報を元にコードを生成します。生成されたコードは、修正せずに使用するファイルとスケルトンコードがあります。このスケルトンコードは、必要なアプリケーションプログラムを記述する際の助けとなるよう作られています。

Standard 版を使用する場合は、ネットワークのコンフィグレーション(IP アドレス、ソケット定義)、デバイスドライバのコンフィグレーション(MAC アドレス、デバイス I/F 定義、ドライバ固有のフィーチャ)、μNet3 初期化ルーチン呼出しなどのコンフィグレーションを、テンプレートソースコード net_cfg.c に記述します。

アプリケーションプログラムを記述した後、ビルドしてロードモジュールを作成します。

**※コンフィグレータでは同時にカーネルのコンフィグレーションも行われますが、本書ではそれに関しては説明しません。適宜「μC3/Compact ユーザーズガイド」を参照してください。**




**開発手順図**

開発手順図中 ┌ ─ ┐ で囲った部分は Compact 版コンフィグレータのソース生成機能によって自動的に作成されます。Standad 版によるコンフィグレーションは net_cfg.c ファイルに設計に沿った情報を入力して実装します。詳しくは３．４．２　　メモリ I/O API (uNet3 ver.3.xx)

## net_memset　　メモリの値設定

**【書式】**

void* net_memset(void* d, int c, SIZE n);

**【パラメータ】**

| | | |
|---|---|---|
| void* | d | 設定するメモリの先頭アドレス |
| int | c | 設定する値 |
| SIZE | n | 設定バイト数 |

**【戻り値】**

| | | |
|---|---|---|
| void* | d | 設定するメモリの先頭アドレス |

**【解説】**

メモリの設定が正常に終了した場合は、引数で指定されるメモリの先頭アドレスを返却して下さい。

## net_memcpy　　メモリのコピー

【書式】

void* net_memcpy(void* d, const void* s, SIZE n);

【パラメータ】

| | | |
|---|---|---|
| void* | d | コピー先アドレス |
| const void* | s | コピー元アドレス |
| SIZE | n | コピーバイト数 |

【戻り値】

| | | |
|---|---|---|
| void* | d | コピー先アドレス |

【解説】

メモリのコピーが正常に終了した場合は、引数で指定されるコピー先アドレスを返却して下さい。

## net_memcmp　　メモリの比較

【書式】

int net_memcmp(const void* d, const void* s, SIZE n);

【パラメータ】

| | | |
|---|---|---|
| const void* | d | 比較メモリアドレス 1 |
| const void* | s | 比較メモリアドレス 2 |
| SIZE | n | 比較バイト数 |

【戻り値】

| | | |
|---|---|---|
| int | | 比較結果 |

【解説】

両メモリから指定されたバイト数分、同じ値の場合は 0 を返却して下さい。そうでない場合は、非 0 を返却して下さい。

第４章　コンフィグレーションを参照して下さい。

## １．４　　サンプルを使ったチュートリアル

μNet3/Compact に同梱しているサンプルを使用してプログラム作成までを説明します。
μNet3/Standard には予め作成された同様のプログラムが同梱されています。

**サンプルの説明**

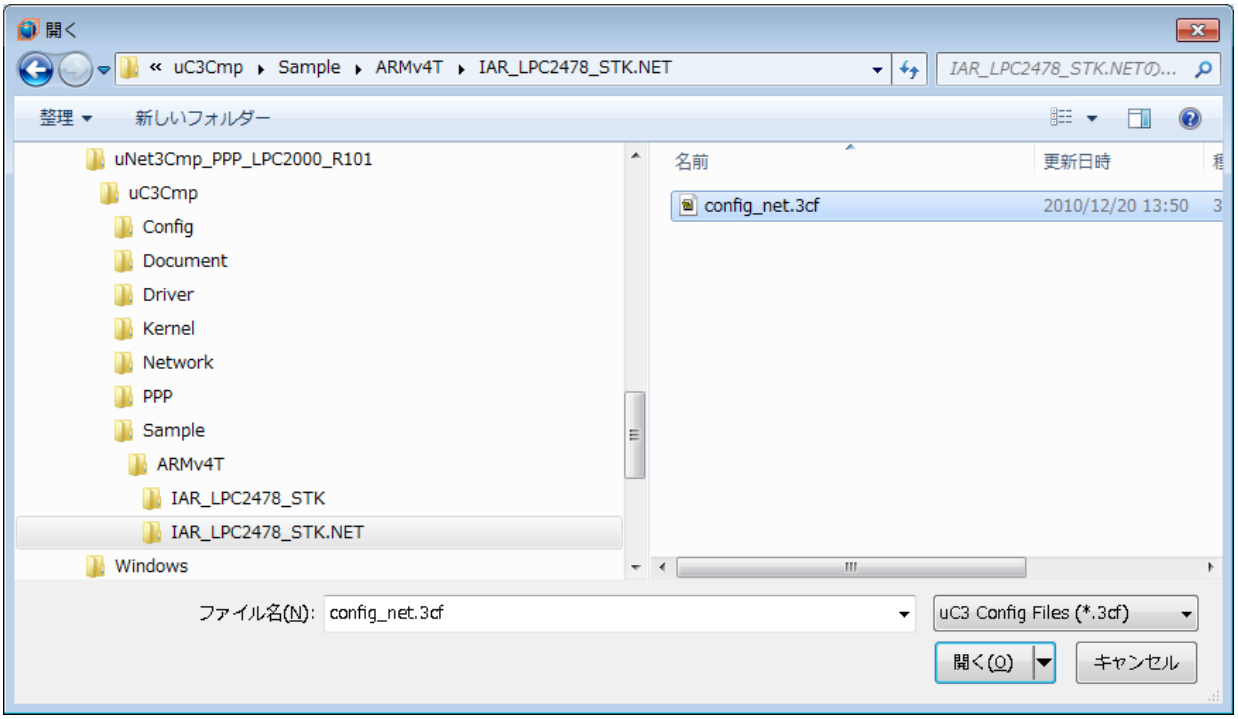
今回は Sample フォルダにある **Sample¥ARMv4T¥IAR_LPC2478_STK.NET（ご使用の環境に合わせて Sample¥XXX.NET フォルダを適宜読み替えて下さい）** を使用します。このサンプルを使って DHCP クライアント,HTTP サーバーを組み合わせたプログラムを作成します。

・DHCP クライアント
　DHCP サーバーから動的に IP アドレスを取得し自ホストに割り当てます。
・HTTP サーバー
　ウェブブラウザから LED の点滅間隔を 100msec 単位で変更させます。

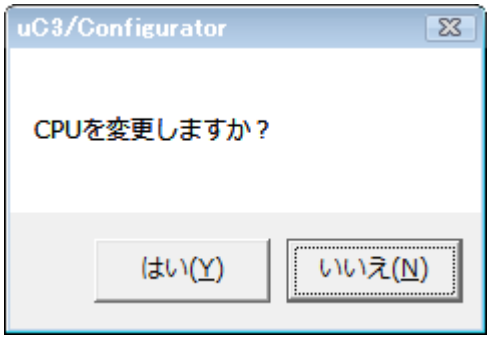
### １．４．１　　コンフィグレーションファイルの読み込み

Config¥uC3conf.exe を起動して、コンフィグレーション済みファイル読み込みます。


「既存のプロジェクトを開く」を選択し「OK ボタン」をクリックしてください。



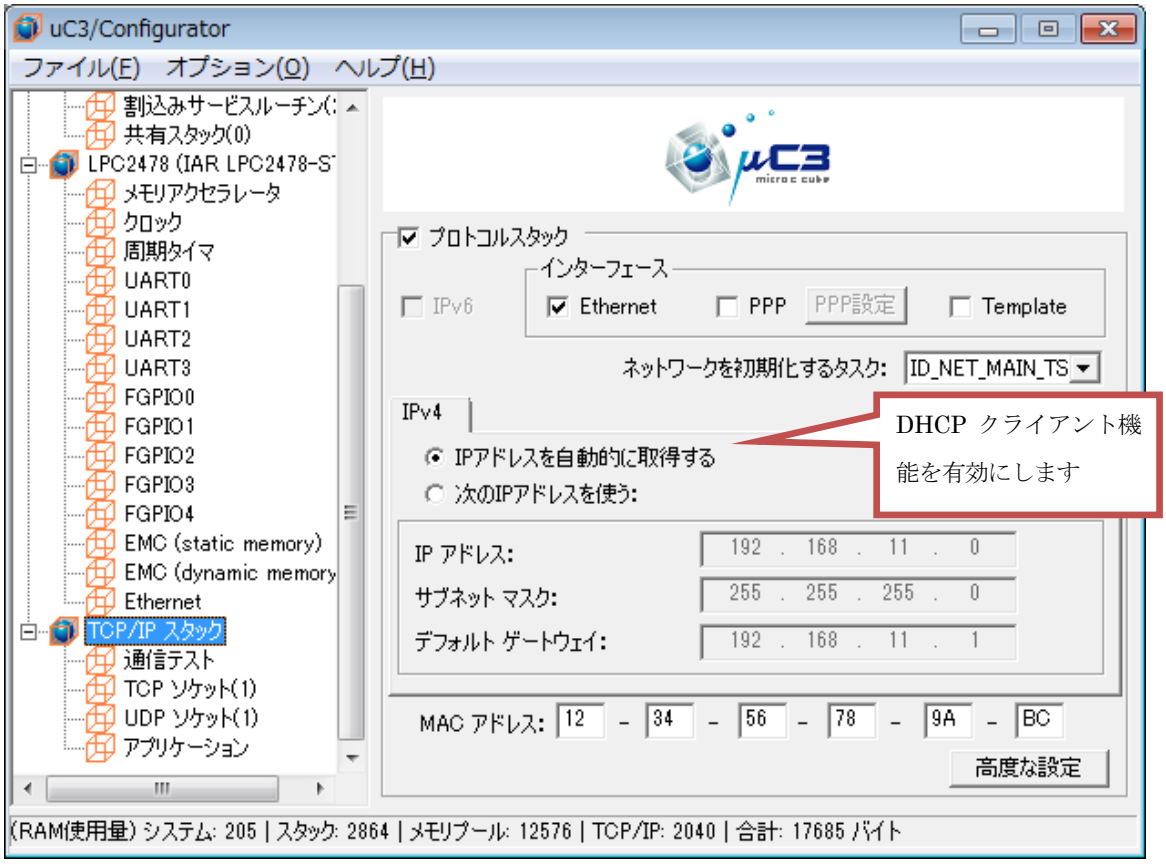

config_net.3cf ファイルを選択し、「開くボタン」をクリックしてください。




CPU 変更を確認する画面が表示されたら「いいえボタン」をクリックしてください。

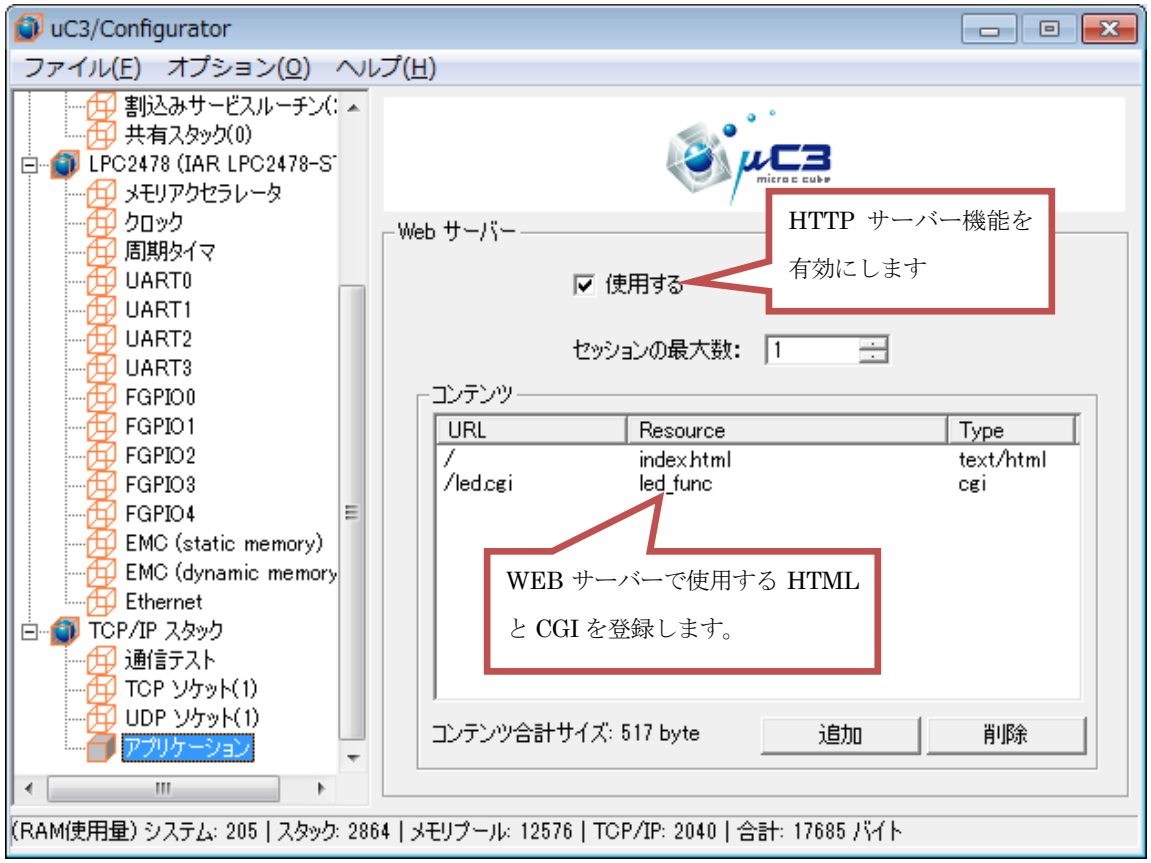
### １．４．２　　コンフィグレータを使った設定

μNet3/Compact のコンフィグレーションを行います。今回は既にコンフィグレーション済みですので設定の変更はしません。




左のツリーより「TCP/IP スタック」をマウスクリックで選択してください。この画面では主にターゲット側の IP アドレスの設定を行います。

今は、「IP アドレスを自動的に取得する」が選択されていることを確認してください。



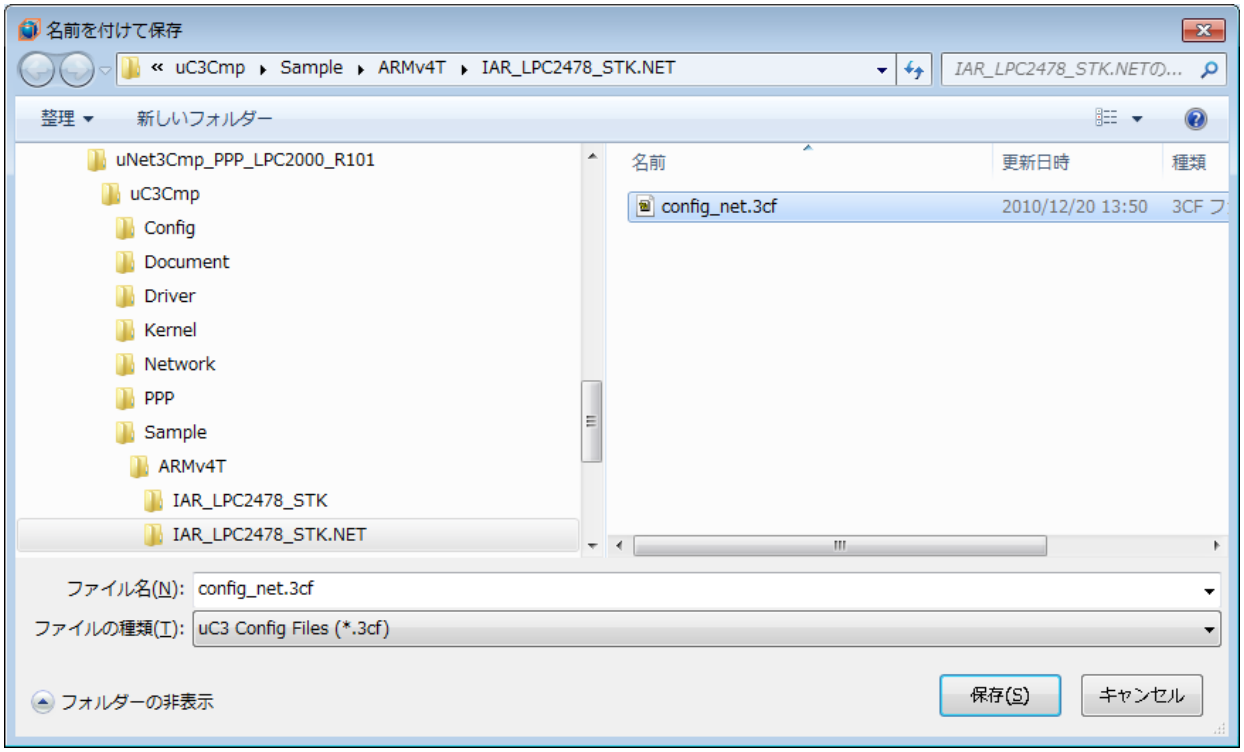

左のツリーより「アプリケーション」をマウスクリックで選択してください。この画面では主に WEB（HTTP）サーバーで使用する HTML や JPG などのファイル、CGI プログラムの関数名などを登録します。WEB サーバーには CGI の機能がサポートされており，LED の点滅はこの機能を使って行います．CGI の機能を使うとブラウザからの要求に合わせて指定した関数の呼び出しを行うことができます。ここでは URL を/led.cgi と指定して関数 led_func()が呼ばれるように指定します。

WEB サーバーを「使用する」が選択され、「コンテンツ」に HTML：index.html と CGI 関数：led_func が登録されていることを確認してください。

### １．４．３　　コンフィグレータ設定の保存

今回は、μNet3/Compact のコンフィグレーション設定の変更は実施しないが、コンフィグレータで設定した内容の確認を実施するために**コンフィグレータ設定の保存**を行います。

「ファイル」メニューより「保存」を選択してください。




今回は、設定の変更を実施していないため、上書きの保存を実施する。上書きの注意メッセージで「はい」ボタンをクリックする。

保存の操作で、コンフィグレータの設定を保存することができます。また同時に、この操作で、プロジェクトファイル（config_net.3cf）と拡張子を「xml」に変えたファイル（ここでは、config_net.xml）が保存されます。

config_net.xml をブラウザで開くことにより、コンフィグレータで設定した情報をブラウザの画面上で確認することができます。ネットワークでの設定例は、下記となります。

＜ネットワークの設定値＞

CPU

| 型番 |
|---|
| LPC1768 |

全般

| ネットワーク使用 | インタフェース | MTUサイズ | ネットワークバッファ数 | ネットワークを初期化するタスク(ID) |
|---|---|---|---|---|
| する | Ethernet | 1500 | 8 | ID_MAIN_TSK |

PPP

| ユーザ名 | パスワード | ダイヤル | COMデバイス | ボーレート | フロー制御 | IPアドレス取得 | ローカルIPアドレス | リモートIPアドレス | DNSアドレス取得 | プライマリDNSアドレス | セカンダリDNSアドレス | 認証プロトコル | VJ圧縮 | リトライ回数 | リトライ間隔(ms) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

ホスト

| DHCP | MACアドレス | IPアドレス | サブネットマスク | ゲートウェイ |
|---|---|---|---|---|
| 有効 | 12-34-56-78-9A-BC | | | |

ソケット

| IDの定義名 | ソケット種別 | ポート番号 | 送信バッファサイズ | 受信バッファサイズ | Connectタイムアウト | Closeタイムアウト | Sendタイムアウト | Receiveタイムアウト |
|---|---|---|---|---|---|---|---|---|
| ID_SOC_HTTP1 | TCP | 80 | 1024 | 1024 | -1 | -1 | 25000 | 25000 |
| ID_SOC_DHCP | UDP | 68 | | | | | 3000 | 3000 |

アプリケーション(WEBサーバー)

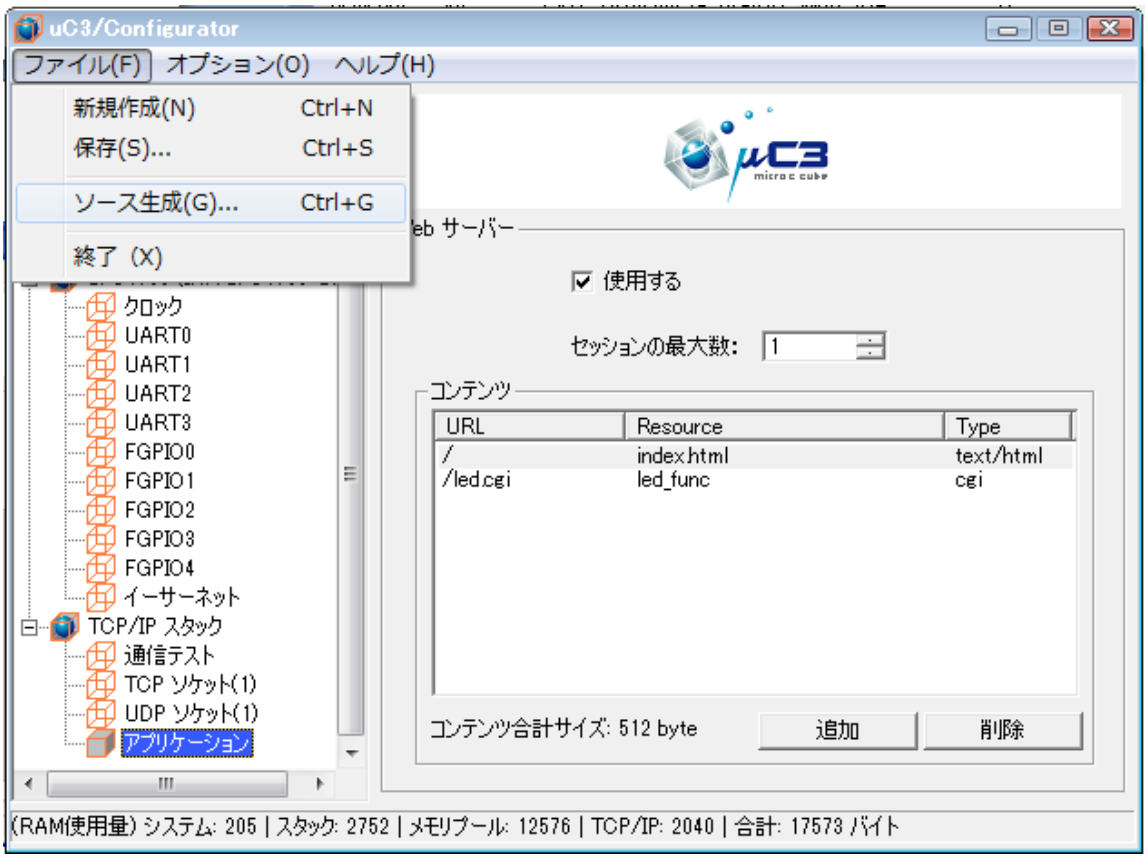
| 使用 | セッション最大数 |
|---|---|
| する | 1 |

| Content-Type | URL | Resource |
|---|---|---|
| text/html | / | .\index.html |
| cgi | /led.cgi | led_func |

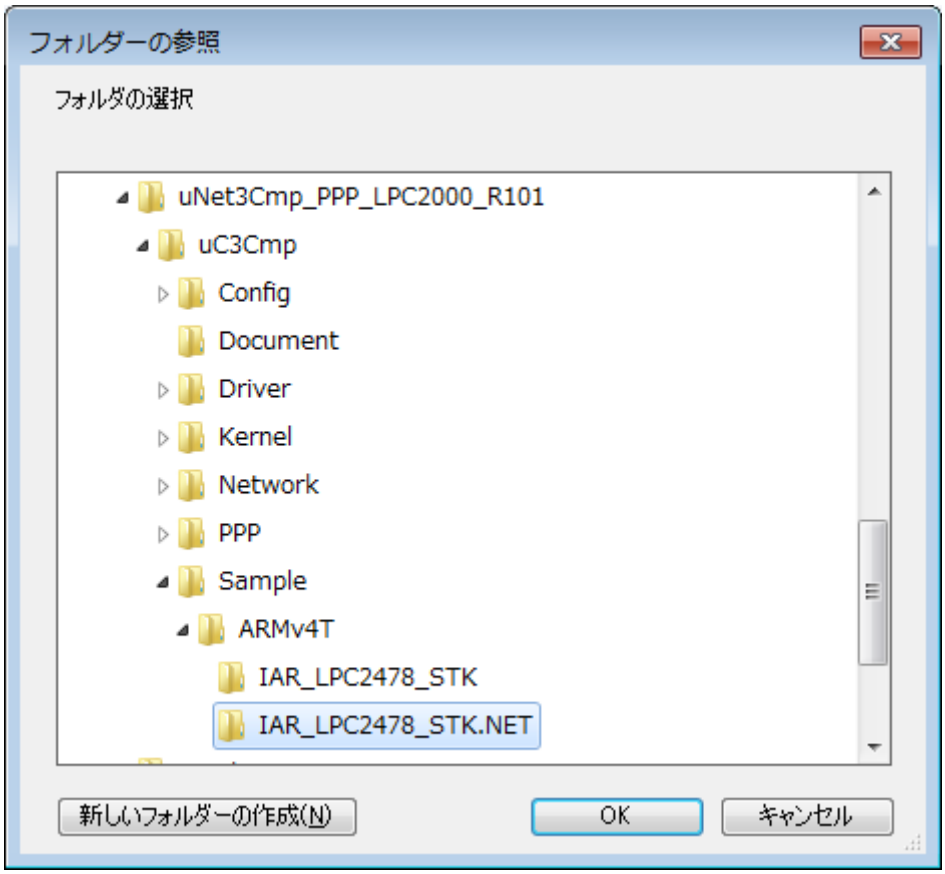
コンフィグレータで設定した内容図

### １．４．４　　ソースコードの生成

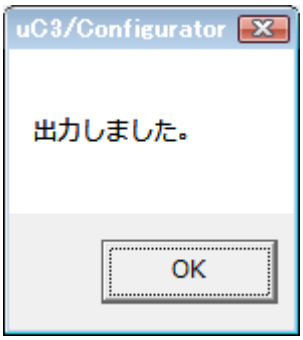
ソースコードの生成をします。



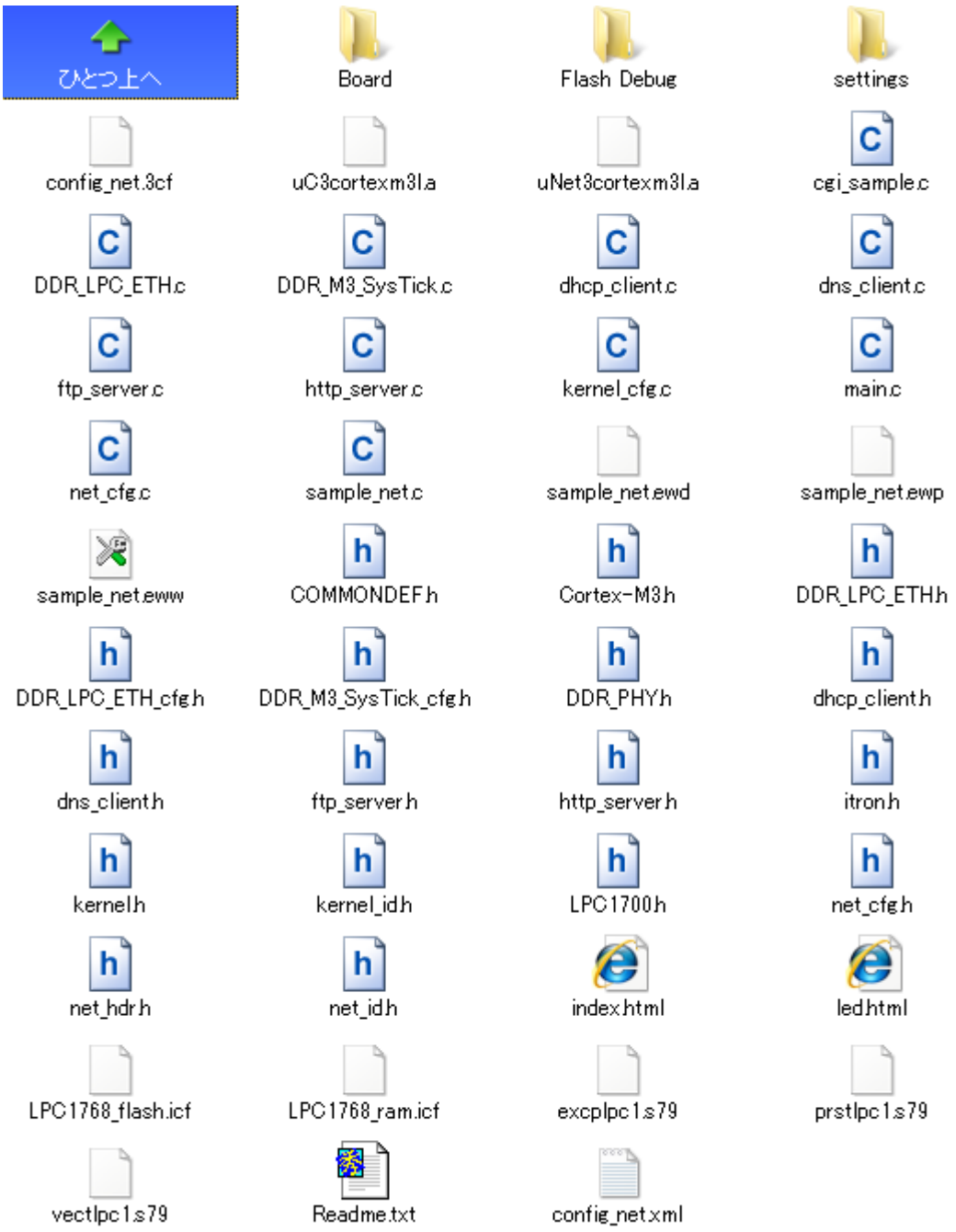

「ファイル」メニューより「ソース生成」を選択してください。




ソースの生成先に Sample¥ARMv4T¥IAR_LPC2478_STK.NET を選択してください。




完了画面が表示されたら、「OK ボタン」をクリックしてください。これでソースコード生成が完了しました。生成されたソースは**生成されたファイル図**のように TCP/IP プロトコルスタックライブラリ、コンフィグレーションファイル等があります。




**生成されたファイル図**

### １．４．４　　プログラムの作成

IAR Embedded Workbench を起動し sample_net.eww を読み込みます。本来はコンフィグレータが生成したスケルトンコード main.c にプログラムを記述するのですが、今回は既にプログラム記述済みである sample_net.c を使用します。

#### CGI プログラムの作成

送られてくる設定値は CGI の機能を使いアプリケーションで受取ります。CGI とはブラウザからの要求に応じてプログラムを起動するための仕組みで，これによりブラウザ側から設定値を受取り，応答をブラウザに返す仕組みを実現できます。

ブラウザから送られてくる、LED 点滅インターバルタイムの設定値はサンプル・コード cgi_sample.c に記述された関数 CgiScriptLedSetting()が変数 LedTmo に設定します。そこで，led_func()に CgiScriptLedSetting()の呼び出し処理を記述し、MainTask に LED 点滅処理を記述します。（**ソースリスト**）

ソースリスト

```
/*********************************
  CGI Script
 *********************************/
extern void CgiScriptLedSetting(T_HTTP_SERVER *http);
extern TMO LedTmo;
void led_func(T_HTTP_SERVER *http)
{
    CgiScriptLedSetting(http);
}

/*********************************
        MainTask
 *********************************/
extern ER net_setup(void);
void MainTask(VP_INT exinf)
{
    /* ネットワーク初期化 */
    net_setup();

    for (;;) {
        /* TODO */
        Led1(LED_ON);
        Led2(LED_ON);
        dly_tsk(LedTmo);
        Led1(LED_OFF);
        Led2(LED_OFF);
        dly_tsk(LedTmo);
    }
}
```

CGI 関数

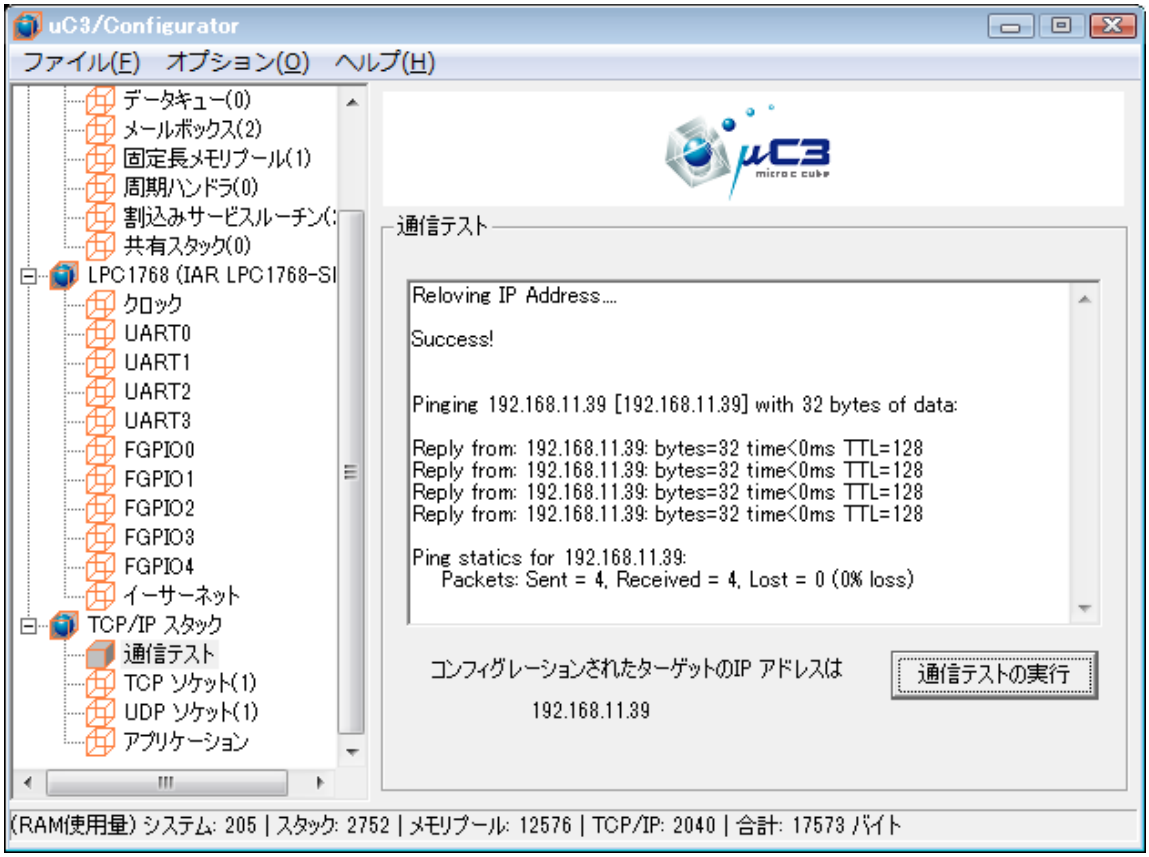
LED 点滅処理

### １．４．５　　WEB サーバーの実行

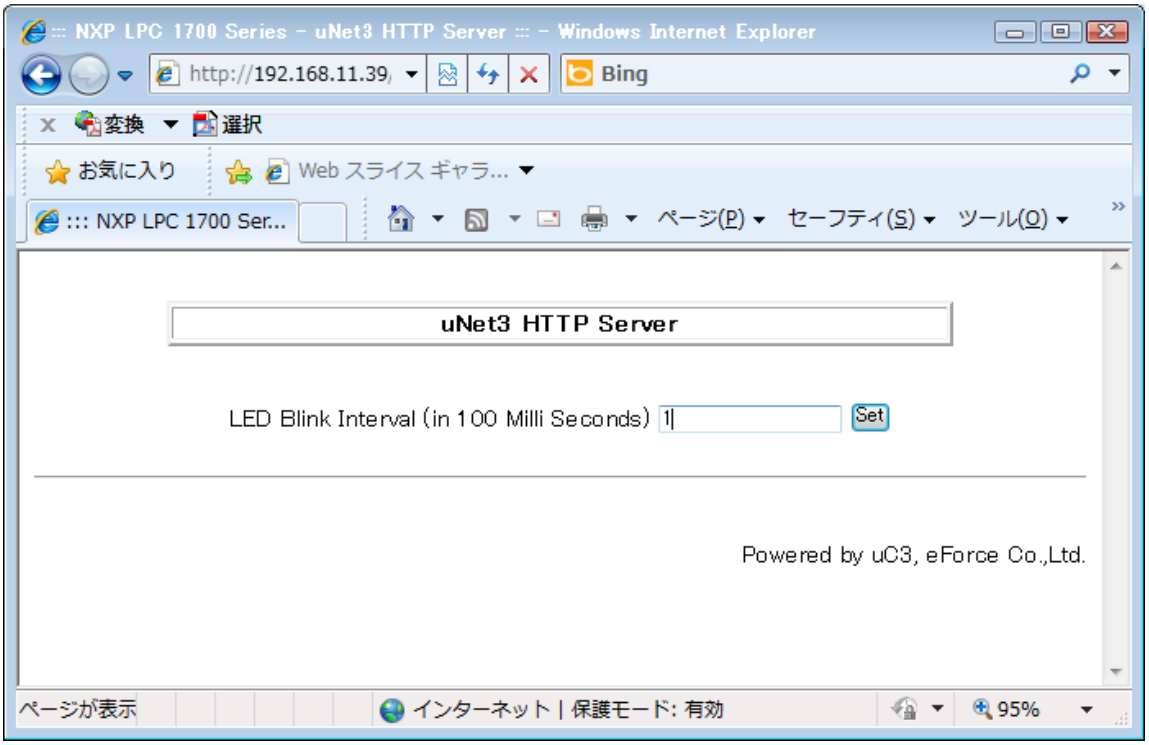
IAR Embedded Workbench で、ビルド対象の構成に「Flash　Debug」を選択し、ビルドしてロードモジュールを作成してください。

ロードモジュールを Flash メモリへ書き込み，プログラムが正しく実行されると、基板の LED が点滅します。まず、TCP/IP スタックが正しく動作しているか、コンフィグレータの通信テストの機能を使って確認します。「通信テスト」を選択し「通信テストの実行」ボタンを押すと，コンフィグレータで設定した内容を元にターゲットへ Ping を行います。通信が正常に行われると、**通信テスト図**のように“Reply from xxx.xxx.xxx.xxx.....”のような応答メッセージが表示されます。これによりターゲット側が正常に応答したことが確認できます。次にブラウザを起動します。ブラウザには設定した IP アドレスを直接入力すると、コンフィグレータで設定した HTML の画面が表示されます（**WEB ページ図**）。ここでインターバルを入力すると、指定したインターバルタイムで LED が点滅されるようになります。

**※PC にウィルスセキュリティソフトがイントールされている場合、ターゲット側プログラムが正常に動作しているにも関わらず通信テストが失敗することがあります、その際はウィルスセキュリティソフトのファイルフォール設定を無効にしてから通信テストを行ってください。**




通信テスト図




WEB ページ図

# 第２章　μNet3 の基本概念

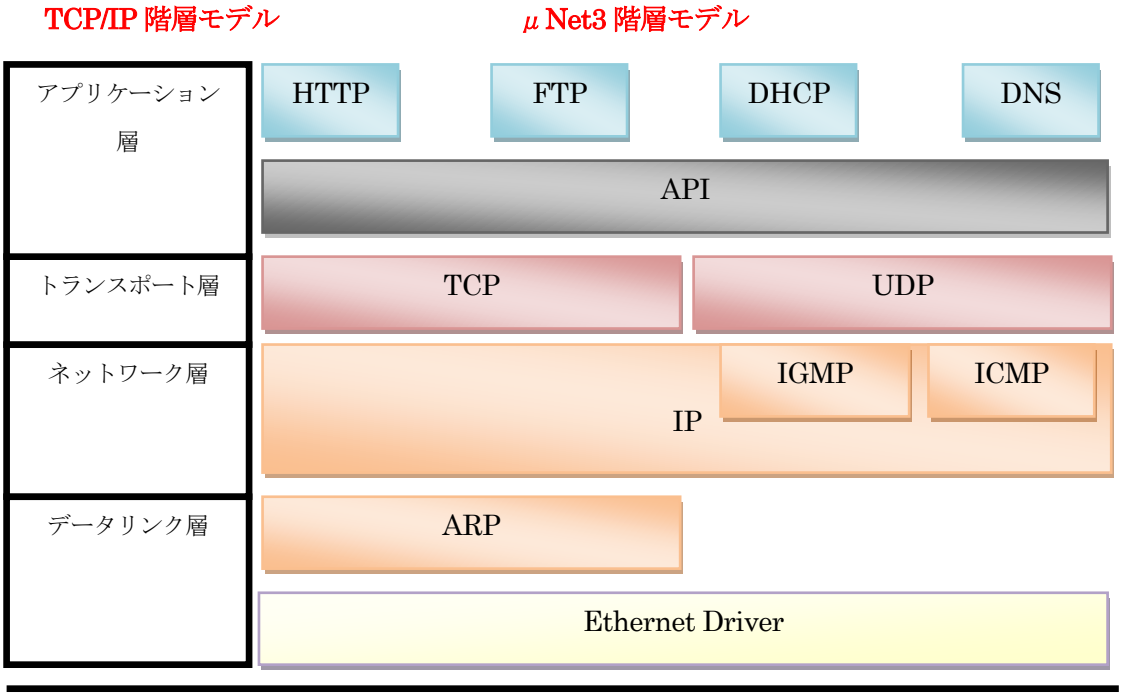
## ２．１　　用語の意味

### ２．１．１　　プロトコル

ネットワーク間でデータを伝達する手順、方法等を定めたものを「プロトコル」と呼びます。μNet3/Compact はこの「プロトコル」(=通信規則)を利用しています。これらの規則は“Request For Comments(通称：RFC)”と呼ばれるもので仕様が公開されています。

### ２．１．２　　プロトコルスタック

ネットワーク上である機能を実現するために必要なプロトコルを選び、階層状に積み上げたソフトウェア群を「プロトコルスタック」と呼びます。μNet3 では次のような階層となっています。




TCP/IP モデルとμNet3 の階層モデル図

### ２．１．３　　IP(Internet Protocol) アドレス

ネットワーク上で各ノードを特定するための論理的な番号を「IP アドレス」と呼びます。「IP アドレス」は 32 ビットのアドレス空間を持ち、192.168.1.32 のように表記します。

**ブロードキャストアドレス**

ブロードキャストとは一つのネットワークに属する全てのノードに対して、同時に同じデータを送る動作(=同報通信)のことを指します。そのブロードキャストのために特殊に割り当てられているアドレスのことを「ブロードキャストアドレス」と呼びます。通常、「ブロードキャストアドレス」にはすべてのビットが“1”の IP アドレス“255.255.255.255”を使用します。

**マルチキャストアドレス**

ブロードキャストが全てのノードにデータを送信するのに対し、特定のグループに対してのみ、データを送信する専用のアドレスのことを「マルチキャストアドレス」と呼びます。

### ２．１．４　MAC(Media Access Control) アドレス

論理アドレスである「IP アドレス」に対し、LAN カードなどのネットワーク機器を識別するために設定されているハードウェア固有の物理アドレスを「MAC アドレス」と呼びます。「MAC アドレス」は 48 ビットのアドレス空間を持ち、12-34-56-78-9A-BC　や 12:34:56:78:9A:BC　のように表記します。

### ２．１．５　ポート番号

ネットワーク通信で通信相手のプログラムを特定する番号のことを「ポート番号」と呼びます。TCP/IP で通信を行なうノードはネットワーク内での住所にあたる IP アドレスを持っていますが、複数のノードと同時に通信するために、補助アドレスとして 0 から 65535 のポート番号を用います。

### ２．１．６　ビックエンディアンとリトルエンディアン

複数バイトで構成されている数値データを、メモリに格納するときの方式のことを「エンディアン」と呼び、最上位バイトから順に格納する方式のことを「ビッグエンディアン」と呼びます。最下位バイトから順に格納する方式のことを「リトルエンディアン」と呼びます。

TCP/IP ではヘッダー情報は「ビッグエンディアン」で送信することが定められています。

### ２．１．７　パケット

データの送受信の単位を「パケット」と呼びます。パケットには二つの情報が含まれており、ひとつは実際のデータが格納された部分（データ領域）と、もうひとつは、そのデータの宛先や送信元情報、エラーチェック情報といった管理用の情報が格納される部分（ヘッダー領域）です。

### ２．１．８　ホストとノード

ネットワークで通信するコンピュータのことを「ホスト」と呼びます。サーバー、クライアント、ハブ、ルーター、アクセスポイント等、ネットワーク節点のことを「ノード」と呼びます。

### ２．１．９　　Address Resolution Protocol（ARP）

論理アドレス（TCP/IP の場合は IP アドレス）から物理アドレス（MAC アドレス）を導き出すためのプロトコルを「ARP」と呼びます。

### ２．１．１０　　Internet　Protocol（IP）

ノード間またはノードとゲートウェイ間の通信を実現するプロトコルを「IP（IP プロトコル）」と呼びます。「IP（IP プロトコル）」は上位層の基礎となる重要なプロトコルです。「IP」の役割は IP アドレスを元に、ルーターなどを経由して宛先にデータを届けることですが、確実に届けるという保証はなく、データ信頼性の確保は上位層の役割となっています。

前述の「IP アドレス」はこの「IP プロトコル」のヘッダーに置かれます。

### ２．１．１１　　Internet Control Message Protocol（ICMP）

「IP」ネットワーク通信で発生したエラーを通知したり、ネットワークの状態を確認する為の機能を提供するプロトコルを「ICMP」と呼びます。よく知られているものに Ping と言われるエコー要求、エコー応答メッセージがあります。

### ２．１．１２　　Internet Group Management Protocol（IGMP）

IP マルチキャストを実現する為のプロトコルを「IGMP」と呼びます。同一のデータを複数のホストに効率よく配送することができます。

### ２．１．１３　　User Datagram Protocol（UDP）

コネクションレス型のデータグラム通信サービスを提供するプロトコルを「UDP」と呼びます。「IP」はアプリケーションとのインタフェースを持っていません。「UDP」はその機能をアプリケーションから使えるようにしたプロトコルです。故に、パケットが相手に届いたことを知らせる手段がないことや、パケットの届く順番が入れ替る可能性があり、データの信頼性は保証されません。

### ２．１．１４　　Transmission Control Protocol（TCP）

コネクション型のストリーム通信サービスを提供するプロトコルを「TCP」と呼びます。「TCP」は IP プロトコルの上位層として、順序制御と誤り訂正や再送・フロー制御といった信頼性のある通信を提供します。

### ２．１．１５　　Dynamic Host Configuration Protocol（DHCP）

ネットワークに接続する際に、IP アドレスなど必要な情報を自動的に割り当てるプロトコルを「DHCP」と呼びます。「DHCP」を使うためには DHCP サーバーを用意し、サーバー側で、あらかじめ DHCP クライアント用に IP アドレスをいくつか用意しておく必要があります（アドレスプール）。

### ２．１．１６　Hyper Text Transfer Protocol（HTTP）

ホームページやウェブサイトの HTML ファイルなどのコンテンツの転送を行うためのプロトコルを「HTTP」と呼びます。「HTTP」は HTML ファイルだけの転送のみならず、WEB ブラウザで表示できる、JPEG、GIF、PNG、ZIP などのバイナリデータの転送も可能です。

### ２．１．１７　File Transfer Protocol（FTP）

ホスト間でファイル転送を行うためのプロトコルを「FTP」と呼びます。

### ２．１．１８　Domain Name System（DNS）

IP アドレスをホスト（ドメイン）名に、ホスト名を IP アドレスに変換する名前解決メカニズムのことを「DNS」と呼びます。「DNS」を利用すると IP アドレスをもとにホスト名を求めたり、ホスト名から IP アドレスを求めたりすることが可能になります。

### ２．１．１９　ソケット

アプリケーションが TCP/IP 通信するための通信窓口のことを「ソケット」と呼びます。「ソケット」は IP アドレスとポート番号等で構成されています。アプリケーションは「ソケット」を指定して回線を開くだけで、通信手順の詳細を気にすることなくデータの送受信を行なうことができます。通信側で使用しているプロトコルによりソケットの種類が存在します。TCP ソケットは TCP プロトコルを使用してデータ通信を実施し、UDP ソケットは UDP プロトコルを使用してデータ通信を実施します。μNet3 では操作対象となる「ソケット」を識別するのに ID 番号を使用します。アプリケーションでは ID 番号を用いてソケット API を呼び出します。

### ２．１．２０　ブロッキングとノンブロッキング

何らかの関数を呼び出したとき、そのアクションが完了するまで戻らないことを「ブロッキングモード」と呼び、完了を待たずに即座に戻ることを「ノンブロキングモード」と呼びます。

例えばμNe3 のソケット API において、「ブロッキング モード」で、rcv_soc 関数を呼び出したタスクはそのアクションが完了する（データが受信できる）まで待ち状態に置かれることになります。「ノンブロッキングモード」では rcv_soc 関数の呼出しは、エラーコード E_WBLK とともに即座に戻り、そのアクションの完了(EV_RCV_SOC)はコールバック関数に通知されます。

μNet3 のソケットのデフォルト動作は「ブロッキングモード」となっており、「ノンブロッキングモード」に変更するには cfg_soc 関数を使用してコールバック関数の登録とコールバックイベントフラグを設定します。

### ２．１．２１　コールバック関数

プロトコルスタックの状態を非同期にアプリケーションに通知する為の関数を「コールバッ

ク関数」と呼びます。

### ２．１．２２　　タスクコンテキスト

μNet3 の全ての API・関数は、タスクコンテキストから呼び出さなければならない。

ネットワークコールバック関数から slp_tsk 等のタスクを待ち状態にするシステムコールを呼び出さないでください。また、ネットワークコールバック関数からμNet3 の全ての API・関数を呼び出さないでください。

### ２．１．２３　　リソース

プログラムで使用する資源のことを「リソース」と呼びます。タスク、セマフォといった「カーネルネルオブジェクト」、メモリなどが該当します。

**※　タスク、セマフォ等の「カーネルオブジェクト」に関しての詳細は「μC3 ユーザーズガイド」を参照ください。**

### ２．１．２４　　MTU

MTU(Maximum Transfer Unit)は、通信ネットワークにおいて、1 回の転送で送信できるデータの最大値を示す値である。そして、MTU は、データリンク層のフレームの最大データサイズを示す。なお、MTU で指定のできる最小の値は 68 バイトとなります。

最大データサイズの指定は、データリンク層で使用するプロトコルに依存し、Ethernet インタフェースでは一般的に 1500 バイトが使用されています。

### ２．１．２５　　MSS

MSS (Maximum Segment Size)は TCP パケットの最大データサイズを示す。そのため、MSS の値は、次式で計算することができます。

MSS = MTU – (IP ヘッダーサイズ+ TCP ヘッダーサイズ（通常は 40 バイト))

Ethernet インタフェースでは一般的に MSS の値は、1460 バイトとなります。

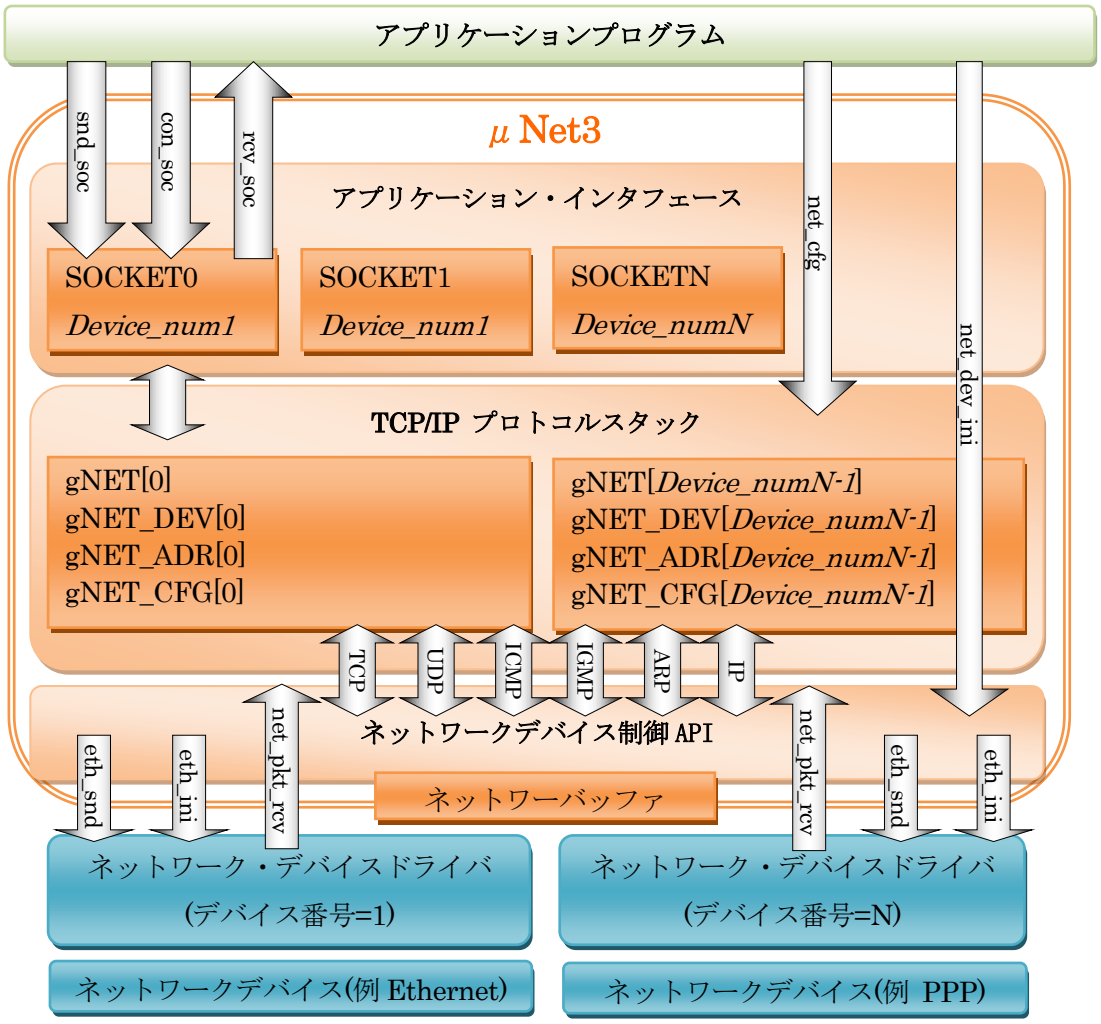
### ２．１．２６　　IP リアセンブリ・フラグメント

IP パケットの最大サイズは 64K バイトとなります。しかし、通信インタフェースの MTU はこれよりも小さい値となっているため、IP モジュールが、IP パケットをいくつかに分割して送信する必要があります。この処理を「IP フラグメンテーション」と呼び、分割された IP パケットのことを「IP フラグメント」と呼びます。

そして、受信側の IP モジュールでは、分割された「IP フラグメント」を連結する必要があり、この処理を「IP リアセンブリ」と呼びます。

## ２．２　　ネットワークシステムのアーキテクチャ

### ２．２．１　　ネットワークシステム構成図




**ネットワークシステムの構成図**

- **アプリケーションプログラム**
  ネットワーク通信するためのユーザアプリケーションプログラムです。DHCP、FTP、Telnet、HTTP などのアプリケーションプロトコルも含まれます。

- **アプリケーション・インタフェース**
  リモートホストとの接続の確立、データの送信・受信等といった様々なネットワークサービスを利用するためのインタフェース(API)を提供します。
  通常アプリケーションではソケット ID やデバイス番号を指定してアプリケーション･インターフェースを使用します。

- **TCP/IP プロトコルスタック**
  このプログラムは TCP、UDP、ICMP、IGMP、IP、ARP といったネットワークプロトコルを処理します。

- **ネットワークデバイス制御 API**
  ネットワークシステムには様々はネットワークデバイスが存在している可能性があり、デバイス毎にデバイスドライバが必要になります。ネットワークデバイス制御 API は、これらのデバイスの違いを吸収し、統一的にアクセスするためのインタフェースを提供します。アプリケーションプログラムからデバイス番号を使用して各デバイスにアクセスします。

- **ネットワーク・デバイスドライバ**
  ネットワークデバイスを制御するプログラムです。この実装の中身はデバイスごとに異なります。
  **※μNet3 では標準で Ethernet と PPP のデバイスドライバを提供しています。**

- **ネットワークデバイス**
  実際のネットワークデータの送信、受信を行うハードウェアです。Ethernet、PPP(RS-232) 、WLAN などがこれに該当します。

- **その他**
  **μNet3** は下記 **μC3** のカーネルオブジェクトを使用しています。

| オブジェクト | オブジェクト ID | 用途 |
|---|---|---|
| タスク | ID_NET_MAIN_TSK | μNet3 起動タスク |
| タスク | ID_TCP_TIM_TSK | μNet3 時間管理タスク |
| タスク | ID_ETH_SND_TSK | Ether ドライバ送信タスク |
| タスク | ID_ETH_RCV_TSK | Ether ドライバ受信タスク |
| タスク | ID_ETH_CTL_TSK | Ether ドライバ制御タスク |
| セマフォ | ID_TCP_SEM | μNet3 リソース制御セマフォ |
| イベントフラグ | ID_ETH_RCV_FLG | Ether ドライバイベントフラグ |
| イベントフラグ | ID_ETH_SND_FLG | Ether ドライバイベントフラグ |
| メールボックス | ID_ETH_SND_MBX | Ether ドライバメールボックス |
| メールボックス | ID_ETH_RCV_MBX | Ether ドライバメールボックス |
| 固定長メモリプール | ID_TCP_MPF | ネットワークバッファ領域 |

## ２．３　　ディレクトリとファイル構成

### ２．３．１　　μNet3 ver.1.xx/μNet3 ver.2.xx のファイル構成

μNet3 に含まれるファイルは次の通りです。

ヘッダーファイル

/Network/Inc

| | |
|---|---|
| net_cfg.h | TCP/IP プロトコルスタックのデフォルトコンフィグレーションマクロ |
| net_hdr.h | TCP/IP プロトコルスタックを使用するための必要な情報が定義<br>**※このヘッダーファイルはアプリケーションのソースファイルに必ず含めてください** |

ソースファイル

/Network/Src

| | |
|---|---|
| net_ini.c | 初期化モジュール |
| net_buf.c | ネットワークバッファ管理 API |
| net_arp.c | ARP プロトコル　モジュール |
| net_ip4.c | IPv4 プロトコル　モジュール |
| net_icmp.c | ICMP プロトコル　モジュール |
| net_igmp.c | IGMPv2 プロトコル　モジュール |
| net_ipr.c | IP 再構築　モジュール |
| net_udp.c | UDP プロトコル　モジュール |
| net_tcp.c | TCP プロトコル　モジュール |
| net_tim.c | プロトコルスタック　タイマーモジュール |
| net_soc.c | ソケット API |
| net_dev.c | デバイスドライバインタフェース |

ライブラリファイル

このフォルダには TCP/IP プロトコルスタックを各種プロセッサモードでビルド済みライブラリとビルド用のプロジェクトファイルが格納されています。

/Network/lib/<**CPU**>

uNet3**xxxx*b***.a

uNet3**xxxx*l***.a

**CPU** は CPU のアーキテクチャ名に依存します。

**xxxx** は CPU のプロセッサモードまたはプロセッサ名に依存します。

‘*b*’はビッグエンディアン、‘*l*’はリトルエンディアンを表します。

アプリケーションプロトコルソースファイル

/Network/NetApp

| | |
|---|---|
| dhcp_client.h | DHCP クライアント マクロ、プロトタイプ、定義等 |
| dhcp_client.c | DHCP クライアント ソースコード |
| ftp_server.h | FTP サーバー　マクロ、プロトタイプ、定義等 |
| ftp_server.c | FTP サーバー ソースコード |
| http_server.h | HTTP サーバー　マクロ、プロトタイプ、定義等 |
| http_server.c | HTTP サーバー ソースコード |
| dns_client.h | DNS クライアント マクロ、プロトタイプ、定義等 |
| dns_client.c | DNS クライアント ソースコード |
| ping_client.h | ICMP エコー要求 マクロ、プロトタイプ、定義など |
| ping_client.c | ICMP エコー要求(ping) ソースコード |
| sntp_client.h | SNTP クライアント マクロ、プロトタイプ、定義等 |
| sntp_client.c | SNTP クライアント ソースコード |
| net_strlib.h | μNet3 提供 String 系ライブラリ関数定義 |
| net_strlib.c | μNet3 提供 String 系ライブラリ関数ソースコード |

/Network/NetApp/ext

| | |
|---|---|
| dhcp_client.h | DHCP クライアントロ、プロトタイプ、定義等 |
| dhcp_client.c | 拡張版 DHCP クライアント ソースコード |

サンプルソースファイル

/Network/sample

| | |
|---|---|
| DDR_TEMPLATE_NET.c | μNet3 ネットワーク・デバイスドライバテンプレートコード |
| DDR_LOOPBACK_NET.c | ループバック用デバイスドライバ |

## ２．３．２　μNet3 ver.3.xx/μNet3 PRO のファイル構成

μNet3 に含まれるファイルは次の通りです。

ヘッダーファイル

/Network/Inc

| | |
|---|---|
| net_sup.h | TCP/IP プロトコルスタックのデフォルトコンフィグレーションマクロ |
| net_def.h | TCP/IP プロトコルスタックの定義（内部制御用） |
| net_sts.h | ネットワーク情報管理の定義（内部制御用） |
| net_sts_id.h | ネットワーク情報管理 ID の定義 |
| net_hdr.h | TCP/IP プロトコルスタックを使用するための必要な情報が定義<br>**※このヘッダーファイルはアプリケーションのソースファイルに必ず含めてください** |

ソースファイル

/Network/Src

| | |
|---|---|
| net_ini.c | 初期化モジュール |
| net_buf.c | ネットワークバッファ管理 API |
| net_arp.c | ARP プロトコル　モジュール |
| net_ip4.c | IPv4 プロトコル　モジュール |
| net_icmp.c | ICMP プロトコル　モジュール |
| net_igmp.c | IGMPv2 プロトコル　モジュール |
| net_ipr.c | IP 再構築　モジュール |
| net_udp.c | UDP プロトコル　モジュール |
| net_tcp.c | TCP プロトコル　モジュール |
| net_tim.c | プロトコルスタック　タイマーモジュール |
| net_soc.c | ソケット API |
| net_dev.c | デバイスドライバインタフェース |
| net_sts.c | ネットワーク情報管理モジュール |

ライブラリファイル

このフォルダには TCP/IP プロトコルスタックを各種プロセッサモードでビルド済みライブラリとビルド用のプロジェクトファイルが格納されています。

/Network/lib/<**CPU**>

uNet3**xxxx*b***.a
uNet3**xxxx*l***.a
***uNet3BSDxxxx*b***.a
***uNet3BSDxxxx*b***.a

/SNMP/lib/<**CPU**>

***SNMPxxxx*b***.a
***SNMPxxxx*l***.a

**CPU** は CPU のアーキテクチャ名に依存します。
**xxxx** は CPU のプロセッサモードまたはプロセッサ名に依存します。
‘***b***’ はビッグエンディアン、‘***l***’ はリトルエンディアンを表します。
***uNet3BSDxxxx*** ライブラリはμNet3 PRO 版にのみ収録されます。
***SNMPxxxxl*** ライブラリはμNet3 PRO 版にのみ収録されます。

アプリケーションプロトコルソースファイル

***太字斜体***で示すファイルはμNet3 PRO 版にのみ収録されます。

/Network/NetApp

| | |
|---|---|
| dhcp_client.h | DHCP クライアント マクロ、プロトタイプ、定義等 |
| dhcp_client.c | DHCP クライアント ソースコード |
| ftp_server.h | FTP サーバー　マクロ、プロトタイプ、定義等 |
| ftp_server.c | FTP サーバー ソースコード |
| http_server.h | HTTP サーバー　マクロ、プロトタイプ、定義等 |
| http_server.c | HTTP サーバー ソースコード |
| dns_client.h | DNS クライアント マクロ、プロトタイプ、定義等 |
| dns_client.c | DNS クライアント ソースコード |
| ping_client.h | ICMP エコー要求 マクロ、プロトタイプ、定義など |
| ping_client.c | ICMP エコー要求(ping) ソースコード |
| sntp_client.h | SNTP クライアント マクロ、プロトタイプ、定義等 |
| sntp_client.c | SNTP クライアント ソースコード |
| net_strlib.h | μNet3 提供 String 系ライブラリ関数定義 |
| net_strlib.c | μNet3 提供 String 系ライブラリ関数ソースコード |
| base64calc.c | BASE64 計算ライブラリ ソースコード |
| base64calc.h | BASE64 計算ライブラリ ソースコード定義など |

| | |
|---|---|
| md5calc.c | MD5 計算ライブラリ ソースコード |
| md5calc.h | MD5 計算ライブラリ ソースコード定義など |
| ***dhcp_server.c*** | DHCP サーバー ソースコード |
| ***dhcp_server.h*** | DHCP サーバー マクロ、プロトタイプ、定義等 |
| ***ftp_client.c*** | FTP クライアント ソースコード |
| ***ftp_client.h*** | FTP クライアント マクロ、プロトタイプ、定義等 |
| ***http_client.c*** | HTTP クライアント ソースコード |
| ***http_client.h*** | HTTP クライアント マクロ、プロトタイプ、定義等 |
| ***smtp_client.c*** | SMTP クライアント ソースコード |
| ***smtp_client.h*** | SMTP クライアント マクロ、プロトタイプ、定義等 |
| ***sntp_server.c*** | SNTP サーバー ソースコード |
| ***sntp_server.h*** | SNTP サーバー マクロ、プロトタイプ、定義等 |
| ***telnet_server.c*** | TELNET サーバー ソースコード |
| ***telnet_server.h*** | TELNET サーバー マクロ、プロトタイプ、定義等 |
| ***tftp_client.c*** | TFTP クライアント ソースコード |
| ***tftp_client.h*** | TFTP クライアント マクロ、プロトタイプ、定義等 |
| ***tftp_server.c*** | TFTP サーバー ソースコード |
| ***tftp_server.h*** | TFTP サーバー マクロ、プロトタイプ、定義等 |
| /Network/NetApp/ext | |
| dhcp_client.h | DHCP クライアントロ、プロトタイプ、定義等 |
| dhcp_client.c | 拡張版 DHCP クライアント ソースコード |
| /Network/NetApp/cfg | |
| ftp_server_cfg.c | FTP サーバーコンフィグレーション |
| ftp_server_cfg.h | FTP サーバーコンフィグレーション |
| ***ftp_client_cfg.h*** | FTP クライアントコンフィグレーション |
| ***http_client_cfg.h*** | HTTP クライアントコンフィグレーション |
| ***smtp_client_cfg.h*** | SMTP クライアントコンフィグレーション |
| ***sntp_server_cfg.h*** | SNTP サーバーコンフィグレーション |
| ***telnet_server_cfg.h*** | TELNET サーバーコンフィグレーション |
| ***tftp_client_cfg.h*** | TFTP クライアントコンフィグレーション |
| ***tftp_server_cfg.h*** | TFTP サーバーコンフィグレーション |
| ***/Network/bsd*** | μNet3 BSD ユーザーズガイド参照 |
| ***/SNMP*** | μNet3 SNMP ユーザーズガイド参照 |

サンプルソースファイル

/Network/sample

| | |
|---|---|
| DDR_TEMPLATE_NET.c | μNet3 ネットワーク・デバイスドライバテンプレートコード |
| DDR_LOOPBACK_NET.c | ループバック用デバイスドライバ |

# 第３章　μNet3 の機能概要

## ３．１　　プロトコルスタック

### ３．１．１　　IP モジュール

IP モジュールでは送られてくるパケットの宛先 IP アドレスが、自ホストの IP アドレスと一致するときだけパケットを受信し処理します。それ以外のパケットは処理しません。

#### IP オプション

μNet3 は IP オプションの内 IGMP ルーター警告オプションのみサポートしています。サポートしていない IP オプションは無視されます。

#### TTL（Time to Live）

μNet3 で TTL のデフォルト値は CFG_IP4_TTL（64）に設定されています。この値は net_cfg()を使って変更することができます。net_cfg()を使って TTL 値を変更した場合、すべてのソケットの TTL 値が変更されます。個々のソケットの TTL 値を変更したい場合は cfg_soc()を使用してください。

#### TOS（Type Of Service）

μNet3 で TOS は CFG_IP4_TOS（0）に設定されています。

#### ブロードキャスト

ブロードキャストの受信可否は net_cfg()を使って変更することができます。初期値は受信可に設定されています。ブロードキャストの送信は常に可能です。ブロードキャストの設定はすべてのソケットに対して有効で、ソケット単位でのブロードキャストの受信可否設定はできません。

ブロードキャストの送受信には UDP ソケットを使用してください。

#### マルチキャスト

マルチキャスト受信を許可するには net_cfg()を使って、参加するマルチキャストグループアドレスを登録します。マルチキャストグループアドレスは CFG_NET_MGR_MAX（8）まで登録することができます。マルチキャストの送信は常に可能です。マルチキャストの設定はすべてのソケットに対して有効で、ソケット単位でのマルチキャストの受信可否設定はできません。

マルチキャスト送信用 TTL は CFG_IP4_MCAST_TTL(１)に設定されています。この値も net_cfg()を使って変更することができます。

マルチキャストのループバックはサポートしていません。

マルチキャストの送受信には UDP ソケットを使用してください。

### MTU

μNet3 では MTU のデフォルト値として CFG_PATH_MTU (1500 バイト) を設定しています。この値は、コンフィグレータで設定することができます。

### IP リアセンブリ・フラグメント

uNet3 では、IP パケットとして、最大サイズはデフォルトとして、1500 バイトとなっています(この値は、ネットワークバッファの値と関連しています)。IP パケットの最大サイズを大きくするためには、ネットワークバッファを大きくする必要があります。例えば、2048byte の UDP データを送受信する場合は、ネットワークバッファの値を、「コントロールヘッダサイズ(100 bytes) + IP ヘッダーサイズ (20bytes) + UDP ヘッダーサイズ (8bytes) + 2048」の計算値よりも大きくする必要があります。

デフォルトの IP リアセンブリ・プロセス・タイムアウト値は、CFG_IP4_IPR_TMO(10 秒)となっています。もし、リアセンブリ・プロセスがこのタイムアウト内に完了しない場合、リアセンブリ処理は取り消され、ICMP エラーメッセージ(タイプ 11：時間超過によるパケット廃棄)がリモートホストに送られます。

デフォルトの IP リアセンブリ・プロセス回数は CFG_NET_IPR_MAX(2)として設定しています。CFG_NET_IPR_MAX の値は、ホストが同時に IP リアセンブリ処理を実施することができる値を示しています。

### IGMP

μNet3 では（ルーターからの）「クエリ（グループ問い合わせ)」対する「レポート（応答)」メッセージの送信までのタイムアウトは CFG_IGMP_REP_TMO （10 秒）に設定されています。

μNet3 は IGMPv2 をサポートしていますが、IGMPv1 互換機能もサポートしています。

IGMPv1 の「クエリ」を受け取った場合、IGMPv1 モードに切り替えて処理を行います。その後、一定時間、IGMPv1 メッセージが無ければ IGMPv2 モードに戻ります。この IGMPv 1 から IGMPv 2 に戻るまでのタイムアウトは CFG_IGMP_V1_TMO （400 秒）に設定されています。

### ICMP

μNet3 は「エコー応答」、「エコー要求」、「時間超過」メッセージをサポートしています。

### ３．１．２　ARP モジュール

#### （1）アドレス解決

μNet3 ではホストの IP アドレスと物理アドレス（MAC アドレス）の対応付けを管理しています。この対応付けの管理表（変換表）を ARP キャッシュと呼びます。ARP キャッシュサイズは CFG_NET_ARP_MAX（8）に設定されています。

IP パケットをネットワークに送信する際、ARP キャッシュを参照し該当する IP アドレスが存在した場合は、そこに記録されている物理アドレスを宛先としてパケットを送信します。IP アドレスが存在しない場合は、IP パケットは一旦キューに保存し、ARP 要求パケットをブロードキャスト送信します。リモートホストから ARP 応答パケットを受信したら、新たに ARP キャッシュに受信した物理アドレスを記録します。その後、キューから IP パケットを取り出し、新たに取得した物理アドレス宛てにパケットを送信します。

また、ARP エントリ情報は最長 ARP_CLR_TMO（20 分間）キャッシュに保持されます。

#### （2）IP アドレス競合検出

RFC5227 に従って、同一リンク内の他のホストと IP アドレスが重複していないかをチェックします。このチェックは LAN インタフェースの起動時や、リンクアップ状態に移行した際にアプリケーションの指示により実行します。またインタフェースに IP アドレスが設定されたのち、他のホストが同じ IP アドレスを使用していた場合には、競合を検出してアプリケーションに通知します。

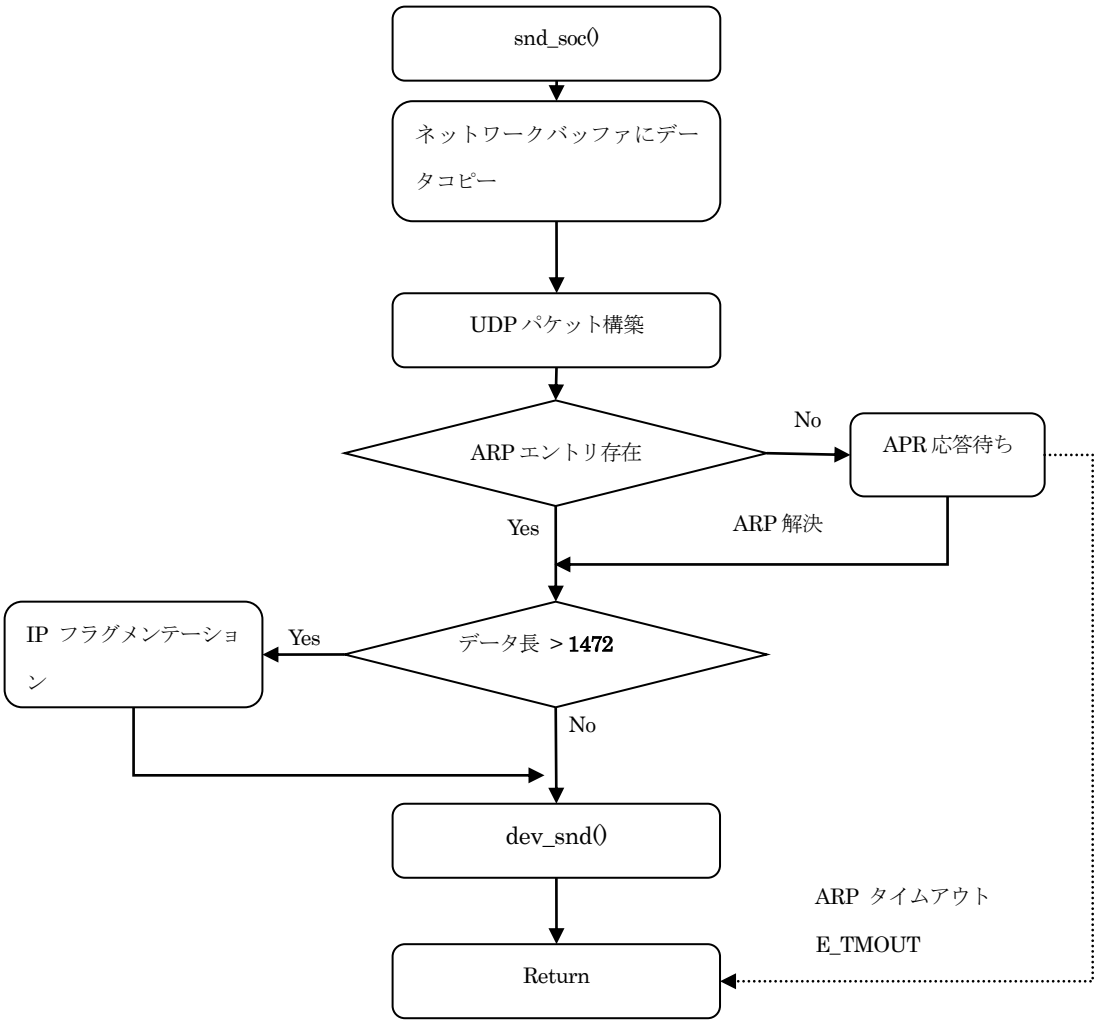
IP アドレスの競合検出には ARP メッセージを使用します。これから使用する IP アドレスが、既に使用されていないかを探知する ARP メッセージを「ARP Probe」と言います。ARP Probe メッセージに対して他のホストが ARP 応答しなかった(競合する IP アドレスが無い)場合には、「ARP Announce」と言うメッセージを送信して IP アドレスを使用することを通知します。

### ３．１．３　UDP モジュール

UDP はリモートホストと接続する事なしにデータの送受信を行います。

#### （1）データの送信

データの送信前には必ず con_soc を使って、送信先(IP アドレス、ポート番号)とソケットの関連付けを行います。その後、snd_soc()を使ってデータを送信します。snd_soc（）の処理フローは下図のようになります。




**UDP ソケット　snd_soc の処理フロー**

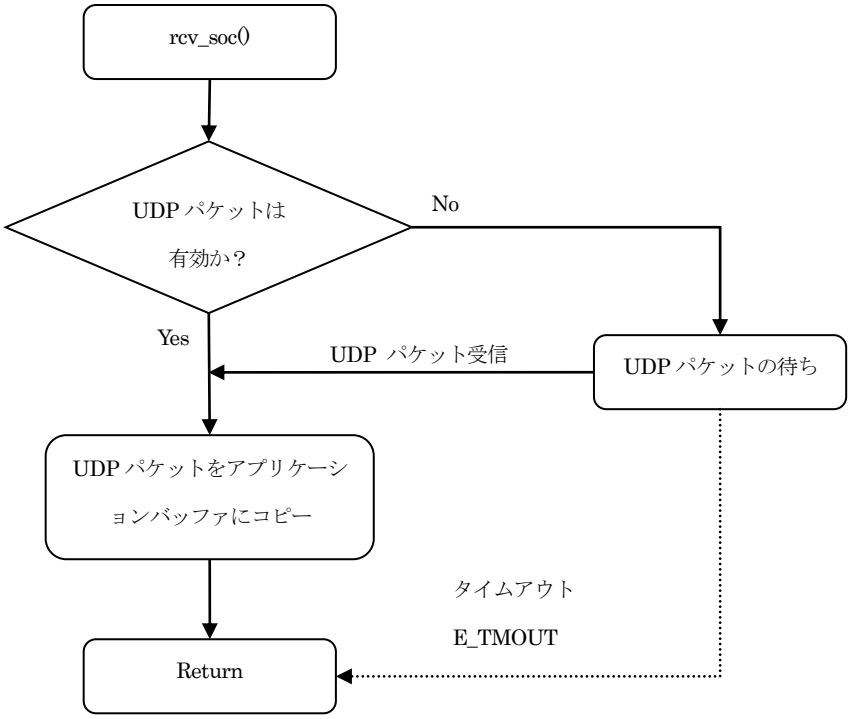
① アプリケーションデータはネットワークバッファーにコピーされ、リモートホストの IP アドレス、ポート番号など UDP ヘッダーを付加し UDP パケットを構築します。

② ARP プロトコルでリモートホストの MAC アドレスが解決出来ないときは、E_TMOUT エラーを返します。

③ デフォルトでは、送信データの最大サイズが 1472 バイト (CFG_PATH_MTU (1500 バイト) – IP ヘッダーサイズ – UDP ヘッダーサイズ)に設定されています。これ以上のサイズを送信する場合は、ネットワークバッファサイズの設定が必要です。詳細は、IP リアセンブリ・フラグメントの項目を参照してください。

### （2）データの受信

データの受信は rcv_soc()を使います。rcv_soc（）の処理フローは下図のようになります。




**UDP ソケット rcv_soc の処理フロー**

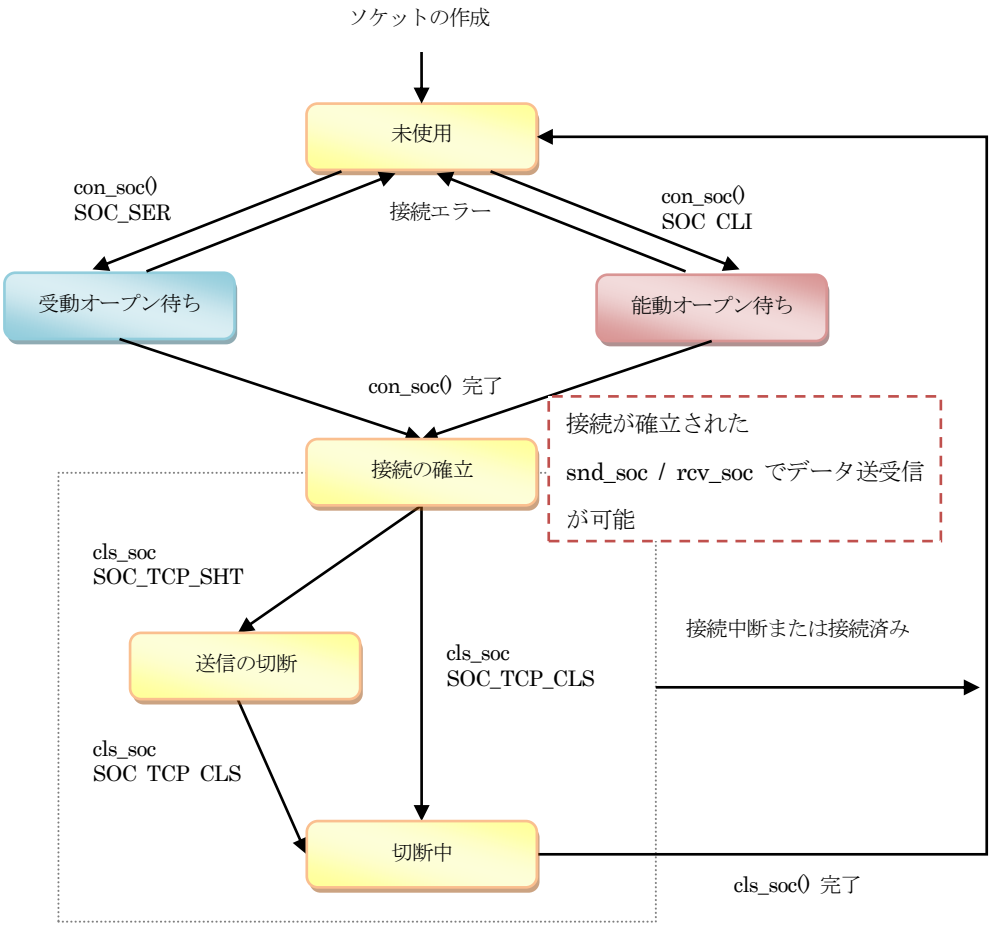
① UDP パケットが未受信なら、UPD パケットの受信待ちになります。この時、ソケットの受信タイムアウトを過ぎたら E_TMOUT を返します。

② 受信したパケットサイズが要求されたデータサイズよりも小さければ、アプリケーションのバッファにコピーします。受信したパケットサイズが要求されたデータサイズよりも大きいときは、要求サイズ分だけアプリケーションのバッファにコピーします。残ったデータは捨てられます。

③ デフォルトでは、受信データの最大サイズが 1472 バイト（CFG_PATH_MTU（1500 バイト）－IP ヘッダーサイズ －UDP ヘッダーサイズ)に設定されています。これ以上のサイズを受信する場合は、ネットワークバッファサイズの設定が必要です。詳細は、IP リアセンブリ・フラグメントの項目を参照してください。

### ３．１．４　　TCP モジュール

TCP は UDP と異なり、コネクション型ですので送信相手と通信路を確保しデータの送受信を行います。TCP のシーケンスは下図のようになります。




TCP のシーケンス

#### （1）接続の確立

TCP 接続には能動接続と、受動接続の二つのモードがあります。能動接続はリモートホストに自ら接続要求します。対して受動接続はリモートホストからの接続を待ちうけます。

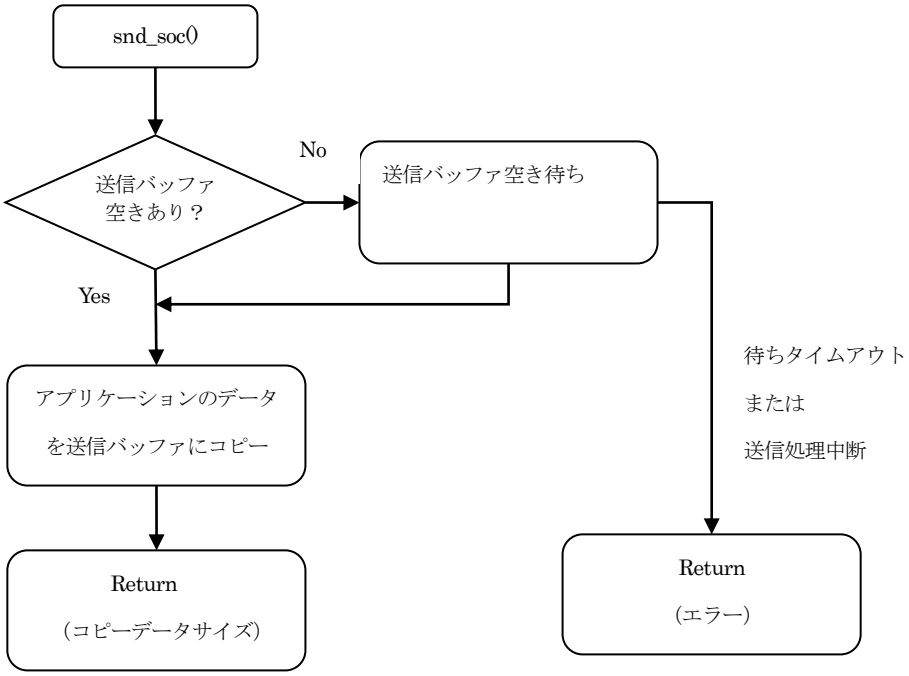
接続には con_soc()を使用し、SOC_CLI で能動接続、SOC_SER で受動接続を指定します。

#### （2）接続の終了

接続を切断するには　cls_soc()を使用します。完全に接続を切断するには SOC_TCP_CLS を送信のみ切断するには SOC_TCP_SHT を指定します。

### （3）データの送信

snd_soc()を使ってデータを送信します。snd_soc () の処理フローは下図のようになります。




**TCP ソケット　snd_soc の処理フロー**

① アプリケーションのデータを TCP 送信バッファにコピーします。コピーが成功したら TCP プロトコルがデータを送信します。リモートホストがデータを受信したら TCP 送信バッファにあるデータはクリアされます。

#### TCP 送信バッファ

送信バッファサイズは TCP ソケット作成時に指定する必要があります。バッファサイズは 4 バイトから 32 キロバイトの範囲で、2 の二乗の単位で指定します。

### （4）データの受信

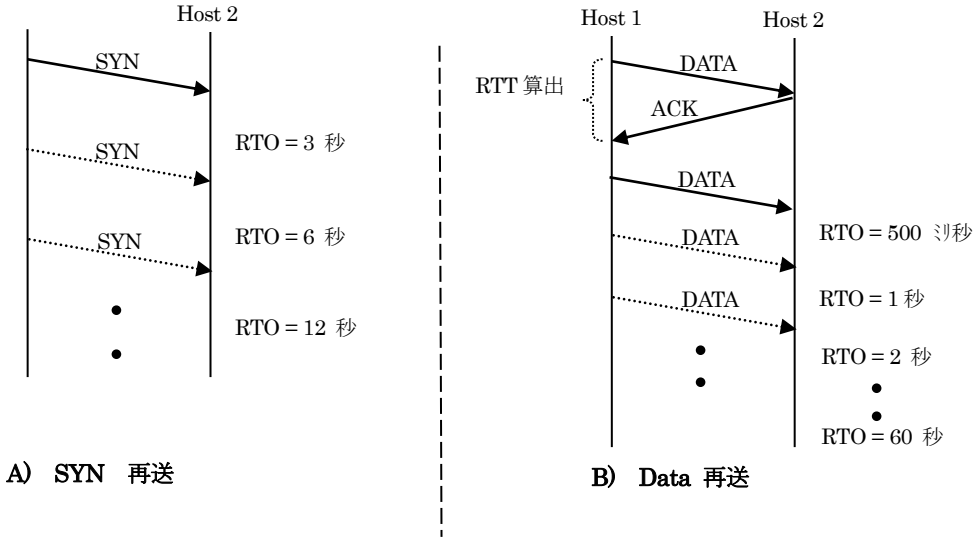
rcv_soc()を使ってデータを送信します。受信した TCP パケットは、まず TCP 受信バッファに登録されます。rcv_soc()が呼ばれたら TCP 受信バッファからアプリケーションのバッファにコピーされます。

#### TCP 受信バッファ（ウィンドウバッファ）

受信バッファサイズは TCP ソケット作成時に指定する必要があります。バッファサイズは 4 バイトから 32 キロバイトの範囲で、2 の二乗の単位で指定します。

### （5）再送タイムアウト

再送タイマのシーケンスは下図のようになります。




再送タイマ例

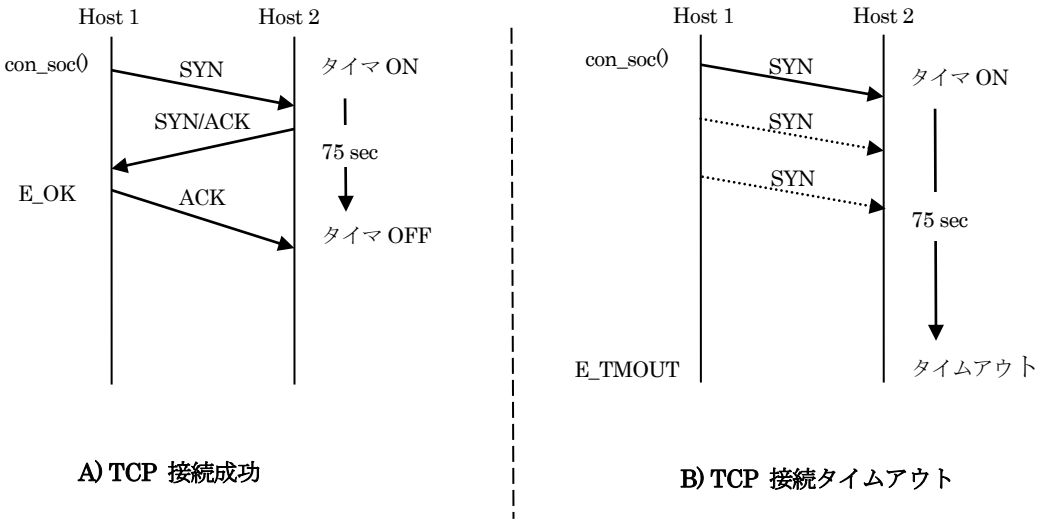
TCP では何らかの原因で一定時間の間 ACK パケットの応答が無い場合、応答が無かったセグメントを再送します。この再送するまでの待ち時間のことを「RTO」（Retransmission Time Out　再送タイムアウト）と呼びます。RTO の初期値は「RTT」（Round Trip Time）と呼ばれる「パケットが相手まで往復する時間」の「４倍＋α」となっています。RTO の値は再送を行うたびに２倍に増やされていきます。

**上図 A** の SYN 再送時、RTT 値は設定されていないので、CFG_TCP_RTO_INI（3 秒）を使用します。**上図 B** のデータ再送では前回の送信成功を元に計算された RTT 値、500 ミリ秒を使用しています。

RTO の範囲は CFG_TCP_RTO_MIN（500 ミリ秒)から CFG_TCP_RTO_MAX（60 秒)に設定されています。

### （6）接続タイムアウト

接続タイマシーケンスは下図のようになります。




接続タイムアウト例

con_soc()呼出し時に、このタイマは起動し 3 ウェイハンドシェイクがタイムアウトまでに完了すれば、E_OK を返します（**A**）。タイムアウトしたら E_TMOUT を返します（**B**）。

接続処理（3 ウェイハンドシェイク）のタイムアウト値は CFG_TCP_CON_TMO（75 秒）に設定されています。

**※TCP ソケットは作成時、接続用ブロキングタイムアウトを指定することができます。この値がタイムアウトした場合、接続処理は直ちに中断され con_soc()は E_TMOUT を返します。**

### （7）送信タイムアウト

送信タイムアウトは CFG_TCP_SND_TMO（64 秒)に設定されています。データ通信中、CFG_TCP_SND_TMO を経っても相手から応答がない場合は接続を切断します。

### （8）切断タイムアウト

切断処理のタイムアウトは CFG_TCP_CLS_TMO（64 秒)に設定されています。cls_soc()が CFG_TCP_CLS_TMO までに完了しなければ、接続は強制切断され cls_soc()は E_TMOUT を返します。

**※TCP ソケットは作成時、切断用ブロキングタイムアウトを指定することができます。この値がタイムアウトした場合、切断処理は直ちに中断され cls_soc()は E_TMOUT を返します。**

### （9）遅延 ACK タイムアウト

遅延 ACK タイムアウトは CFG_TCP_ACK_TMO (200 ミリ秒)に設定されています。

### （１０）TCP 輻輳制御

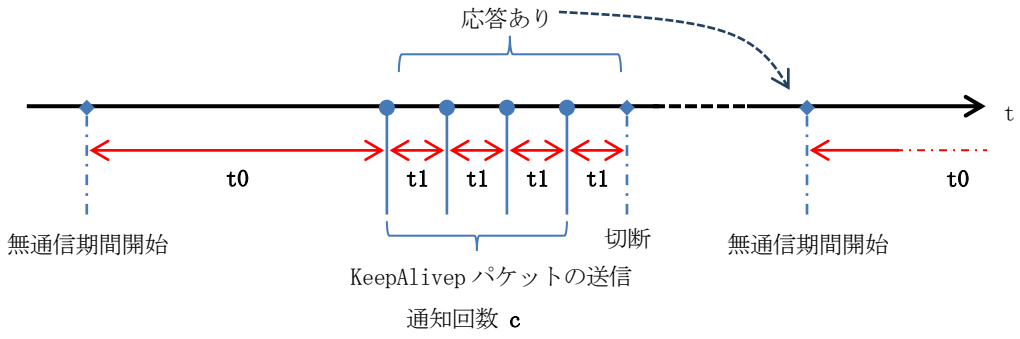
μNet3 は高速再送/高速復帰をサポートしています。重複 ACK 数は CFG_TCP_DUP_CNT (4)に設定されています。

### （１１）Maximum　Segment Size（MSS）

MSS は CFG_TCP_MSS　（1460 バイト）に設定されています。

### （１２）Keep Alive 機能

μNet3 は KeepAlive 機能をサポートしています。




$t_0$=KeepAlive 起動時間、　$t_1$=KeepAlive 通知間隔、　c=KeepAlive 通知回数

**KeepAlive パケット動作**

KeepAlive 機能が有効な場合(**c** > 0)、無通信状態で $t_0$ 時間を経過すると KeepAlive パケットを接続先ホストに送信します。その後接続先から応答を得るか、**c** 回の再送を繰り返すまで $t_1$ 時間間隔で KeepAlive パケットの送信を続けます。

KeepAlive パケットを **c** 回再送して応答が無い場合、接続先ホストとの TCP コネクションを切断します。接続先ホストから応答が有った場合、TCP コネクションの接続を維持します。（上図右の無通信期間開始に遷移）

KeepAlive 機能が無効な場合(**c** ＝0)、TCP コネクションは自動的に切断することはありません。

## ３．２　　ネットワーク・デバイスドライバ

μNet3 では各種ネットワークデバイスに対応できるようデバイスドライバの共通インタフェースを提供しています。このインタフェースに従い作成されたデバイスドライバはμNet3 で使用することが可能となります。

具体的に、プロトコルスタックは **T_NET_DEV 構造体**を通じてデバイスドライバにアクセスしますので、予め **T_NET_DEV 構造体**にデバイス名、デバイス番号、デバイスドライバの関数といった情報を登録しておきます。プロトコルスタックはデバイス番号により **T_NET_DEV** に登録されたデバイスを特定しアクセスします。

ネットワーク・デバイスドライバの詳細に関しては、「uNet3Ethernet ドライバインタフェース」ガイドを参照して下さい。

### ３．２．１　　デバイス構造体

```
typedef struct t_net_dev {
        UB      name[8];    /* デバイス名 */
        UH      num;        /* デバイス番号 */
        UH      type;       /* デバイスタイプ */
        UH      sts;        /*予約 */
        UH      flg;        /* 予約 */
        FP      ini;        /* dev_ini 関数へのポインタ*/
        FP      cls;        /* dev_cls 関数へのポインタ*/
        FP      ctl;        /* dev_ctl 関数へのポインタ*/
        FP      ref;        /* dev_ref 関数へのポインタ*/
        FP      out;        /* dev_snd 関数へのポインタ*/
        FP      cbk;        /* dev_cbk 関数へのポインタ*/
        UW      *tag;       /* 予約 */
        union   cfg;        /* MAC アドレス */
        UH      hhdrsz;     /* デバイスヘッダーサイズ */
        UH      hhdrofs;    /* ネットワークバッファ書き込み位置 */
        VP      opt;        /* ドライバ拡張領域（μNet3/ver3 以降)*/
} T_NET_DEV;
```

#### （1）デバイス番号

デバイスを特定するためにユニークな番号をセットします。プロトコルスタックはこの番号を使ってデバイスにアクセスします。**デバイス番号は必ず１から連番でつける必要があります。**

#### （2）デバイス名

デバイスを特定するために名前をセットします。デバイス名の長さは８バイト以内です。

例）eth0、eth1 など。

### （3）デバイスタイプ

ネットワークデバイスのタイプをセットします。以下のようなものがあります。

| デバイスタイプ | 意味 |
|---|---|
| NET_DEV_TYPE_ETH | Ethernet デバイス |
| NET_DEV_TYPE_PPP | PPP デバイス |

### （4）デバイスドライバ関数

デバイスドライバは以下に示す関数をサポートする必要があります。これらの関数はプロトコルスタックから適宜呼び出されます。

| プロトタイプ | 説明 | 必須 |
|---|---|---|
| ER dev_ini(UH dev_num) | デバイスの初期化 | 必須 |
| ER dev_cls(UH dev_num) | デバイスの解放 | 必須ではない |
| ER dev_snd(UH dev_num, T_NET_BUF *pkt) | パケットをネットワークに送信 | 必須 |
| ER dev_ctl(UH dev_num, UH opt, VP val) | デバイスの制御 | 必須ではない |
| ER dev_ref(UH dev_num, UH opt, VP val) | デバイス状態取得 | 必須ではない |
| void dev_cbk(UH dev_num, UH opt, VP val) | デバイスからのイベント通知（コールバック関数） | 必須ではない |

### （5）MAC アドレス

ハードウェアを特定するためのユニークな値をセットします。

```
union {
        struct {
            UB    mac[6];   /* MAC アドレス */
        }eth;
} cfg;
```

### ３．２．２　インタフェース

| dev_ini | デバイスの初期化 |
|---|---|

**【書式】**

ER ercd = dev_ini(UH dev_num);

**【パラメータ】**

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |

**【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 |
| E_OBJ | 既に初期済み |
| E_PAR | T_NET_DEV に不正な値が設定された |
| <0 | その他エラー（実装依存） |

**【解説】**

デバイスの初期化を行います。この関数はプロトコルスタックからデバイスを初期化するために呼ばれます。この関数が呼ばれる前に T_NET_DEV にデバイス情報が登録されている必要があります。

## dev_cls　　デバイスの解放

【書式】

ER ercd = dev_cls(UH dev_num);

【パラメータ】

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |

【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 |
| E_OBJ | 既に解放済み |

【解説】

デバイスを解放します。

## dev_ctl　　デバイスの制御

**【書式】**

ER ercd = dev_ctl(UH dev_num, UH opt, VP val);

**【パラメータ】**

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |
| UH | opt | 制御コード |
| VP | val | 設定する値 |

**【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 |
| E_PAR | 不正なパラメータ |
| E_OBJ | 既に解放済み |

**【解説】**

この関数の動作は実装依存になります。

## dev_ref　　デバイスの状態取得

**【書式】**

ER ercd = dev_ref(UH dev_num, UH opt, VP val);

**【パラメータ】**

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |
| UH | opt | 状態コード |
| VP | val | 取得する値 |

**【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 |
| E_PAR | 不正なパラメータ |
| E_OBJ | 既に解放済み |

**【解説】**

この関数の動作は実装依存になります。

## dev_snd　　パケットの送信

**【書式】**

ER ercd = dev_snd(UH dev_num, T_NET_BUF *pkt);

**【パラメータ】**

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |
| T_NET_BUF | *pkt | ネットワークバッファへのポインタ |

**【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_WBLK | パケットはキューへ登録された（エラーではない） |
| E_ID | デバイス番号が不正 |
| E_PAR | 不正なパラメータ |
| E_TMOUT | パケットの送信がタイムアウト |
| E_OBJ | 既にデバイス状態が不正 |

**【解説】**

この関数はパケットを Ethernet に送信します。

**実装例**

```
ER dev_snd(UH dev_num, T_NET_BUF *pkt)
{
  /* Ethernet フレーム(IP/TCP/UDP)をコピー */
  memcpy(txframe,  pkt->hdr,  pkt->hdr_len);
  /* ネットワークへ送信 */
  xmit_frame(txframe);
  return E_OK;
}
```

上の例では、プロトコルスタックの処理がデバイスドライバによってブロキングされてしまいます。次の例ではキューを使いブロッキングしない例を示します。

**ノンブロキング例**

```
ER dev_snd(UH dev_num, T_NET_BUF *pkt)
{
   queue_tx(pkt);        /* パケットをキューに登録 */
   return E_WBLK;   /* ノンブロッキング */
}

void queue_tx_task(void)
{
   dequeue_tx(pkt); /* キューからパケット取り出し */
   /* Ethernet フレーム(IP/TCP/UDP)をコピー */
   memcpy(txframe,   pkt->hdr,   pkt->hdr_len);
   xmit_frame(txframe);   /* ネットワークへ送信 */
   if (transmission timeout) {
      pkt->ercd = E_TMOUT;     /* タイムアウトセット */
   }
   net_buf_ret(pkt);
}
```

dev_snd では送信処理は行わず、パケットはキューに登録して E_WBLK を返します。実際のパケット送信処理は別タスクで行い、ネットワークバッファの解放もそこで行うようにします。

## dev_cbk　　デバイスのイベント通知

**【書式】**

void dev_cbk(UH dev_num, UH opt, VP val);

**【パラメータ】**

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |
| UH | opt | イベントコード |
| UH | val | イベント値 |

**【戻り値】**

なし

**【エラーコード】**

なし

**【解説】**

デバイスドライバからアプリケーションにイベントを通知するための関数です。この関数は実装依存です。

### ３．２．３　　パケットのルーティング

デバイスドライバから上位プロトコルスタックへパケットを転送するには、次の API を使用します。

**※この API はアプリケーションからは使用できません。**

| net_pkt_rcv | プロトコルスタックへパケット転送 |
|---|---|

**【書式】**

void net_pkt_rcv(T_NET_BUF *pkt);

**【パラメータ】**

| | | |
|---|---|---|
| T_NET_BUF | *pkt | ネットワークバッファへのポインタ |

**【戻り値】**

なし

**【エラーコード】**

なし

**【解説】**

この関数は上位プロトコルへのパケットを転送します。次の例ではデバイスドライバから上位プロトコルスタックへパケットを転送する例を示します。

**例**

```
/* ネットワークバッファの確保 */
T_NET_BUF *pkt;
net_buf_get(&pkt, len, TMO);

/* 受信した Ethernet ヘッダーをネットワークバッファへセット */
pkt->hdr = pkt->buf + 2;
pkt->hdr_len = ETH_HDR_SZ;
memcpy(pkt->hdr, rx_frame, pkt->hdr_len);

/* 受信した IP ペイロードをネットワークバッファへセット */
pkt->dat = pkt->hdr + pkt->hdr_len;
pkt->dat_len = rx_frame_len – pkt->hdr_len;
memcpy(pkt->dat, rx_frame + pkt->hdr_len, pkt->dat_len);

/* デバイス情報のセット*/
```

```
pkt->dev = dev;

/* プロトコルスタックへネットワークバッファの送信 */
net_pkt_rcv(pkt);
```

ネットワークバッファの解放は net_pkt_rcv()内で行われます。net_pkt_rcv()はタスクコンテキストから呼ばれなければなりません。

### ３．２．４　　ループバックインタフェース

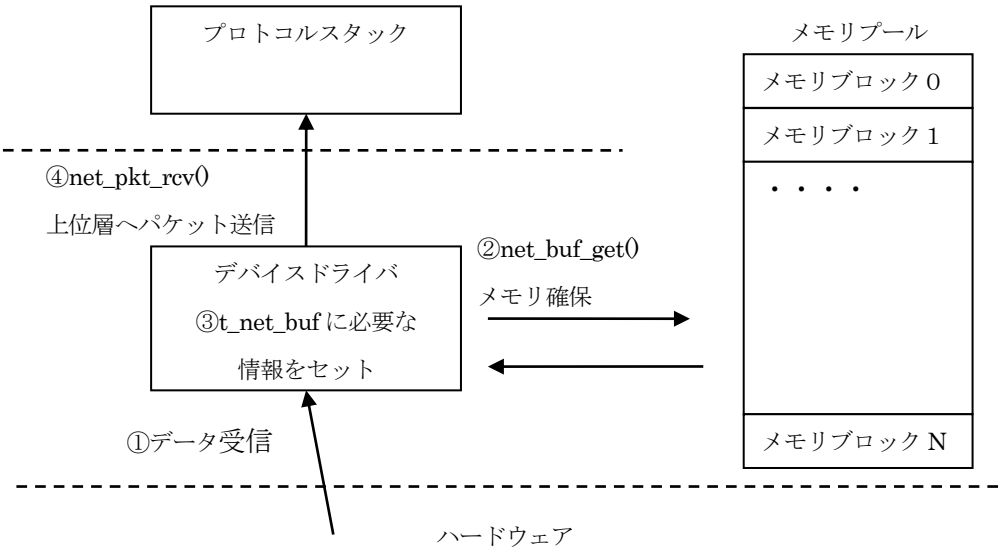
μNet3 ではパケットをネットワーク・デバイスドライバで折り返すループバックインタフェースを提供します。DDR_LOOPBACK_NET.c を使用することで、そのインタフェースから送信するパケットはμNet3 に通知されます。

一般の(127.0.0.1 で表される) ループバックインタフェースとは異なり、μNet3 では静的な IP アドレスと MAC アドレスを設定します。μNet3/Compact でループバックインタフェースを使用するには、コンフィグレータのインタフェースの登録時にデバイスタイプに「Loopback」を選択します。

ループバックインタフェースからパケットを受信するには、そのパケットの宛先が割り当てた IP アドレスでなければなりません。またループバックインタフェースを使用する場合でも ARP を使ったアドレス解決が行われます。

## ３．３　　メモリ管理

プロトコルスタックではメモリ管理にネットワークバッファを使用しています。ネットワークバッファを使うことにより動的にメモリの空きブロックを確保することが可能になります。下図にメモリ確保の例を示します。まず始めにハードウェアからデータを受信したデバイスドライバはネットワークバッファ API を使用して、メモリを確保(net_buf_get)します。次に確保したメモリに必要情報をセットし、上位層のプロトコルスタックへパケットを送信(net_pkt_rcv)します。




メモリ確保例の図

### ３．３．１　　ネットワークバッファ

μNet3/Compact ではブロックサイズが 1472byte の固定長メモリプールを最大 8 つ使用しています。ネットワークバッファはこのメモリプールからメモリを確保したり、解放したりする仕組みを提供します。

**ネットワークバッファの構造（T_NET_BUF）**

```
typedef struct t_net_buf {
        UW          *next;      /* 予約 */
        ID          mpfid;      /* メモリプール ID */
        T_NET       *net;       /* ネットワークインタフェース */
        T_NET_DEV   *dev;       /* ネットワークデバイス */
        T_NET_SOC   *soc;       /* ソケット */
        ER          ercd;       /* エラーコード */
        UH          flg;        /* プロトコルスタック制御用フラグ */
        UH          seq;        /* フラグメントシーケンス */
        UH          dat_len;    /* パケットのデータサイズ */
        UH          hdr_len;    /* パケットのヘッダーサイズ */
        UB          *dat;       /* パケット(buf)内のデータ位置を指す */
        UB          *hdr;       /* パケット(buf)内のヘッダー位置を指す */
        UB          buf[];      /* 実際のパケット */
} T_NET_BUF ;
```

各プロトコル間、プロトコルとデバイスドライバ間のパケットの送受信には、**T_NET_BUF** を使用します。

TCP/IP で実際のパケットデータは‘**buf**’に格納されており、‘***dat**’、‘***hdr**’、‘**hdr_len**’‘**dat_len**’は それにアクセスするために使用します。

### ３．３．２　ネットワークバッファ API

※このネットワークバッファ API はアプリケーションから使用することはできません。

| net_buf_get | ネットワークバッファの確保 |
|---|---|

【書式】

ER ercd = net_buf_get(T_NET_BUF **buf, UH len, TMO tmo);

【パラメータ】

| | | |
|---|---|---|
| T_NET_BUF | **buf | メモリ確保するバッファのアドレス |
| UH | len | 確保するバイト数 |
| UH | tmo | タイムアウト指定 |

【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 | 不正なパラメータ値が設定された |
| E_NOMEM | メモリ確保できない |
| E_TMOUT | タイムアウト |

【解説】

メモリプールよりメモリを確保します。確保したメモリのアドレスは buf に返します。

## net_buf_ret　　ネットワークバッファの解放

**【書式】**

void net_buf_ret(T_NET_BUF *buf) ;

**【パラメータ】**

| | | |
|---|---|---|
| T_NET_BUF | *buf | メモリ解放するバッファのアドレス |

**【戻り値】**

なし

**【エラーコード】**

なし

**【解説】**

メモリプールにメモリを返却します。ネットワークバッファとソケットが関連づけられている場合は、ソケットにメモリ解放イベントを通知します。

## ３．４　　メモリＩ/O 処理

μNet3 は動作するデバイスやコンパイル環境に依存しないように、プロトコル処理で発生する連続したメモリへの書き込みや、比較処理はユーザ側で定義します。たとえば DMA 機能を備えたデバイスの場合、標準ライブラリ関数である memcpy()は使用せず、DMA 転送でメモリコピーを実行することができます。

(※本機能はμNet3 のバージョン 2.0 以降に限ります。)

### ３．４．１　　メモリＩ/O API (uNet3 ver.2.xx)

| net_memset | メモリの値設定 |
|---|---|

【書式】

VP net_memset(VP d, int c, UINT n);

【パラメータ】

| | | |
|---|---|---|
| VP | d | 設定するメモリの先頭アドレス |
| int | c | 設定する値 |
| UINT | n | 設定バイト数 |

【戻り値】

| | | |
|---|---|---|
| VP | d | 設定するメモリの先頭アドレス |

【解説】

メモリの設定が正常に終了した場合は、引数で指定されるメモリの先頭アドレスを返却して下さい。

## net_memcpy　　メモリのコピー

【書式】

VP net_memcpy(VP d, VP s, UINT n);

【パラメータ】

| | | |
|---|---|---|
| VP | d | コピー先アドレス |
| VP | s | コピー元アドレス |
| UINT | n | コピーバイト数 |

【戻り値】

| | | |
|---|---|---|
| VP | d | コピー先アドレス |

【解説】

メモリのコピーが正常に終了した場合は、引数で指定されるコピー先アドレスを返却して下さい。

## net_memcmp　　メモリの比較

【書式】

int net_memcmp(VP d, VP s, UINT n);

【パラメータ】

| | | |
|---|---|---|
| VP | d | 比較メモリアドレス 1 |
| VP | s | 比較メモリアドレス 2 |
| UINT | n | 比較バイト数 |

【戻り値】

| | | |
|---|---|---|
| int | | 比較結果 |

【解説】

両メモリから指定されたバイト数分、同じ値の場合は 0 を返却して下さい。そうでない場合は、非 0 を返却して下さい。

### ３．４．２　メモリ I/O API (uNet3 ver.3.xx)

| net_memset | メモリの値設定 |
|---|---|

【書式】

void* net_memset(void* d, int c, SIZE n);

【パラメータ】

| | | |
|---|---|---|
| void* | d | 設定するメモリの先頭アドレス |
| int | c | 設定する値 |
| SIZE | n | 設定バイト数 |

【戻り値】

| | | |
|---|---|---|
| void* | d | 設定するメモリの先頭アドレス |

【解説】

メモリの設定が正常に終了した場合は、引数で指定されるメモリの先頭アドレスを返却して下さい。

## net_memcpy　　メモリのコピー

【書式】

void* net_memcpy(void* d, const void* s, SIZE n);

【パラメータ】

| | | |
|---|---|---|
| void* | d | コピー先アドレス |
| const void* | s | コピー元アドレス |
| SIZE | n | コピーバイト数 |

【戻り値】

| | | |
|---|---|---|
| void* | d | コピー先アドレス |

【解説】

メモリのコピーが正常に終了した場合は、引数で指定されるコピー先アドレスを返却して下さい。

## net_memcmp　　メモリの比較

【書式】

int net_memcmp(const void* d, const void* s, SIZE n);

【パラメータ】

| | | |
|---|---|---|
| const void* | d | 比較メモリアドレス 1 |
| const void* | s | 比較メモリアドレス 2 |
| SIZE | n | 比較バイト数 |

【戻り値】

| | | |
|---|---|---|
| int | | 比較結果 |

【解説】

両メモリから指定されたバイト数分、同じ値の場合は 0 を返却して下さい。そうでない場合は、非 0 を返却して下さい。

# 第４章　コンフィグレーション

## ４．１　　μNet3/Compact(μNet3 ver.1.xx)のコンフィグレーション

μNet3/Compact を使用する場合、システム設計で決定される各オブジェクトのパラメータとなるコンフィグレーション情報をコンフィグレータに入力し、必要なソースファイルとアプリケーションプログラムのテンプレートになるスケルトンコードを生成することが可能です。

本章では TCP/IP プロトコルスタックのコンフィグレーションについて説明しています。カーネル、デバイスに関するコンフィグレーションについては「μC3/Compact ユーザーズガイド」を参照して下さい。

### ４．１．１　　コンフィグレータの起動

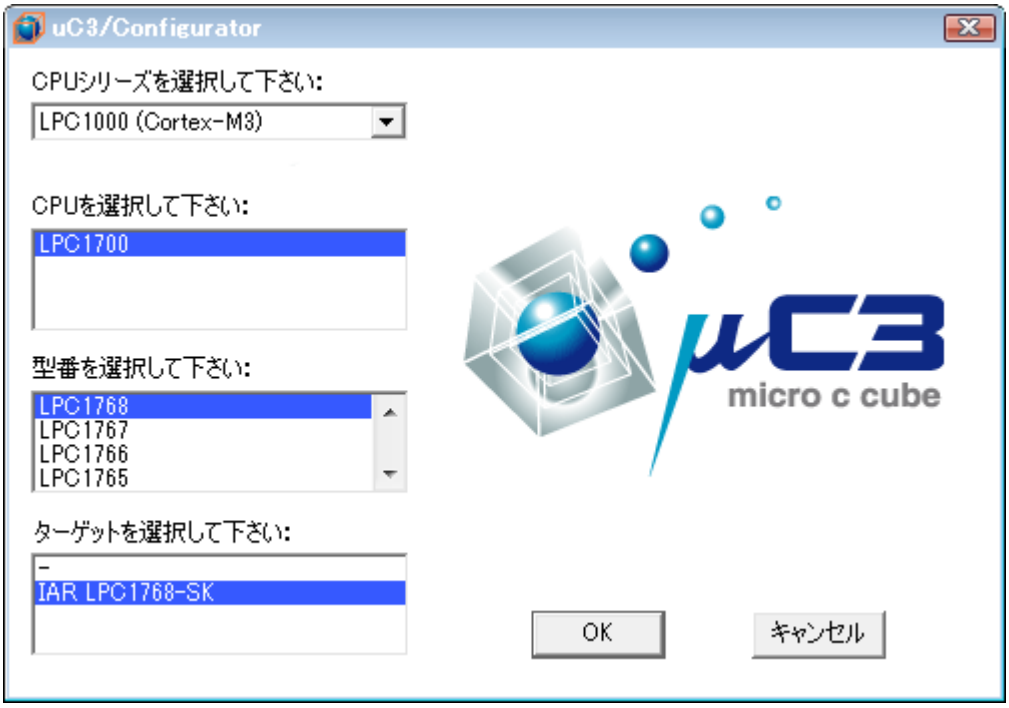
「uC3conf.exe」をダブルクリックし、起動してください。

#### A．新規でプロジェクトを生成する場合

「新規プロジェクトの生成」を選択後に「OK」をクリックし、「CPU 選択」へ進みます。

**CPU 選択**

リストよりＣＰＵのシリーズと型番を選択後に「OK」をクリックし、「メイン画面」へ進みます。

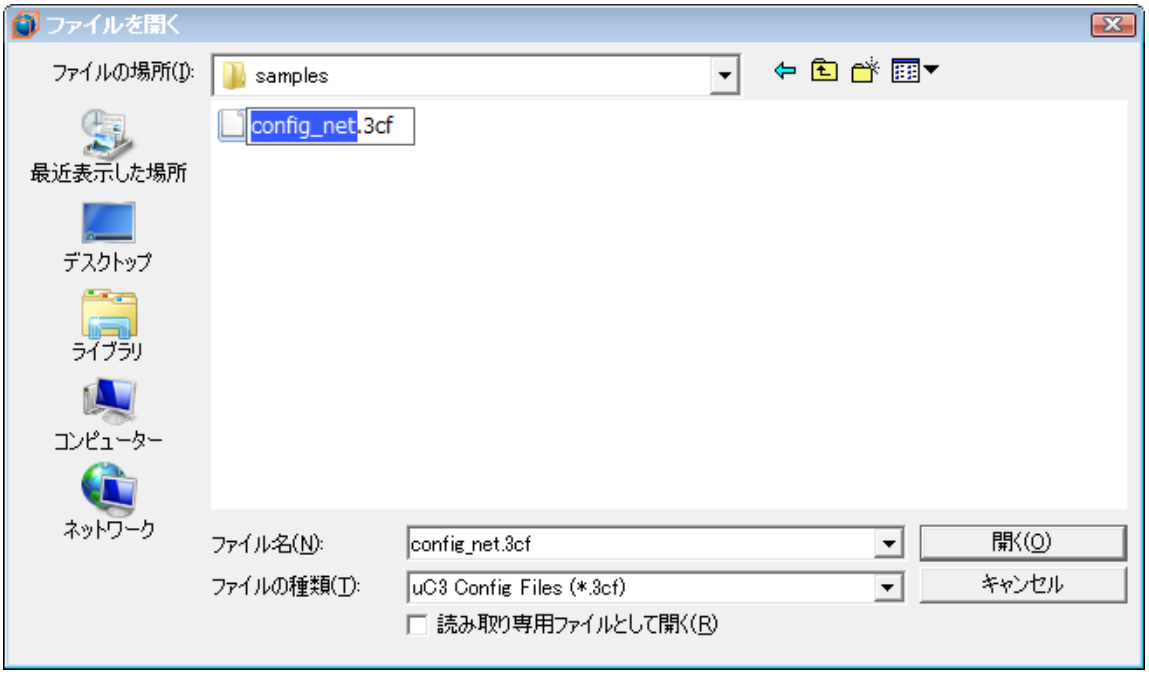


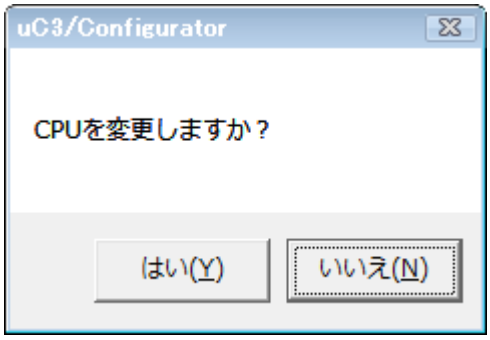
### B．既存のプロジェクトを開く場合

「既存のプロジェクトを開く」を選択後に「OK」をクリックし、「ファイルを開く」へ進みます。


### ファイルを開く

保存されていたプロジェクトファイル（拡張子.3cf）を選択後に「開く」をクリックし、「メイン画面」へ進みます。






CPU 変更を選択する画面が表示されたら「いいえ」をクリックしてください。

### C．メイン画面

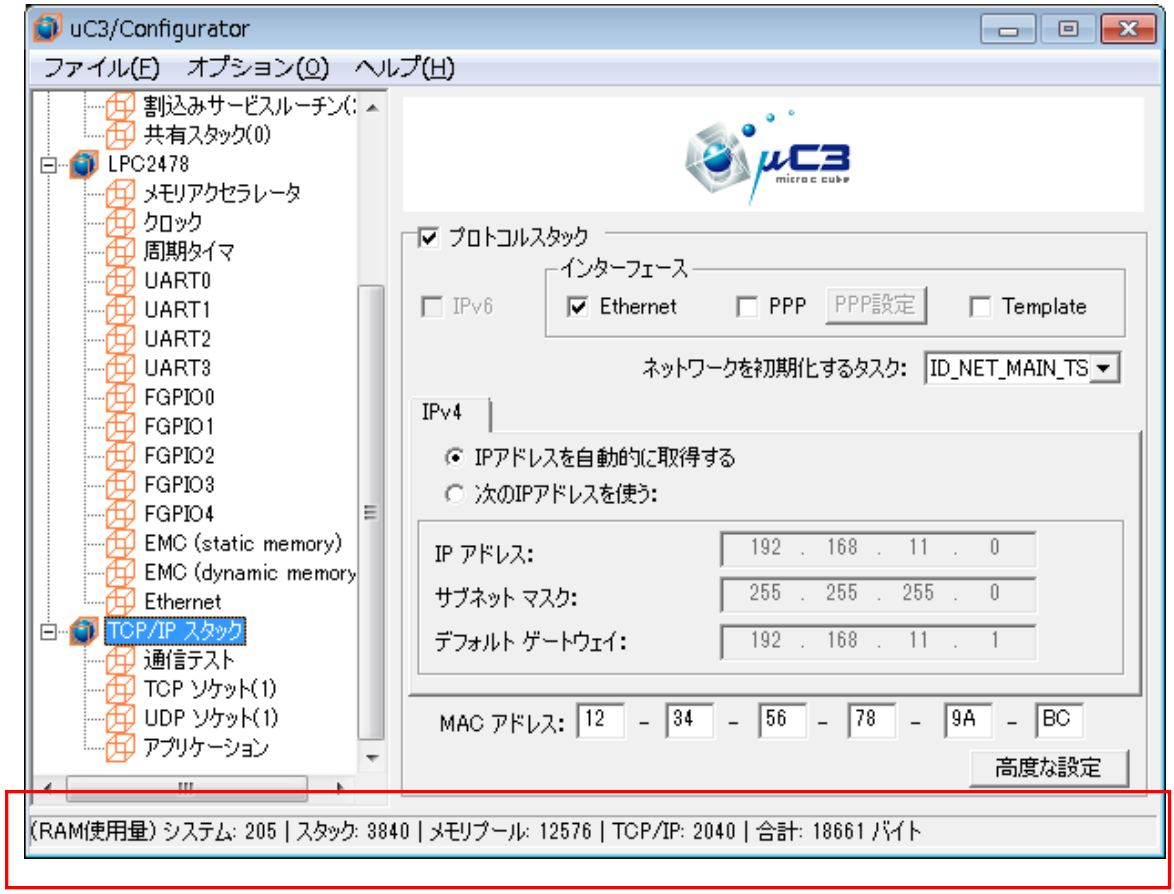
起動後はプロジェクトの参照と編集が可能なメイン画面になります。ツリー表示のオブジェクト名をクリックすることで各オブジェクトのコンフィグレーション画面に切り替えることができます。

ここには、TCP/IP プロトコルスタックやカーネル、プロセッサ依存部などのコンフィグレーションがあります。

## ４．１．２　　TCP/IP プロトコルスタックの設定

TCP/IP プロトコルスタックには、TCP/IP プロトコルスタックの全体コンフィグレーションと TCP ソケット、UDP ソケット、アプリケーションのコンフィグレーションがあります。TCP ソケット、UDP ソケットのコンフィグレーション画面では、１つのソケットが１つのタブに対応しています。

下端のステータスバーには、システムで RAM メモリの使用量が常に表示されます。以下はコンフィグレーション画面の例です。

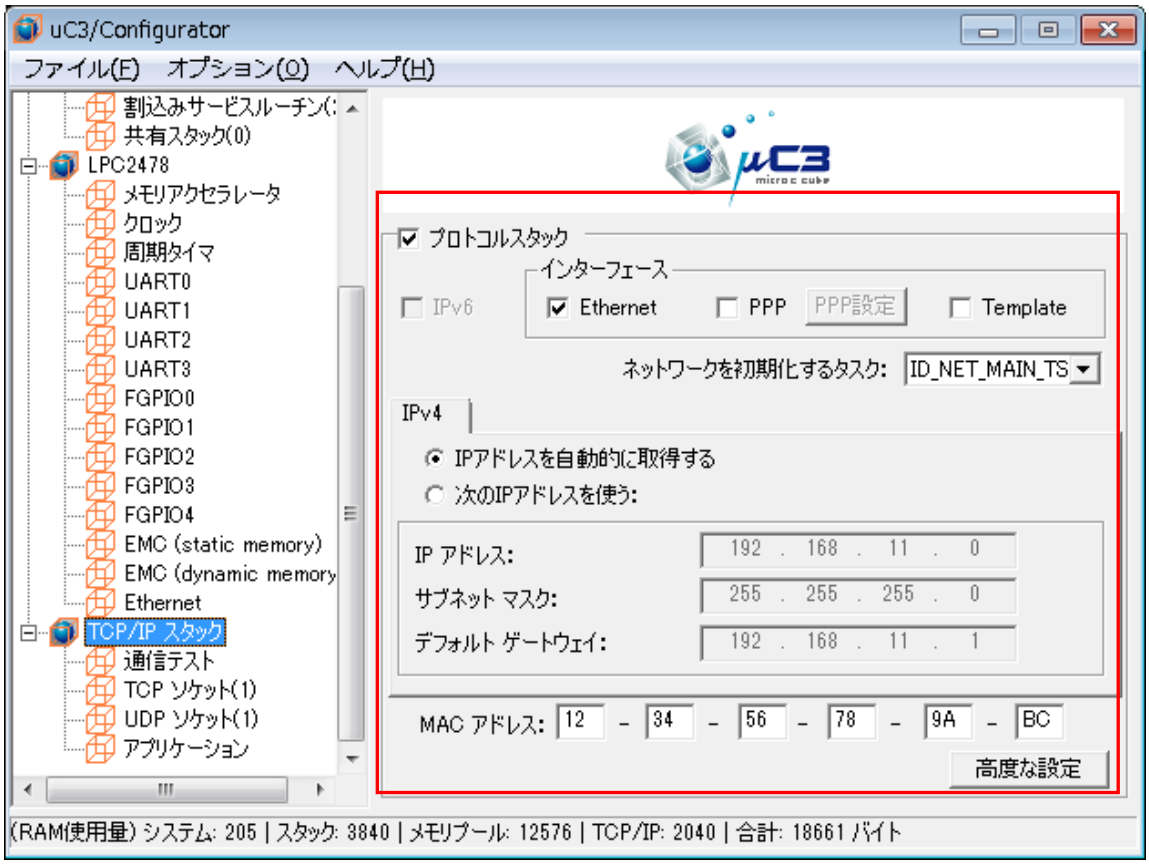



**RAM 使用量**

| 項目 | 内容 |
|---|---|
| システム | カーネル自体が使用するメモリ、オブジェクトの管理領域、データキューのバッファ |
| スタック | システムスタック、タスク個別のスタック、共有スタック |
| メモリプール | 固定長メモリプールのメモリプール領域 |
| TCP/IP | TCP/IP プロトコルスタックで使用するソケット等の使用量 |

## ４．１．３　　全般のコンフィグレーション

ツリー表示の TCP/IP スタックをクリックすると、TCP/IP プロトコルスタック全般のコンフィグレーション画面が表示されます。




### プロトコルスタック

プロトコルスタックの使用有無をチェックボックスで指定してください。チェックを ON にするとプロコルスタックが有効になります。OFF にすると無効になります。

**※OFF にすると、今まで設定した内容がクリアされますのでご注意ください。**

### インタフェース

使用するインタフェースを選択します。Ethernet インタフェースを使用する場合は、Ethernet をチェックします。

また独自のネットワークデバイスを実装する場合は、Template をチェックすることでμNet3 とのインタフェースが実装された空のネットワーク・デバイスドライバが使用できます。

ここで選択されたインタフェースはソケットを生成する際に選択可能になります。

### ネットワークを初期化するタスク

ネットワークを初期化するタスクを選択します。ここは、指定を外すことが可能になっています。指定を外した場合、ネットワーク初期化の処理「net_setup()」を、追記することが必要

となります。ネットワークを初期化するタスクのスタックサイズは 300 バイト以上に設定してください。

**ホスト設定-IP アドレス**

ホストの IP アドレスを指定してください。“IP アドレスを自動的に取得する”を選択すると、DHCP を用いて自動的に IP アドレスを設定します。この時、DHCP 用の UDP ソケットが自動的に追加されます。“次の IP アドレスを使う”を選択した場合は固定 IP アドレスを指定してください。設定値は Ethernet インタフェースに対して有効となります。

**ホスト設定-MAC アドレス**

ホストの MAC アドレスを指定してください。入力は各オクテット単位で行います。設定値は Ethernet インタフェースに対して有効となります。

**「高度な設定」ボタン**

「PATH MTU サイズ」及び「ネットワークバッファ数」を指定する画面が表示されます。


**PATH MTU サイズ**

PATH MTU を指定してください。本書の「第２章　μNet3/Compact の基本概念」及び「第３章 μNet3/Compact の機能概要」を参照して、設定を行ってください。

【補足】

PATH MTU サイズの設定が、固定長メモリプール”ID_TCP_MPF”の「メモリブロックサイズ」に自動的に反映されます。

メモリブロックサイズは、下記により計算されています。

メモリブロックサイズ ＝PATH_MTU＋ 管理情報サイズ

**ネットワークバッファ数**

ネットワークバッファ数を指定してください。この数値は、TCP/IP プロトコルスタックが使用するネットワークバッファのメモリブロック数を指定します。本書の「第３章　μNet3/Compact の機能概要」の「３．３　ネットワークバッファ」を参照して、設定を行ってください。

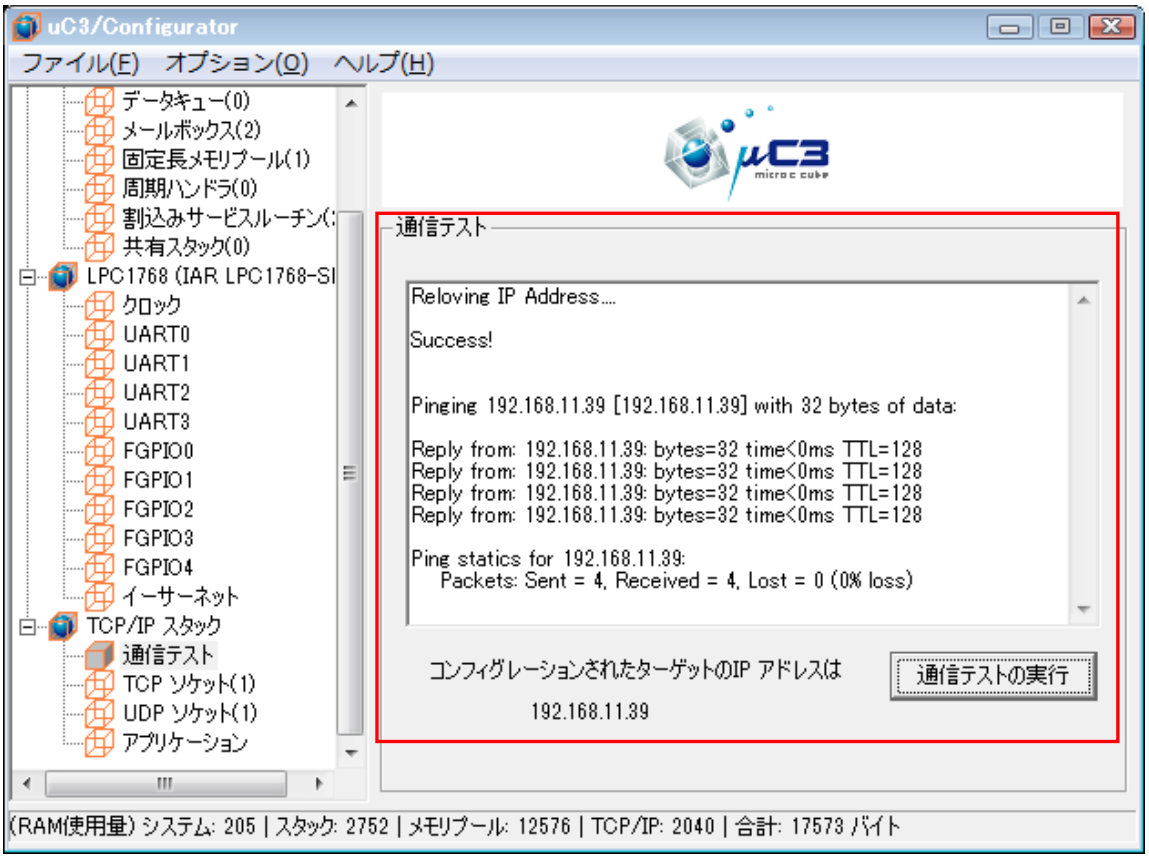
【補足】

ネットワークバッファ数の設定が、固定長メモリプール”ID_TCP_MPF”の「メモリブロック数」に自動的に反映されます。

## ４．１．４　　通信テスト

ツリー表示の通信テストをクリックすると、通信テスト画面が表示されます。

ここではコンフィグレーションは行わず、ターゲットに対して通信テストを行います。



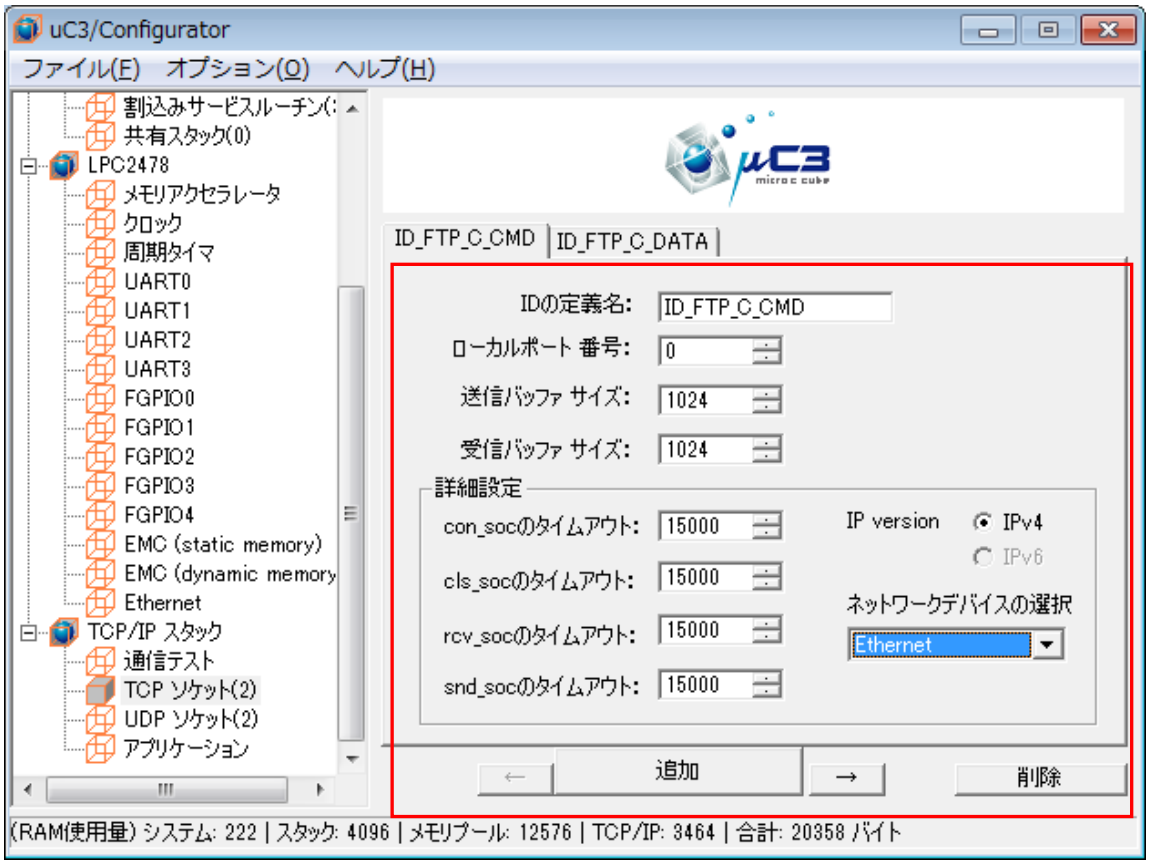

**「通信テストの実行」ボタン**

４．３で指定したホスト IP アドレスに対して PING を用いて通信テストを行います。通信テストを行うにはターゲットにプログラムが実装されている必要があります。通信テスト結果は画面に表示されます。

**※PC にウィルスセキュリティソフトがイントールされている場合、ターゲット側プログラムが正常に動作しているにも関わらず失敗することがあります、その際はウィルスセキュリティソフトのファイルフォール設定を無効にしてから通信テストを行ってください。**

## ４．１．５　　TCP ソケットのコンフィグレーション

ツリー表示の TCP ソケットをクリックすると、TCP ソケットのコンフィグレーション画面が表示されます。




### ID の定義名

ソケットの ID 番号を表す任意の定義名を指定してください。この定義名は net_id.h 内でマクロ定義されます。

### ローカルポート番号

ソケットのポート番号を指定してください。

### 送信バッファサイズ

ソケットの送信バッファサイズを指定してください。

### 受信バッファサイズ

ソケットの受信バッファサイズを指定してください。

### con_soc のタイムアウト

con_soc　API のタイムアウト時間をミリ秒（ms）単位で指定してください。－1 を指定する

と con_soc が正常終了かエラーになるまで返りません。この設定はブロッキングモード時のみ有効です。

### cls_soc のタイムアウト

cls_soc API のタイムアウト時間をミリ秒（ms）単位で指定してください。－1 を指定すると cls_soc が正常終了かエラーになるまで返りません。この設定はブロッキングモード時のみ有効です。

### rcv_soc のタイムアウト

rcv_soc API のタイムアウト時間をミリ秒（ms）単位で指定してください。－1 を指定すると rcv_soc が正常終了かエラーになるまで返りません。この設定はブロッキングモード時のみ有効です。

### snd_soc のタイムアウト

snd_soc API のタイムアウト時間をミリ秒（ms）単位で指定してください。－1 を指定すると snd_soc が正常終了かエラーになるまで返りません。この設定はブロッキングモード時のみ有効です。

### ネットワークデバイスの選択

インタフェースでチェックしたネットワークデバイスを選択します。ソケットは必ず一つのネットワークデバイスに関連付ける必要があります。もし何も選択されていない場合はソケット生成時にネットワークデバイスを特定しません。この場合の動作は５．４　ソケット API を参照して下さい。

### 「追加」ボタン

新しいソケットとそれに対応するタブを追加します。

### 「削除」ボタン

現在選択されているタブのソケットを削除します。

### 「←」ボタン

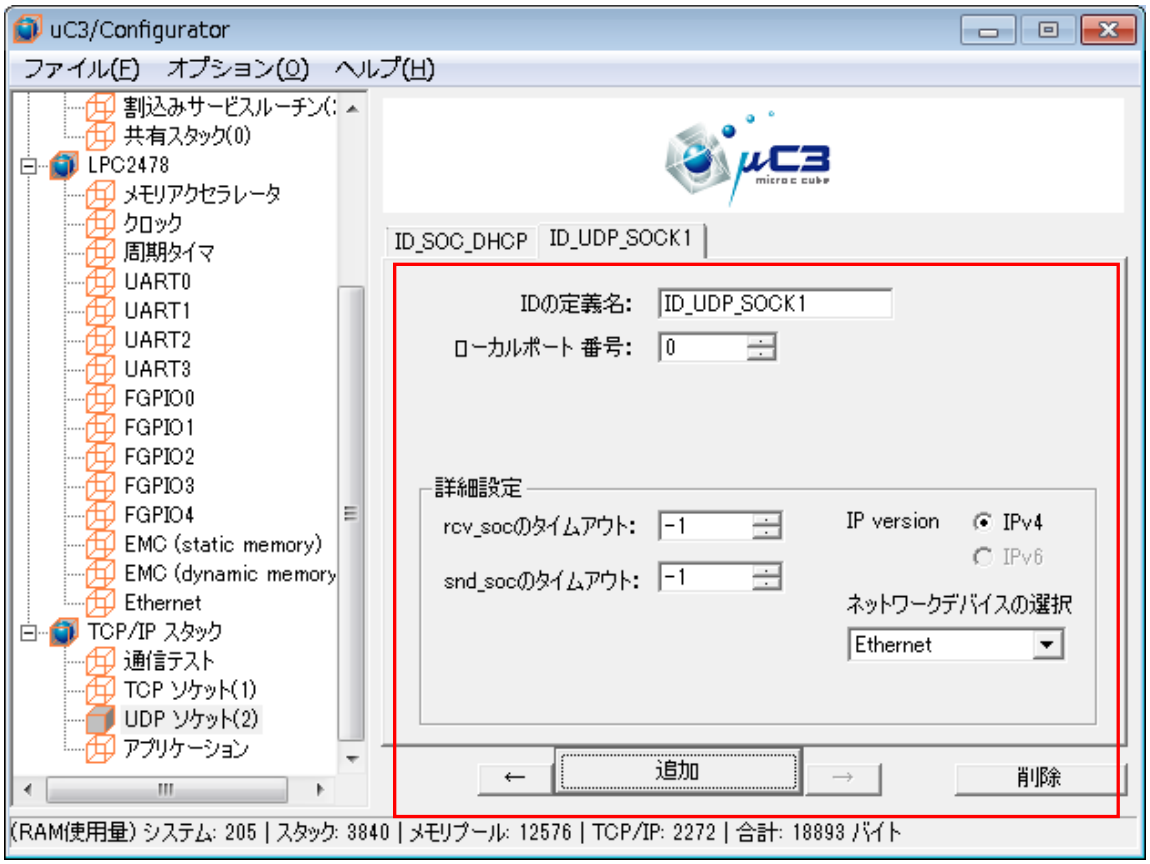
現在選択されているタブを左へ移動します。

### 「→」ボタン

現在選択されているタブを右へ移動します。

## ４．１．６　　UDP ソケットのコンフィグレーション

ツリー表示の UDP ソケットをクリックすると、UDP ソケットのコンフィグレーション画面が表示されます。




### ID の定義名

ソケットの ID 番号を表す任意の定義名を指定してください。この定義名は net_id.h 内でマクロ定義されます。

### ローカルポート番号

ソケットのポート番号を指定してください。

### rcv_soc のタイムアウト

rcv_soc　API のタイムアウト時間をミリ秒（ms）単位で指定してください。－1 を指定すると rcv_soc が正常終了かエラーになるまで返りません。この設定はブロッキングモード時のみ有効です。

### snd_soc のタイムアウト

snd_soc API のタイムアウト時間をミリ秒（ms）単位で指定してください。－1 を指定すると snd_soc が正常終了かエラーになるまで返りません。この設定はブロッキングモード時のみ有効です。

### ネットワークデバイスの選択

インタフェースでチェックしたネットワークデバイスを選択します。ソケットは必ず一つのネットワークデバイスに関連付ける必要があります。もし何も選択されていない場合はソケット生成時にネットワークデバイスを特定しません。この場合の動作は５．４　ソケット API を参照して下さい。

### 「追加」ボタン

新しいソケットとそれに対応するタブを追加します。

### 「削除」ボタン

現在選択されているタブのソケットを削除します。

### 「←」ボタン

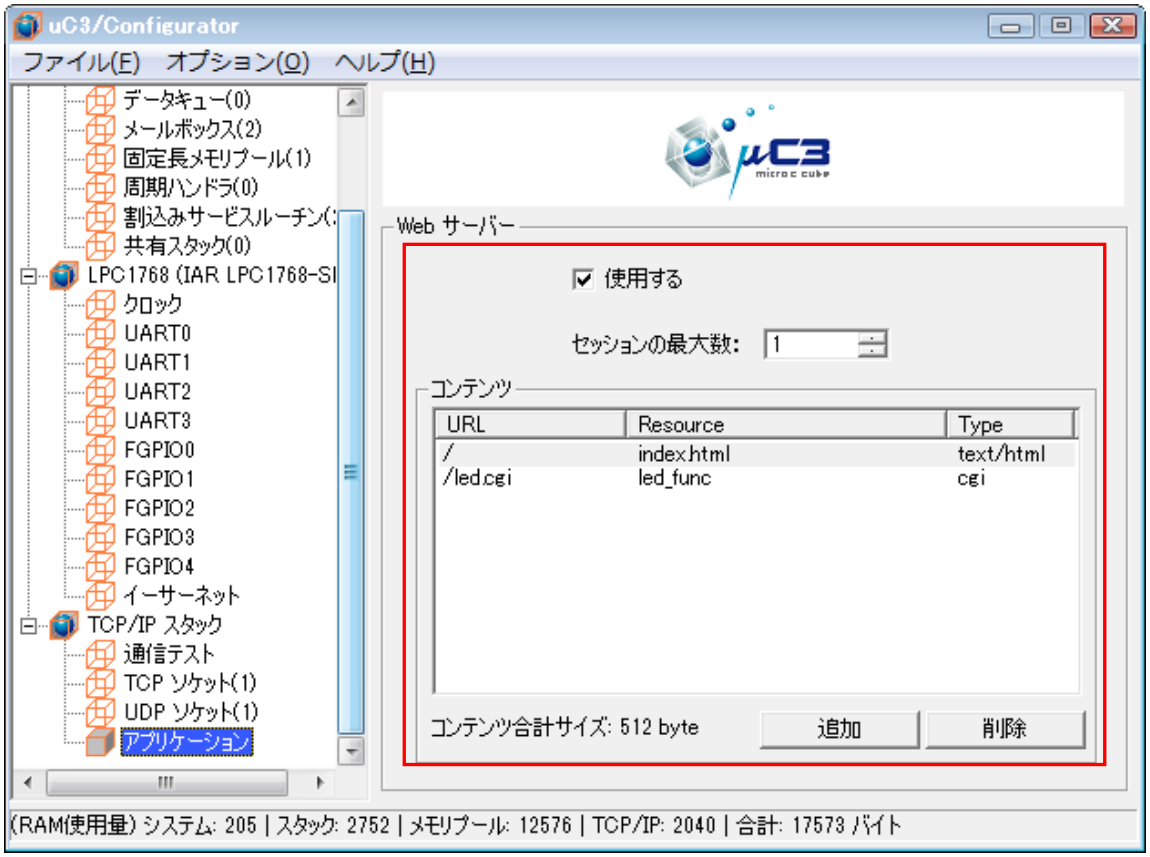
現在選択されているタブを左へ移動します。

### 「→」ボタン

現在選択されているタブを右へ移動します。

## ４．１．７　　アプリケーションのコンフィグレーション

ツリー表示のアプリケーションをクリックすると、アプリケーションのコンフィグレーション画面が表示されます。このコンフィグレーションを行うことにより簡単に WEB サーバーアプリケーションを作成することができます。




### WEB サーバーの使用有無

WEB サーバーの使用有無を指定してください。チェックを ON にすると WEB サーバーが有効になります。OFF にすると無効になります。WEB サーバーを有効にすると HTTP 用 TCP ソケットが自動的に追加されます。

### セッションの最大数

WEB サーバーへの接続するセッションの上限を指定してください。指定できる最大値は 2 です。

### コンテンツ一覧

現在登録されているコンテンツが表示されています。登録できるコンテンツ数は最大 50 個です。コンテンツ一覧上で変更したいコンテンツをマウス左ダブルクリックすると、コンテンツを変更することができます。

### コンテンツ合計サイズ

現在登録されているコンテンツの合計サイズが表示されます。このサイズが ROM サイズ上限を超えないようにしてください。

### 「追加」ボタン

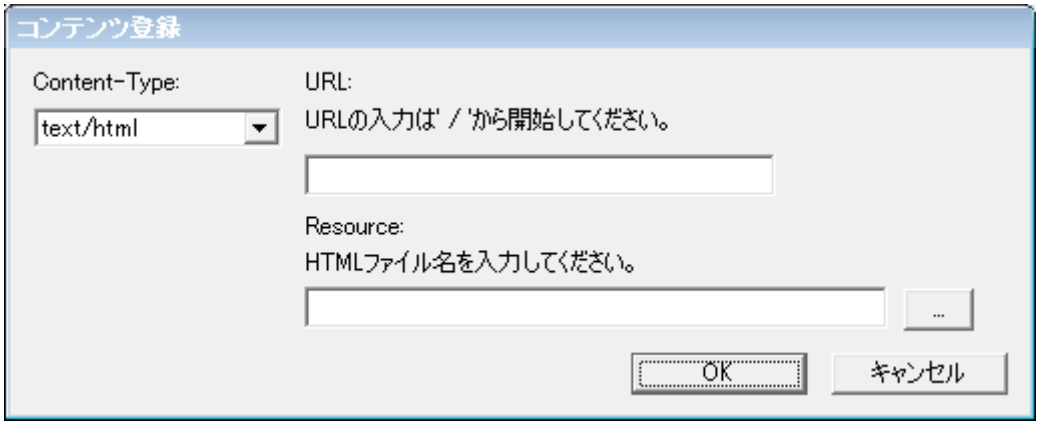
新しいコンテンツを追加します。

### 「削除」ボタン

現在選択されているコンテンツを削除します。

### コンテンツの追加、変更

「追加」ボタンをクリックするか、一覧に登録されているコンテンツをダブルクリックすると次の登録画面が表示されます。




#### 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）を指定してください。コンテンツタイプは次の中から選択します。

text/html
image/gif
image/jpeg
cgi

#### URL

コンテンツの URL を指定してください。URL の入力は‘ / ’から開始してください。入力可能文字数は 12 文字までです。

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てください。

- 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てください。もしくは「…」ボタンをクリックすると“ファイル選択画面”が表示されますので、そこでファイルを指定することもできます。
- 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てください。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：　" 　`{}*@;+:*,.#$%&'¥"!?~^=|/¥<>()"**

OK ボタン

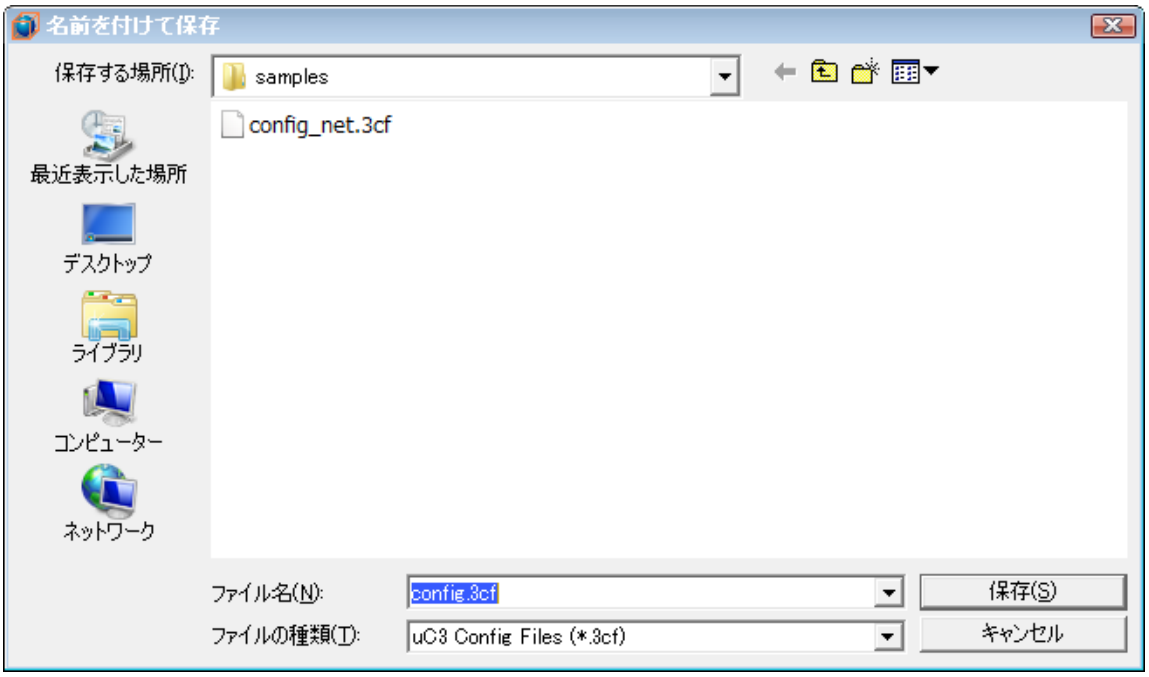
コンテンツを登録します。

キャンセルボタン

コンテンツ登録をせずに画面を閉じます。

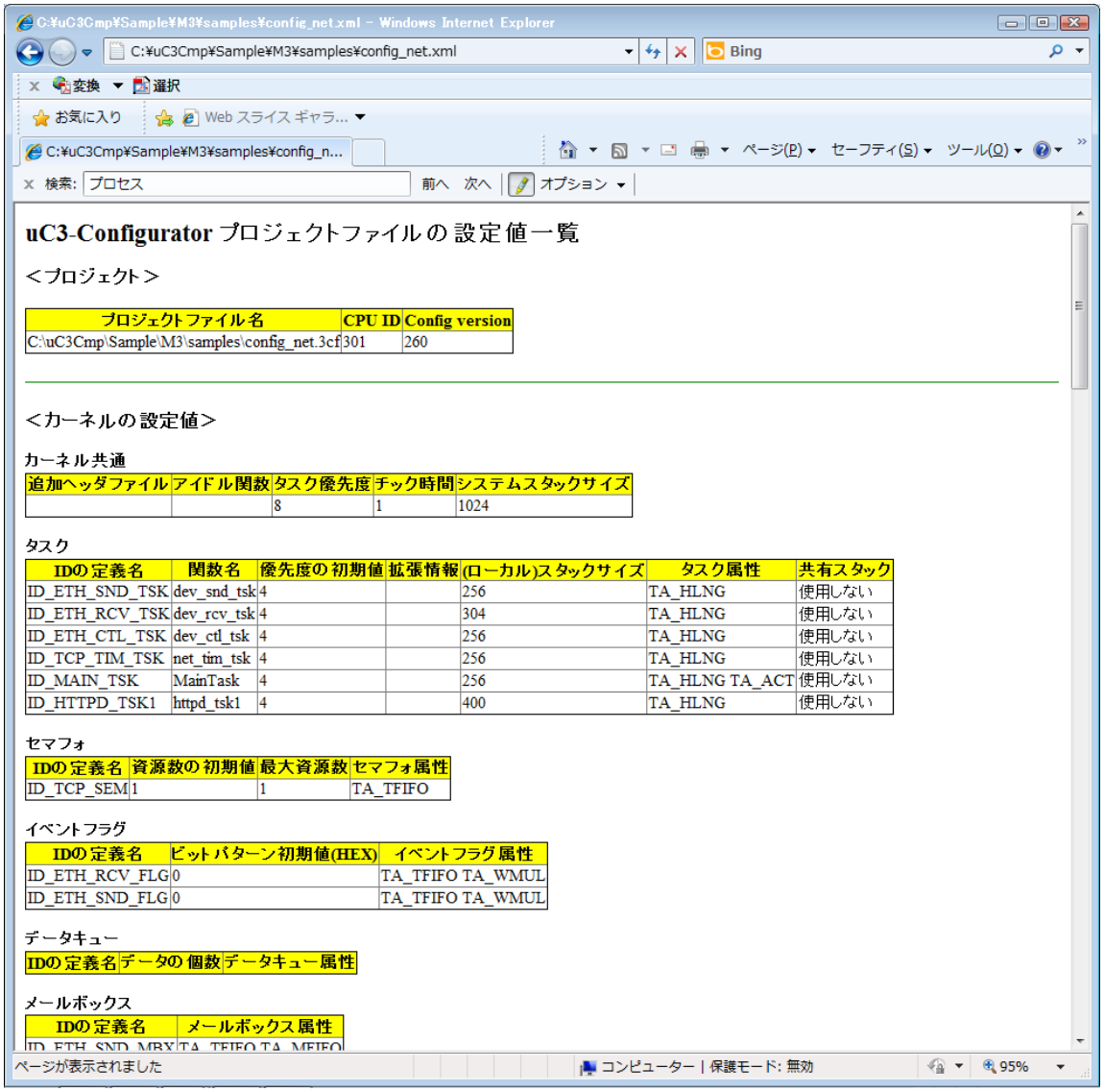
## ４．１．８　　プロジェクトファイルの保存

「ファイル」→「保存…(S)」により、「名前を付けて保存画面」を開き、プロジェクトファイルの保存先フォルダを指定し、「OK」をクリックします。




保存されるファイルは、プロジェクトファイル（デフォルト config..3cf）と拡張子を「xml」に変えたファイルが保存されます。

このファイルをブラウザで開くことにより、コンフィグレーション情報を確認することができます。




## uC3-Configurator プロジェクトファイルの 設定値一覧

＜プロジェクト＞

| プロジェクトファイル名 | CPU ID | Config version |
|---|---|---|
| C:\uC3Cmp\Sample\M3\samples\config_net.3cf | 301 | 260 |

＜カーネルの設定値＞

カーネル共通

| 追加ヘッダファイル | アイドル関数 | タスク優先度 | チック時間 | システムスタックサイズ |
|---|---|---|---|---|
| | | 8 | 1 | 1024 |

タスク

| IDの定義名 | 関数名 | 優先度の初期値 | 拡張情報 | (ローカル)スタックサイズ | タスク属性 | 共有スタック |
|---|---|---|---|---|---|---|
| ID_ETH_SND_TSK | dev_snd_tsk | 4 | | 256 | TA_HLNG | 使用しない |
| ID_ETH_RCV_TSK | dev_rcv_tsk | 4 | | 304 | TA_HLNG | 使用しない |
| ID_ETH_CTL_TSK | dev_ctl_tsk | 4 | | 256 | TA_HLNG | 使用しない |
| ID_TCP_TIM_TSK | net_tim_tsk | 4 | | 256 | TA_HLNG | 使用しない |
| ID_MAIN_TSK | MainTask | 4 | | 256 | TA_HLNG TA_ACT | 使用しない |
| ID_HTTPD_TSK1 | httpd_tsk1 | 4 | | 400 | TA_HLNG | 使用しない |

セマフォ

| IDの定義名 | 資源数の初期値 | 最大資源数 | セマフォ属性 |
|---|---|---|---|
| ID_TCP_SEM | 1 | 1 | TA_TFIFO |

イベントフラグ

| IDの定義名 | ビットパターン初期値(HEX) | イベントフラグ属性 |
|---|---|---|
| ID_ETH_RCV_FLG | 0 | TA_TFIFO TA_WMUL |
| ID_ETH_SND_FLG | 0 | TA_TFIFO TA_WMUL |

データキュー

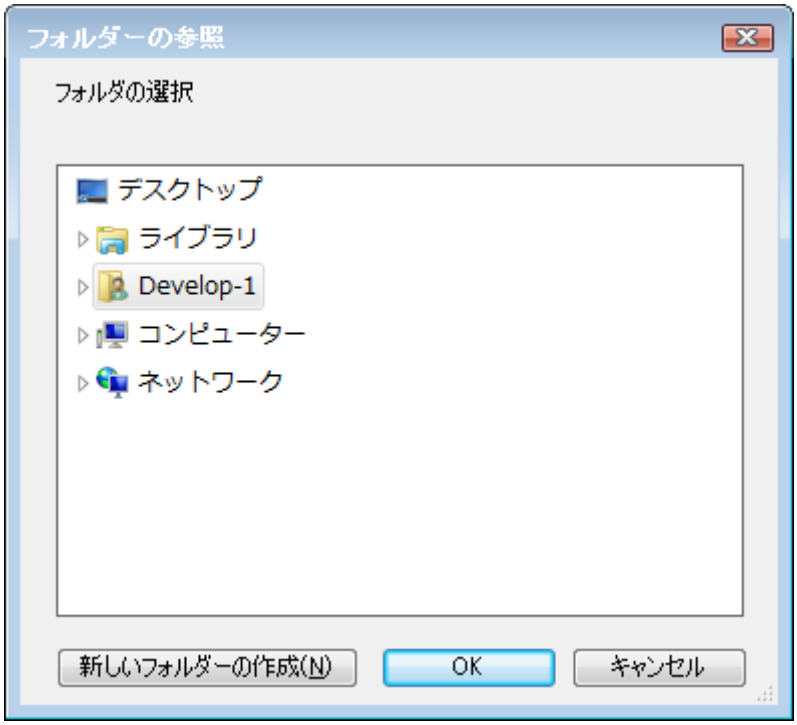
| IDの定義名 | データの個数 | データキュー属性 |
|---|---|---|

メールボックス

| IDの定義名 | メールボックス属性 |
|---|---|
| ID_ETH_SND_MBX | TA_TFIFO TA_MFIFO |

## ４．１．９　　ソース生成

「ファイル」→「ソース生成...(G)」により、「フォルダの参照画面」を開き、生成するファイルを展開する任意のフォルダを指定し、「OK」をクリックします。




スケルトンコード main.c が既に存在していた場合には、編集済みのアプリケーションファイルを上書きで誤って消去しないよう、確認のメッセージが表示されます。

**【推奨】**

スケルトンコードの上書きによる消去を防ぐため、スケルトンコードを直接編集せず、テンプレートとして用いてアプリケーションプログラムを作成することを推奨します。

### Ａ．生成されるファイル

| ファイル | 内容 |
|---|---|
| net_cfg.c | プロトコルスタックのコンフィグレーションコード<br>ソケット定義、IP アドレス定義、MAC アドレス定義等 |
| net_id.h | ソケット ID 定義ヘッダーファイル |
| net_hdr.h | プロトコルスタックのヘッダーファイル |
| main.c | main()、初期設定関数、タスクやハンドラなどのスケルトンコード |
| プロトコルスタックライブラリ | プロトコルスタックの API 群をまとめたライブラリ |
| アプリケーションプロトコルソースファイル | HTTP、DHCP、DNS、FTP プロトコルの API 群をまとめたコード |

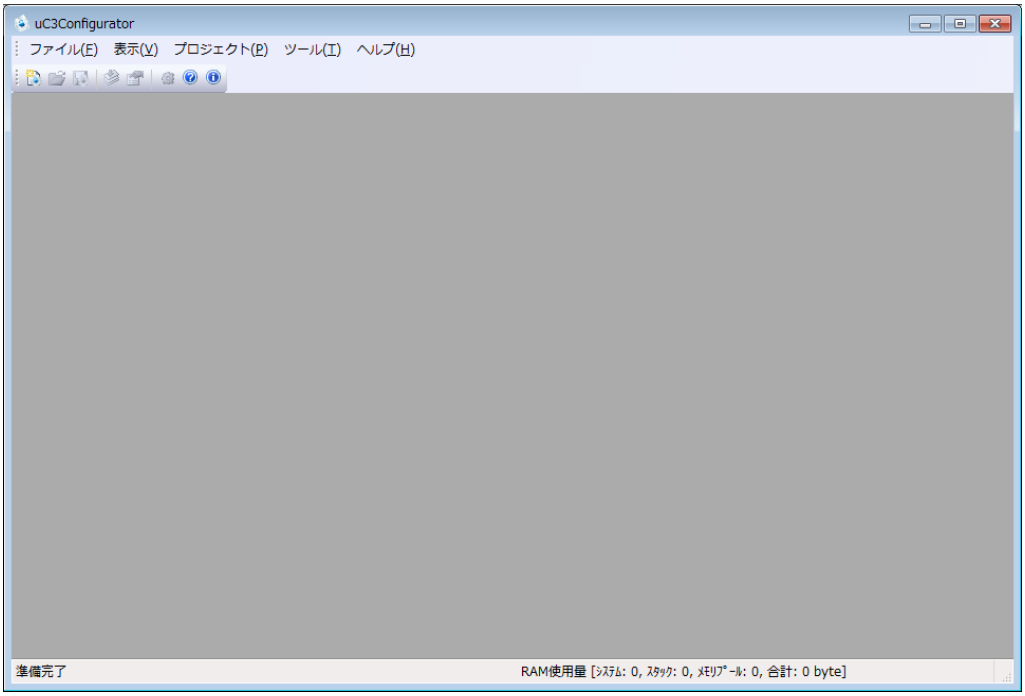
**※上記以外にもカーネル、プロセッサに関連するファイルは生成されますが、本書ではそれに関しては説明しません。適宜「μC3/Compact ユーザーズガイド」を参照してください。**

これらの生成されるファイルは、コンフィグレーションやプロセッサ、さらにはデバイスによっても異なります。

## ４．２　　μNet3/Compact(μNet3 ver.2)のコンフィグレーション

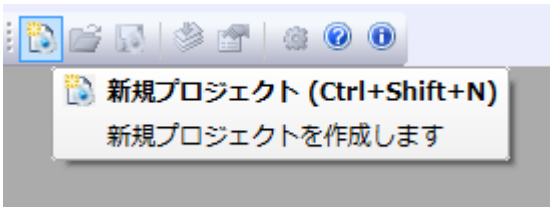
### ４．２．１　　コンフィグレータの起動

Configurator.exe」をダブルクリックし、起動してください。



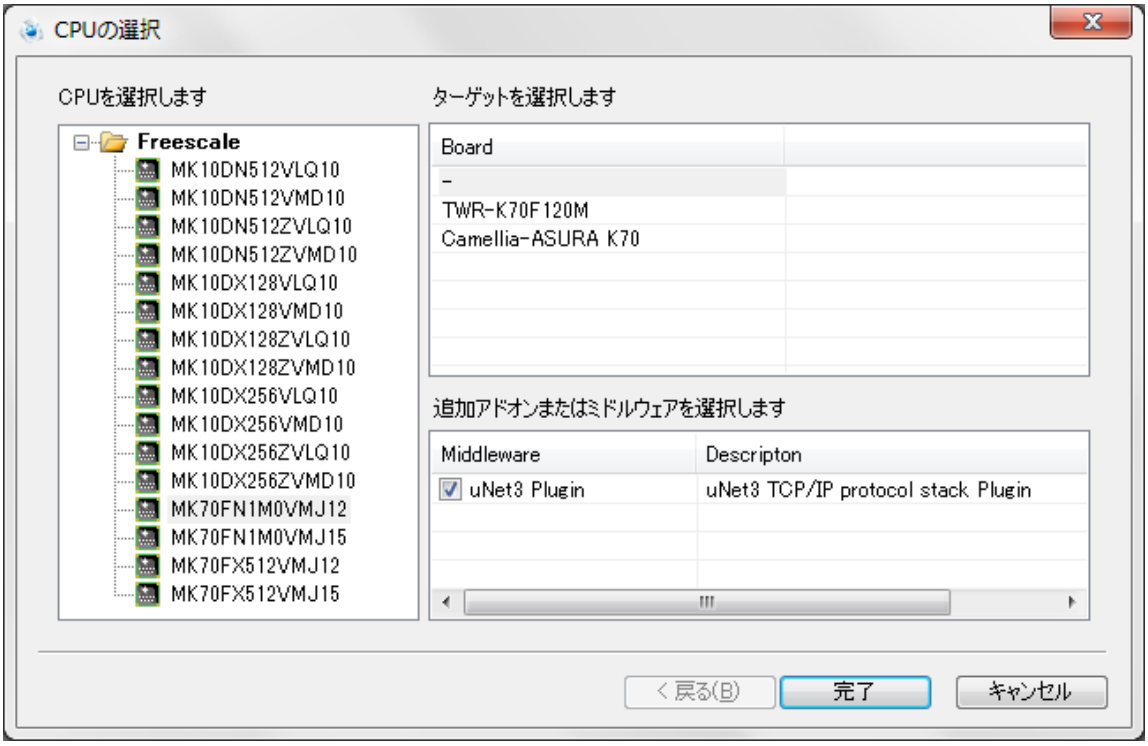

#### Ａ．新規でプロジェクトを生成する場合

ツールバーの「新規プロジェクト」をクリックし、「CPU の選択」へ進みます。



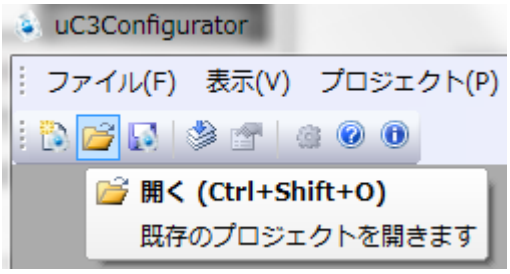
### CPU 選択

リストよりベンダー、CPU 型番、ターゲットを選択し、追加するミドルウェアでは「uNet3」にチェックします。「完了」をクリックし「メイン画面」へ進みます。



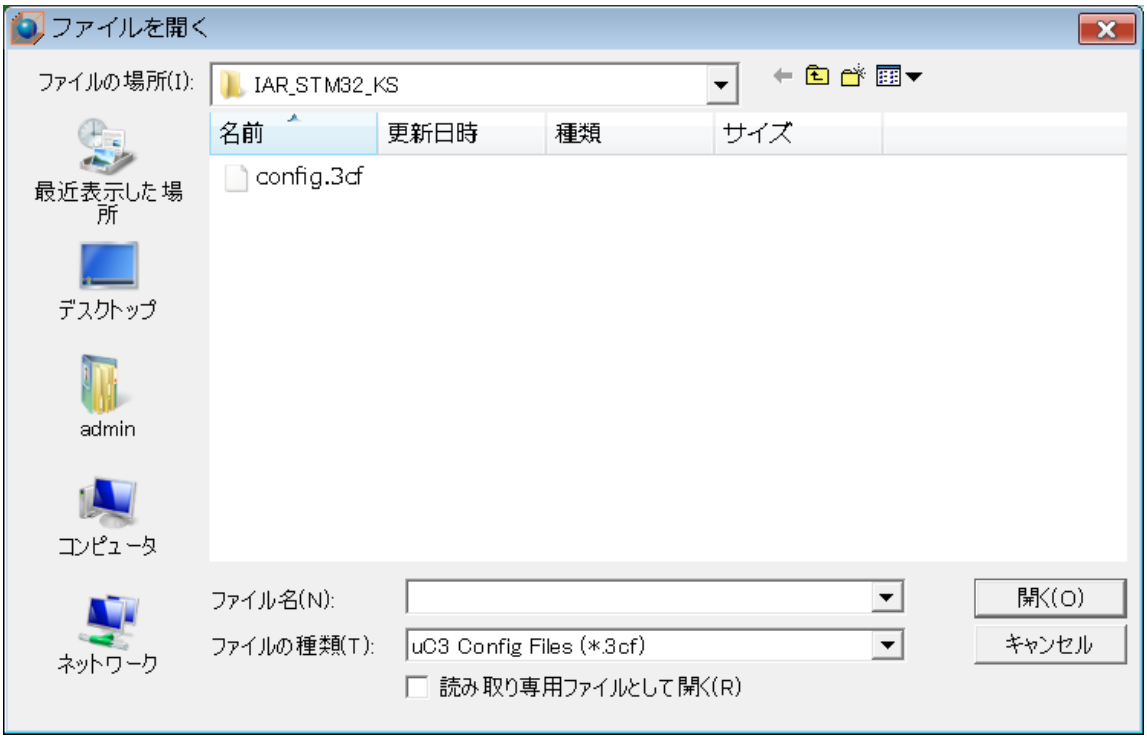
### B．既存のプロジェクトを開く場合

ツールバーの「開く」をクリックし、「ファイルを開く」へ進みます。



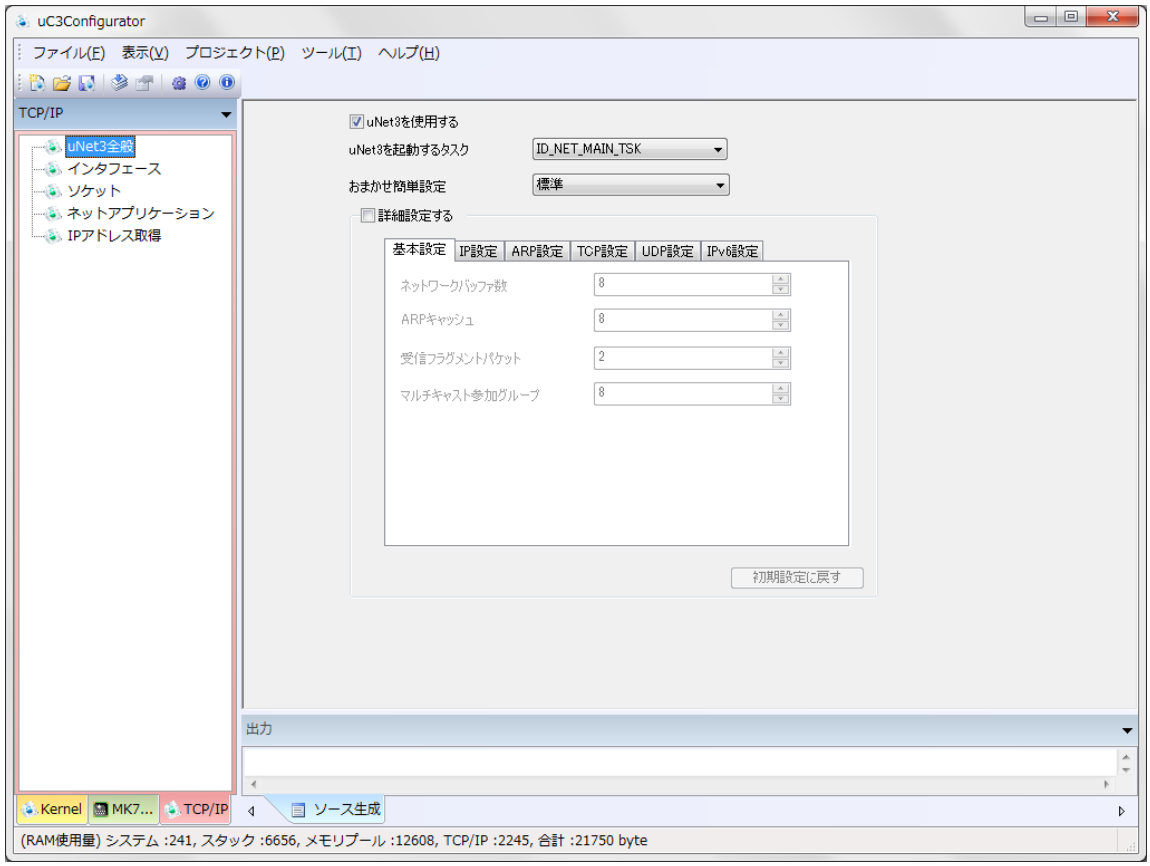

**ファイルを開く**

保存されていたプロジェクトファイル（拡張子.3cf）を選択後に「開く」をクリックし、「メイン画面」へ進みます。



### C．メイン画面

起動後はプロジェクトの参照と編集が可能なメイン画面になります。左側のメニュー画面から TCP/IP タブを選択します。



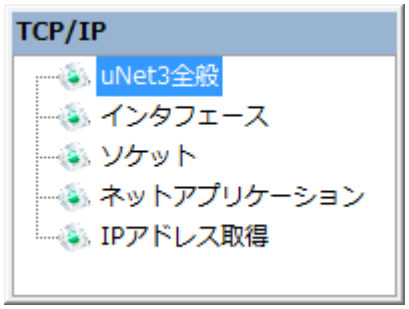
### ４．２．２　　TCP/IP プロトコルスタックの設定

TCP/IP プロトコルスタックには「uNet3 全般」、「インタフェース」、「ソケット」、「ネットアプリケーション」のコンフィグレーションが可能です。

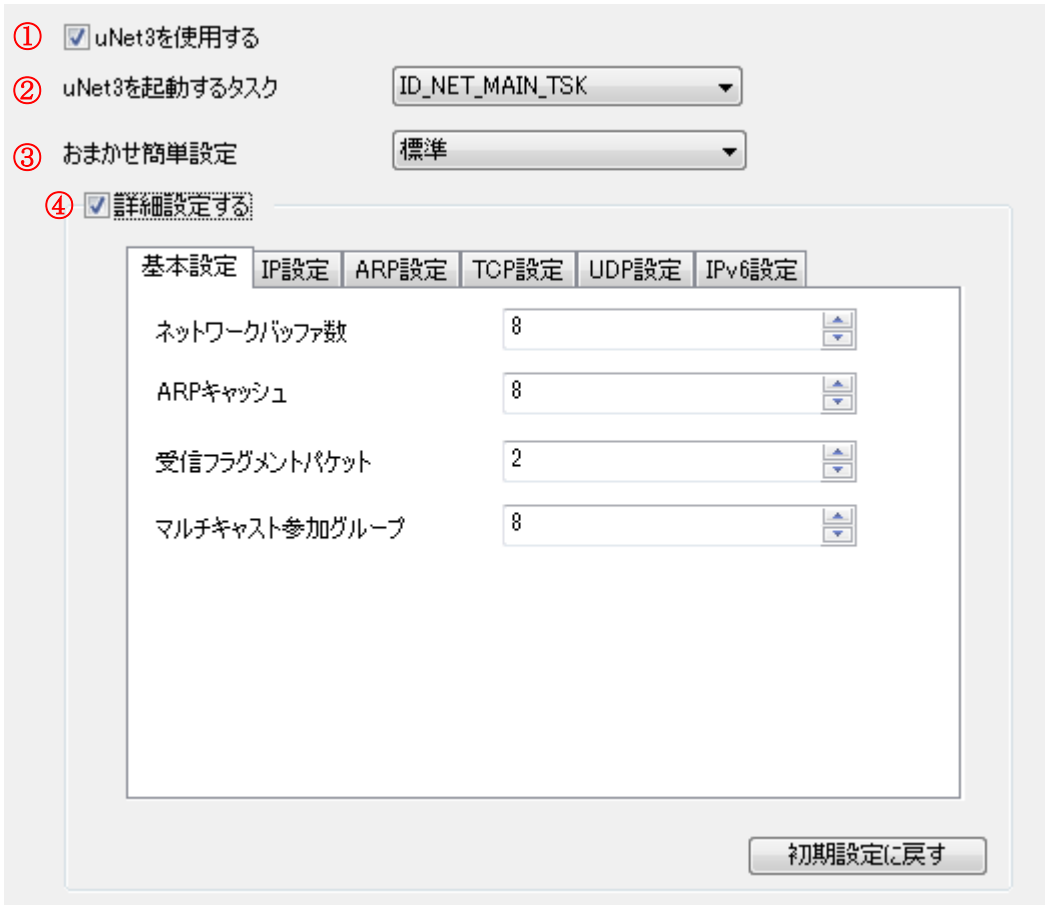
### ４．２．３　　μNet3 全般の設定

メニュー画面のμNet3 全般をクリックすると、TCP/IP プロトコルスタック全般のコンフィグレーション画面が表示されます。

メニュー画面




コンフィグレーション画面



**① μNet3 を使用する**

μNet3 の使用有無をチェックボックスで指定してください。チェックを ON にするとμNet3 が有効になります。OFF にすると無効になります。

**※OFF にすると、今まで設定した内容がクリアされますのでご注意ください。**

**② μNet3 を起動するタスク**

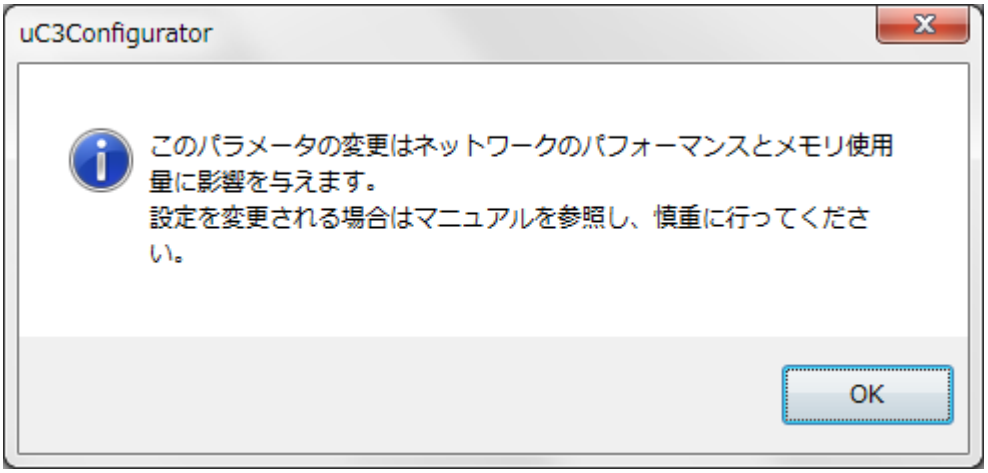
μNet3 を起動するタスクを選択します。ここは、指定を外すことが可能になっています。指定を外した場合、ネットワーク初期化の処理「net_setup()」を、追記することが必要となります。ネットワークを初期化するタスクのスタックサイズは 300 バイト以上に設定してください。

**③ おまかせ簡単設定**

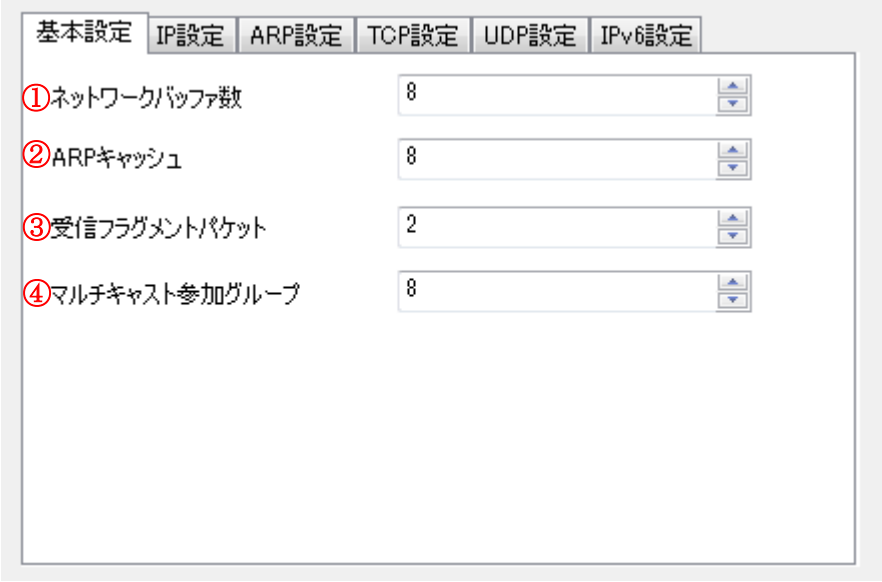
各種テーブルサイズなどシステム要件に合わせて一括して設定することができます。「標準設定」を選択した場合テーブルサイズはデフォルトの値になります。「省メモリ設定」を選択した場合μNet3 が動作する必要最小限のサイズになります。

**④ 詳細設定する**

μNet3 が提供する各種プロトコルの詳細な設定を行うことができます。



【基本設定】




**① ネットワークバッファ数**

ネットワークバッファ数を指定してください。この数値は、TCP/IP プロトコルスタックが使用するネットワークバッファのメモリブロック数を指定します。本書の「第３章　μNet3/Compact の機能概要」の「３．３　ネットワークバッファ」を参照して、設定を行ってください。

**② ARP キャッシュ**

ARP キャッシュの数を指定して下さい。

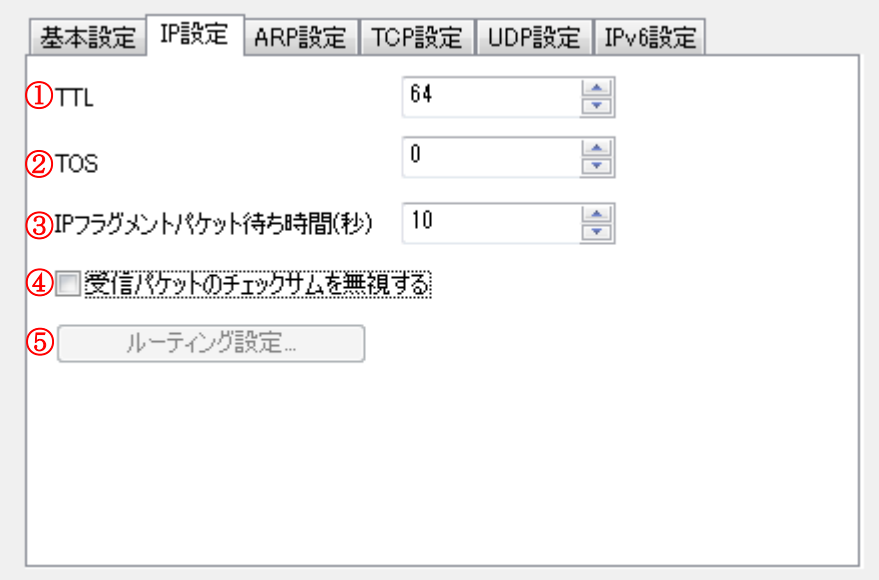
**③ 受信フラグメントパケット**

同時に待つことができる異なる ID のフラグメントパケット数を設定して下さい。

**④ マルチキャスト参加グループ**

参加するマルチキャストグループの最大数を設定して下さい。

【IP 設定】




**① TTL**

送信する IP ヘッダーの TTL 値を設定します。ソケット単位で設定する場合は cfg_soc()を使用して下さい。

**② TOS**

送信する IP ヘッダーの TOS 値を設定します。ソケット単位で設定する場合は cfg_soc()を使用して下さい。

**③ IP フラグメントパケット待ち時間**

IP リアセンブル処理において残りの IP フラグメントパケットを待つ時間を設定して下さい。

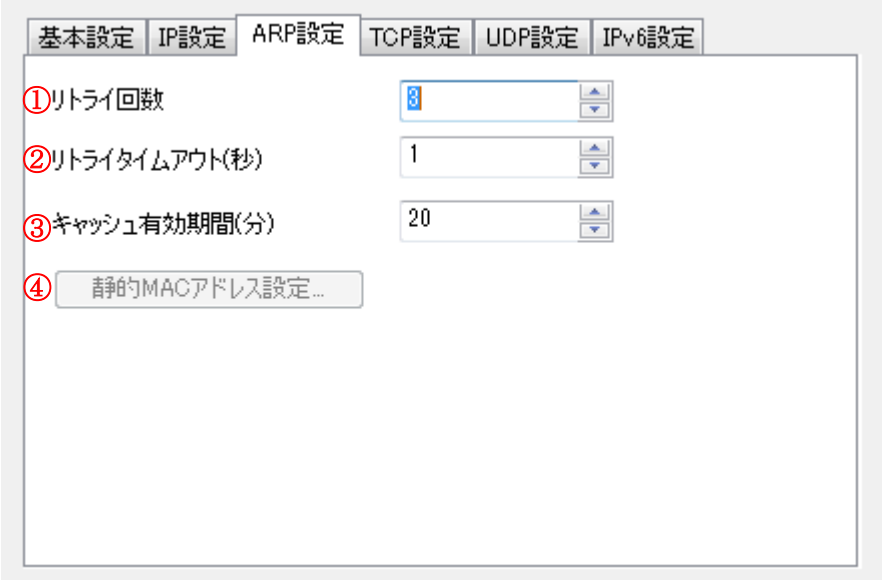
**④ 受信パケットのチェックサムを無視する**

受信した IP パケットのチェックサム値を検証しません。

**⑤ ルーティング設定**

ルーティング設定機能は使えません。

【ARP 設定】




**① リトライ回数**

ARP 応答を得るまでにリトライする ARP の送信回数を設定して下さい。

**② リトライタイムアウト**

ARP 応答を得るまでにリトライする ARP の送信間隔を設定して下さい。

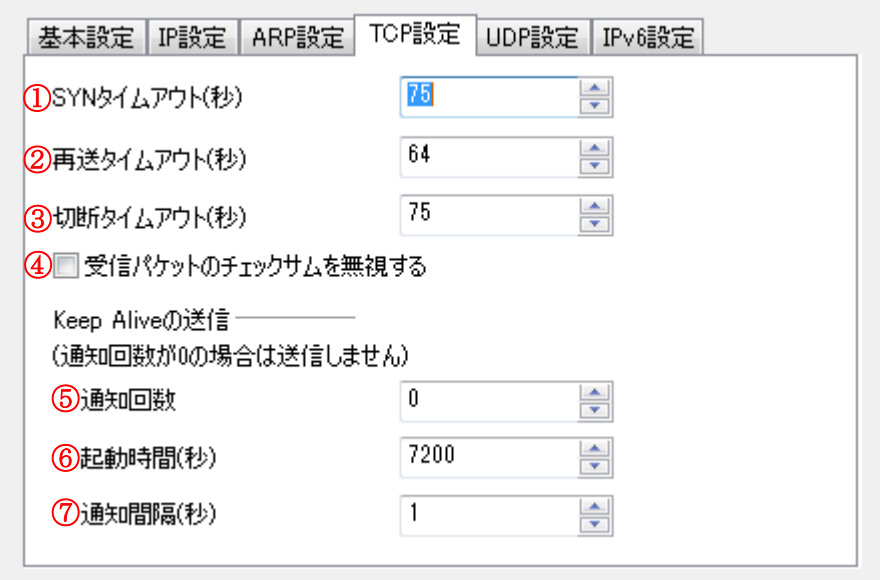
**③ キャッシュ有効期間**

ARP キャッシュに保持する時間を設定して下さい。

**④ 静的 MAC アドレス設定**

静的 MAC アドレス設定機能は使えません。

【TCP 設定】




**① SYN タイムアウト**

TCP アクティブオープン時の接続プロセス時間を設定します。

**② 再送タイムアウト**

TCP データ送信時の再送終了時間を設定します。

**③ 切断タイムアウト**

TCP クローズ時の切断プロセス時間を設定します。

**④ 受信パケットのチェックサムを無視する**

受信した TCP パケットのチェックサム値を検証しません。

**⑤ Keep Alive 通知回数**

切断するまでに送信する Keep Alive の送信回数を設定して下さい。(0 の場合 Keep Alive は起動しません。)

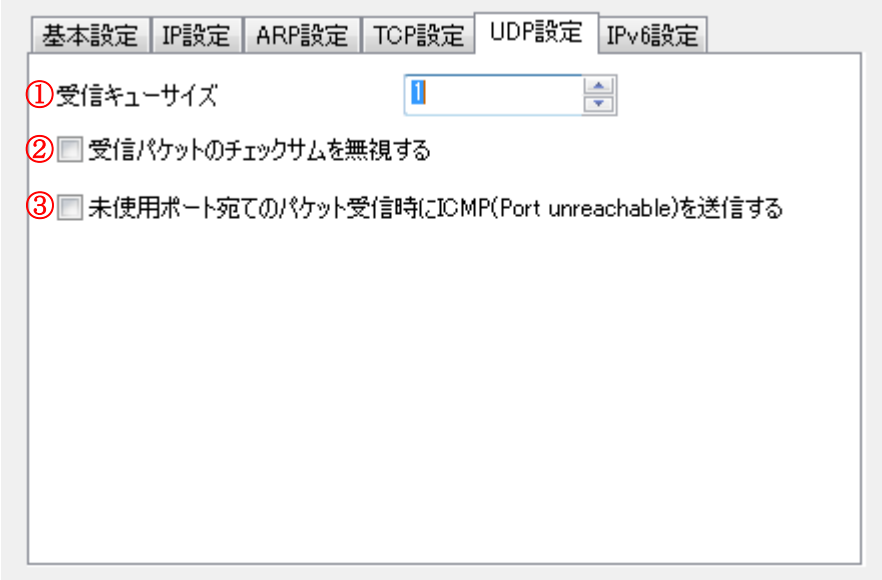
**⑥ Keep Alive 起動時間**

無通信期間の開始後、最初の Keep Alive パケットを送信するまでの時間を設定して下さい。

**⑦ Keep Alive 通知間隔**

Keep Alive パケットの送信間隔時間を設定して下さい。

【UDP 設定】




**① 受信キューサイズ**

rcv_soc()するまでソケット内にキューイングできる受信パケットの数を設定して下さい。

**② 受信パケットのチェックサムを無視する**

受信した UDP パケットのチェックサム値を検証しません。

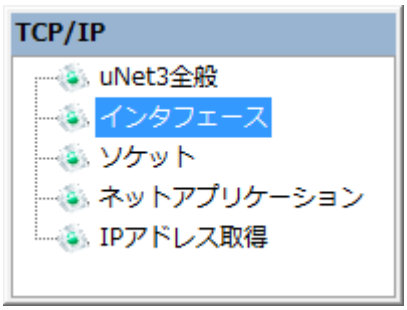
**③ 未使用ポート宛てのパケット受信時に ICMP(Port unreachable)を送信する**

未使用ポート宛てのパケットを受信した場合、パケット送信元のアドレスに ICMP(Port unreachable)を送信します。

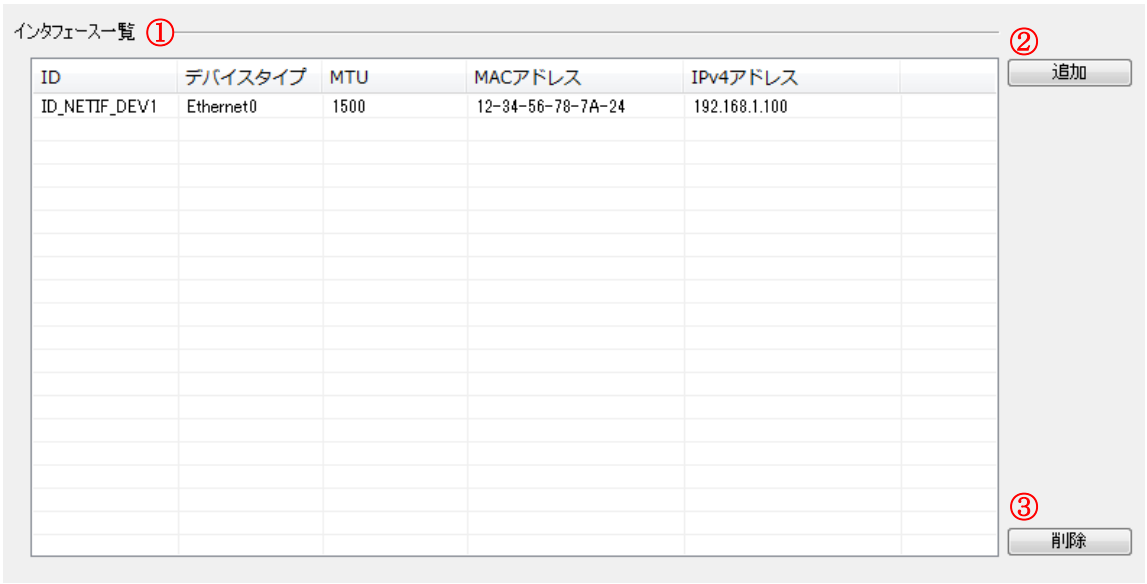
## ４．２．４　　インタフェースの設定

メニュー画面のインタフェースをクリックすると、Ethernet や PPP など各デバイスインタフェースの設定画面が表示されます。

メニュー画面




コンフィグレーション画面




### ① インタフェース一覧

現在設定されているインタフェースの一覧が表示されます。リスト内インタフェースをダブルクリックすることで編集画面を表示します。

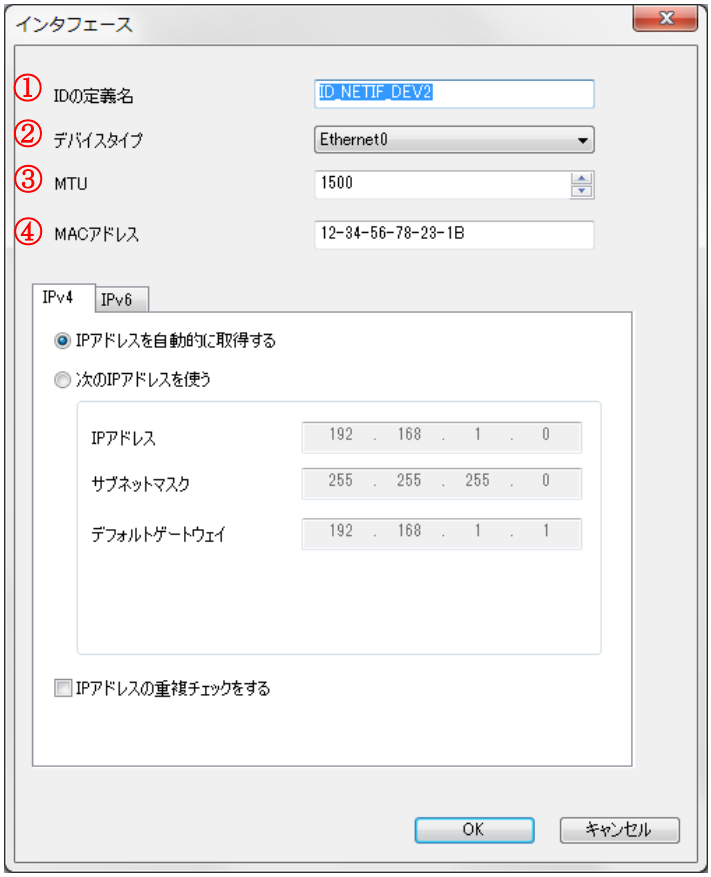
### ② 追加

新規に追加するインタフェースの編集画面を表示します。

### ③ 削除

選択されたインタフェースの設定を削除します。

【インタフェース】




① ID の定義名

インタフェースの ID 番号を表す任意の定義名を指定してください。この定義名は net_id.h 内でマクロ定義されます。

② デバイスタイプ

デバイスタイプ(リンクタイプ)をチップでサポートしているものから選択します。Template を選択するとμNet3 とのインタフェースが実装された空のネットワーク・デバイスドライバが使用できます。また Loopback を選択するとドライバレベルで送信パケットを折り返し受信するネットワーク・デバイスドライバが使用できます。

③ MTU

PATH MTU を指定してください。本書の「第２章　μNet3/Compact の基本概念」及び「第３章 μNet3/Compact の機能概要」を参照して、設定を行ってください。

④ MAC アドレス

ホストの MAC アドレスを指定してください。入力は各オクテット単位で行います。設定値は Ethernet インタフェースに対して有効となります。

【インタフェース IPv4】


### ⑤ IP アドレス

ホストの IP アドレスを指定してください。“IP アドレスを自動的に取得する”を選択すると、DHCP を用いて自動的に IP アドレスを設定します。この時、DHCP 用の UDP ソケットが自動的に追加されます。“次の IP アドレスを使う”を選択した場合は固定 IP アドレスを指定してください。設定値は Ethernet インタフェースに対して有効となります。

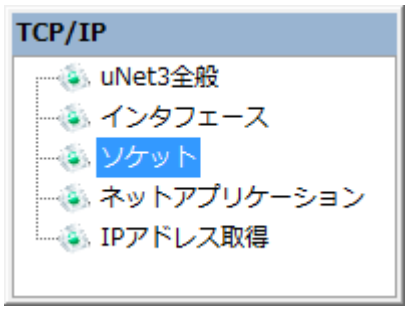
### ⑥ IP アドレスの重複チェックをする

μNet3 起動時にホストに設定された IP アドレスが同じネットワーク上に存在しないかを ARP を送信することで検出します。また起動後に他の端末が同じ IP アドレスを使用して ARP を送信した場合これを検知し、コールバック関数を通してホストに通知します。

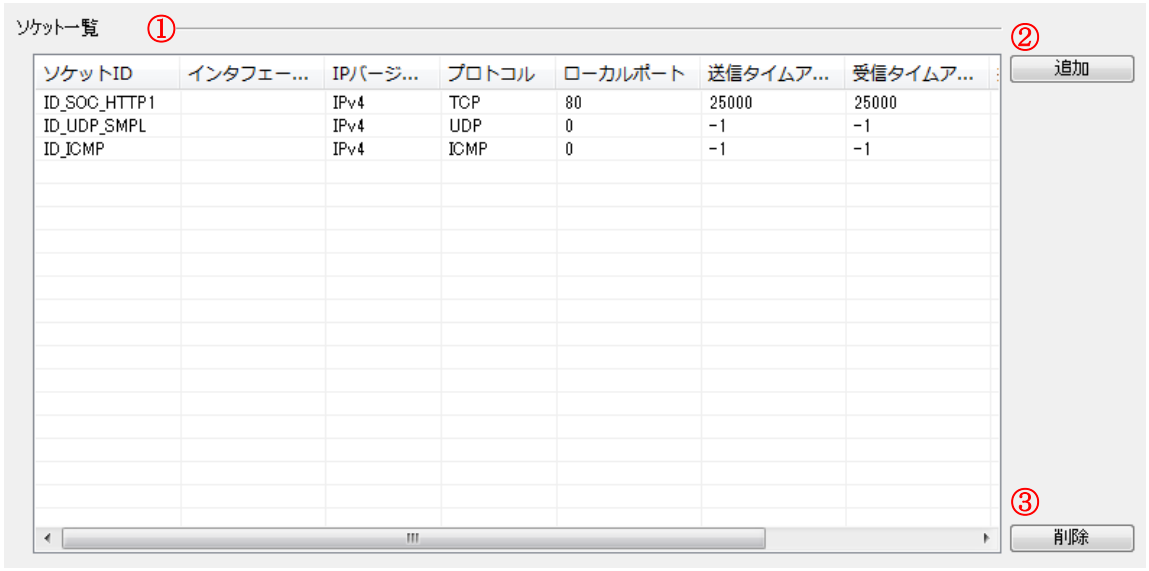
## ４．２．５　　ソケットの設定

メニュー画面のソケットをクリックすると、TCP と UDP のソケット設定画面が表示されます。

メニュー画面




コンフィグレーション画面




### ① ソケット一覧

現在設定されているソケットの一覧が表示されます。リスト内ソケットをダブルクリックすることで編集画面を表示します。

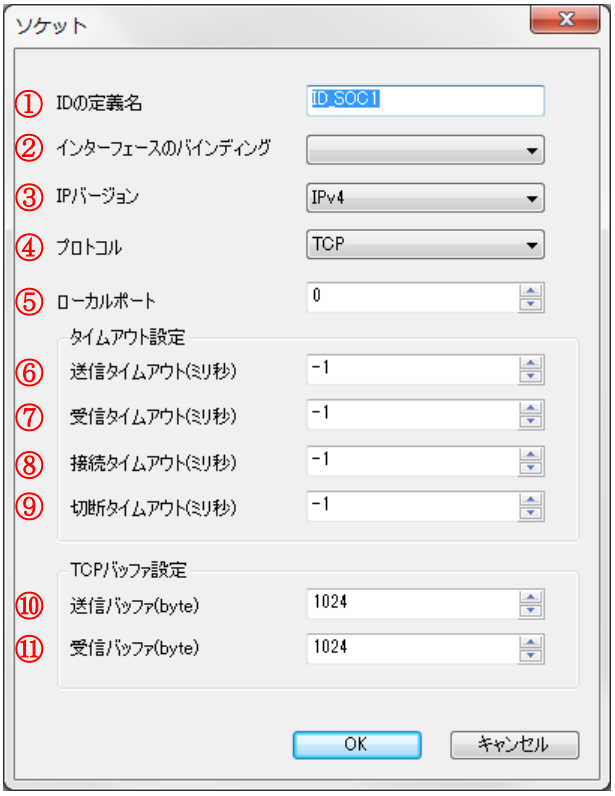
### ② 追加

新規に追加するソケットの編集画面を表示します。

### ③ 削除

選択されたソケットの設定を削除します。

【ソケット】




### ① ID の定義名

ソケットの ID 番号を表す任意の定義名を指定してください。この定義名は net_id.h 内でマクロ定義されます。

### ② インタフェースのバインディング

インタフェース設定で追加したものを選択します。ソケットは必ず一つのインタフェースに関連付ける必要があります。もし何も選択されていない場合はソケット生成時にネットワークデバイスを特定しません。この場合の動作は５．４　　ソケット API を参照して下さい。

### ③ IP バージョン

IPv4 もしくは IPv6 を選択します。（IPv6 はμNet3-IPv6 パッケージしか選択できません）

### ④ プロトコル

TCP もしくは UDP を選択します。

### ⑤ ローカルポート

ソケットのポート番号を指定してください。

### ⑥ 送信タイムアウト

snd_soc API のタイムアウト時間をミリ秒（ms）単位で指定してください。－1 を指定すると snd_soc が正常終了かエラーになるまで返りません。この設定はブロッキングモード時のみ有効です。

### ⑦ 受信タイムアウト

rcv_soc API のタイムアウト時間をミリ秒（ms）単位で指定してください。－1 を指定すると snd_soc が正常終了かエラーになるまで返りません。この設定はブロッキングモード時のみ有効です。

### ⑧ 接続タイムアウト

con_soc API のタイムアウト時間をミリ秒（ms）単位で指定してください。－1 を指定すると con_soc が正常終了かエラーになるまで返りません。この設定は TCP ソケットのブロッキングモード時のみ有効です。

### ⑨ 切断タイムアウト

cls_soc API のタイムアウト時間をミリ秒（ms）単位で指定してください。－1 を指定すると con_soc が正常終了かエラーになるまで返りません。この設定は TCP ソケットのブロッキングモード時のみ有効です。

### ⑩ 送信バッファ

ソケットの送信バッファサイズを指定してください。この設定は TCP ソケットのみ有効です。

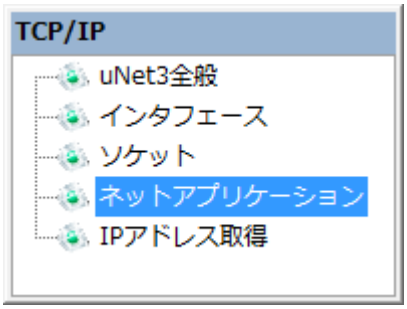
### ⑪ 受信バッファ

ソケットの受信バッファサイズを指定してください。この設定は TCP ソケットのみ有効です。

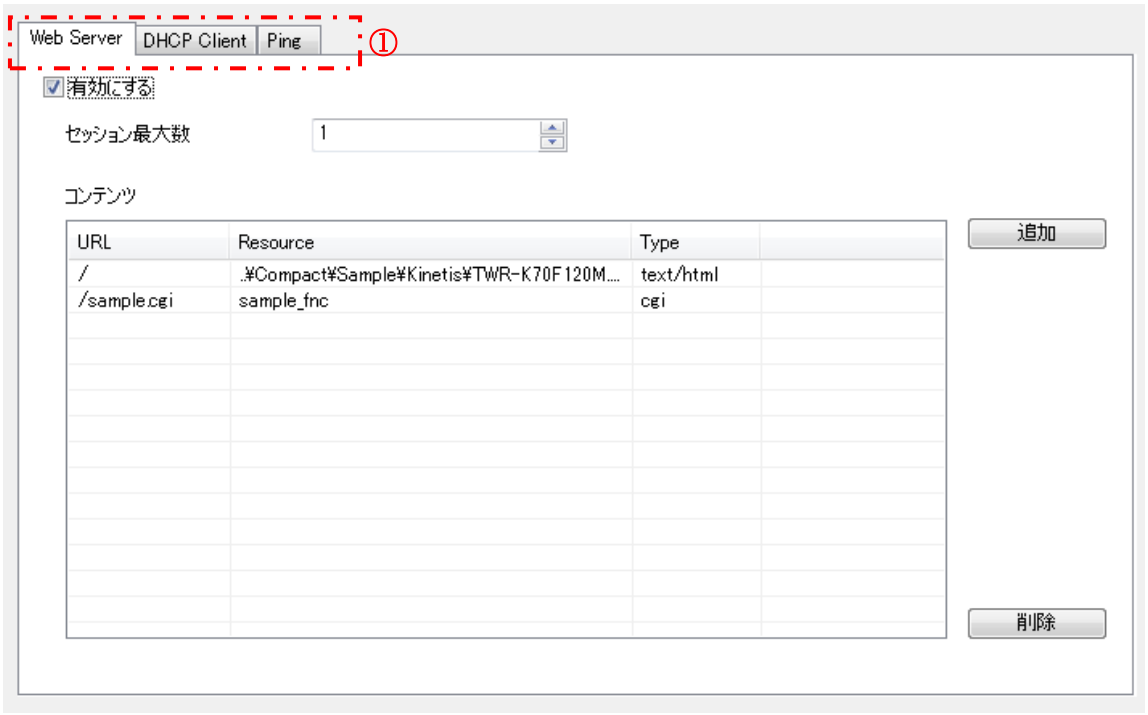
## ４．１．６　　ネットアプリケーションの設定

メニュー画面のネットアプリケーションをクリックすると、μNet3 が提供するネットワークアプリケーションの設定ができます。

メニュー画面



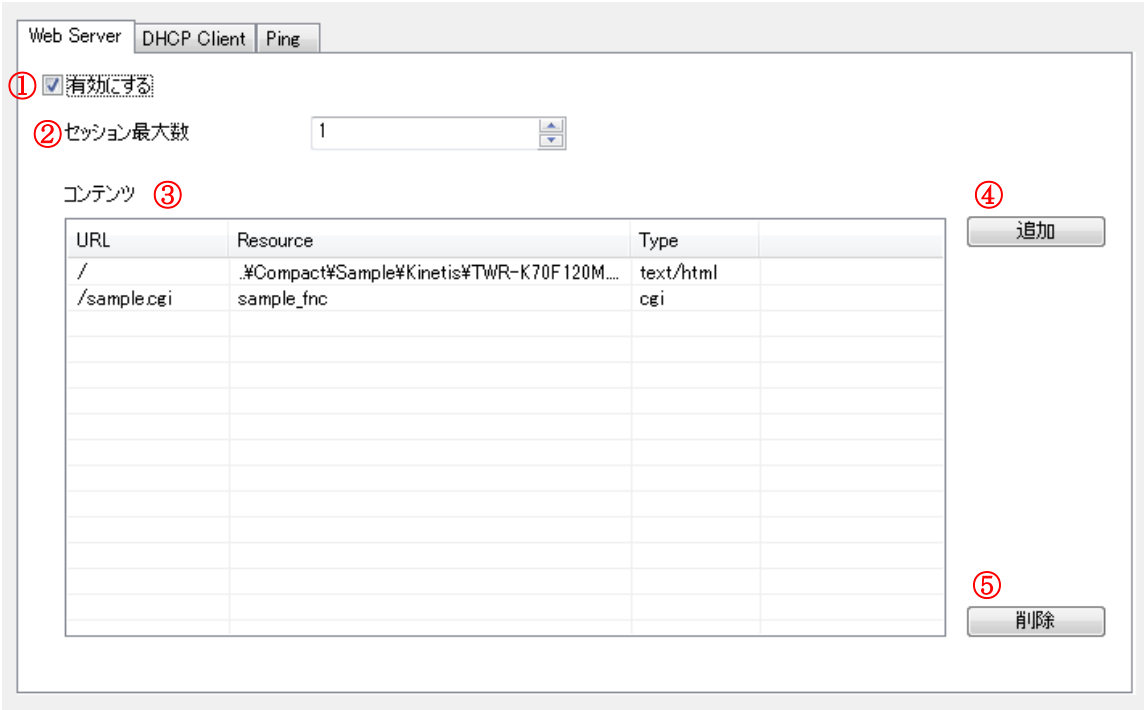

コンフィグレーション画面




### ① アプリケーションタブ

各アプリケーションのコンフィグレーションを行います。

【Web Server】




### ① 有効にする

WEB サーバーの使用有無を指定してください。チェックを ON にすると WEB サーバーが有効になります。OFF にすると無効になります。WEB サーバーを有効にすると HTTP 用 TCP ソケットが自動的に追加されます。

### ② セッション最大数

WEB サーバーへの接続するセッションの上限を指定してください。指定できる最大値は 50 です。

### ③ コンテンツ一覧

現在登録されているコンテンツが表示されています。登録できるコンテンツ数は最大 50 個です。コンテンツ一覧上で変更したいコンテンツをマウス左ダブルクリックすると、コンテンツを変更することができます。

### ④ 追加

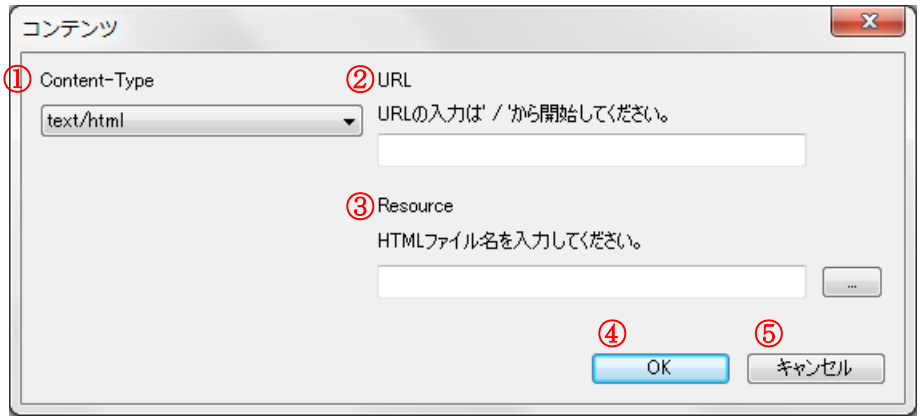
新しいコンテンツを追加します。

### ⑤ 削除

現在選択されているコンテンツを削除します。

**コンテンツの追加、変更**

「追加」ボタンをクリックするか、一覧に登録されているコンテンツをダブルクリックすると次の登録画面が表示されます。




① 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）を指定してください。コンテンツタイプは次の中から選択します。

text/html

image/gif

image/jpeg

cgi

② URL

コンテンツの URL を指定してください。URL の入力は'/'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）

③ Resource

コンテンツのリソースを指定してください。

・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てください。もしくは「…」ボタンをクリックすると“ファイル選択画面”が表示されますので、そこでファイルを指定することもできます。

・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てください。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：　" ` { } * @ ; + : * , . # $ % & ' ¥ " ! ? ~ ^ = | / ¥ < > ( ) "

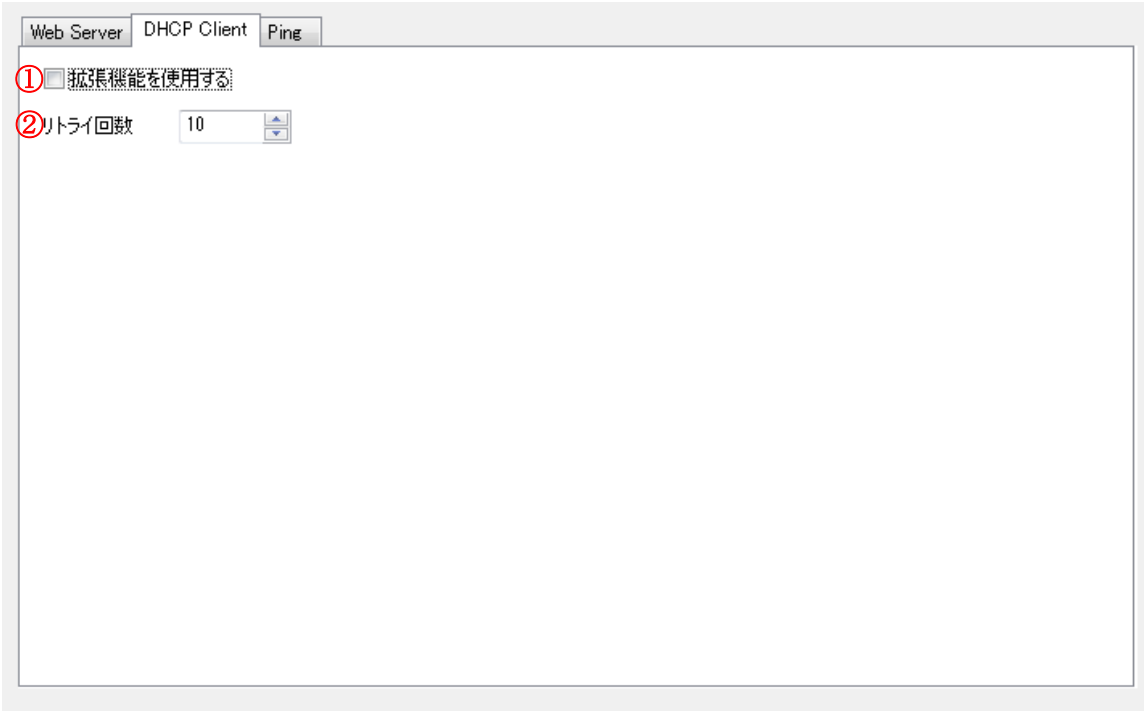
④ OK

コンテンツを登録します。

**⑤ キャンセル**

コンテンツ登録をせずに画面を閉じます。

【DHCP Client】




**① 拡張機能を使用する**

拡張版の DHCP クライアントを使用します。拡張機能についてはネットワークアプリケーション「DHCP クライアント拡張版」を参照して下さい。

**② リトライ回数**

DHCP クライアントがタイムアウトした場合のリトライ回数を設定します。

【Ping】

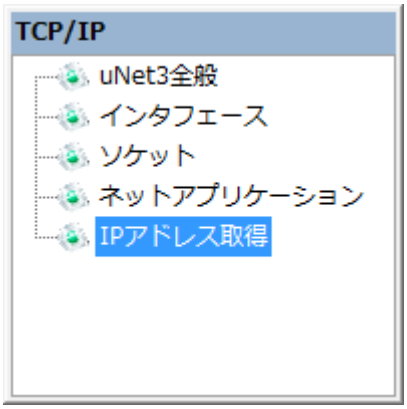

① ICMP Echo 要求を使用する

Ping 用の ICMP ソケットを生成します。

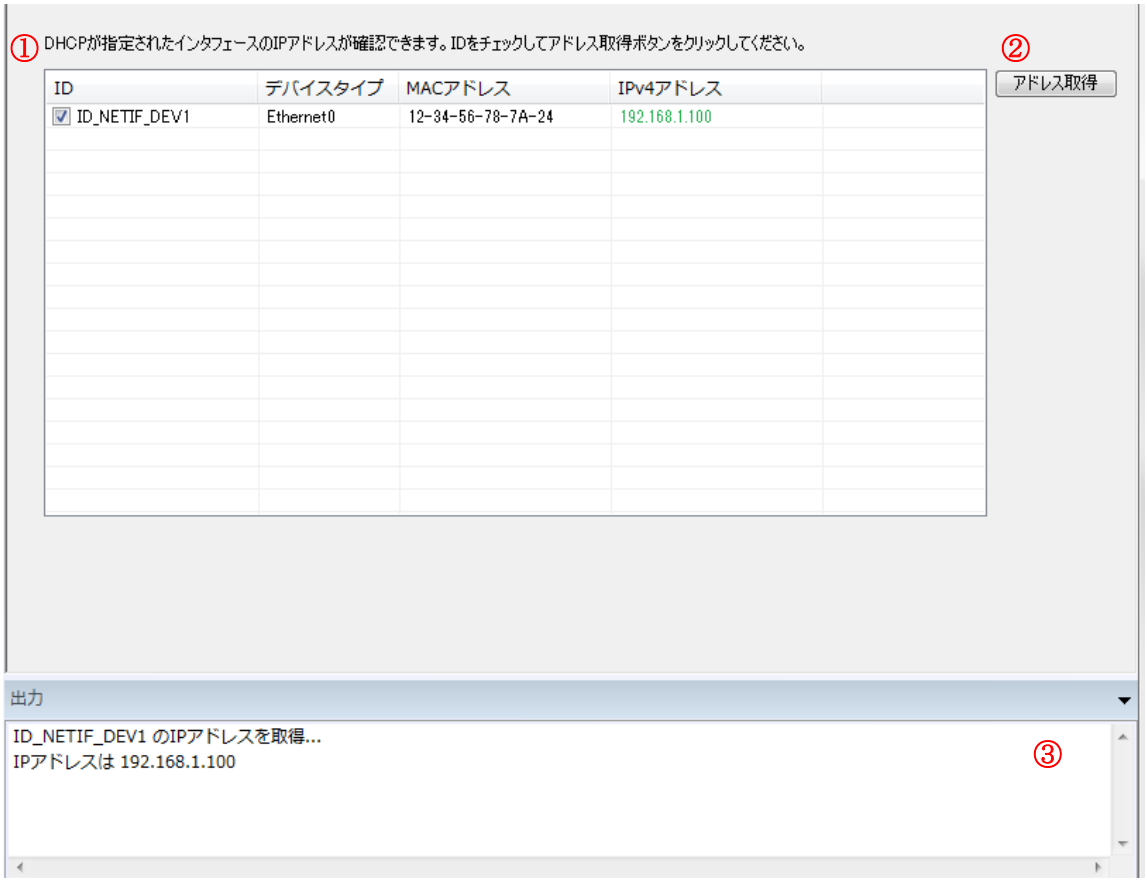
## ４．２．７　　IP アドレス取得の実施

メニュー画面の IP アドレス取得をクリックすると、DHCP クライアント有効状態のターゲットの IP アドレス取得が実施できます。

メニュー画面




コンフィグレーション画面



### ① インタフェース一覧

インタフェース一覧です。チェック有のものが IP アドレス取得対象となります。

### ② アドレス取得

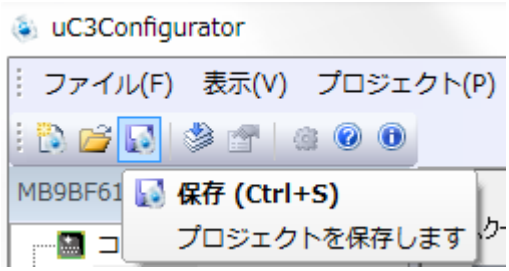
対象インタフェースの IP アドレスを取得します。

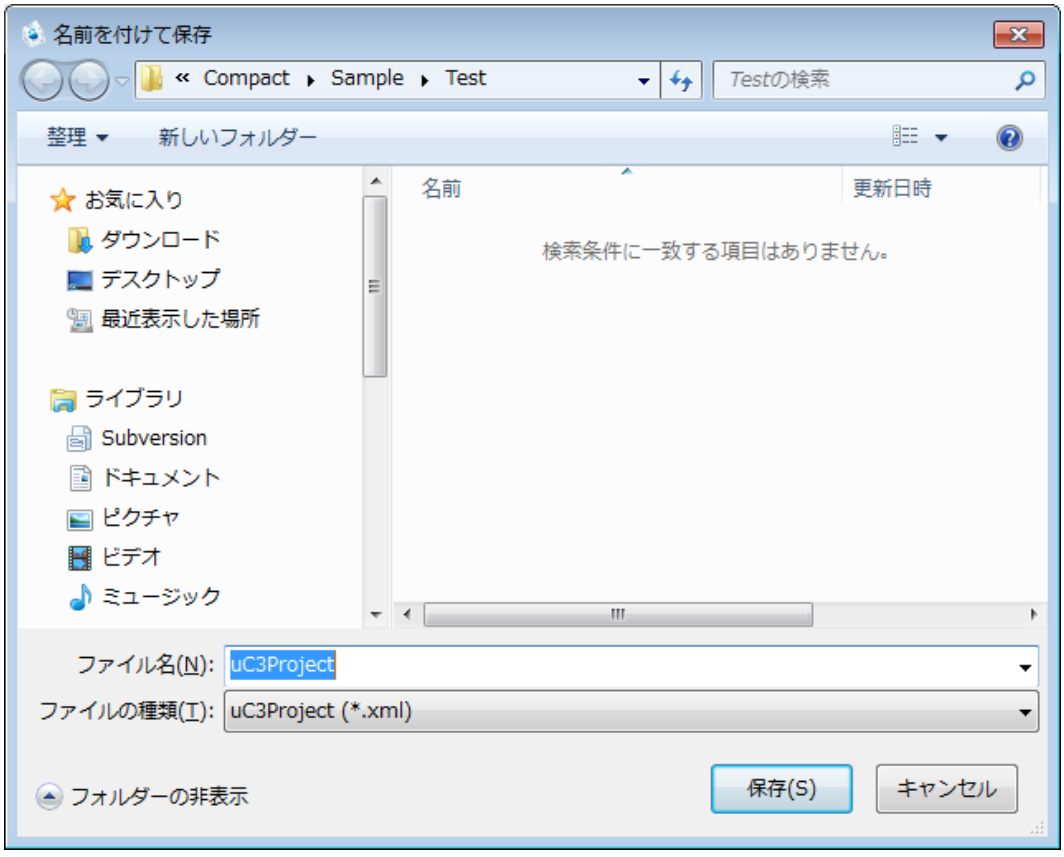
### ③ 出力画面

対象インタフェースの IP アドレス取得結果を表示します。

## ４．２．８　　プロジェクトファイルの保存

ツールバーの「保存」をクリックし、「名前を付けて保存画面」を開き、プロジェクトファイルの保存先フォルダを指定し、「OK」をクリックします。




保存されるファイルは、プロジェクトファイル（デフォルト config..3cf）と拡張子を「xml」に変えたファイルが保存されます。

このファイルをブラウザで開くことにより、コンフィグレーション情報を確認することができます。

# uC3/Configurator プロジェクトファイルの設定値一覧

[使用プラグイン]

| ファイル名 |
|---|
| D:\20120523_NewConfig\uC3\Configurator\Compact\Kernel\ARM\CortexM3\uC3CmpCortexM3.plugin |
| D:\20120523_NewConfig\uC3\Configurator\Compact\CPU\Fujitsu\MB9BXXX\uC3CmpCpuMB9Bxxxx.plugin |

---

[カーネルの設定値]

カーネル全般

| カーネル割込みレベル | タスク優先度 | チック時間 | 初期化関数 | アイドル関数 | 追加ヘッダファイル | タイムイベントハンドラスタックサイズ(CSTACK) | システムハンドラスタックサイズ(HSTACK) |
|---|---|---|---|---|---|---|---|
| 0 | 8 | 1 | | | | 1024 | 1024 |

タスク

| IDの定義名 | 関数名 | 優先度の初期値 | 拡張情報 | (ローカル)スタックサイズ | タスク属性 | 共有スタック |
|---|---|---|---|---|---|---|
| ID_TASK1 | Task1 | 1 | | 256 | TA_HLNG \| TA_ACT | |
| ID_TASK2 | Task2 | 1 | | 256 | TA_HLNG \| TA_ACT | |

セマフォ

| IDの定義名 | 資源数の初期値 | 最大資源数 | セマフォ属性 |
|---|---|---|---|

イベントフラグ

| IDの定義名 | ビットパターン初期値(HEX) | イベントフラグ属性 |
|---|---|---|

データキュー

| IDの定義名 | データの個数 | データキュー属性 |
|---|---|---|

メールボックス

| IDの定義名 | メールボックス属性 |
|---|---|

固定長メモリプール

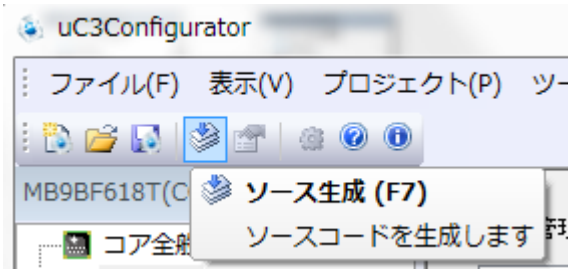
| IDの定義名 | メモリブロック数 | メモリブロックのサイズ | 固定長メモリプール属性 |
|---|---|---|---|

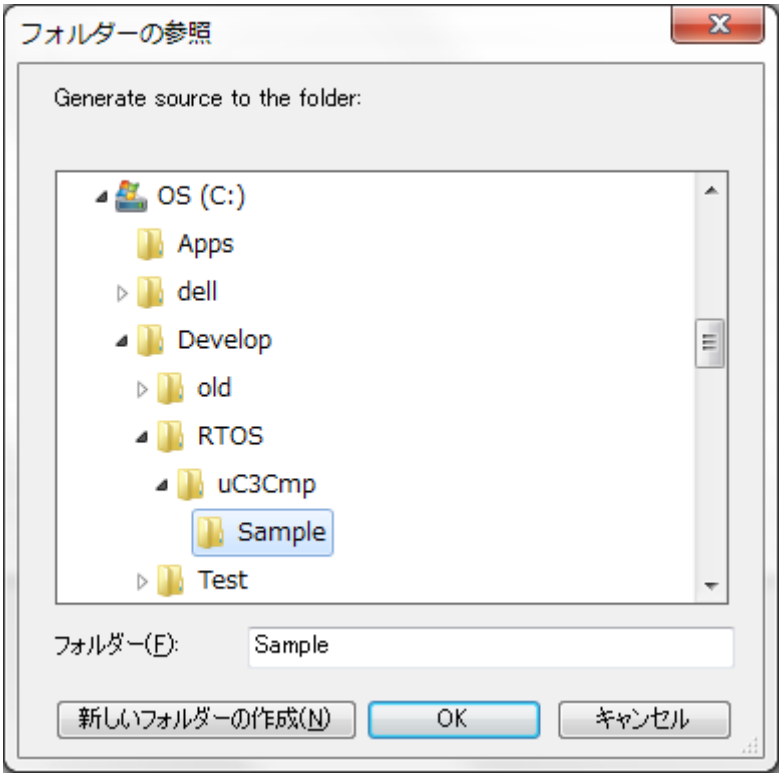
周期ハンドラ

| IDの定義名 | 関数名 | 拡張情報 | 起動周期 | 起動位相 | 周期ハンドラ属性 |
|---|---|---|---|---|---|

## ４．２．９　　ソース生成

ツールバーの「ソース生成」をクリックし、「フォルダの参照画面」を開き、生成するファイルを展開する任意のフォルダを指定し、「OK」をクリックします。




スケルトンコード main.c が既に存在していた場合には、編集済みのアプリケーションファイルを上書きで誤って消去しないよう、確認のメッセージが表示されます。

**【推奨】**

スケルトンコードの上書きによる消去を防ぐため、スケルトンコードを直接編集せず、テンプレートとして用いてアプリケーションプログラムを作成することを推奨します。

### A．生成されるファイル

| ファイル | 内容 |
|---|---|
| net_cfg.c | プロトコルスタックのコンフィグレーションコード<br>ソケット定義、IP アドレス定義、MAC アドレス定義等 |
| net_id.h | ソケット ID 定義ヘッダーファイル |
| net_hdr.h | プロトコルスタックのヘッダーファイル |
| main.c | main()、初期設定関数、タスクやハンドラなどのスケルトンコード |
| プロトコルスタックライブラリ | プロトコルスタックの API 群をまとめたライブラリ |
| アプリケーションプロトコルソースファイル | HTTP、DHCP、DNS、FTP プロトコルの API 群をまとめたコード |

**※上記以外にもカーネル、プロセッサに関連するファイルは生成されますが、本書ではそれに関しては説明しません。適宜「μC3/Compact ユーザーズガイド」を参照してください。**

これらの生成されるファイルは、コンフィグレーションやプロセッサ、さらにはデバイスによっても異なります。

## ４．３　　μNet3/Standard のコンフィグレーション

μNet3/Standard を使用する場合、IP アドレス、送信バッファサイズといったパラメータを net_cfg.c に定義することにより(※)、μNet3 のコンフィグレーションを行います。Sample フォルダにある net_cfg.c をテンプレートとし、システム設計に従い各パラメータを変更してください。

（※）μNet3/ver3 ではコンフィグレーション値を net_cfg.h に定義します。

### ４．３．１　　コンフィグレーション一覧

以下にコンフィグレーション可能なパラメータの一覧を示します。

| コンフィグレーション定義 | デフォルト | コンフィグレーション内容 |
|---|---|---|
| CFG_NET_DEV_MAX | 1 | データリンクデバイス数 |
| CFG_NET_SOC_MAX | 10 | 使用ソケットの上限数 |
| CFG_NET_TCP_MAX | 5 | 使用 TCP ソケットの上限数（※1）、(※3) |
| CFG_NET_ARP_MAX | 8 | ARP エントリ数 |
| CFG_NET_MGR_MAX | 8 | マルチキャストエントリ数 |
| CFG_NET_IPR_MAX | 2 | IP 再構築キュー数 |
| CFG_NET_BUF_SZ | 1576 | ネットワークバッファサイズ（※2) |
| CFG_NET_BUF_CNT | 8 | ネットワークバッファ数 |
| CFG_NET_BUF_OFFSET | 2 | ネットワークバッファデータ書き込み位置（※2) |
| CFG_PATH_MTU | 1500 | MTU サイズ（※2）、(※4) |
| CFG_ARP_RET_CNT | 3 | ARP リトライ回数（※4) |
| CFG_ARP_RET_TMO | 1(sec) | ARP リトライタイムアウト（※4) |
| CFG_ARP_CLR_TMO | 20(minu) | ARP キャッシュクリアタイムアウト（※4) |
| CFG_IP4_TTL | 64 | IP ヘッダーTTL 値（※4) |
| CFG_IP4_TOS | 0 | IP ヘッダーTOS 値（※4) |
| CFG_IP4_IPR_TMO | 10(sec) | IP リアセンブルパケット待ち時間（※4) |
| CFG_IP4_MCAST_TTL | 1 | IP ヘッダーTTL(マルチキャストパケット)（※4) |
| CFG_IGMP_V1_TMO | 400(sec) | IGMPV1 タイムアウト（※4) |
| CFG_IGMP_REP_TMO | 10(sec) | IGMP レポートタイムアウト（※4) |
| CFG_TCP_MSS | 1460 | MSS （※4) |
| CFG_TCP_RTO_INI | 3(sec) | TCP リトライタイムアウト初期値（※4) |
| CFG_TCP_RTO_MIN | 500(msec) | TCP リトライタイムアウト最小値（※4) |
| CFG_TCP_RTO_MAX | 60(sec) | TCP リトライタイムアウト最大値（※4) |
| CFG_TCP_RTO_MAX | 60(sec) | TCP リトライタイムアウト最大値（※4) |
| CFG_TCP_SND_WND | 1024 | 送信バッファサイズ（※3）、(※4) |

| コンフィグレーション定義 | デフォルト | コンフィグレーション内容 |
|---|---|---|
| CFG_TCP_RCV_WND | 1024 | 受信バッファサイズ(Window サイズ) (※4) |
| CFG_TCP_DUP_CNT | 4 | リトライ開始重複 ACK 数 (※4) |
| CFG_TCP_CON_TMO | 75(sec) | SYN タイムアウト (※4) |
| CFG_TCP_SND_TMO | 64(sec) | 送信タイムアウト (※4) |
| CFG_TCP_CLS_TMO | 75(sec) | FIN タイムアウト (※4) |
| CFG_TCP_CLW_TMO | 20(sec) | Close Wait タイムアウト (※4) |
| CFG_TCP_ACK_TMO | 200(msec) | ACK タイムアウト (※4) |
| CFG_TCP_KPA_CNT | 0 | 切断するまでの KeepAlive 通知回数 |
| CFG_TCP_KPA_INT | 1(sec) | KeepAlive を開始後の通知間隔 |
| CFG_TCP_KPA_TMO | 7200(sec) | KeepAlive を開始するまでの無通信時間 |
| CFG_PKT_RCV_QUE | 1 | 受信パケットキューイング数 (※4) |
| CFG_PKT_CTL_FLG | 0 | 受信パケットチェックサム検証無効フラグ |
| CFG_ARP_PRB_WAI | 1000(msec) | net_acd()実行時の ARP Probe 送信待ち時間 |
| CFG_ARP_PRB_NUM | 3(times) | net_acd()実行時の ARP Probe 送信回数 |
| CFG_ARP_PRB_MIN | 1000(msec) | 次の ARP Probe 送信までの最小待ち時間 |
| CFG_ARP_PRB_MAX | 2000(msec) | 次の ARP Probe 送信までの最大待ち時間 |
| CFG_ARP_ANC_WAI | 2000(msec) | ARP Probe を送信してからの検出待ち時間 |
| CFG_ARP_ANC_NUM | 2(times) | ARP Announce の送信回数 |
| CFG_ARP_ANC_INT | 2000(ms) | 次の ARP Announce 送信までの待ち時間 |

※1 CFG_NET_SOC_MAX 以下である必要があります。また CFG_NET_SOC_MAX との差分が非 TCP ソケットの上限数になります。

※2 ネットワークバッファのサイズは、ネットワークバッファ管理構造体サイズ(44Byte)に、MTU、データリンクヘッダサイズ(Ethernet の場合は 14Byte)および、データ書き込み位置を合わせたサイズより大きい必要があります。

※3 TCP の送信バッファは使用するデバイスに関わらず、共通でグローバル変数 UB gTCP_SND_BUF[ ]を使用します。gTCP_SND_BUF[ ]は CFG_TCP_SND_WND × CFG_NET_TCP_MAX でこのサイズを決定します。もし TCP 送信バッファサイズの異なるデバイスを複数使用する場合は、その最大値に合わせて gTCP_SND_BUF[]を設定する必要がありあます。

※4 使用するデバイス単位に設定することが可能です。デバイス番号-1 をインデックスとして gNET_CFG[]に設定します。

### ４．３．２　　IP アドレス

IP アドレスを設定します。ネットワークデバイス毎に IP アドレスが必要になりますので、IP アドレスは CFG_NET_DEV_MAX 分登録してください。

IP アドレス:192.168.1.10、ゲートウェイ：192.168.1.1、サブネットマスク:255.255.255.0 の設定例は下記の通りとなります。

```
T_NET_ADR gNET_ADR[] = {
  {  /*   for Device 1   */
     0x0,              /*  必ず 0 を指定  */
     0x0,              /*  必ず 0 を指定  */
     0xC0A8000A, /* IP アドレス 192.168.1.10 を設定     */
     0xC0A80001,  /*  ゲートウェイ 192.168.1.1       */
     0xFFFFFF00, /*  サブネットマスク 255.255.255.0 */
  }
};
```

### ４．３．３　　デバイスドライバ

デバイスドライバを設定します。デバイスドライバは CFG_NET_DEV_MAX 分登録してください。

```
T_NET_DEV gNET_DEV[]   =   {
  {..}
}
```

詳細は**３．２　ネットワーク・デバイスドライバ**を参照してください。

### ４．３．４　　プロトコルスタック情報テーブル

次のようにプロトコルスタック広域変数を設定します。

```
const VP net_inftbl[] = {
   0,                          /*  必ず 0 を指定  */
   (VP)gNET_SOC,          /*  ソケットを使用しない場合は NULL を指定  */
   (VP)gNET_TCP,          /*  ソケットを使用しない場合は NULL を指定  */
   (VP)gNET_IPR,           /* IP 再構築機能を使用しない場合は NULL を指定  */
   (VP)gNET_MGR,          /* IGMP を使用しない場合は NULL を指定  */
   (VP)gTCP_SND_BUF,   /* TCP ソケットを使用しない場合は NULL を指定  */
};
```

## ４．３．５　　μC3 リソース

μNet3 では次のカーネルオブジェクトを使用します。

| | | |
|---|---|---|
| c_net_tsk | タスク | TCP/IP のタイマイベントに使用 |
| c_net_sem | セマフォ | TCP 排他制御に使用 |
| c_net_mpf | メモリプール | ネットワークバッファーに使用 |

## ４．３．６　　ネットワーク情報管理リソース（μNet3 ver.3 以降）

μNet3 ではネットワークの情報を管理するためにデバイスやソケットの数に応じた情報管理領域を以下のように広域変数で用意する必要があります。（μNet3 ver.3 以降）

| | | |
|---|---|---|
| T_NET_STS_DEV | net_cfg_sts_dev[CFG_NET_DEV_MAX] | デバイス情報 |
| T_NET_STS_IFS | net_cfg_sts_ifs[CFG_NET_DEV_MAX] | TCP/IP 情報 |
| T_NET_STS_IFS | net_cfg_sts_ifs_tmp[CFG_NET_DEV_MAX] | TCP/IP 情報(テンポラリ) |
| T_NET_STS_ARP | net_cfg_sts_arp[CFG_NET_ARP_MAX] | ARP 情報 |
| T_NET_STS_SOC | net_cfg_sts_soc[CFG_NET_SOC_MAX] | ソケット情報 |
| VP | net_cfg_sts_ptr[4 + CFG_NET_DEV_MAX] | ステータスポインタ |
| VP | net_cfg_sts_ptr_tmp[4 + CFG_NET_DEV_MAX] | ステータスポインタ |
| T_NET_STS_CFG | T_NET_STS_CFG gNET_STS_CFG = {<br>net_cfg_sts_dev,<br>net_cfg_sts_ifs,<br>net_cfg_sts_ifs_tmp,<br>net_cfg_sts_arp,<br>net_cfg_sts_soc,<br>net_cfg_sts_ptr,<br>net_cfg_sts_ptr_tmp,<br>0<br>}; | ネットワーク情報管理テーブル |

# 第５章　アプリケーションプログラミングインタフェースの説明

## ５．１　　プロトコルスタックの初期化

TCP/IP プロトコルスタックを使用するにはプロトコルスタックの初期化とネットワークデバイスの初期化が必要になります。基本的には次のように初期化します。

**初期化コード例)**

```
/* プロトコルスタックの初期化 */
        ercd    =    net_ini();
        if (ercd != E_OK) {
           return ercd;
        }
/* ネットワークデバイス(デバイス番号 N)の初期化 */
        ercd    =    net_dev_ini(N);
        If (ercd != E_OK) {
           return ercd;
        }
```

初期化コードは net_cfg.c の net_setup 関数で実施しています。

## ５．２　ネットワーク・インタフェース API

### net_ini　TCP/IP プロトコルスタックの初期化

【書式】

ER ercd = net_ini(void);

【パラメータ】

なし

【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】

| | |
|---|---|
| < 0 | 初期化失敗 |

【解説】

プロトコルスタックで使用するリソースを初期化します。プロトコルスタックで使用するカーネルオブジェクト（タスク、メモリプール、セマフォ）も同時に生成初期化されます。また、プロトコルスタックで使用する広域変数には初期値がセットされます。

プロトコルスタックを使用する場合にはどの API よりも先に、この API を発行する必要があります。

## net_cfg　　ネットワーク・インタフェースのパラメータ設定

**【書式】**

ER ercd = net_cfg(UH num, UH opt, VP val);

**【パラメータ】**

| | | |
|---|---|---|
| UH | num | デバイス番号 |
| UH | opt | パラメータコード |
| VP | val | 設定する値 |

**【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_NOSPT | 不正なパラメータコード |
| E_ID | デバイス番号正しくない |
| E_NOMEM | マルチキャストテーブルがいっぱい |

**【解説】**

IP アドレスやサブネットマスク、ブロードキャストアドレス、マルチキャスト、その他基本的な設定を行います。

**設定例**

```
net_cfg(1, NET_BCAST_RCV, (VP)1);  /* ブロードキャストの受信を有効 */
```

| パラメータコード | データタイプ | 意味 |
|---|---|---|
| NET_IP4_CFG | T_NET_ADR | IP アドレス、サブネットマスク、ゲートウェイを設定します。val には T_NET_ADR のポインタを渡してください。 |
| NET_IP4_TTL | UB | TTL（Time to Live）を設定します。<br>デフォルトは 64 が設定されています。 |
| NET_BCAST_RCV | UB | ブロードキャストの受信の可否を設定します。“1”を設定した場合は受信可能となり、“0”を設定した場合は不可となります。 |
| NET_MCAST_JOIN | UW | 参加するマルチキャストグループの IP アドレスを登録します。 |
| NET_MCAST_DROP | UW | 脱退するマルチキャストグループの IP アドレスを設定します。 |
| NET_MCAST_TTL | UB | マルチキャスト送信で使用する TTL を設定します。 |
| NET_ACD_CBK | コールバック関数ポインタ | 運用中に IP アドレス競合を検出したことをコールバック通知する関数を設定します。<br>この設定により競合検出の通知機能が有効になります。 |

## net_ref　　ネットワーク・インタフェースのパラメータ参照

**【書式】**

ER ercd = net_ref(UH num, UH opt, VP val);

**【パラメータ】**

| | | |
|---|---|---|
| UH | num | デバイス番号 |
| UH | opt | パラメータコード |
| VP | val | 取得する値のバッファへのポインタ |

**【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_NOSPT | 不正なパラメータコード |
| E_ID | デバイス番号正しくない |

**【解説】**

IP アドレスやサブネットマスク、ブロードキャストアドレス、そのほかの基本的な設定の確認を行います。

**設定例**

```
UB bcast;
net_ref(1, NET_BCAST_RCV, (VP)&bcast);  /* ブロードキャストの受信状態 */
```

| パラメータコード | データタイプ | 意味 |
|---|---|---|
| NET_IP4_CFG | T_NET_ADR | IP アドレス、サブネットマスク、ゲートウェイを取得します。val には T_NET_ADR のポインタを渡してください。 |
| NET_IP4_TTL | UB | TTL（Time to Live）を取得します。 |
| NET_BCAST_RCV | UB | ブロードキャストの受信の状態を取得します。 |
| NET_MCAST_TTL | UB | マルチキャスト送信 TTL を取得します。 |

## net_acd　　IP アドレスの競合探知

【書式】

ER ercd = net_acd(UH dev_num, T_NET_ACD *acd);

【パラメータ】

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |
| T_NET_ACD | *acd | アドレス競合情報 |

【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_ID | デバイス番号不正 |
| E_PAR | パラメータ不正 |
| E_OBJ | 重複呼び出し、ホスト IP 未設定時の呼び出し |
| E_TMOUT | ARP 送信タイムアウト |
| E_SYS | IP アドレス競合検出 |
| E_OK | IP アドレス競合非検出 |

【解説】

dev_num で指定されるデバイスの、IP アドレスの競合検出を行います。

IP アドレスの競合を検出した場合、引数の競合情報には相手側の MAC アドレスが格納されます。

この API とは別に非同期で IP アドレスの競合を検出したい場合は、net_cfg()API でコールバック関数を登録する必要があります。

※本関数は最大で約 10 秒、競合アドレスの検出を試みるため専用タスクで呼び出すことをお勧めします。

## acd_cbk　　IP アドレス競合検出時のコールバック関数

**【書式】**

ER acd_cbk(T_NET_ACD* acd);

**【戻り値】**

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

**【パラメータ】**

| | | |
|---|---|---|
| T_NET_ACD | *acd | アドレス競合情報 |

**【解説】**

この関数は運用中に IP アドレスの競合を検出した場合に呼び出されます。引数の競合情報には、競合したホストの MAC アドレスが格納されます。

IP アドレスの競合に対して、自身のホストでその IP アドレスを使用し続ける場合は E_OK を返却して下さい。それ以外の場合は E_SYS を返却して下さい。

コールバック関数は ARP パケットを受信したタスク(Ethernet ドライバの受信タスク)上で呼ばれます。そのためコールバック関数は即座に終了して下さい。また IP アドレス探知中(net_acd()実行中)はコールバック関数が呼ばれることはありません。

**使用例**

```
#define    ID_DEVNUM_ETHER   1    /* デバイス番号 */
/* アドレス競合検出時のコールバック関数 */
ER acd_detect(T_NET_ACD * acd)
{
    return E_OK;
}

/* ネットワーク初期化関数 */
ER net_setup(void)
{
    ER ercd;
    T_NET_ACD acd;

    ercd = net_ini();
    if (ercd != E_OK) {
        return ercd;
    }
    ercd = net_dev_ini(ID_DEVNUM_ETHER);
    if (ercd != E_OK) {
        return ercd;
    }

    /* IP アドレス競合探知 */
    ercd = net_acd(ID_DEVNUM_ETHER, &acd);
    if (ercd == E_OK) {
        /* IP アドレスの競合無し */
        /* IP アドレス競合検出時のコールバック関数設定 */
        net_cfg(ID_DEVNUM_ETHER, NET_ACD_CBK, (VP)acd_detect);
    }
    else if (ercd == E_SYS) {
        /* MAC アドレスが acd.mac のホストと IP が競合 */
    } else {
        /* IP アドレスの競合探知に失敗 */
    }
    return ercd;
}
```

## ５．３　　ネットワークデバイス制御 API

ネットワークデバイス制御 API はアプリケーションからデバイスドライバに統一的にアクセスするためのインタフェースを提供します。各デバイスには、本 API に‘デバイス番号’を指定してアクセスします。デバイス番号とは、デバイスを識別するための固有の番号です。

### net_dev_ini　　ネットワークデバイスの初期化

**【書式】**

ER ercd = net_dev_ini(UH dev_num);

**【パラメータ】**

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |

**【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】**

| | |
|---|---|
| < 0 | 初期化失敗 |

**【解説】**

dev_num を使って特定のデバイスを初期化します。net_dev_ini は、実際にはデバイスドライバの dev_ini を使ってデバイスを初期化します。

正常終了すると、そのネットワークデバイスを通じてパケットの処理が可能となります。

## net_dev_cls　　ネットワークデバイスの解放

**【書式】**

ER ercd = net_dev_cls(UH dev_num);

**【パラメータ】**

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |

**【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】**

| | |
|---|---|
| < 0 | 解放失敗 |

**【解説】**

dev_num を使って特定のデバイスを解放します。net_dev_cls は、実際にはデバイスドライバの dev_cls を使ってデバイスを解放します。

## net_dev_ctl　　ネットワークデバイスの制御

**【書式】**

ER ercd = net_dev_ctl(UH dev_num, UH opt, VP val);

**【パラメータ】**

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |
| UH | opt | 制御コード |
| VP | val | 設定する値 |

**【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】**

| | |
|---|---|
| < 0 | 失敗 |

**【解説】**

dev_num を使って特定のデバイスを制御します。net_dev_ctl はデバイスドライバの dev_ctl を呼び出しているだけですので、実際の動作はデバイスドライバの実装に依存します。

## net_dev_sts　　ネットワークデバイスの状態取得

**【書式】**

ER ercd = net_dev_sts(UH dev_num, UH opt, VP val);

**【パラメータ】**

| | | |
|---|---|---|
| UH | dev_num | デバイス番号 |
| UH | opt | 状態コード |
| VP | val | 取得する値 |

**【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】**

| | |
|---|---|
| < 0 | 失敗 |

**【解説】**

dev_num を使って特定のデバイスの状態を取得します。net_dev_sts はデバイスドライバの dev_ref を呼び出しているだけですので、具体的な動作はデバイスドライバの実装に依存します。

## ５．４　　ソケット API

アプリケーションは、ソケット API を使用してリモートホストとの TCP/UDP データのやり取りを行います。

ソケットは生成もしくは接続時に**デバイス番号**を使って、接続するネットワークデバイスを指定する必要があります。デバイス番号に 0 を指定した場合は「デバイスを特定しない」という意味を持ち、送信と受信でソケット／ネットワークデバイス間のインタフェース選択動作が異なります。またソケット生成時に 0 以外のデバイス番号を指定した場合には、接続時にデバイス番号を指定する必要はありません。

例として N 個のネットワークデバイス(N は 2 以上)で構成されたシステム上で、ソケット API を使用した場合、以下の表の通りデバイスを使用します。

| | 生成時のデバイス番号(※1) | 接続時のデバイス番号(※2) | 使用するデバイス |
|---|---|---|---|
| **ソケット送信動作**<br>snd_soc()や TCP クライアントの con_soc()(SYN 送信) | 0 | 0 | デバイス番号 1(先頭) |
| | 0 | N | デバイス番号 N |
| | N | ANY | デバイス番号 N |
| **ソケット受信動作**<br>rcv_soc()や TCP サーバーの con_soc()(SYN 受信) | 0 | 0 | 通知したデバイス(※3) |
| | 0 | N | デバイス番号 N |
| | N | ANY | デバイス番号 N |

※1　μNet3/Compact の場合は、コンフィグレータでソケットを追加する時にネットワークデバイスを指定します。Standard の場合は、con_soc() API の引数 host->num で指定します。

※2　con_soc() API の引数 host->num で指定します。UDP ソケットで受信する場合は con_soc() API を呼び出す必要はありません。

※3　ソケット生成時も接続時にもデバイス番号を指定していないソケットは、ポート番号とプロトコルが一致すればどのデバイスからでもパケットを受信することが可能です。この場合ソケットはパケットを通知したデバイスを以降の動作で使用します。

## cre_soc　　ソケットの生成

【書式】

ER ercd = cre_soc(UB proto, T_NODE *host);

【パラメータ】

| | | |
|---|---|---|
| UH | proto | プロトコル種別 |
| T_NODE | *host | ローカルホスト情報 |

【戻り値】

| | | |
|---|---|---|
| ER | ercd | 生成されたソケットの ID（>0）またはエラーコード |

【エラーコード】

| | |
|---|---|
| E_NOMEM | ソケットを作ることができない(ソケット最大数を超えている) |
| E_PAR | ‘host’が不正 |
| E_NOSPT | ’proto’が不正 |

【T_NODE】

使用するローカルポート番号とデバイスインタフェースを指定します。

| | | | |
|---|---|---|---|
| UH | port | ポート番号 | ローカルホストのポート番号。1 から 65535 の値または PORT_ANY を指定する。PORT_ANY が指定された場合、ポート番号はプロトコルスタックで決定する。 |
| UH | ver | IP バージョン | 0 を指定（IP_VER4 を使う） |
| UB | num | デバイス番号 | 使用したいデバイスのデバイス番号を指定 |
| UW | ipa | IP アドレス | 0 を指定（ローカル IP アドレスを使う） |

【proto】

生成するソケットのプロトコル種別

| | |
|---|---|
| IP_PROTO_TCP | TCP ソケット |
| IP_PROTO_UDP | UDP ソケット |

**【解説】**

この API は指定したプロトコルのソケットを作ります。

| TCP ソケット生成例 |
| --- |
| T_NODE　host; |
| host.num = 1; |
| host.port = 7; |
| host.ver = IP_VER4; |
| host.ipa = INADDR_ANY; |
| cre_soc(IP_PROTO_TCP, &host); |

※ソケットの生成関数は Standard 版でのみ提供される機能です。Compact 版をご使用の場合はコンフィグレータでソケットを定義する必要があります。

## del_soc　　ソケット削除

**【書式】**

ER ercd = del_soc(UH sid);

**【パラメータ】**

| | | |
|---|---|---|
| UH | sid | ソケットを識別する ID |

**【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 | 不正 ID 番号 |
| E_NOEXS | ソケットが存在しない(ソケットが作られていない) |
| E_OBJ | ソケットの状態が不正 |

**【解説】**

この API は指定した ID のソケットを削除します。TCP ソケットを削除する時、事前に cls_soc()を呼び出してソケットを閉じてください。

※ソケットの削除関数は Standard 版でのみ提供される機能です。Compact 版をご使用の場合はソケットを動的に削除することはできません。

## con_soc　　ソケットの接続

**【書式】**

ER ercd = con_soc(UH sid, T_NODE *host, UB con_flg)；

**【パラメータ】**

| | | |
|---|---|---|
| UH | sid | ソケットを識別する ID |
| T_NODE | *host | リモートホスト情報 |
| UB | con_flg | 接続モード |

**【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_ID | 不正 ID 番号 |
| E_NOEXS | ソケットが存在しない(ソケットが作られていない) |
| E_PAR | host または con_flg が不正 |
| E_OBJ | ソケットの状態が不正（既に接続済みのソケットに対してこの API を呼んだ時など） |
| E_TMOUT | 接続処理がタイムアウトした |
| E_WBLK | ノンブロッキングモードで処理 |
| E_CLS | リモートホストから接続拒否 |
| E_RLWAI | 接続処理が中止された |
| E_QOVR | 既に con_soc()実行中 |

**【T_NODE】**

リモートホストと使用するデバイスインタフェースを指定します。

| | | | |
|---|---|---|---|
| UH | port | ポート番号 | リモートホストのポート番号（1 から 65535） |
| UH | ver | IP バージョン | 0 を指定 |
| UB | num | デバイス番号 | 使用したいデバイスのデバイス番号 |
| UW | ipa | IP アドレス | リモートホストの IP アドレス |

**【con_flg】**

接続を待ち受ける(サーバー)、能動的（クライアント）に接続するかを指定します。

UDP ソケットの場合は常に 0 を指定してください。

| | |
|---|---|
| SOC_CLI | リモートホストに接続する（能動接続） |
| SOC_SER | 接続を待ち受ける（受動接続） |

【解説】

このAPIは使用するプロトコルによって振る舞いが異なります。

TCPの時は、リモートホストとの接続の確立を行い、UDPの時は、データ送信先とソケットとの関連付けを行います。

TCPサーバーソケットの接続例

```
T_NODE    remote = {0};                    /* 0でクリア */
con_soc(ID, &remote, SOC_SER);
```

TCPクライアントソケットの接続例

```
T_NODE    remote;
remote.port = 100;                         /* リモートホストのポート番号 */
remote.ver = IP_VER4;
remote.num = 1;                            /* 使用するデバイス番号を指定 */
remote.ipa = ip_aton("192.168.11.1"); /*    リモートホストのIPアドレス */
con_soc(ID, &remote, SOC_CLI);
```

## cls_soc　　ソケットの切断

**【書式】**

ER ercd = cls_soc(UH sid, UB cls_flg);

**【パラメータ】**

| | | |
|---|---|---|
| UH | sid | ソケットを識別する ID |
| UB | cls_flg | 切断モード |

**【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_ID | 不正 ID 番号 |
| E_NOEXS | ソケットが存在しない(ソケットが作られていない) |
| E_PAR | ‘cls_flg が不正 |
| E_OBJ | ソケットの状態が不正（未接続状態でこの API を呼んだ時など） |
| E_TMOUT | 切断処理がタイムアウトした |
| E_WBLK | ノンブロッキングモードで処理 |
| E_CLS | リモートホストから接続の強制終了 |
| E_RLWAI | 切断処理が中止された |
| E_QOVR | 既に cls_soc()実行中 |

**【cls_flg】**

このパラメータは TCP ソケットのみに有効です。

| | |
|---|---|
| SOC_TCP_CLS | ソケットを切断する。（接続を終了する） |
| SOC_TCP_SHT | 送信処理のみを無効にする。受信は可能。(SOC_TCP_SHT を使用して、送信処理をのみを停止した後、完全に接続を終了したい場合、SOC_TCP_CLS を使用して完全に接続を終了する必要があります) |

**【解説】**

この API は使用するプロトコルによって振る舞いが異なります。

TCP の時は、リモートホストとの接続を切断し、UDP の時は、ソケットに関連付けられたデータの送受信先の情報をクリアします。（この後、UDP データの送信を行うことはできません。）

## cfg_soc　　ソケットのパラメータ設定

【書式】

ER ercd = cfg_soc(UH sid, UB code, VP val) ;

【パラメータ】

| | | |
|---|---|---|
| UH | sid | ソケットを識別する ID |
| UB | code | パラメータコード |
| VP | val | 設定する値 |

【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 | 不正 ID 番号 |
| E_NOEXS | ソケットが存在しない(ソケットが作られていない) |
| E_NOSPT | 不正なパラメータコード |
| E_PAR | 不正なパラメータ値 |
| E_OBJ | ソケットの状態が不正 |

【解説】

次に示すパラメータの設定が可能です。設定する値は VP 型へキャストして渡してください。

**設定例**

```
UB ttl = 32;
cfg_soc(ID, SOC_IP_TTL, (VP)ttl);
```

| パラメータコード | データタイプ | 意味 |
|---|---|---|
| SOC_TMO_CON | TMO | con_soc の呼出しタイムアウト |
| SOC_TMO_CLS | TMO | cls_soc の呼出しタイムアウト |
| SOC_TMO_SND | TMO | snd_soc の呼出しタイムアウト |
| SOC_TMO_RCV | TMO | rcv_soc 呼出しタイムアウト |
| SOC_IP_TTL | UB | IP ヘッダーの TTL（Time to Live）を設定 |
| SOC_IP_TOS | UB | IP ヘッダーの TOS（Type of Server）を設定 |
| SOC_CBK_HND | 関数へのポインタ | コールバック関数の登録 |
| SOC_CBK_FLG | UH | コールバックイベントフラグのビットパターンを設定(設定する値は、以下を参照) |
| SOC_PRT_LOCAL | UH | ローカルポート番号の変更 |

| コールバックイベントフラグビット | 意味 |
|---|---|
| EV_SOC_CON | con_soc()をノンブロッキングモードに設定<br>（TCP ソケットのみ） |
| EV_SOC_CLS | cls_soc()をノンブロッキングモードに設定<br>（TCP ソケットのみ） |
| EV_SOC_SND | snd_soc()をノンブロッキングモードに設定 |
| EV_SOC_RCV | rcv_soc()をノンブロッキングモードに設定 |

コールバックイベントフラグビットについては、複数ビットの設定が可能です。複数設定する場合 OR で設定してください。以下に設定例を示します。

例) ercd = cfg_soc(ソケット ID, SOC_CBK_FLG, (VP)(EV_SOC_CON|EV_SOC_SND|EV_SOC_RCV|EV_SOC_CLS));

ノンブロッキングに設定したソケットイベントはそのイベントのソケットタイムアウトが無効になります。

コールバックイベントフラグビットを有効にするときは、SOC_CBK_HND でコールバック関数の登録が必要となります。コールバック関数はについては、以下を参照してください。

## soc_cbt　　コールバック関数

【書式】

UW soc_cbt(UH sid, UH event, ER ercd);

【パラメータ】

| | | |
|---|---|---|
| UH | sid | ソケットを識別する ID |
| UH | event | コールバックイベントフラグビット |
| ER | ercd | エラーコード |

このコールバック関数は、TCP/IP スタックから呼び出されます。なお、ノンブロッキングモードのソケット API を実行した場合、API 処理が待ち状態になる必要がある時、待ち状態とはならずに、E_WBLK の値が返ります。このとき、TCP/IP スタックからは、コールバック関数で処理が終わったことを通知します。

| コールバックイベントフラグビット(event) | エラーコード(ercd) | 意味 |
|---|---|---|
| EV_SOC_CON | E_OK | con_soc()処理が正常終了 |
| | < 0 | con_soc()処理がエラーで終了。この時のエラー内容については con_soc()のエラーコードを参照してください。 |
| EV_SOC_CLS | E_OK | cls_soc()処理が正常終了 |
| | < 0 | cls_soc()処理がエラーで終了。この時のエラー内容ついては cls_soc()のエラーコードを参照してください。 |
| EV_SOC_SND | > 0 | UDP ソケット：<br>snd_soc()処理が正常終了<br><br>TCP ソケット：<br>TCP 送信バッファに空きがある場合、空きのサイズを'ercd'値で表す。再び snd_soc()を呼び出して、送信データを TCP 送信バッファにコピーすることが出来る。 |
| | <= 0 | snd_soc()処理がエラーで終了。この時のエラー内容については snd_soc()のエラーコードを参照してください。 |
| EV_SOC_RCV | > 0 | UDP ソケット：<br>UDP ソケットには受信データが存在する。受信データのサイズを'ercd'値で表す。再び rcv_soc()を呼び出してデータを受信することが出来る。<br><br>TCP ソケット：<br>TCP ソケットには受信データが存在する。受信データのサイズを'ercd'値で表す。再び rcv_soc()読んでデータを受信することが出来る。 |
| | <= 0 | rcv_soc()処理がエラーで終了。この時のエラー内容については rcv_soc()のエラーコードを参照してください。 |

※コールバック関数からμNet3 の全ての API・関数を呼び出すことはできません。(コールバック関数は割り込みハンドラと同じように考えて使用してください)

## ref_soc　　ソケットのパラメータ参照

【書式】

ER ercd = ref_soc(UH sid, UB code, VP val) ;

【パラメータ】

| | | |
|---|---|---|
| UH | Sid | ソケットを識別する ID |
| UB | Code | パラメータコード |
| VP | val | 取得する値のバッファへのポインタ |

【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_ID | 不正 ID 番号 |
| E_NOEXS | ソケットが存在しない(ソケットが作られていない) |
| E_NOSPT | 不正なパラメータコード |
| E_PAR | 不正なパラメータ値（val が NULL の場合） |
| E_OBJ | ソケットの状態が不正（ソケットを参照することができない） |

**【解説】**

次に示すパラメータの参照が可能です。取得する値は VP 型へキャストして渡してください。

**リモートホスト情報取得例**

```
T_NODE remote;
ref_soc(ID, SOC_IP_REMOTE, (VP)&remote);
```

| パラメータコード | データタイプ | 意味 |
|---|---|---|
| SOC_TMO_CON | TMO | con_soc の呼出しタイムアウト |
| SOC_TMO_CLS | TMO | cls_soc の呼出しタイムアウト |
| SOC_TMO_SND | TMO | snd_soc の呼出しタイムアウト |
| SOC_TMO_RCV | TMO | rcv_soc 呼出しタイムアウト |
| SOC_IP_LOCAL | T_NODE | ローカルホストの IP アドレスとポート番号を取得 |
| SOC_IP_REMOTE | T_NODE | リモートホストの IP アドレスとポート番号を取得 |
| SOC_IP_TTL | UB | TTL（Time to Live）を取得 |
| SOC_IP_TOS | UB | TOS(Type Of Service)を取得 |
| SOC_RCV_PKT_INF | T_RCV_PKT_INF | ソケットが受信した最新のパケット情報を取得(TCP ソケットの場合は取得不可) |
| SOC_PRT_LOCAL | UH | ローカルポート番号の参照 |

マルチキャストアドレスとユニキャストアドレスを持つソケットで、直前のパケットを受信した IP アドレスを知るには次のように参照して下さい。

**受信 IP アドレス取得例**

```
T_RCV_PKT_INF rcv_pkt_inf;
ref_soc(ID, SOC_RCV_PKT_INF, (VP)&rcv_pkt_inf);
if (rcv_pkt_inf.dst_ipa == MULTICASTADDRESS) {
    /* マルチキャストアドレスで受信 */
}
```

## abt_soc　　ソケット処理の中止

**【書式】**

ER ercd = abt_soc(UH sid, UB code);

**【パラメータ】**

| | | |
|---|---|---|
| UH | sid | ソケットを識別する ID |
| UB | code | 制御コード |

**【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_ID | 不正 ID 番号 |
| E_NOEXS | ソケットが存在しない(ソケットが作られていない) |
| E_NOSPT | 不正な制御コード |
| E_OBJ | ソケットの状態が不正 |

**【解説】**

この API は、con_soc、cls_soc、snd_soc、rcv_soc の待ち状態をキャンセルすることができます。 キャンセルされた API は、E_RLWAI を返します。

| 制御コード | 意味 |
|---|---|
| SOC_ABT_CON | con_soc()処理の中止 |
| SOC_ABT_CLS | cls_soc()処理の中止 |
| SOC_ABT_SND | snd_soc()処理の中止 |
| SOC_ABT_RCV | rcv_soc()処理の中止 |
| SOC_ABT_ALL | すべてのソケットの処理の中止 |

## snd_soc　　データの送信

**【書式】**

ER ercd = snd_soc(UH sid, VP data, UH len);

**【パラメータ】**

| | | |
|---|---|---|
| UH | sid | ソケットを識別する ID |
| VP | data | 送信データのポインタ |
| UH | len | 送信データサイズ |

**【戻り値】**

| | | |
|---|---|---|
| ER | ercd | 実際に送信されたデータサイズ(>0)またはエラーコード |

**【エラーコード】**

| | |
|---|---|
| E_ID | 不正 ID 番号 |
| E_NOEXS | ソケットが存在しない(ソケットが作られていない) |
| E_PAR | 不正な送信データか、送信データサイズが指定されていない |
| E_OBJ | ソケットの状態が不正 |
| E_TMOUT | 送信処理がタイムアウト |
| E_WBLK | ノンブロッキンモードで処理 |
| E_CLS | リモートホストから接続の強制終了 |
| E_RLWAI | 送信処理が中止された |
| E_NOMEM | メモリ不足 |
| E_QOVR | 既に snd_soc()実行中 |
| EV_ADDR | 送信先デフォルト G/W が不明 |

**【解説】**

この API はリモートホストにデータを送信します。処理が成功した時は、実際に送信されたデータサイズを返します。それ以外の時はエラーコードを返します。

TCP ソケットの場合、この API はデータをプロトコルスタック内部にコピーし、そのコピーしたサイズを返します。(返されるデータサイズは引数で指定された len 以下です)。詳細は、「３．１．４　TCP モジュール」を参照してください。

UDP ソケットの場合、データはネットワークに送信されその送信サイズを返します。詳細は、「３．１．３　UDP モジュール」を参照してください。

## rcv_soc　　データの受信

【書式】

ER ercd = rcv_soc(UH sid, VP data, UH len);

【パラメータ】

| | | |
|---|---|---|
| UH | sid | ソケットを識別する ID |
| VP | data | 受信データへのポインタ |
| UH | len | 受信データサイズ |

【戻り値】

| | | |
|---|---|---|
| ER | ercd | 実際に受信したデータサイズ(>0)またはエラーコード |

【エラーコード】

| | |
|---|---|
| E_ID | 不正 ID 番号 |
| E_NOEXS | ソケットが存在しない(ソケットが作られていない) |
| E_PAR | 不正な受信データか、受信データサイズが指定されていない |
| E_OBJ | ソケットの状態が不正 |
| E_TMOUT | 受信処理がタイムアウト |
| E_WBLK | ノンブロッキンモードで処理 |
| E_CLS | リモートホストから接続の強制終了 |
| E_RLWAI | 受信処理が中止された |
| E_QOVR | 既に rcv_soc()実行中 |
| 0 | 接続が切断された |

【解説】

この API はリモートホストから送信されたデータを受信します。

TCP の場合、受信可能な最大サイズはコンフィグレータで指定した“受信バッファサイズ”です。詳細は、「３．１．４　TCP モジュール」を参照してください。

UDP の場合､受信可能な最大サイズは 1472 bytes(デフォルト MTU – IP ヘッダーサイズ – UDP ヘッダーサイズ)になります。詳細は、「３．１．３　UDP モジュール」を参照してください。

## ５．５　その他 API

| htons | 16 ビット値をネットワークバイトオーダーへ変換 |
|---|---|

【書式】

UH htons(UH val);

【パラメータ】

| | | |
|---|---|---|
| UH | val | ホストバイトオーダーの 16 ビット値 |

【戻り値】

| | |
|---|---|
| UH | ネットワークバイトオーダーの 16 ビット値 |

| htonl | 32 ビット値をネットワークバイトオーダーへ変換 |
|---|---|

【書式】

UW htonl(UW val);

【パラメータ】

| | | |
|---|---|---|
| UW | val | ホストバイトオーダーの 32 ビット値 |

【戻り値】

| | |
|---|---|
| UW | ネットワークバイトオーダーの 32 ビット値 |

## ntohs　16 ビット値をホストバイトオーダーへ変換

【書式】

UH ntohs(UH val);

【パラメータ】

| | | |
|---|---|---|
| UH | val | ネットワークバイトオーダーの 16 ビット値 |

【戻り値】

| | |
|---|---|
| UH | ホストバイトオーダーの 16 ビット値 |

## ntohl　32 ビット値をホストバイトオーダーへ変換

【書式】

UW ntohl(UW val);

【パラメータ】

| | | |
|---|---|---|
| UW | val | ネットワークバイトオーダーの 32 ビット値 |

【戻り値】

| | |
|---|---|
| UW | ホストバイトオーダーの 32 ビット値 |

## ip_aton　ドット表記の IPv4 アドレス文字列を 32 ビット値に変換

【書式】

UW ip_aton(const char *str);

【パラメータ】

| | | |
|---|---|---|
| char * | str | ドット表記の IPv4 アドレス文字列へのポインタ |

【戻り値】

| | | |
|---|---|---|
| UW | > 0 | 正常終了（変換後 32 ビット値） |

【エラーコード】

| | |
|---|---|
| 0 | 不正な IP アドレスが指定された |

## ip_ntoa　32 ビット値の IPv4 アドレスをドット表記の IPv4 アドレス文字列に変換

【書式】

void ip_ntoa(const char *str, UW ipaddr);

【パラメータ】

| | | |
|---|---|---|
| char * | str | 変換後、IP アドレス文字列を受け取るポインタ |
| UW | ipaddr | IP アドレスの 32 ビット値 |

【戻り値】

なし

【解説】

正常終了した時は、str に文字列がセットされる。エラーの時、str は NULL となる。

## ip_byte2n　　IPv4 アドレスの配列を 32 ビット値に変換

【書式】

UW ip_byte2n(char *ip_array);

【パラメータ】

| | | |
|---|---|---|
| char * | ip_array | IP アドレスのバイト値配列へのポインタ |

【戻り値】

| | | |
|---|---|---|
| UW | > 0 | 正常終了（変換後 32 ビット値） |

【エラーコード】

| | |
|---|---|
| 0 | 不正な IP アドレスが指定された |

## ip_n2byte　　IPv4 アドレスの 32 ビット値を配列に変換

【書式】

void ip_n2byte(char *ip_arry, UW ip);

【パラメータ】

| | | |
|---|---|---|
| char * | ip_arry | IP アドレスのバイト値配列へのポインタ |
| UW | ip | IP アドレスの 32 ビット値 |

【戻り値】

なし

【解説】

正常終了した時は、ip_array に値がセットされる。エラーの時、ip_array は NULL となる。

# 第６章　付録

## ６．１　　パケット形式

### (1) T_NODE

通信端点の情報

```
typedef struct t_node {
        UH      port;   /* ソケットのポート番号 */
        UB      ver;    /* IP バージョン*/
                        /* 必ず IP_VER4 指定 */
        UB      num;    /* デバイス番号*/
        UW      ipa;    /* IP アドレス */
} T_NODE;
```

### (2) T_NET_ADR

ネットワークアドレスの情報

```
typedef struct t_net_adr {
        UB      ver;    /* IP バージョン*/
                        /* 必ず IP_VER4 指定 */
        UB      mode;   /* 予約 */
        UW      ipaddr; /* IP アドレス */
        UW      gateway;/* ゲートウェイ　*/
        UW      mask;   /* サブネットマスク */
} T_NET_ADR;
```

### (3) T_NET_DEV

ネットワークデバイスの情報

```
typedef struct t_net_dev {
        UB      name[8];/* デバイス名 */
        UH      num;    /* デバイス番号 */
        UH      type;   /* デバイスタイプ */
        UH      sts;    /*予約 */
        UH      flg;    /* 予約 */
        FP      ini;    /* dev_ini 関数へのポインタ*/
        FP      cls;    /* dev_cls 関数へのポインタ*/
        FP      ctl;    /* dev_ctl 関数へのポインタ*/
        FP      ref;    /* dev_ref 関数へのポインタ*/
```

```
        FP          out;        /* dev_snd 関数へのポインタ*/
        FP          cbk;        /* dev_cbk 関数へのポインタ*/
        UW          *tag;       /* 予約 */
        union {                 /* MAC アドレス */
          struct {
              UB    mac[6];
          }eth;
        } cfg;
        UH          hhdrsz;     /* デバイスヘッダ―サイズ */
        UH          hhdrofs;    /* ネットワークバッファ書き込み位置*/
        VP          opt;        /* ドライバ拡張領域（μNet3/ver3 以降)*/
    } T_NET_DEV;
```

### (4) T_NET_BUF

ネットワークバッファーの情報

```
typedef struct t_net_buf {
        UW          *next;      /* 予約 */
        ID          mpfid;      /* メモリプール ID */
        T_NET       *net;       /* ネットワークインタフェース */
        T_NET_DEV   *dev;       /* ネットワークデバイス */
        T_NET_SOC   *soc;       /* ソケット */
        ER          ercd;       /* エラーコード */
        UH          flg;        /* ブロードキャスト・マルチキャストフラグ */
        UH          seq;        /* フラグメントシーケンス */
        UH          dat_len;    /* パケットのデータサイズ */
        UH          hdr_len;    /* パケットのヘッダーサイズ */
        UB          *dat;       /* パケット(buf)内のデータ位置を指す */
        UB          *hdr;       /* パケット(buf)内のヘッダー位置を指す */
        UB          buf[];      /* 実際のパケット */
    } T_NET_BUF ;
```

### (5) T_RCV_PKT_INF

受信パケット情報

```
typedef struct t_rcv_pkt_inf{
        UW      src_ipa;        /* パケットの送信元 IP アドレス*/
        UW      dst_ipa;        /* パケットの送信先 IP アドレス*/
        UH      src_port;       /* パケットの送信元ポート番号*/
```

```
    UH      dst_port;       /* パケットの送信先ポート番号*/
    UB      ttl;            /* パケットの IP ヘッダーTTL*/
    UB      tos;            /* パケットの IP ヘッダーTOS*/
    UB      ver;            /* パケットの IP ヘッダーバージョン*/
    UB      num;            /* パケットの受信デバイス番号/
} T_RCV_PKT_INF;
```

## ６．２　定数とマクロ

### (1) IP アドレス

| | |
|---|---|
| ADDR_ANY | 0 の IP アドレス |
| IP_VER4 | IP バージョン４ |

### (2) ポート番号

| | |
|---|---|
| PORT_ANY | 0 のポート番号 |

### (3) IP プロトコル

| | |
|---|---|
| IP_PROTO_TCP | TCP プロトコル |
| IP_PROTO_UDP | UDP プロトコル |
| IP_PROTO_ICMP | ICMP プロトコル |

### (4) ネットワーク・インタフェース制御

| | |
|---|---|
| NET_IP4_CFG | IP アドレス、サブネットマスク等の設定と確認 |
| NET_IP4_TTL | TTL の設定と確認 |
| NET_BCAST_RCV | ブロードキャストの受信の設定と確認 |
| NET_MCAST_JOIN | マルチキャストグループへの参加 |
| NET_MCAST_DROP | マルチキャストグループからの脱退 |
| NET_MCAST_TTL | マルチキャスト送信で使用する TTL の設定 |

### (5) ソケットのパラメータ

| | |
|---|---|
| SOC_IP_TTL | ソケットの TTL の設定と確認 |
| SOC_IP_TOS | ソケットの TOS の設定と確認 |
| SOC_TMO_SND | snd_soc のブロッキングタイムアウトの設定と確認 |
| SOC_TMO_RCV | rcv_soc のブロッキングタイムアウトの設定と確認 |
| SOC_TMO_CON | con_soc のブロッキングタイムアウトの設定と確認 |
| SOC_TMO_CLS | cls_soc のブロッキングタイムアウトの設定と確認 |
| SOC_IP_LOCAL | ローカルホストの IP アドレスとポート番号を取得 |
| SOC_IP_REMOTE | リモートホストの IP アドレスとポート番号を取得 |
| SOC_CBK_HND | コールバック関数の登録 |
| SOC_CBK_FLG | コールバックイベントの指定 |
| SOC_RCV_PKT_INF | 受信パケット情報を取得 |

### (6) ソケットの接続モード

| | |
|---|---|
| SOC_CLI | リモートホストに接続する（能動接続） |
| SOC_SER | 接続を待ち受ける（受動接続） |

### (7) ソケットの終了モード

| | |
|---|---|
| SOC_TCP_CLS | ソケットを切断する。（接続を終了する） |
| SOC_TCP_SHT | 送信処理のみを無効にする。受信は可能。 |

### (8) ソケットの中止モード

| | |
|---|---|
| SOC_ABT_CON | con_soc()の中止 |
| SOC_ABT_CLS | cls_soc()の中止 |
| SOC_ABT_SND | snd_soc()の中止 |
| SOC_ABT_RCV | rcv_soc()の中止 |
| SOC_ABT_ALL | すべてのソケットの処理の中止 |

### (9) コールバックイベント

| | |
|---|---|
| EV_SOC_CON | con_soc()をノンブロッキングモードにする。 |
| EV_SOC_CLS | cls_soc()をノンブロッキングモードにする。 |
| EV_SOC_SND | snd_soc()をノンブロッキングモードにする。 |
| EV_SOC_RCV | rcv_soc()をノンブロッキングモードにする。 |

## ６．３　エラーコード一覧

| | | |
|---|---|---|
| E_NOSPT | -9 | 未サポート機能 |
| E_PAR | -17 | パラメータエラー |
| E_ID | -18 | 不正 ID 番号 |
| E_NOMEM | -33 | メモリ不足 |
| E_OBJ | -41 | オブジェクト状態エラー |
| E_NOEXS | -42 | オブジェクト未生成 |
| E_QOVR | -43 | キューイングオーバフロー |
| E_RLWAI | -49 | 待ち状態の強制解除 |
| E_TMOUT | -50 | ポーリング失敗またはタイムアウト |
| E_CLS | -52 | 待ちオブジェクトの状態変化 |
| E_WBLK | -57 | ノンブロッキング受付 |
| E_BOVR | -58 | バッファオーバフロー |
| EV_ADDR | -98 | デフォルト G/W 未設定 |

## ６．４　API 一覧

| API 名 | |
|---|---|
| **A）ネットワーク・インタフェース** | |
| net_ini | TCP/IP プロトコルスタックの初期化 |
| net_cfg | ネットワーク・インタフェースのパラメータ設定 |
| net_ref | ネットワーク・インタフェースのパラメータ参照 |
| net_acd | IP アドレス重複検出 |
| **B）ネットワークデバイス制御** | |
| net_dev_ini | ネットワークデバイスの初期化 |
| net_dev_cls | ネットワークデバイスの解放 |
| net_dev_ctl | ネットワークデバイスの制御 |
| net_dev_sts | ネットワークデバイスの状態取得 |
| **C） ソケット** | |
| cre_soc | ソケットの生成(Standard 版のみ) |
| del_soc | ソケットの削除(Standard 版のみ) |
| con_soc | ソケットの接続 |
| cls_soc | ソケットの切断 |
| snd_soc | データの送信 |
| rcv_soc | データの受信 |
| cfg_soc | ソケットのパラメータ設定 |
| ref_soc | ソケットのパラメータ参照 |
| abt_soc | ソケット処理の中止 |
| **D） その他** | |
| ip_aton | ドット表記の IPv4 アドレス文字列を 32 ビット値に変換 |
| ip_ntoa | 32 ビット値の IPv4 アドレスをドット表記の IPv4 アドレス文字列に変換 |
| ip_byte2n | IPv4 アドレスの配列を 32 ビット値に変換 |
| ip_n2byte | IPv4 アドレスの 32 ビット値を配列に変換 |
| htons | 16 ビット値をネットワークバイトオーダーへ変換 |
| ntohs | 16 ビット値をホストバイトオーダーへ変換 |
| htonl | 32 ビット値をネットワークバイトオーダーへ変換 |
| ntohl | 32 ビット値をホストバイトオーダーへ変換 |

# 索引


*A*

abt_soc ..... **144**
acd_cbk ..... **125**
**ARP** ..... **25**, **38**

*C*

cfg_soc ..... **138**
cls_soc ..... **137**
con_soc ..... **135**
cre_soc ..... 132

*D*

del_soc ..... **134**
dev_cbk ..... **54**
dev_cls ..... **49**
dev_ctl ..... **50**
dev_ini ..... **48**
dev_ref ..... **51**
dev_snd ..... **52**
**DHCP** ..... **25**
**DNS** ..... **26**

*F*

**FTP** ..... **26**

*H*

htonl ..... **147**
htons ..... **147**
**HTTP** ..... **26**

*I*

**ICMP** ..... **25**, **37**
**IGMP** ..... **25**, **37**
**IP** ..... **25**
ip_aton ..... **149**
ip_byte2n ..... **150**
ip_n2byte ..... **150**
ip_ntoa ..... **149**
**IP アドレス** ..... **23**
**IP アドレス競合検出** ..... 38
**IP 再構築とフラグメンテーション** ..... **37**

*M*

**MAC アドレス** ..... **24**, **47**

*N*

net_acd ..... **124**
net_buf_get ..... 60
net_buf_ret ..... **61**
net_cfg ..... **121**
net_dev_cls ..... **128**
net_dev_ctl ..... **129**
net_dev_ini ..... **127**
net_dev_sts ..... **130**
net_ini ..... **120**
net_memcmp(ver2) ..... 63
net_memcmp(ver3) ..... 10, 65
net_memcpy(ver2) ..... 63
net_memcpy(ver3) ..... 10, 65
net_memset(ver2) ..... 62
net_memset(ver3) ..... 8, 64
net_pkt_rcv ..... **55**
net_ref ..... **123**
ntohl ..... **148**
ntohs ..... **148**

*R*

rcv_soc ..... **146**
ref_soc ..... **142**

*S*

snd_soc ..... **145**
soc_cbt ..... **140**

*T*

TCP.....25, 41
TOS.....36
TTL.....36

*U*

UDP.....25, 38

*あ*

アドレス解決.....38

*こ*

コールバック関数.....26

*し*

受信バッファサイズ.....102
受信バッファサイズ.....75

*そ*

送信バッファ.....42, 102
送信バッファサイズ.....75
ソケット.....26, 76, 78, 101, 131

*て*

デバイスタイプ.....47
デバイスドライバ関数.....47
デバイス番号.....46, 127, 131
デバイス名.....46

*ね*

ネットワーク・デバイスドライバ.....46
ネットワークバッファー.....56, 59

*の*

ノード.....24
ノンブロッキング.....26

*は*

パケット.....24

*ひ*

ビックエンディアン.....24

*ふ*

ブロードキャスト.....36
ブロードキャストアドレス.....24
ブロッキング.....26
プロトコル.....23
プロトコルスタック.....23, 119

*ほ*

ポート番号.....24
ホスト.....24

*ま*

マルチキャスト.....36
マルチキャストアドレス.....24

*り*

リソース.....27
リトルエンディアン.....24

*る*

ループバックインタフェース.....57

**μNet3 ユーザーズガイド**

---

2009 年 6 月　初版
2010 年 5 月　第２版
2010 年 12 月　第３版
2011 年 5 月　第４版
2012 年 7 月　第５版
2012 年 10 月　第６版
2013 年 1 月　第７版
2014 年 2 月　第８版
2014 年 4 月　第９版
2014 年 5 月　第１０版

イー・フォース株式会社　http://www.eforce.co.jp/
〒103-0006 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