お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注 1. 本資料において使用されている「当社」とは、ルネサスエレクトロニクス株式会社およびルネサスエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

RENESAS

ユーザーズ・マニュアル

RX-NET

ネットワーク・ライブラリ

TCP/IP Ver.1.30

対象デバイス

V850シリーズ™

対象リアルタイムOS

RX850 Pro Ver.3.15

資料番号　U15083JJ4V0UM00（第4版）
発行年月　August 2002　CP(K)

〔メ　モ〕

目次要約

V850 シリーズ，V853，V850/SA1，V850/SB1，V850/SB2，V850/SF1，V850/SV1，V850E/MS1，V850E/MS2，V850E/MA1, V850E/MA2, V850E/IA1, V850E/IA2は，日本電気株式会社の商標です。

PC/ATは米国IBM社の商標です。

Windows, Windows NTは，米国Microsoft Corporationの米国およびその他の国における登録商標または商標です。

Green Hills Softwareは米国Green Hills Software, Inc.の商標です。

その他，記載の会社名／製品名は，各社の商標，または，登録商標です。

本製品は外国為替および外国貿易管理法の規定により規制貨物等（または役務）に該当しますので，
日本国外に輸出する場合には，同法に基づき日本国政府の輸出許可が必要です。

- **本資料の内容は予告なく変更することがありますので，最新のものであることをご確認の上ご使用ください。**
- **文書による当社の承諾なしに本資料の転載複製を禁じます。**
- **本資料に記載された製品の使用もしくは本資料に記載の情報の使用に際して，当社は当社もしくは第三者の知的財産権その他の権利に対する保証または実施権の許諾を行うものではありません。上記使用に起因する第三者所有の権利にかかわる問題が発生した場合，当社はその責を負うものではありませんのでご了承ください。**
- **本資料に記載された回路，ソフトウエア，及びこれらに付随する情報は，半導体製品の動作例，応用例を説明するためのものです。従って，これら回路・ソフトウエア・情報をお客様の機器に使用される場合には，お客様の責任において機器設計をしてください。これらの使用に起因するお客様もしくは第三者の損害に対して，当社は一切その責を負いません。**

M7A 98.8

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

はじめに

対象者　このマニュアルは，V850シリーズの応用システムを設計，開発するユーザを対象とします。

目　的　このマニュアルは，次の構　成に示すRX-NETの機能をユーザに理解していただくことを目的としています。

構　成　このマニュアルは，大きく分けて次の内容で構成しています。

- ●概　説
- ●インストレーション
- ●システム構築
- ●ネットワーク通信機能
- ●API関数
- ●RX-NET（TCP/IP）依存部

読み方　このマニュアルの読者には，電気，論理回路，マイクロコンピュータ，C 言語，アセンブリ言語に関する一般知識が必要です。

V850 シリーズのハードウエア機能，命令機能を知りたいとき
→各製品のユーザーズ・マニュアルを参照してください。

凡　例　注　：本文中につけた注の説明
注意　：気をつけて読んでいただきたい内容
備考　：本文の補足説明
数の表記　：2 進数 …XXXX または B'XXXX
10 進数…XXXX
16 進数…0xXXX または H'XXXX
2 のべき数を示す接頭語（アドレス空間，メモリ容量）：
K（キロ）：2^{10}＝1024
M（メガ）：2^{20}＝1024^2
G（ギガ）：2^{30}＝1024^3

関連資料　このマニュアルを使用する場合は，次の資料もあわせてご覧ください。
関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。
あらかじめご了承ください。

開発ツールに関する資料（ユーザーズ・マニュアル）　(1/2)

資　料　名	資料番号	
	和文	英文
IE-703002-MC（V853™, V850/SA1™, V850/SB1™, V850/SB2™, V850/SC1™, V850/SC2™, V850/SC3™,V850/SF1™, V850/SV1™用インサーキット・エミュレータ）	U11595J	U11595E

開発ツールに関する資料（ユーザーズ・マニュアル） (2/2)

資　料　名		資料番号	
		和文	英文
IE-V850E-MC（V850E/IA1™,V850E/IA2™用インサーキット･エミュレータ）， IE-V850E-MC-A（V850E/MA1™,V850E/MA2™用インサーキット・エミュレータ）		U14487J	U14487E
IE-703003-MC-EM1（V853用インサーキット・エミュレータ・オプション・ボード）		U11596J	U11596E
IE-703017-MC-EM1（V850/SA1用インサーキット・エミュレータ・オプション・ボード）		U12898J	U12898E
IE-703037-MC-EM1（V850/SB1,V850/SB2用インサーキット・エミュレータ・オプション・ボード）		U14151J	U14151E
IE-703040-MC-EM1（V850/SV1用インサーキット・エミュレータ・オプション・ボード）		U14337J	U14337E
IE-703079-MC-EM1（V850/SF1用インサーキット・エミュレータ・オプション・ボード）		U15447J	U15447E
IE-703102-MC（V850E/MS1™,V850E/MS2™用インサーキット・エミュレータ）		U13875J	U13875E
IE-703102-MC-EM1, IE-703102-MC-EM1-A（V850E/MS1，V850E/MS2用インサーキット･エミュレータ・オプション・ボード）		U13876J	U13876E
IE-703107-MC-EM1（V850E/MA1,V850E/MA2用インサーキット・エミュレータ・オプション・ボード）		U14481J	U14481E
IE-703116-MC-EM1（V850E/IA1用インサーキット・エミュレータ・オプション・ボード）		U14700J	U14700E
CA850 Ver.2.50　Cコンパイラ・パッケージ	操作編	U16053J	作成予定
	C言語編	U16054J	U16054E
	PM plus編	U16055J	作成予定
	アセンブリ言語編	U16042J	U16042E
ID850 Ver.2.50　統合ディバッガ	操作編　Windowsベース	U15181J	U15181E
SM850 Ver.2.50　システム・シミュレータ	操作編　Windowsベース	U15182J	U15182E
SM850 Ver.2.00以上　システム・シミュレータ	外部部品ユーザ・オープン・インタフェース仕様編	U14873J	U14873E
RX850 Ver.3.13以上　リアルタイムOS	基礎編	U13430J	U13430E
	インストレーション編	U13410J	U13410E
	テクニカル編	U13431J	U13431E
RX850 Pro Ver.3.13　リアルタイムOS	基礎編	U13773J	U13773E
	インストレーション編	U13774J	U13774E
	テクニカル編	U13772J	U13772E
RX-NET　ネットワーク・ライブラリ（TCP/IP）Ver.1.30		このマニュアル	−
RX-NET　ネットワーク・ライブラリ（PPP）		U15303J	−
RX-NET　ネットワーク・ライブラリ（DNS）		U15304J	−
RX-NET　ネットワーク・ライブラリ（DHCP）		U15382J	−
RX-NET　ネットワーク・ライブラリ（SMTP）		U15505J	−
RX-NET　ネットワーク・ライブラリ（POP）		U15539J	−
RX-NET Ver.1.00 ネットワーク・ライブラリ（telnet）		U16085J	−
RX-NET Ver.1.10 ネットワーク・ライブラリ（FTP）		U15946J	−
RX-NET Ver.1.00 ネットワーク・ライブラリ（WebServer）		U16294J	−
RD850　Ver.3.01　タスク・ディバッガ		U13737J	U13737E
RD850 Pro Ver.3.01　タスク・ディバッガ		U13916J	U13916E
AZ850 Ver.3.10　システム・パフォーマンス・アナライザ		U14410J	U14410E
PG-FP3　フラッシュ・メモリ・プログラマ		U13502J	U13502E
PG-FP4　フラッシュ・メモリ・プログラマ		U15260J	U15260E

目次

図目次

表目次

第 1 章　概説

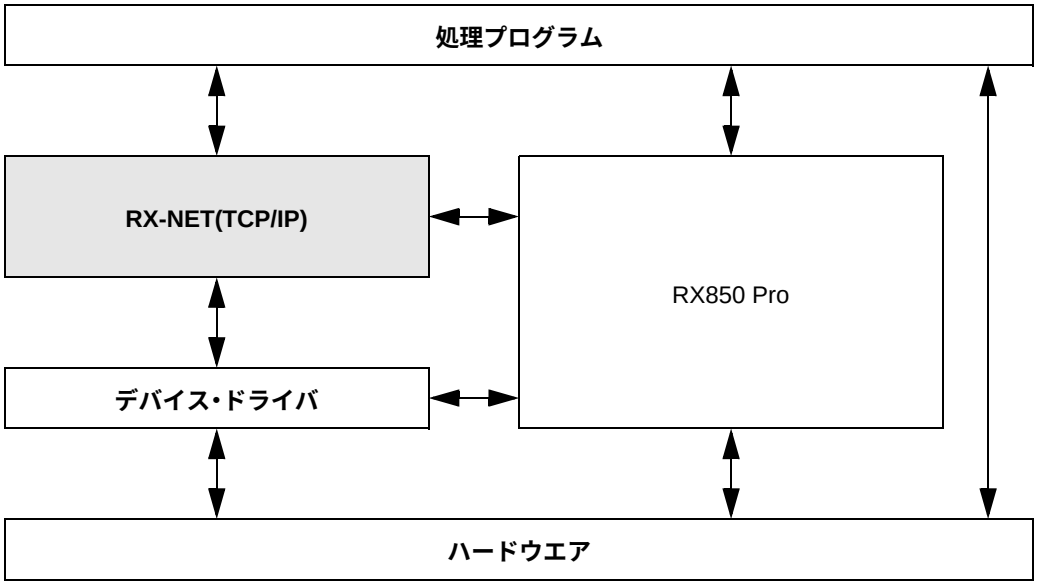
1.1　概要

RX-NET(TCP/IP) は，TCP/IP インタフェースの業界標準として位置付けられている “ ソケット・インタフェース ” に準拠した設計が行われており，組み込み型制御用リアルタイム・オペレーティング・システム RX850 Pro (μITRON3.0 仕様準拠，NEC 製) 上で動作する処理プログラムに対し，ネットワーク通信を実現するためのアプリケーション・プログラム・インタフェース関数 (Application Program Interface 関数) を提供しています。

したがって，ユーザは RX-NET(TCP/IP) が提供する API 関数を利用することにより，複数の処理プログラム間におけるネットワーク通信を容易に実現することが可能となります。

図 1-1 に，RX-NET(TCP/IP) の位置付けを示します。

図 1-1　RX-NET(TCP/IP) の位置付け



1.2　特徴

以下に，RX-NET(TCP/IP) の特徴を示します。

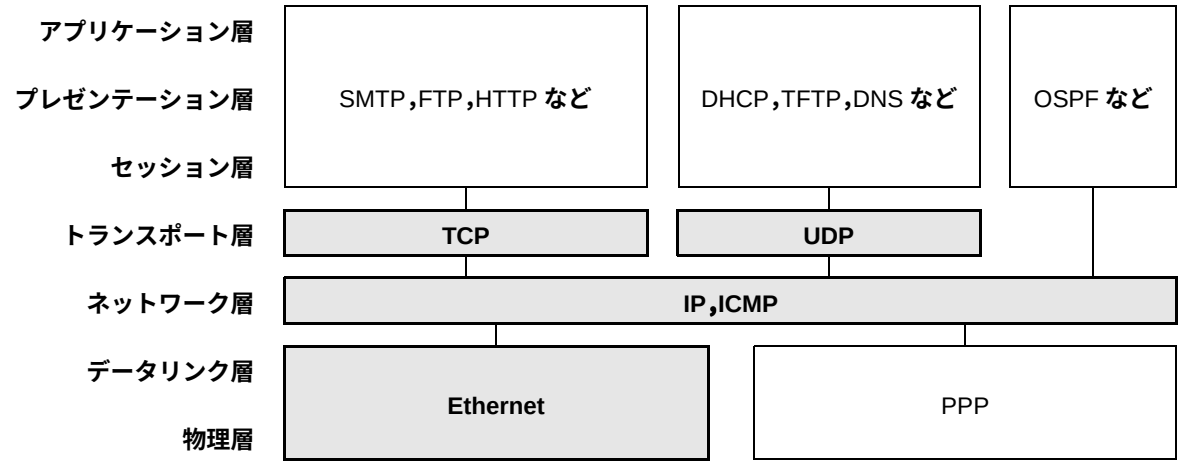
- RFC に準拠

  RX-NET(TCP/IP) では，RFC に準拠した設計が行われています。
  なお，RFC とは，インターネットに関する研究開発機関 IETF (Internet Engineering Task Force) が取りまとめた公開技術文書であり，電子メール，ファイル転送などのプロトコル仕様 (情報交換を行う際に必要な手順，および，規約) の他にも，各種サービス，ガイドラインといった多岐に渡った情報 (ネットワーク技術の実装と運用に主眼を置いた情報) が記載されています。

- ネットワーク通信機能をサポート

  RX-NET(TCP/IP) では，ネットワーク通信を実現する際に必要となる各種 API 関数を提供しています。したがって，RX-NET(TCP/IP) を組み込んだ機器は，インターネットなどへのネットワーク接続が可能となります。
  図 1-2 に，RX-NET(TCP/IP) の階層的位置付けを示します。

図 1-2　RX-NET(TCP/IP) の階層的位置付け



- 高い移植性

  RX-NET(TCP/IP) が処理を実行するうえで必要となるユーザの実行環境 / アプリケーション・システムに依存した処理については，RX-NET(TCP/IP) 依存部 (ユーザ・オウン・コーディング部) として切り出し，サンプル・ソース・ファイルを提供しています。
  これにより，実行環境への移植性を向上させるとともに，カスタマイズ化を容易なものとしています。

- マルチタスク処理を意識した設計

  RX-NET(TCP/IP) が提供する API 関数では，マルチタスク処理を考慮した設計が行われています。このため，ユーザが処理プログラムを記述する際，API 関数の発行に伴うタスク間の排他制御などを意識する必要がありません。

1.3　実行環境

以下に，RX-NET(TCP/IP) が処理を実行するうえで必要となるハードウエアを示します。

- プロセッサ

  以下に，RX-NET(TCP/IP) が処理を実行するうえで必要となるプロセッサを示します。

  V850 シリーズ V850E/xxx

- 周辺コントローラ

  RX-NET(TCP/IP) では，処理を実行するうえで，特定の周辺コントローラは必要ありません。

- メモリ容量

  以下に，RX-NET(TCP/IP) が処理を実行するうえで必要となるメモリ容量を示します。

  RX-NET(TCP/IP) のテキスト領域　：約 50 K バイト
  RX-NET(TCP/IP) のデータ領域　：約 150 K バイト

1.4　開発環境

以下に，RX-NET(TCP/IP) を使用した処理プログラムを開発するうえで必要となるハードウエア，および，ソフトウエアを示します。

- ハードウエア
  - ホスト・マシン (下記のいずれか)

    PC-98NX/PC09821 シリーズ　：Windows 2000，98，Me，XP，NT 4.0
    IBM-PC/AT 互換機　：Windows 2000，98，Me，XP，NT 4.0
    SPARC station　：Solaris Rel.2.5.*x*

- ソフトウエア
  - リアルタイム OS

    RX850 Pro Ver.3.15 以上　：NEC 製

  - C コンパイラ・パッケージ (下記のいずれか)

    CA850 Ver.2.41 以上　：NEC 製
    CCV850E Ver.1.8.9 Rel.4.0.2 以上　：米国 Green Hills Softwareoyobi, Inc. 製

第 2 章　インストレーション

本章では，RX-NET(TCP/IP) の提供媒体に格納されているファイル群をユーザの開発環境 (ホスト・マシン) 上にインストールする際の手順について解説しています。

2.1　概要

RX-NET(TCP/IP) の提供媒体は，ホスト・マシンの種類 (Windows ベース，UNIX ベース) に併せて計 2 種類が用意されています。
表 2-1 に，RX-NET(TCP/IP) の提供形式一覧を示します。

表 2-1　RX-NET(TCP/IP) の提供形式

ホスト・マシン	提供形式	提供媒体
Windows ベース • PC-98NX/PC-9821 シリーズ • IBM-PC/AT 互換機	CA850 対応版オブジェクト・ファイル形式 CCV850E 対応版オブジェクト･ファイル形式	CD-ROM
UNIX ベース • SPARC station	CA850 対応版オブジェクト・ファイル形式 CCV850E 対応版オブジェクト･ファイル形式	CD-ROM

注意　ホスト･マシンの種類別に用意された提供媒体には,2 種類 (CA850 対応版オブジェクト･ファイル形式,CCV850E 対応版オブジェクト・ファイル形式) の RX-NET(TCP/IP) が格納されています。したがって，提供媒体からホスト・マシン上にファイル群をインストールする際には，ユーザが使用する C コンパイラ・パッケージに対応した RX-NET(TCP/IP) をインストールする必要があります。

2.2　インストール手順

RX-NET(TCP/IP) の提供媒体に格納されているファイル群のインストール手順は，ホスト・マシンの種類 (Windows ベース，UNIX ベース) により異なります。
そこで，以降に，ホスト・マシンが Windows ベースの場合，UNIX ベースの場合のインストール手順をそれぞれに示します。

2.2.1　Windows ベース

以下に，RX-NET(TCP/IP) の提供媒体に格納されているファイル群をホスト・マシン (Windows ベース：PC-98NX/PC-9821 シリーズ，IBM-PC/AT 互換機) 上にインストールする際の手順を示します。

1) Windows の起動
ホスト・マシン，および，周辺機器などの電源を投入し，Windows を起動します。

2) 提供媒体のセット
RX-NET(TCP/IP) の提供媒体をホスト･マシンの該当デバイス装置 (CD-ROM ドライブ) にセットすることにより，セットアップ・プログラムが自動実行します。
以降，モニタ画面に表示されるメッセージに従ってインストール作業を実行します。

注意　セットアップ・プログラムが自動実行しない場合には，RX-NET(TCP/IP) の提供媒体のディレクトリ RX-NET_V850E_NEC\DISK1 に格納されている SETUP.EXE を起動します。

3) ファイル群の確認
Windows の標準アプリケーション Explorer などを用いて，RX-NET(TCP/IP) の提供媒体に格納されていたファイル群がホスト・マシン上にインストールされたことを確認します。
なお，各ディレクトリについての詳細は，「2.3　ディレクトリ構成」を参照してください。

2.2.2　UNIX ベース

以下に，RX-NET(TCP/IP) の提供媒体に格納されているファイル群をホスト・マシン (UNIX ベース：SPARC station) 上にインストールする際の手順を示します。
ただし，入力例中の “ % ” はシェル・プロンプトを，“ △ ” はスペース・キーの入力を，“ <Enter> ” はエンター・キーの入力を表しています。

1) ホスト・マシンへのログイン
ホスト・マシンにログインします。

```
%
```

2) ディレクトリの移動
cd コマンドを実行し，インストール用ディレクトリに移動します。
なお，下記入力例では，インストール用ディレクトリとして /usr/local を指定しています。

注意　インストール用ディレクトリのパーミッション (read，write，execute) はインストール作業者に対して許可状態である必要があります。そこで，インストール用ディレクトリのパーミッションが不許可状態であった場合には，chmod コマンドを実行し，パーミッションを不許可状態から許可状態に変更します。

```
% cd △ /usr/local <Enter>
```

3) 提供媒体のセット
RX-NET(TCP/IP) の提供媒体をホスト・マシンの該当デバイス装置 (CD-ROM ドライブ) にセットします。

4) デバイスのマウント
mount コマンドを実行し，該当デバイス装置に対応したデバイスをマウントします。
なお，下記入力例では，該当デバイス装置のデバイス名 (スペシャル・ファイル名) として /dev/rst8 を，マウント・ディレクトリとして /cdrom を指定しています。

注意　ホスト・マシンによっては，“ デバイスのマウント ”が自動的に行われるものがあります。このような場合，mount コマンドを実行する必要はありません。

```
% mount △ /dev/rst8 △ /cdrom <Enter>
```

5) ファイル群のインストール
tar コマンドを実行し，マウント・ディレクトリ /cdrom 下の圧縮ファイルをインストール用ディレクトリに展開します。
ただし，提供媒体には，以下に示した 2 種類の圧縮ファイルが格納されています。

- CA850 対応版
- CCV850E 対応版

そこで，ユーザが使用する C コンパイラ・パッケージが NEC 製 CA850 の場合は圧縮ファイル nec/rxnet.tar を，米国 Green Hills Software, Inc. 製 CCV850E の場合は圧縮ファイル ghs/rxnet.tar を展開します。

【 CA850 対応版の場合 】

```
% tar △ -xvof △ /cdrom/RXNET/nec/rxnet.tar <Enter>
```

【 CCV850E 対応版の場合 】

```
% tar △ -xvof △ /cdrom/RXNET/ghs/rxnet.tar <Enter>
```

6) ファイル群の確認
ls コマンドを実行し，RX-NET(TCP/IP) の提供媒体に格納されていたファイル群がホスト・マシン上にインストールされたことを確認します。
なお，各ディレクトリについての詳細は，「2.3　ディレクトリ構成」を参照してください。

```
% ls △ -CFR △ /usr/local/nectools32 <Enter>
```

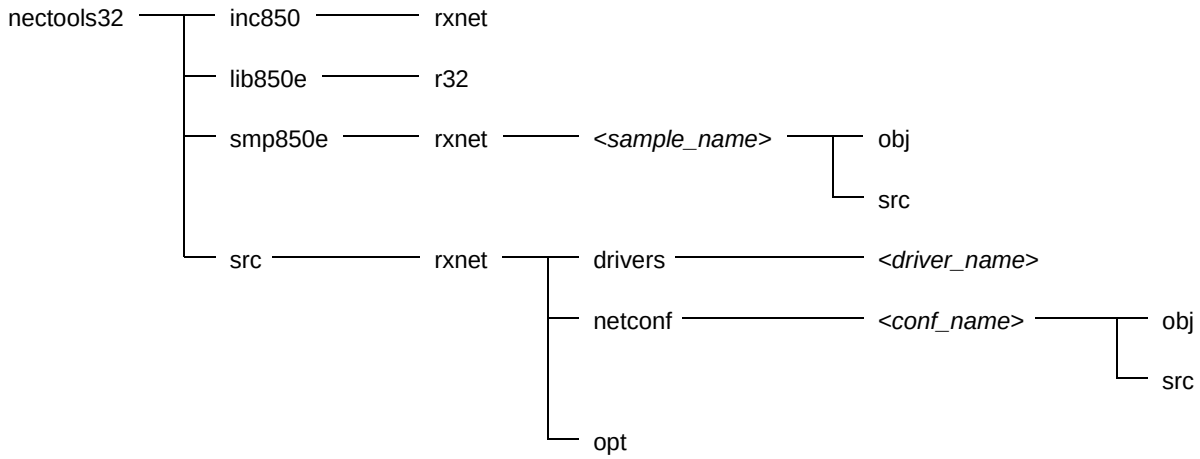
2.3　ディレクトリ構成

RX-NET(TCP/IP) の提供媒体に格納されているファイル群のディレクトリ構成は，ユーザが使用する C コンパイラ・パッケージの種類 (NEC 製 CA850，米国 Green Hills Software, Inc. 製 CCV850E) により異なります。
そこで，以降に，C コンパイラ・パッケージが CA850 の場合，CCV850E の場合のディレクトリ構成をそれぞれに示します。

2.3.1　CA850 対応版

図 2-1 に，RX-NET(TCP/IP) の提供媒体 (CA850 対応版) に格納されているファイル群をホスト・マシン上にインストールした際に生成されるディレクトリ構成を示します。

図 2-1　ディレクトリ構成 (CA850 対応版)



以下に，各ディレクトリの概要を示します。

1) nectools32\inc850
RX-NET(TCP/IP) の標準ヘッダ・ファイルが格納されているディレクトリです。

rxnet.h	： RX-NET(TCP/IP) 用標準ヘッダ・ファイル

2) nectools32\inc850\rxnet
RX-NET(TCP/IP) のヘッダ・ファイルが格納されているディレクトリです。

3) nectools32\lib850e\r32
TCP/IP ライブラリ (32 レジスタ・モード) が格納されているディレクトリです。

librxnet.a	： TCP/IP ライブラリ
libnodhcp.a	： DHCP ダミー・エントリ・ライブラリ
rxnetcfg_<*driver_name*>.o	： デバイス・ドライバ・オブジェクト

4) nectools32\smp850e\rxnet\<*sample_name*>\obj
ロード・モジュールを生成するためのメイク・ファイル Makefile が格納されているディレクトリです。
なお，本ディレクトリにおいて，make コマンドを実行することにより，ロード・モジュール sample.out が本ディレクトリに生成されます。

Makefile	： ロード・モジュール用メイク・ファイル

5) nectools32\smp850e\rxnet\<*sample_name*>\src
サンプル・プログラムのソース・ファイル，および，ヘッダ・ファイルが格納されているディレクトリです。

6) nectools32\src\rxnet\drivers\<*driver_name*>
ネットワーク物理層に該当するデバイス・ドライバのソース・ファイル，および，ヘッダ・ファイルが格納されているディレクトリです。

7) nectools32\src\rxnet\netconf\<*conf_name*>\obj
デバイス・ドライバ・オブジェクト (ネットワーク物理層に該当するデバイス・ドライバ，コンフィギュレーション情報依存処理部) を生成するためのメイク・ファイル Makefile が格納されているディレクトリです。

なお，本ディレクトリにおいて，make コマンドを実行することにより，デバイス・ドライバ・オブジェクト rxnetcfg_<*driver_name*>.o が本ディレクトリ，および，ディレクトリ nectools32\lib850e\r32 に生成されます。

Makefile	：デバイス・ドライバ・オブジェクト用メイク・ファイル

8) nectools32\src\rxnet\netconf\<*conf_name*>\src
コンフィギュレーション情報依存処理部のソース・ファイル，および，ヘッダ・ファイルが格納されているディレクトリです。

9) nectools32\src\rxnet\opt
RX-NET(TCP/IP) の C 言語マクロ・ヘッダ・ファイル，および，静的設定情報ヘッダ・ファイルが格納されているディレクトリです。

libconf.h	：C 言語マクロ・ヘッダ・ファイル
fnsconfig.h	：静的設定情報ヘッダ・ファイル

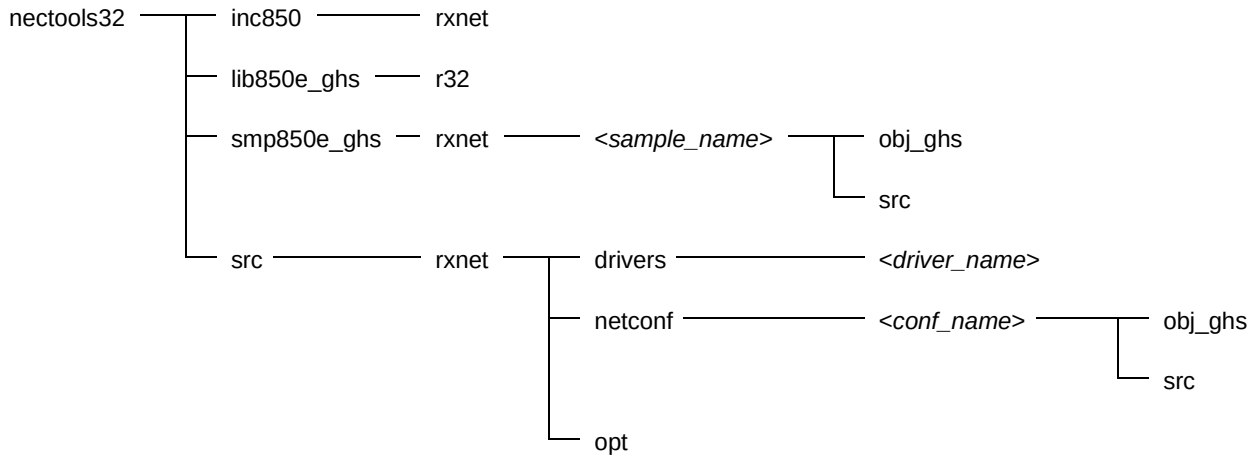
注意　<*sample_name*>，<*driver_name*>，<*conf_name*> についての詳細は，下記に示したテキスト・ファイルを参照してください。

<*sample_name*>	：nectools32\smp850e\rxnet\README
<*driver_name*>	：nectools32\src\rxnet\drivers\README
<*conf_name*>	：nectools32\src\rxnet\netconf\README

2.3.2　CCV850E 対応版

図 2-2 に，RX-NET(TCP/IP) の提供媒体 (CCV850E 対応版) に格納されているファイル群をホスト・マシン上にインストールした際に生成されるディレクトリ構成を示します。

図 2-2　ディレクトリ構成 (CCV850E 対応版)



以下に，各ディレクトリの概要を示します。

1) nectools32\inc850
RX-NET(TCP/IP) の標準ヘッダ・ファイルが格納されているディレクトリです。

rxnet.h	： RX-NET(TCP/IP) 用標準ヘッダ・ファイル

2) nectools32\inc850\rxnet
RX-NET(TCP/IP) のヘッダ・ファイルが格納されているディレクトリです。

3) nectools32\lib850e_ghs\r32
TCP/IP ライブラリ (32 レジスタ・モード) が格納されているディレクトリです。

librxnet.a	： TCP/IP ライブラリ
libnodhcp.a	： DHCP ダミー・エントリ・ライブラリ
rxnetcfg_<*driver_name*>.o	： デバイス・ドライバ・オブジェクト

4) nectools32\smp850e_ghs\rxnet\<*sample_name*>\obj_ghs
ロード・モジュールを生成するためのビルド・ファイル sample.bld が格納されているディレクトリです。
なお，本ディレクトリの sample.bld を用いることにより，ロード・モジュール sample.out が本ディレクトリに生成されます。

sample.bld	： ロード・モジュール用ビルド・ファイル

5) nectools32\smp850e_ghs\rxnet\<*sample_name*>\src
サンプル・プログラムのソース・ファイル，および，ヘッダ・ファイルが格納されているディレクトリです。

6) nectools32\src\rxnet\drivers\<*driver_name*>
ネットワーク物理層に該当するデバイス・ドライバのソース・ファイル，および，ヘッダ・ファイルが格納されているディレクトリです。

7) nectools32\src\rxnet\netconf\<*conf_name*>\obj_ghs
デバイス・ドライバ・オブジェクト (ネットワーク物理層に該当するデバイス・ドライバ，コンフィギュレーション情報依存処理部) を生成するためのビルド・ファイル netconf.bld が格納されているディレクトリです。
なお，本ディレクトリの netconf.bld を用いることにより，デバイス・ドライバ・オブジェクト rxnetcfg_<*driver_name*>.o が本ディレクトリ，および，ディレクトリ nectools32\lib850e_ghs\r32 に生成されます。

netconf.bld	： デバイス・ドライバ・オブジェクト用ビルド・ファイル

8) nectools32\src\rxnet\netconf\<*conf_name*>\src
コンフィギュレーション情報依存処理部のソース・ファイル，および，ヘッダ・ファイルが格納されているディレクトリです。

9) nectools32\src\rxnet\opt
RX-NET(TCP/IP) **の** C **言語マクロ・ヘッダ・ファイル，および，静的設定情報ヘッダ・ファイルが格納されているディレクトリです。**

libconf.h	**：** C **言語マクロ・ヘッダ・ファイル**
fnsconfig.h	**：静的設定情報ヘッダ・ファイル**

注意 <*sample_name*>**，**<*driver_name*>**，**<*conf_name*> **についての詳細は，下記に示したテキスト・ファイルを参照してください。**

<*sample_name*>	**：** nectools32\smp850e_ghs\rxnet\README
<*driver_name*>	**：** nectools32\src\rxnet\drivers\README
<*conf_name*>	**：** nectools32\src\rxnet\netconf\README

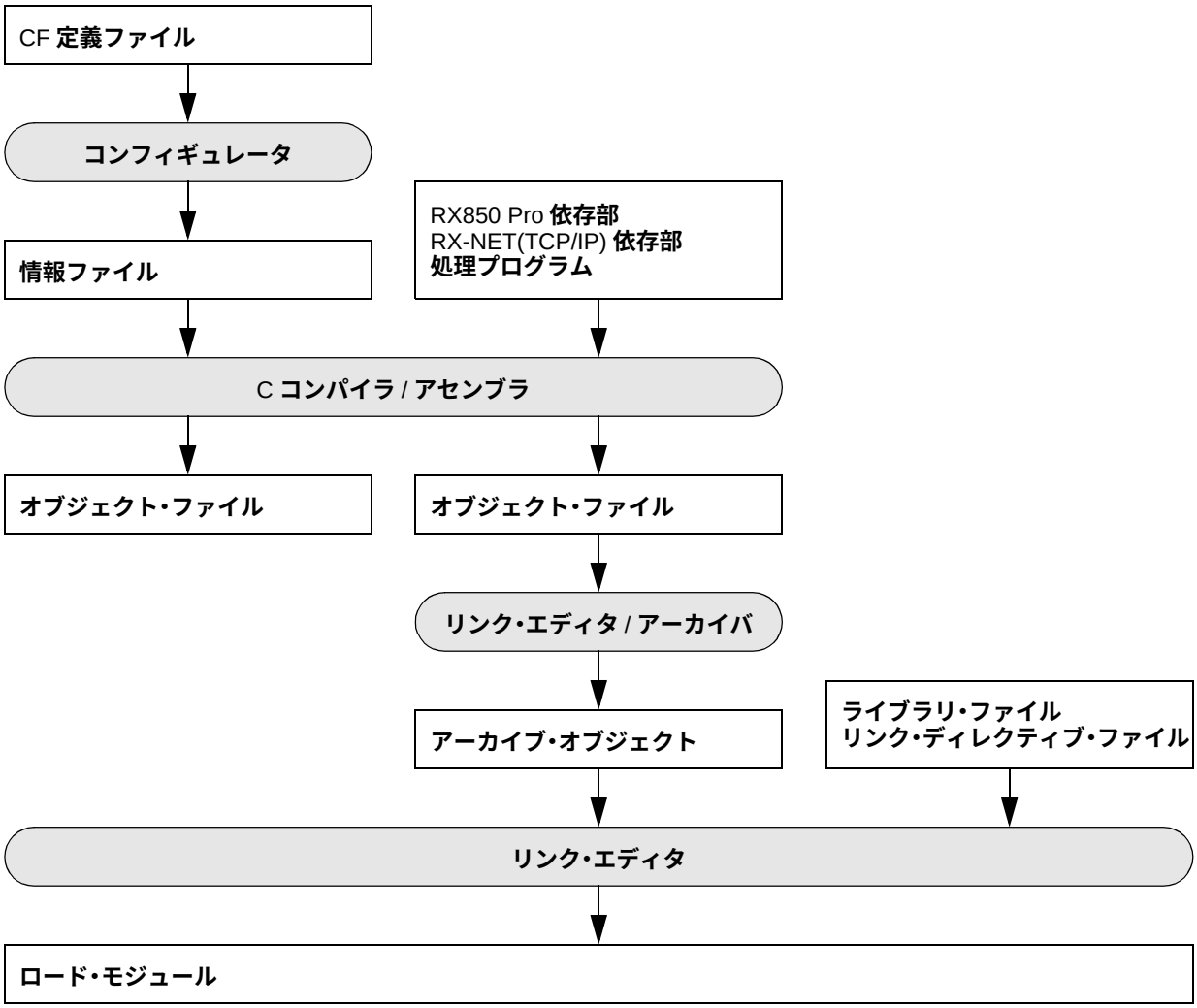
第 3 章　システム構築

本章では，RX-NET(TCP/IP) を使用したネットワーク・アプリケーション (ロード・モジュール) の構築手順を解説しています。

3.1　概要

システム構築とは，RX-NET(TCP/IP) の提供媒体からユーザの開発環境 (ホスト・マシン) 上にインストールしたファイル群を用いてネットワーク・アプリケーション (ロード・モジュール) を生成することです。
図 3-1 に，RX-NET(TCP/IP) のシステム構築手順を示します。

図 3-1　システム構築手順



3.2　CF 定義ファイルの記述

組み込み型制御用リアルタイム・オペレーティング・システム RX850 Pro(μITRON3.0 仕様準拠，NEC 製) の管理下で動作する処理プログラムを作成する場合，RX850 Pro に提供するコンフィギュレーション情報 (リアルタイム OS 情報，SIT 情報，SCT 情報) を保持した CF 定義ファイルが必要となります。

なお，RX-NET(TCP/IP) では 1 個のタスク，1 個の周期起動ハンドラ，9 種類のシステム・コールを，RX-NET(TCP/IP) 依存部では 1 個のタスク，1 個の間接起動割り込みハンドラ，5 種類のシステム・コールを利用して各種機能を実現しています。

- SIT 情報
  - システム情報
    RX-NET(TCP/IP) 用に " タスクの自動割り付け ID 番号 " として 1 個分を確保。
    RX-NET(TCP/IP) 依存部用に " タスクの自動割り付け ID 番号 " として 1 個分を確保。
  - システム最大値情報
    RX-NET(TCP/IP) 用に " タスクの最大生成数 " として 1 個分を確保。
    RX-NET(TCP/IP) 用に " 周期起動ハンドラの最大登録数 " として 1 個分を確保。
    RX-NET(TCP/IP) 依存部用に " タスクの最大生成数 " として 1 個分を確保。
    RX-NET(TCP/IP) 依存部用に " 間接起動割り込みハンドラの最大登録数 " として 1 個分を確保。
- SCT 情報
  - タスク管理機能情報
    RX-NET(TCP/IP) 用，RX-NET(TCP/IP) 依存部用に " cre_tsk，sta_tsk " を定義。
    RX-NET(TCP/IP) 依存部用に " dis_dsp，ena_dsp，get_tid " を定義。
  - タスク付属同期機能情報
    RX-NET(TCP/IP) 依存部用に " slp_tsk，wup_tsk " を定義。
  - 割り込み処理管理機能情報
    RX-NET(TCP/IP) 依存部用に " def_int " を定義。
  - 時間管理機能情報
    RX-NET(TCP/IP) 用に " get_tim，dly_tsk，def_cyc " を定義。
  - システム管理機能情報
    RX-NET(TCP/IP) 用に " ref_sys " を定義。

注意 1　CF 定義ファイルを記述する際の注意事項，および，コンフィギュレーション情報についての詳細は，「RX850 Pro ユーザーズ・マニュアル　インストレーション編」を参照してください。

注意 2　RX-NET(TCP/IP) では，CF 定義ファイルのサンプル・ソース・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\smp850e\rxnet\<*sample_name*>\src
sys.cf　　：CF 定義ファイル

【CCV850E 対応版の場合】
nectools32\smp850e_ghs\rxnet\<*sample_name*>\src
sys.cf　　：CF 定義ファイル

3.3　情報ファイルの生成

「3.2　CF 定義ファイルの記述」で作成された CF 定義ファイルに対して RX850 Pro が提供するユーティリティ・ツール (コンフィギュレータ cf850pro) を実行し，情報ファイル (システム情報テーブル，システム・コール・テーブル，システム情報ヘッダ・ファイル) を生成します。

以下に，シェル・プロンプトのコマンド・ラインから cf850pro を実行する際の入力例 (CF 定義ファイル sys.cf を読み込んだのち，システム情報テーブル sit.s，システム・コール・テーブル svc.s，システム情報ヘッダ・ファイル sys.h を出力) を示します。

ただし，入力例中の " C> " はシェル・プロンプトを，" △ " はスペース・キーの入力を，" <Enter> " はエンター・キーの入力を表しています。

```
C> cf850pro △ -i △ sit.s △ -c △ svc.s △ -d △ sys.h △ sys.cf △ <Enter>
```

注意 1　コンフィギュレータ cf850pro の起動オプション，および，実行方法についての詳細は，「RX850 Pro ユーザーズ・マニュアル　インストレーション編」を参照してください。

注意 2　RX-NET(TCP/IP) では，情報ファイルを生成するためのサンプル・コマンド・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\smp850e\rxnet\<*sample_name*>\obj
Makefile　　：ロード・モジュール用メイク・ファイル

【CCV850E 対応版の場合】
nectools32\smp850e_ghs\rxnet\<*sample_name*>\obj_ghs
sample.bld　　：ロード・モジュール用ビルド・ファイル

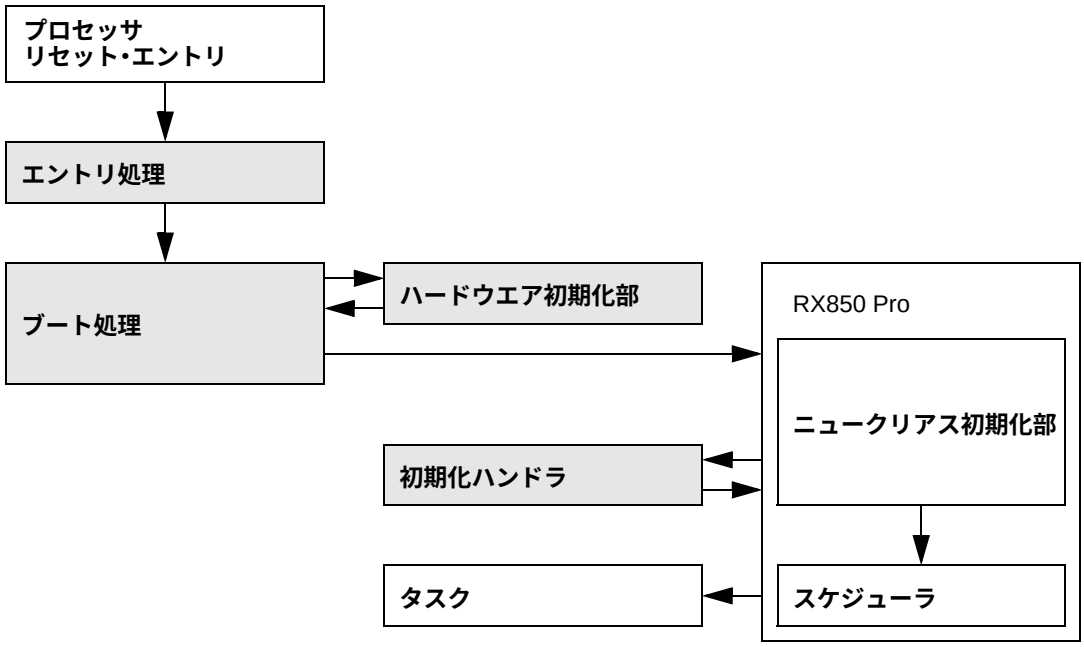
3.4　RX850 Pro 依存部の記述

RX-NET(TCP/IP) では，RX850 Pro が提供する機能を利用して各種機能を実現しています。また，ユーザが記述した処理プログラムは，RX850 Pro の管理下でその処理を実行することになります。

したがって，RX850 Pro を正常に動作させるうえで必要となる RX850 Pro 依存部 (ユーザ・オウン・コーディング部) の記述が必要となります。

図 3-2 に，RX850 Pro 依存部の処理の流れを示します。

図 3-2　RX850 Pro 依存部の処理の流れ



以下に，RX850 Pro 依存部の一覧を示します。

- エントリ処理
  割り込みが発生した際にプロセッサが強制的に制御を移すハンドラ･アドレスに対して該当処理(ブート処理,RX850 Pro が提供する割り込み処理管理機能，直接起動割り込みハンドラ) への分岐処理を割り付けるために用意された処理ルーチンです。

- ブート処理
  RX850 Pro が処理を実行するうえで必要となる最低限の初期化処理を行うために用意された処理ルーチンであり,エントリ処理 (プロセッサのリセット・エントリに割り付けられた分岐処理) から呼び出されます。

- ハードウエア初期化部
  RX850 Pro が処理を実行するうえで必要となるハードウエアの初期化処理を行うために用意された処理ルーチンであり，ブート処理から呼び出されます。

なお，RX850 Pro では，一定周期で発生するタイマ割り込みを利用して時間管理を行っています。そこで，RX850 Pro が時間管理に利用するタイマ割り込みを発生するハードウエア (リアルタイム・パルス・ユニット，または，タイマ・コントローラ) に対しては，CF 定義ファイル作成時にシステム情報で定義した基本クロック周期でタイマ割り込みが発生するような設定を行う必要があります。

- 初期化ハンドラ

  ユーザの実行環境 / アプリケーション・システムに依存した初期化処理を行うために用意された処理ルーチンであり，ニュークリアス初期化部から呼び出されます。
  なお，RX850 Pro では，初期化ハンドラを " タスク " として位置付けています。

注意 1　RX850 Pro 依存部を記述する際の注意事項についての詳細は，「RX850 Pro ユーザーズ・マニュアル　インストレーション編」を参照してください。

注意 2　RX-NET(TCP/IP) では，初期化ハンドラのサンプル・ソース・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\smp850e\rxnet\<*sample_name*>\src
varfunc.c　　：初期化ハンドラ

【CCV850E 対応版の場合】
nectools32\smp850e_ghs\rxnet\<*sample_name*>\src
varfunc.c　　：初期化ハンドラ

3.5　RX-NET(TCP/IP) 依存部の記述

RX-NET(TCP/IP) では，RX-NET(TCP/IP) が提供する機能を実現する際に必要となる基本情報，および，ドライバ関数を RX-NET(TCP/IP) 依存部 (ユーザ・オウン・コーディング部) として切り出しています。
したがって，RX-NET(TCP/IP) が提供する機能を利用した処理プログラムを作成する場合，RX-NET(TCP/IP) を正常に動作させるうえで必要となる RX-NET(TCP/IP) 依存部 (ユーザ・オウン・コーディング部) の記述が必要となります。
以下に，RX-NET(TCP/IP) 依存部の一覧を示します。

- 基本情報

  RX-NET(TCP/IP) では，RX-NET(TCP/IP) が提供する機能を実現する際に必要となる基本情報 (ヒープ情報，ソケット情報など) をユーザ・オウン・コーディング部として切り出しています。
  以下に，RX-NET(TCP/IP) のユーザ・オウン・コーディング部として切り出される基本情報の一覧を示します。

  - ヒープ情報
  - ソケット情報
  - イベント・テーブル情報
  - 周期起動ハンドラ情報
  - タスク情報
  - デバイス・ドライバ・エントリ・テーブル情報
  - 生存時間情報

- ドライバ関数

  RX-NET(TCP/IP) では，RX-NET(TCP/IP) が提供する機能を実現する際に必要となるドライバ関数をユーザ・オウン・コーディング部として切り出しています。
  表 3-1 に，ユーザ・オウン・コーディング部として切り出されるドライバ関数の一覧を示します。

表 3-1　ドライバ関数

ドライバ関数名	機能概要
devname_init	so_initialize を発行した際に呼び出される初期化関数
devname_updown	ll_config，または，ll_unconfig を発行した際に呼び出される開始 / 終了関数
devname_start	RX-NET(TCP/IP) が送信処理を実行した際に呼び出される送信関数
devname_Thread	ネットワーク割り込みハンドラから呼び出されるネットワーク・タスク
devname_intr	受信割り込み，送信完了割り込み，エラー割り込みが発生した際に呼び出されるネットワーク割り込みハンドラ

注意 1　RX-NET(TCP/IP) 依存部についての詳細は，「第 6 章　RX-NET(TCP/IP) 依存部」を参照してください。

注意 2　RX-NET(TCP/IP) では，RX-NET(TCP/IP) のサンプル・ソース・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c
rxnetconf.c　　：基本情報

nectools32\src\rxnet\drivers\<*driver_name*>\
*devname*api.c　　：ドライバ関数
*devname*isrsub.c　：ドライバ関数

【CCV850E 対応版の場合】
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c
rxnetconf.c　　：基本情報

nectools32\src\rxnet\drivers\<*driver_name*>\
*devname*api.c　　：ドライバ関数
*devname*isrsub.c　：ドライバ関数

3.6　処理プログラムの記述

ネットワーク・アプリケーションとして実現すべき処理 (処理プログラム) を記述します。
なお，処理プログラムは，用途別に以下のように分類 / 区別されています。

- タスク
  RX850 Pro の管理下で実行可能な処理プログラムの最小単位です。

- 直接起動割り込みハンドラ
  割り込みが発生した際，RX850 Pro を介在させることなく起動される割り込み処理専用ルーチンです。
  なお，RX850 Pro では，直接起動割り込みハンドラを " タスク " とは独立したもの (非タスク) として位置付けています。このため，割り込みが発生した際には，システム内で最高優先度を持つタスクが実行中であっても，その処理は中断され，直接起動割り込みハンドラに制御が移ります。

- 間接起動割り込みハンドラ
  割り込みが発生した際に RX850 Pro による割り込み前処理 (レジスタの退避，スタックの切り替えなど) を行わせたのちに起動される割り込み処理専用ルーチンです。
  なお，RX850 Pro では，間接起動割り込みハンドラを " タスク " とは独立したもの (非タスク) として位置付けています。このため，割り込みが発生した際には，システム内で最高優先度を持つタスクが実行中であっても，その処理は中断され，間接起動割り込みハンドラに制御が移ります。

- 周期起動ハンドラ
  一定の時間が経過した際に起動される周期処理専用ルーチンです。
  なお，RX850 Pro では，周期起動ハンドラを " タスク " とは独立したもの (非タスク) として位置付けています。このため，一定の時間が経過した際には，システム内で最高優先度を持つタスクが実行中であっても，その処理は中断され，周期起動ハンドラに制御が移ります。

- 拡張 SVC ハンドラ
  ユーザが記述した関数を拡張システム・コールとして RX850 Pro に登録した処理ルーチンです。
  なお，RX850 Pro では，拡張 SVC ハンドラを " 拡張 SVC ハンドラを呼び出した処理プログラム (タスク，非タスク) の延長線 " として位置付けています。

- 拡張 SVC ハンドラ用インタフェース・ルーチン
  処理プログラム (タスク，非タスク) から 4 個以上の引き継ぎデータを持った拡張 SVC ハンドラを呼び出す際に必要となるインタフェース・ルーチンです。

注意 1　処理プログラムを記述する際の注意事項についての詳細は，「RX850 Pro ユーザーズ・マニュアル　基礎編」を参照してください。

注意 2　RX-NET(TCP/IP) では，処理プログラムのサンプル・ソース・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\smp850e\rxnet\<*sample_name*>\src

task.c　　：タスク

【CCV850E 対応版の場合】
nectools32\smp850e_ghs\rxnet\<*sample_name*>\src
task.c　　：タスク

3.7　オブジェクト・ファイルの生成

「3.2　CF 定義ファイルの記述」～「3.6　処理プログラムの記述」で作成された C 言語ソース・ファイル / アセンブリ言語ソース・ファイルに対して C コンパイラ / アセンブラを実行し，リロケータブルなオブジェクト・ファイルを生成します。

注意 1　C コンパイラ / アセンブラの起動オプション，および，実行方法についての詳細は，使用する C コンパイラ・パッケージのユーザーズ・マニュアルを参照してください。

注意 2　RX-NET(TCP/IP) では，オブジェクト・ファイルを生成するためのサンプル・コマンド・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\smp850e\rxnet\<*sample_name*>\obj
Makefile　　：ロード・モジュール用メイク・ファイル

【CCV850E 対応版の場合】
nectools32\smp850e_ghs\rxnet\<*sample_name*>\obj_ghs
sample.bld　　：ロード・モジュール用ビルド・ファイル

3.8　アーカイブ・オブジェクトの生成

「3.7　オブジェクト・ファイルの生成」において作成したリロケータブルなオブジェクト・ファイルのうち，1 個のオブジェクトとしてまとめることが可能なものについてはリンク・エディタ / アーカイバを実行し，アーカイブ・オブジェクトを生成します。

注意 1　リンク・エディタ / アーカイバの起動オプション，および，実行方法についての詳細は，使用する C コンパイラ・パッケージのユーザーズ・マニュアルを参照してください。

注意 2　RX-NET(TCP/IP) では，アーカイブ・オブジェクトを生成するためのサンプル・コマンド・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\smp850e\rxnet\<*sample_name*>\obj
Makefile　　：ロード・モジュール用メイク・ファイル

【CCV850E 対応版の場合】
nectools32\smp850e_ghs\rxnet\<*sample_name*>\obj_ghs
sample.bld　　：ロード・モジュール用ビルド・ファイル

3.9　リンク・ディレクティブ・ファイルの記述

リンク・エディタが行うアドレス割り付けをユーザが固定化するためのファイル (リンク・ディレクティブ・ファイル) を記述します。

注意 1　リンク・ディレクティブ・ファイルを記述する際の注意事項についての詳細は，使用する C コンパイラ・パッケージのユーザーズ・マニュアルを参照してください。

注意 2　RX-NET(TCP/IP) では，リンク・ディレクティブ・ファイルのサンプル・ソース・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\smp850e\rxnet\<*sample_name*>\src
sample.dir　　：リンク・ディレクティブ・ファイル

【CCV850E 対応版の場合】
nectools32\smp850e_ghs\rxnet\<*sample_name*>\obj_ghs
sample.lx　　：リンク・ディレクティブ・ファイル

3.10　ロード・モジュールの生成

「3.7　オブジェクト・ファイルの生成」～「3.8　アーカイブ・オブジェクトの生成」で作成したオブジェクト・ファイル，アーカイブ・オブジェクト，「3.9　リンク・ディレクティブ・ファイルの記述」で作成したリンク・ディレクティブ・ファイルの他に，C コンパイラ・パッケージ，RX850 Pro，RX-NET(TCP/IP) などが提供しているライブラリ・ファイルに対してリンク・エディタを実行し，ロード・モジュールを生成します。

注意 1　**リンク・エディタの起動オプション，および，実行方法についての詳細は，使用する** C **コンパイラ・パッケージのユーザーズ・マニュアルを参照してください。**

注意 2　RX-NET(TCP/IP) **では，ロード・モジュールを生成するためのサンプル・コマンド・ファイルを提供しています。**

【CA850 **対応版の場合】**
nectools32\smp850e\rxnet\<*sample_name*>\obj
Makefile　　**：ロード・モジュール用メイク・ファイル**

【CCV850E **対応版の場合】**
nectools32\smp850e_ghs\rxnet\<*sample_name*>\obj_ghs
sample.bld　　**：ロード・モジュール用ビルド・ファイル**

第 4 章　ネットワーク通信機能

本章では，RX-NET(TCP/IP) が提供しているネットワーク通信機能について解説しています。

4.1　概要

ソケットは BSD UNIX で考え出された抽象的な概念であり，ユーザが記述した処理プログラムからネットワーク・プロトコルに対する動的な操作を可能とするためのものです。

したがって，ソケットが生成されると複数の処理プログラム間におけるネットワーク通信 (メッセージの送受信：一方の通信用エンド・ポイントから送信，他方の通信用エンド・ポイントで受信) が可能となります。

なお，RX-NET(TCP/IP) では，以下に示したタイプのソケットを生成，操作，解放するための API 関数を提供しています。

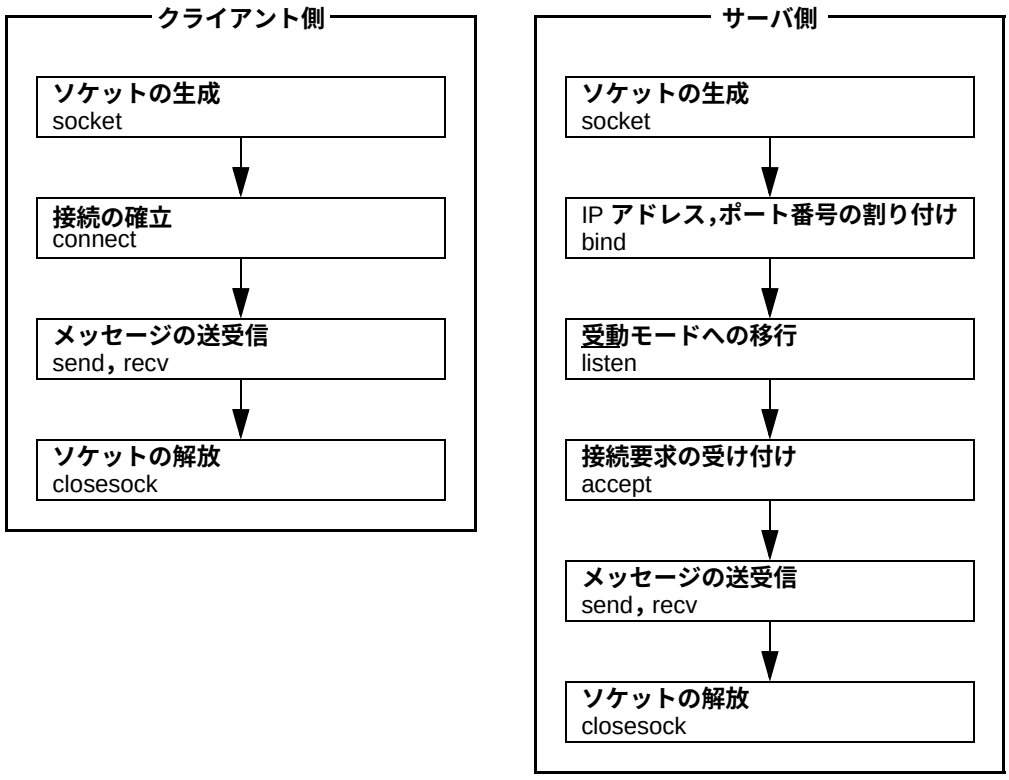
- バーチャル・サーキット型 (TCP)
- データグラム型 (UDP)

4.2　処理の流れ

RX-NET(TCP/IP) におけるネットワーク通信は，“ クライアント・サーバ・モデル ” に準じた方法で行われます。

図 4-1 に，RX-NET(TCP/IP) が提供している API 関数を利用したネットワーク通信 (メッセージの送受信) の実現例を示します。

図 4-1　ネットワーク通信の実現例



4.3　RX-NET API 関数

4.3.1　RX-NET(TCP/IP) の初期化

RX-NET(TCP/IP)の初期化は,以下に示したAPI関数を処理プログラム(タスク)から発行することにより実現されます。

- so_initialize

  RX-NET(TCP/IP) が提供する機能を実現するうえで必要となる各種初期化処理を実行します。

  なお，RX-NET(TCP/IP) では，RX-NET(TCP/IP) の初期化処理として，RX-NET(TCP/IP) が提供している機能を実現する際に必要となる各種資源の生成，および，デバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されている全てのドライバ関数 devname_init の呼び出しを行っています。

  以下に，本 API 関数の記述例を示します。

  なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        so_initialize ( ) ;                     /* RX-NET(TCP/IP) の初期化 */
        ext_tsk ( ) ;                           /* タスクの終了処理 */
}
```

注意 1　デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「6.2.6　デバイス・ドライバ・エントリ・テーブル情報」を参照していください。

注意 2　ドライバ関数 devname_init についての詳細は，「6.4.1　外部インタフェース仕様 devname_init」を参照してください。

4.3.2　ネットワーク・インタフェースの起動 / 遮断

ネットワーク・インタフェースの起動 / 遮断は，以下に示した API 関数を処理プログラム (タスク) から発行することにより実現されます。

- ll_config

  パラメータ *ipaddress*，*ipmask*，*mtu* で指定された情報をもとに，パラメータ *devname* で指定されたネットワーク・インタフェースを初期化したのち，起動します。

  なお，RX-NET(TCP/IP) では，ネットワーク・インタフェースの初期化処理として，RX-NET(TCP/IP) が提供している機能を実現する際に必要となる各種資源の生成，および，パラメータ *devname* で指定されたデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているドライバ関数 devname_updown の呼び出しを行っています。

  以下に，本 API 関数の記述例を示します。

  なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        char            devname [ ] = "s91s" ;                  /* 変数の宣言，初期化 */
        u32             ipaddress = inet_addr ( “192.168.0.1” ) ;       /* 変数の宣言，初期化 */
        u32             ipmask = inet_addr ( "255.255.255.0" ) ;        /* 変数の宣言，初期化 */
        unsigned        mtu = 1500 ;                            /* 変数の宣言，初期化 */

        so_initialize ( ) ;                             /* RX-NET(TCP/IP) の初期化 */
```

```
                                                    /* ネットワーク・インタフェースの起動 */
        ll_config ( devname , ipaddress , ipmask , mtu , 0x0 ) ;
        ext_tsk ( ) ;                               /* タスクの終了処理 */
}
```

注意 1　パラメータ *devname* に指定可能なネットワーク・デバイス名は，あらかじめデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているネットワーク・デバイス名に限られます。
なお，デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「**6.2.6　デバイス・ドライバ・エントリ・テーブル情報**」を参照してください。

注意 2　パラメータ *mtu* に 0x0，または，1500 以上の値を指定した場合，RX-NET(TCP/IP) は最大転送単位 (MTU：Maximum Tran sfer Unit) のディフォールト値 1500 が指定されていたものとして処理を行います。

注意 3　ドライバ関数 devname_updown についての詳細は，「**6.4.1　外部インタフェース仕様 devname_updown**」を参照してください。

- ll_unconfig

パラメータ *devname*，*ipaddress*，*ipmask* で指定されたネットワーク・インタフェースを遮断します。
なお，RX-NET(TCP/IP) では，ネットワーク・インタフェースの遮断処理として，RX-NET(TCP/IP) が提供している機能を実現する際に必要となる各種資源の削除，および，パラメータ *devname* で指定されたデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているドライバ関数 devname_updown の呼び出しを行っています。
以下に，本 API 関数の記述例を示します。
なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        char            devname [ ] = "s91s" ;                          /* 変数の宣言，初期化 */
        u32             ipaddress = inet_addr ( “192.168.0.1” ) ;       /* 変数の宣言，初期化 */
        u32             ipmask = inet_addr ( "255.255.255.0" ) ;        /* 変数の宣言，初期化 */
        unsigned        mtu = 1500 ;                                    /* 変数の宣言，初期化 */

        so_initialize ( ) ;                             /* RX-NET(TCP/IP) の初期化 */
                                                        /* ネットワーク・インタフェースの起動 */
        ll_config ( devname , ipaddress , ipmask , mtu , 0x0 ) ;

        ..........
        ..........
        ..........


                                                        /* ネットワーク・インタフェースの遮断 */
        ll_unconfig ( devname , ipaddress , ipmask ) ;
        ext_tsk ( ) ;                                   /* タスクの終了処理 */
}
```

注意 1　パラメータ *devname* に指定可能なネットワーク・デバイス名は，あらかじめデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているネットワーク・デバイス名に限られます。
なお，デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「**6.2.6　デバイス・ドライバ・エントリ・テーブル情報**」を参照してください。

注意 2　ドライバ関数 devname_updown についての詳細は，「**6.4.1　外部インタフェース仕様 devname_updown**」を参照してください。

4.3.3　IP アドレスの登録 / 登録解除

IPアドレスの登録/登録解除は,以下に示したAPI関数を処理プログラム(タスク)から発行することにより実現されます。

- ll_route

　パラメータ *devname* で指定されたネットワーク・インタフェースのルーティング・テーブルにパラメータ *gateway* で指定されたルータの IP アドレスを登録します。
　以下に，本 API 関数の記述例を示します。
　なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include        <stdrx85p.h>                /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>                   /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>               /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        char            devname [ ] = "s91s" ;                  /* 変数の宣言，初期化 */
        u32             ipaddress = inet_addr ( “192.168.0.1” ) ;   /* 変数の宣言，初期化 */
        u32             ipmask = inet_addr ( "255.255.255.0" ) ;    /* 変数の宣言，初期化 */
        unsigned        mtu = 1500 ;                            /* 変数の宣言，初期化 */
        u32             gateway = inet_addr ( "10.30.178.254" ) ;   /* 変数の宣言，初期化 */

        so_initialize ( ) ;                                     /* RX-NET(TCP/IP) の初期化 */
                                                                /* ネットワーク・インタフェースの起動 */
        ll_config ( devname , ipaddress , ipmask , mtu , 0x0 ) ;
        ll_route ( devname , gateway , 0x0 , 0x0 , 0x0 ) ;      /* IP アドレスの登録 */
        ext_tsk ( ) ;                                           /* タスクの終了処理 */
}
```

　注意　パラメータ *devname* に指定可能なネットワーク・デバイス名は，あらかじめデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているネットワーク・デバイス名に限られます。
　なお，デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「**6.2.6　デバイス・ドライバ・エントリ・テーブル情報**」を参照してください。

- ll_del_static_route

　パラメータ *devname* で指定されたネットワーク・インタフェースのルーティング・テーブルからパラメータ *gateway* で指定されたルータの IP アドレスを削除 (登録解除) します。
　以下に，本 API 関数の記述例を示します。
　なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include        <stdrx85p.h>                /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>                   /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>               /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        char            devname [ ] = "s91s" ;                  /* 変数の宣言，初期化 */
        u32             ipaddress = inet_addr ( “192.168.0.1” ) ;   /* 変数の宣言，初期化 */
        u32             ipmask = inet_addr ( "255.255.255.0" ) ;    /* 変数の宣言，初期化 */
        u32             gateway = inet_addr ( "10.30.178.254" ) ;   /* 変数の宣言，初期化 */

        so_initialize ( ) ;                                     /* RX-NET(TCP/IP) の初期化 */
                                                                /* ネットワーク・インタフェースの起動 */
        ll_config ( devname , ipaddress , ipmask , 0x0 , 0x0 ) ;
        ll_route ( devname , gateway , 0x0 , 0x0 , 0x0 ) ;      /* IP アドレスの登録 */

        ……
        ……
```

```
        ..........................

                                                        /* IP アドレスの登録解除 */
        ll_del_static_route ( devname , gateway , 0x0 , 0x0 ) ;
        ext_tsk ( ) ;                                   /* タスクの終了処理 */
}
```

注意　パラメータ *devname* **に指定可能なネットワーク・デバイス名は，あらかじめデバイス・ドライバ・エントリ・テーブル情報** ndevsw **に登録されているネットワーク・デバイス名に限られます。**
なお，デバイス・ドライバ・エントリ・テーブル情報 ndevsw **についての詳細は，「6.2.6　デバイス・ドライバ・エントリ・テーブル情報」を参照してください。**

4.3.4　ICMP ECHO パケットの送信

ICMP ECHO **パケットの送信は，以下に示した** API **関数を処理プログラム** (**タスク**) **から発行することにより実現されます。**

- ping

パラメータ *ipaddress* **で指定されたホスト・マシンにパラメータ** *data_len***，および，パラメータ** *npackets* **で指定された** ICMP ECHO **パケットを送信します。**
なお，本 API **関数からの復帰処理は，パラメータ** *npackets* **で指定された数の** ICMP ECHO REPLY **パケットを受信した際，または，本** API **関数の発行から** 1000 **ミリ秒が経過した際に行われます。**

4.3.5　IP アドレスの変換

IP **アドレスの変換は，以下に示した** API **関数を処理プログラム** (**タスク**) **から発行することにより実現されます。**

- inet_addr

パラメータ *ipaddress* **で指定されたドット形式の** IP **アドレスをネットワーク・バイト・オーダ形式の** IP **アドレスに変換します。**
なお，本 API **関数の処理が正常終了した際には，“ ネットワーク・バイト・オーダ形式に変換された** IP **アドレス ” が戻り値として返されます。**
以下に，本 API **関数の記述例を示します。**
なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             address ;                               /* 変数の宣言 */
        u8              ipaddress [ ] = "192.168.0.1" ;         /* 変数の宣言，初期化 */

        address = inet_addr ( ipaddress ) ;             /*IP アドレスの変換 */
        ext_tsk ( ) ;                                   /* タスクの終了処理 */
}
```

注意　ドット形式の IP **アドレスとは，“** 202.247.5.136 **” などといったドット “** . **” で区切られた数字から構成されている** IP **アドレスを意味しています。**

4.4　ソケット API 関数

4.4.1　ソケットの生成 / 解放

ソケットの生成 / 解放は，以下に示した API 関数を処理プログラム (タスク) から発行することにより実現されます。

- socket

  パラメータ *type* で指定された情報をもとに，ネットワーク通信を実現するうえで必要となるソケット (通信用エンド・ポイント) をブロッキング・モードで生成します。

  なお，本 API 関数では，ソケットの生成処理として，以下に示した値でソケット・オプションの初期化を行っています。

  - IP ネットワーク・レベル用ソケット・オプション

サービス・タイプ	：通常の信頼性，通常のスループット，通常の遅延，ルーチンを優先
断片化制御タイプ	：OFF
生存時間の最大値	：MAXTTL(単位：秒)
生存時間の最小値	：MINTTL(単位：秒)
セキュリティ・オプション	：OFF
Loose Source とルートの記録	：OFF
Strict Source とルートの記録	：OFF
ルートの記録	：OFF
ストリーム識別子	：OFF
インターネット・タイムスタンプ	：OFF

  - ソケット・インタフェース・レベル用ソケット・オプション

アドレスの再利用	：OFF
キープ・アライブ・モード	：OFF
Don't Route オプション	：OFF
Linger オプション	：OFF
Throughput オプション	：OFF
Expedite オプション	：OFF

  また，本 API 関数の処理が正常終了した際には，“ 生成したソケットのソケット記述子 ”が戻り値として返されます。
  以下に，本 API 関数の記述例を示します。
  なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include      <stdrx85p.h>              /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include      <rxnet.h>                 /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include      <fnsconfig.h>             /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int         sd , errp ;                         /* 変数の宣言 */
        int         type = SOCK_STREAM ;                /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;  /* ソケットの生成 */
        ext_tsk ( ) ;                                   /* タスクの終了処理 */
}
```

注意 1　パラメータ *errp* で指定された領域には，以下に示したエラー・コードが格納されます。

ENOERR	(0x0)	：正常終了
EMFILE	(0x17)	：最大ソケット生成数を超えています
EPROTONOSUPPORT	(0x58)	：システム予約領域 *protocol* の指定が不正です
ESOCKTNOSUPPORT	(0x59)	：ソケット・タイプ *type* の指定が不正です
EAFNOSUPPORT	(0x5b)	：アドレス・ファミリ *af* の指定が不正です
ENETDOWN	(0x5e)	：ネットワークがダウンしています

注意 2　生存時間の最大値 MAXTTL，および，最小値 MINTTL についての詳細は，「**6.2.7　生存時間情報**」を参照してください。

注意 3　ソケット・オプションの設定処理についての詳細は「5.5.2　ソケット API 関数 setsopt」を，IP アドレス，ポート番号の割り付け処理についての詳細は「5.5.2　ソケット API 関数 bind」を参照してください。

- closesock

パラメータ *sd* で指定されたソケットで確立している接続を遮断 (送受信両方向の接続を遮断) したのち，該当ソケットの解放処理を実行します。
なお，RX-NET(TCP/IP) では，ソケットの解放処理として，ソケット用各種リソースの解放を行っています。
以下に，本 API 関数の記述例を示します。
なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;      /* ソケットの生成 */

        .....
        .....
        .....

        closesock ( sd ) ;                                      /* ソケットの解放 */
        ext_tsk ( ) ;                                           /* タスクの終了処理 */
}
```

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　本 API 関数を発行した際，対象ソケットの Linger オプションが ON であり，かつ，Linger タイムアウトが 0x0 以外の値であった場合には，接続の遮断処理，および，該当ソケットの解放処理は Linger タイムアウトで指定された時間が経過するまで遅延されます。

注意 3　本 API 関数では，正常終了時の後処理として，RX-NET(TCP/IP) に対する CLOSE_NOTIFY イベントの発生通知を行います。

4.4.2　ソケット・オプションの設定 / 獲得

ソケット・オプションの設定 / 獲得は，以下に示した API 関数を処理プログラム (タスク) から発行することにより実現されます。

- setsopt

パラメータ *level*，*optname*，および，*optval* で指定された情報をもとに，パラメータ *sd* で指定されたソケットのソケット・オプションを変更します。
以下に，パラメータ *optname* で指定されたソケット・オプションの種類別 (IP_O_TOS，IP_O_FRAF，IP_O_MAXTTL など) に，パラメータ *optval* に設定する値の意味を示します。

IP_O_TOS　　**…サービス・タイプ**

【 信頼性 (ビット 2) 】
IP_RELIABLE　　**： 高い信頼性**

【 スループット (ビット 3) 】
IP_THROUGHPUT　**： 最効率のスループット**

【 遅延 (ビット 4) 】
IP_DELAY　　**： 最短の遅延**

【 優先順位 (ビット 5 ～ 7) 】
IP_ROUTINE　　**： ルーチン**
IP_PRIORITY　　**： 優先度**

IP_IMMED　：即時
IP_FLSHO　：フラッシュを無視
IP_FLASH　：フラッシュ
IP_CRITIC　：CRITIC/ECP
IP_IC　：インターネットワーク制御
IP_NC　：ネットワーク制御

IP_O_FRAG　…断片化制御タイプ
0x0　：OFF
0x1　：ON

IP_O_MAXTTL　…生存時間の最大値
0x0 ～ 0xff　：生存時間 (単位：ミリ秒)

IP_O_MINTTL　…生存時間の最小値
0x0 ～ 0xff　：生存時間 (単位：ミリ秒)

IP_O_SECURE　…セキュリティ・オプション
0x0　：OFF
0x1　：ON

IP_O_LSRR　…Loose Source とルートの記録
0x0　：OFF
0x1　：ON

IP_O_SSRR　…Strict Source とルートの記録
0x0　：OFF
0x1　：ON

IP_O_RR　…ルートの記録
0x0　：OFF
0x1　：ON

IP_O_STREAM　…ストリーム識別子
0x0　：OFF
0x1　：ON

IP_O_TIME　…インターネット・タイムスタンプ
0x0　：OFF
0x1　：ON

SO_LINGER　…Linger オプション
0x0　：OFF
0x1　：ON

以下に，本 API 関数の記述例を示します。
なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include      <stdrx85p.h>          /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include      <rxnet.h>             /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include      <fnsconfig.h>         /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp , optval ,  opt_len ;          /* 変数の宣言 */
        int             type = SOCK_STREAM ;                     /* 変数の宣言，初期化 */
        int             level = SOL_SOCKET ;                     /* 変数の宣言，初期化 */
        int             optname = ~SO_REUSEADDR ;                /* 変数の宣言，初期化 */
        lingeropt       optval ;                                 /* データ構造体の宣言 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;      /* ソケットの生成 */
        setsopt ( sd , level , optname , NULL , 0x0 ) ;     /* ソケット・オプションの設定 */

        optval.linger_on = 0x1 ;                                 /* データ構造体の初期化 */
```

```
        optval.linger_timeout = 120                          /* データ構造体の初期化 */

                                                             /* ソケット・オプションの設定 */
        setsopt ( sd , level , optname , ( char * ) &optval , sizeof ( lingeropt ) ) ;
        ext_tsk ( ) ;                                        /* タスクの終了処理 */
}
```

注意 1　**ソケット記述子** *sd* **には，**API **関数** socket **の戻り値を設定します。**

注意 2　**ソケット・オプション** *optname* **に** IP_O_SECURE **を指定した場合，**_optval_ **には** IP **ソケット・セキュリティ・オプション情報** ipos_t **へのポインタを設定します。**
なお，IP **ソケット・セキュリティ・オプション情報** ipos_t **についての詳細は，「5.4.4　IP ソケット・セキュリティ・オプション情報」を参照してください。**

注意 3　**ソケット・オプション** *optname* **に** IP_O_TIME **を指定した場合，**_optval_ **には** IP **ソケット・タイムスタンプ・オプション情報** ipot_t **へのポインタを設定します。**
なお，IP **ソケット・タイムスタンプ・オプション情報** ipot_t **についての詳細は，「5.4.5　IP ソケット・タイムスタンプ・オプション情報」を参照してください。**

注意 4　**ソケット・オプション** *optname* **に** SO_LINGER **を指定した場合，**_optval_ **には** Linger **オプション情報** lingeropt **へのポインタを設定します。**
なお，Linger **オプション情報** lingeropt **についての詳細は，「5.4.6　Linger オプション情報」を参照してください。**

注意 5　**ソケット・オプション** *optname* **に以下に示した何れかの値を指定した場合，**_optval_ **には** NULL **を，**_opt_len_ **には** 0x0 **を設定します。**

SO_REUSEADDR	SO_KEEPALIVE	SO_DONTROUTE
SO_LINGER	SO_THROUGHPUT	SO_EXPEDITE
~SO_REUSEADDR	~SO_KEEPALIVE	~SO_DONTROUTE
~SO_LINGER	~SO_THROUGHPUT	~SO_EXPEDITE

- getsopt

パラメータ *sd* **で指定されたソケットに設定されているソケット・オプションの値をパラメータ** *optval* **で指定された領域に格納します。**
なお，獲得するソケット・オプションの種類については，パラメータ *level*，**および，パラメータ** *optname* **の組み合わせで指定します。**
以下に，本 API **関数の記述例を示します。**
なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp , optval ,  opt_len ;         /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        int             level = SOL_SOCKET ;                    /* 変数の宣言，初期化 */
        int             optname = SO_REUSEADDR ;                /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;          /* ソケットの生成 */
                                                                /* ソケット・オプションの獲得 */
        getsopt ( sd , level , optname , ( char * ) &optval , &opt_len ) ;

        if ( optname & optval ) {
                /* アドレスの再利用が可能な場合の処理 */
                ……
                ……
                ……
        } else {
                /* アドレスの再利用が不可能な場合の処理 */
```

```
                .....
                .....
                .....
        }

        ext_tsk ( ) ;                                   /* タスクの終了処理 */
}
```

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　獲得したソケット・オプションの値 *optval* についての詳細は，「5.5.2　ソケット API 関数 socket」を参照してください。

4.4.3　IP アドレス，ポート番号の割り付け / 獲得

IP アドレス，ポート番号の割り付け / 獲得は，以下に示した API 関数を処理プログラム (タスク) から発行することにより実現されます。

- bind

  パラメータ *sd* で指定されたソケットにパラメータ *name* で指定されたアドレス構造体 sockaddr_in **に格納されている** IP **アドレス，ポート番号を割り付けます。**
  以下に，本 API **関数の記述例を示します。**
  なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        struct          sockaddr_in     name ;                  /* データ構造体の宣言 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;          /* ソケットの生成 */

        name.sin_family = AF_INET ;                             /* データ構造体の初期化 */
        name.sin_addr.s_addr = htonl ( 0x0 ) ;                  /* データ構造体の初期化 */
        name.sin_port = htons ( 2048 ) ;                        /* データ構造体の初期化 */

                                                                /* IP アドレス，ポート番号の割り付け */
        bind ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        ext_tsk ( ) ;                                           /* タスクの終了処理 */
}
```

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　アドレス構造体 sockaddr_in **についての詳細は，「5.4.3　インターネット・アドレス用アドレス構造体」を参照してください。**

- getsockname

パラメータ *sd* **で指定されたソケットの** IP **アドレス,ポート番号をパラメータ** *name* **で指定されたアドレス構造体** sockaddr_in **に格納します。**

以下に，本 API **関数の記述例を示します。**

なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>                /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>                   /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>               /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        struct          sockaddr_in     name ;                  /* データ構造体の宣言 */
        int             backlog = 0x5 ;                         /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;          /* ソケットの生成 */

        name.sin_family = AF_INET ;                             /* データ構造体の初期化 */
        name.sin_addr.s_addr = htonl ( 0x0 ) ;                  /* データ構造体の初期化 */
        name.sin_port = htons ( 2048 ) ;                        /* データ構造体の初期化 */
                                                                /* IP アドレス，ポート番号の割り付け */
        bind ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        listen ( sd , backlog ) ;                               /* 受動モードへの移行 */
                                                                /* 接続要求の受け付け */
        accept ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) , &errp ) ;
                                                                /* IP アドレス，ポート番号の獲得 */
        getsockname ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        ext_tsk ( ) ;                                           /* タスクの終了処理 */
}
```

注意 1　**ソケット記述子** *sd* **には，**API **関数** socket **の戻り値を設定します。**

注意 2　**アドレス構造体** sockaddr_in **についての詳細は，「5.4.3　インターネット・アドレス用アドレス構造体」を参照してください。**

- getpeername

パラメータ *sd* **で指定されたソケットと接続が確立している遠隔側ソケット** (API **関数** connect**，または，**accept **の発行により接続が確立した遠隔側ソケット**) **の** IP **アドレス，ポート番号をパラメータ** *name* **で指定されたアドレス構造体** sockaddr_in **に格納します。**

以下に，本 API **関数の記述例を示します。**

なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>                /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>                   /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>               /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        struct          sockaddr_in     name ;                  /* データ構造体の宣言 */
        int             backlog = 0x5 ;                         /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;          /* ソケットの生成 */

        name.sin_family = AF_INET ;                             /* データ構造体の初期化 */
        name.sin_addr.s_addr = htonl ( 0x0 ) ;                  /* データ構造体の初期化 */
```

```
        name.sin_port = htons ( 2048 ) ;                        /* データ構造体の初期化 */

                                                                /* IP アドレス，ポート番号の割り付け */
        bind ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        listen ( sd , backlog ) ;                               /* 受動モードへの移行 */
                                                                /* 接続要求の受け付け */
        accept ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) , &errp ) ;
                                                                /* 遠隔側 IP アドレス，ポート番号の獲得 */
        getpeername ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        ext_tsk ( ) ;                                           /* タスクの終了処理 */
}
```

注意 1　**ソケット記述子 *sd* には，**API **関数** socket **の戻り値を設定します。**

注意 2　**アドレス構造体** sockaddr_in **についての詳細は，「5.4.3　インターネット・アドレス用アドレス構造体」を参照してください。**

4.4.4　モードの移行

モードの移行は，以下に示した API **関数を処理プログラム** (**タスク**) **から発行することにより実現されます。**

- listen

パラメータ *sd* で指定されたソケットにパラメータ *backlog* で指定された接続要求の最大保留数を設定したのち，受動モードへと移行させます。
以下に，本 API **関数の記述例を示します。**
なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        struct          sockaddr_in     name ;                  /* データ構造体の宣言 */
        int             backlog = 0x5 ;                         /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;          /* ソケットの生成 */

        name.sin_family = AF_INET ;                             /* データ構造体の初期化 */
        name.sin_addr.s_addr = htonl ( 0x0 ) ;                  /* データ構造体の初期化 */
        name.sin_port = htons ( 2048 ) ;                        /* データ構造体の初期化 */

                                                                /* IP アドレス，ポート番号の割り付け */
        bind ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        listen ( sd , backlog ) ;                               /* 受動モードへの移行 */
        ext_tsk ( ) ;                                           /* タスクの終了処理 */
}
```

注意 1　**ソケット記述子 *sd* には，**API **関数** socket **の戻り値を設定します。**

注意 2　**パラメータ *backlog* に指定可能な値は，**0x1 **～** 0x5 **に限られます。なお，パラメータ *backlog* に** 0x5 **よりも大きな値を指定した場合，**RX-NET(TCP/IP) **は** 0x5 **が指定されていたものとして処理を行います。**

注意 3　API **関数** socket **の発行により生成されたソケットの状態は，能動モード** (**クライアントが利用可能な状態**)**，受動モード** (**サーバが利用可能な状態**) **のいずれの状態でもありません。**

- blocking

  パラメータ *sd* **で指定されたソケットをブロッキング・モードへと移行させます。**
  これにより，API **関数** (connect，accept，send，sendto，recv，recvfrom **など**) **は，特定の条件が成立するまでの間，要求動作の実行が中断 (ブロック) されます。**
  以下に，本 API **関数の記述例を示します。**
  なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;  /* ソケットの生成 */
        nonblocking ( sd ) ;                            /* ノンブロッキング・モードへの移行 */

        .....
        .....
        .....

        blocking ( sd ) ;                               /* ブロッキング・モードへの移行 */
        ext_tsk ( ) ;                                   /* タスクの終了処理 */
}
```

  注意 1　**ソケット記述子** *sd* **には，**API **関数** socket **の戻り値を設定します。**

  注意 2　**本** API **関数では，ブロッキング・モードへの移行要求がキューイングされません。このため，既に対象ソケットがブロッキング・モードへの移行していた場合には，何も処理は行わず，エラーとしても扱いません。**

  注意 3　API **関数** socket **では，該当ソケットを " ブロッキング・モード " で生成しています。**

- nonblocking

  パラメータ *sd* **で指定されたソケットをノンブロッキング・モードへと移行させます。**
  これにより，API **関数** (connect，accept，send，sendto，recv，recvfrom **など**) **は，即時に特定の条件が成立しなかった場合，異常終了 (戻り値，または，エラー・コードとして** EWOULDBLOCK **を返却**) **となります。**
  以下に，本 API **関数の記述例を示します。**
  なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;  /* ソケットの生成 */
        nonblocking ( sd ) ;                            /* ノンブロッキング・モードへの移行 */
        ext_tsk ( ) ;                                   /* タスクの終了処理 */
}
```

  注意 1　**ソケット記述子** *sd* **には，**API **関数** socket **の戻り値を設定します。**

  注意 2　**本** API **関数では，ノンブロッキング・モードへの移行要求がキューイングされません。このため，既に対象ソケットがノンブロッキング・モードへの移行していた場合には，何も処理は行わず，エラーとしても扱いません。**

注意 3　API **関数** socket **では，該当ソケットを “ ブロッキング・モード ” で生成しています。**

4.4.5　接続の確立 / 遮断

接続の確立 / 遮断は，以下に示した API **関数を処理プログラム** (**タスク**) **から発行することにより実現されます。**

- connect

パラメータ *sd* **で指定されたソケットとパラメータ** *name* **で指定された遠隔側ソケットとの接続を確立します。**
ただし，本 API **関数を発行した際，遠隔側ソケットにおいて “** accept **の発行** (**接続要求の受付**) **” が行われていなかった場合には，遠隔側ソケットの待機ソケット・キューに接続要求がキューイングされます。**
以下に，本 API **関数の記述例を示します。**
なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>                    /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>                       /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>                   /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        struct          sockaddr_in     name ;                  /* データ構造体の宣言 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;          /* ソケットの生成 */

        name.sin_family = AF_INET ;                             /* データ構造体の初期化 */
        name.sin_addr.s_addr = htonl ( REMOTE_IP_ADDRESS ) ;    /* データ構造体の初期化 */
        name.sin_port = htons ( 2048 ) ;                        /* データ構造体の初期化 */

                                                                /* 接続の確立 */
        connect ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        ext_tsk ( ) ;                                           /* タスクの終了処理 */
}
```

注意 1　**ソケット記述子** *sd* **には，**API **関数** socket **の戻り値を設定します。**

注意 2　**本** API **関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，接続の確立処理を実行するうえで必要な条件が整っていなかった場合には，接続の確立処理は行わず，戻り値として** EWOULDBLOCK **を返しています。**

注意 3　**アドレス構造体** sockaddr_in **についての詳細は，「5.4.3　インターネット・アドレス用アドレス構造体」を参照してください。**

- shutdown

パラメータ *sd* **で指定されたソケット** (**バーチャル・サーキット型** SOCK_STREAM) **で確立している接続のうち，パラメータ** *direction* **で指定された方向の接続を遮断します。**
以下に，本 API **関数の記述例を示します。**
なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>                    /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>                       /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>                   /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        struct          sockaddr_in     name ;                  /* データ構造体の宣言 */
```

```
        int             direction = 0x2 ;                       /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;          /* ソケットの生成 */

        name.sin_family = AF_INET ;                             /* データ構造体の初期化 */
        name.sin_addr.s_addr = htonl ( REMOTE_IP_ADDRESS ) ;    /* データ構造体の初期化 */
        name.sin_port = htons ( 2048 ) ;                        /* データ構造体の初期化 */

                                                                /* 接続の確立 */
        connect ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;

        ..........
        ..........
        ..........

        shutdown ( sd , direction ) ;                           /* 接続の遮断 */
        ext_tsk ( ) ;                                           /* タスクの終了処理 */
}
```

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　本 API 関数を発行した際，対象ソケットの Linger オプションが ON であり，かつ，Linger タイムアウトが 0x0 以外の値であった場合には，接続の遮断処理は Linger タイムアウトで指定された時間が経過するまで遅延されます。

注意 3　本 API 関数では，ソケットの解放処理 (ソケット用各種リソースの解放) が行われません。このため，対象ソケットを解放する際には，API 関数 closesock の発行が必要となります。

4.4.6　接続要求の受け付け / 削除

接続要求の受け付け / 削除は，以下に示した API 関数を処理プログラム (タスク) から発行することにより実現されます。

- accept

パラメータ *sd* で指定されたソケットに対して発行されている遠隔側ソケットからの接続要求 (待機ソケット・キューの先頭にキューイングされている接続要求) を受け付けたのち，パラメータ *name* で指定されたアドレス構造体 sockaddr_in に遠隔側ソケットの IP アドレス，ポート番号を格納します。

なお，本 API 関数の処理が正常終了した際には，" 接続要求が受け付けられた遠隔側ソケットのソケット記述子 " が戻り値として返されます。

以下に，本 API 関数の記述例を示します。

なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        struct          sockaddr_in     name ;                  /* データ構造体の宣言 */
        int             backlog = 0x5 ;                         /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;          /* ソケットの生成 */

        name.sin_family = AF_INET ;                             /* データ構造体の初期化 */
        name.sin_addr.s_addr = htonl ( 0x0 ) ;                  /* データ構造体の初期化 */
        name.sin_port = htons ( 2048 ) ;                        /* データ構造体の初期化 */

                                                                /* IP アドレス，ポート番号の割り付け */
```

```
        bind ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        listen ( sd , backlog ) ;                          /* 受動モードへの移行 */
                                                           /* 接続要求の受け付け */
        accept ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) , &errp ) ;
        ext_tsk ( ) ;                                      /* タスクの終了処理 */
}
```

注意 1　**ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。**

注意 2　**本 API 関数では，パラメータ *name*，および，パラメータ *name_len* に** 0x0 (NULL **ポインタ) が指定された場合には，接続要求の受け付け処理のみを行い，遠隔側ソケットの IP アドレス，ポート番号の格納処理は行いません。**

注意 3　**パラメータ *errp* で指定された領域には，以下に示したエラー・コードが格納されます。**

ENOERR	(0x0)	**：正常終了**
EINVAL	(0x16)	**：パラメータの指定が不正です**
EMFILE	(0x17)	**：最大ソケット生成数を超えています**
EWOULDBLOCK	(0x50)	**：対象ソケットがノンブロッキング・モードです**
EOPNOTSUPP	(0x5a)	**：ソケット・タイプの指定が不正です**
ENETDOWN	(0x5e)	**：ネットワークがダウンしています**

注意 4　**本 API 関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，接続要求の受け付け処理を実行するうえで必要な条件が整っていなかった場合には，接続要求の受け付け処理は行わず，戻り値として** ERR **を，エラー・コードとして** EWOULDBLOCK **を返しています。**

注意 5　**アドレス構造体** sockaddr_in **についての詳細は，「5.4.3　インターネット・アドレス用アドレス構造体」を参照してください。**

- reject

パラメータ *sd* で指定されたソケットの待機ソケット・キューの先頭にキューイングされている接続要求を削除 (接続の確立拒否) します。

以下に，本 API **関数の記述例を示します。**

なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        struct          sockaddr_in     name ;                  /* データ構造体の宣言 */
        int             backlog = 0x5 ;                         /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;          /* ソケットの生成 */

        name.sin_family = AF_INET ;                             /* データ構造体の初期化 */
        name.sin_addr.s_addr = htonl ( 0x0 ) ;                  /* データ構造体の初期化 */
        name.sin_port = htons ( 2048 ) ;                        /* データ構造体の初期化 */

                                                                /* IP アドレス，ポート番号の割り付け */
        bind ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        listen ( sd , backlog ) ;                               /* 受動モードへの移行 */
                                                                /* 接続要求の受け付け */
        accept ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) , &errp ) ;

        ..........
        ..........
        ..........
```

```
        errp = reject ( sd ) ;                          /* 接続要求の削除 */

        if ( errp == ENOERR ) {
                /* 接続要求がキューイングされていた場合の処理 */
                ·····
                ·····
                ·····
        } else {
                /* 接続要求がキューイングされていなかった場合の処理 */
                ·····
                ·····
                ·····
        }

        ext_tsk ( ) ;                                   /* タスクの終了処理 */
}
```

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　本 API 関数では，削除要求のキューイングが行われません。このため，本 API 関数を発行した際，対象ソケットの待機ソケット・キューに接続要求がキューイングされていなかった場合には，戻り値として EINVAL を返しています。

4.4.7　メッセージの送信 / 受信

メッセージの送信 / 受信は,以下に示した API 関数を処理プログラム (タスク) から発行することにより実現されます。

- send

パラメータ *sd* で指定されたソケットと接続が確立している遠隔側ソケット (API 関数 connect，または，accept の発行により接続が確立した遠隔側ソケット) に対して *buf* で指定されたメッセージを *flags* で指定された方法で送信します。
なお，本 API 関数の処理が正常終了した際には,“ 実際に送信したメッセージのサイズ (単位：バイト) ”が戻り値として返されます。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　パラメータ *errp* で指定された領域には，以下に示したエラー・コードが格納されます。

ENOERR	(0x0)	：正常終了
EINVAL	(0x16)	：パラメータの指定が不正です
EWOULDBLOCK	(0x50)	：対象ソケットがノンブロッキング・モードです
ENETDOWN	(0x5e)	：ネットワークがダウンしています
ENOBUFS	(0x63)	：メッセージのサイズが大きすぎます
ENOTCONN	(0x65)	：接続が確立していません
ESHUTDOWN	(0x66)	：shutdown の発行により接続が遮断されているため，メッセージを送信することができません

注意 3　本 API 関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，メッセージの送信処理を実行するうえで必要な条件が整っていなかった場合には，メッセージの送信処理は行わず，戻り値として ERR を，エラー・コードとして EWOULDBLOCK を返しています。

注意 4　本 API 関数を発行した際，対象ソケットがデータグラム型 SOCK_DGRAM であった場合には，送信可能なメッセージの最大サイズは 0x5c0 バイトとなります。
このため，メッセージを格納した領域のサイズ *buf_len* に 0x5c0 以上の値を指定した場合には，メッセージの送信処理は行わず，戻り値として ERR を，エラー・コードとして ENOBUFS を返しています。

- sendto

パラメータ *sd* で指定されたソケットと接続が確立している遠隔側ソケット (パラメータ *name* で指定された遠隔側ソケット) に対してパラメータ *buf* で指定されたメッセージをパラメータ *flags* で指定された方法で送信します。
なお，本 API 関数の処理が正常終了した際には,“ 実際に送信したメッセージのサイズ (単位：バイト) ”が戻り値として返されます。

注意 1　**ソケット記述子** *sd* **には，**API **関数** socket **の戻り値を設定します。**

注意 2　**パラメータ** *errp* **で指定された領域には，以下に示したエラー・コードが格納されます。**

ENOERR	(0x0)	**：正常終了**
EINVAL	(0x16)	**：パラメータの指定が不正です**
EWOULDBLOCK	(0x50)	**：対象ソケットがノンブロッキング・モードです**
ENETDOWN	(0x5e)	**：ネットワークがダウンしています**
ENOBUFS	(0x63)	**：メッセージのサイズが大きすぎます**
ENOTCONN	(0x65)	**：接続が確立していません**
ESHUTDOWN	(0x66)	**：**shutdown **の発行により接続が遮断されているため，メッセージを送信することができません**

注意 3　**本** API **関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，メッセージの送信処理を実行するうえで必要な条件が整っていなかった場合には，メッセージの送信処理は行わず，戻り値として** ERR **を，エラー・コードとして** EWOULDBLOCK **を返しています。**

注意 4　**本** API **関数を発行した際，対象ソケットがデータグラム型** SOCK_DGRAM **であった場合には，送信可能なメッセージの最大サイズは** 0x5c0 **バイトとなります。**
このため，メッセージを格納した領域のサイズ *buf_len* **に** 0x5c0 **以上の値を指定した場合には，メッセージの送信処理は行わず，戻り値として** ERR **を，エラー・コードとして** ENOBUFS **を返しています。**

注意 5　**アドレス構造体** sockaddr_in **についての詳細は，「5.4.3　インターネット・アドレス用アドレス構造体」を参照してください。**

- recv

パラメータ *sd* **で指定されたソケットと接続が確立している遠隔側ソケット** (API **関数** connect**，または，** accept **の発行により接続が確立した遠隔側ソケット**) **からパラメータ** *flags* **で指定された方法でメッセージを受信し，パラメータ** *buf* **で指定された領域に格納します。**
なお，本 API **関数の処理が正常終了した際には，“ 実際に受信したメッセージのサイズ** (**単位：バイト**) **”が戻り値として返されます。**
以下に，本 API **関数の記述例を示します。**
なお，記述例中の ext_tsk **は，**RX850 Pro **が提供しているシステム・コールです。**

```
#include        <stdrx85p.h>                /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>                   /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>               /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        struct          sockaddr_in     name ;                  /* データ構造体の宣言 */
        int             backlog = 0x5 ;                         /* 変数の宣言，初期化 */
        char            *buf ;                                  /* 変数の宣言 */
        extern  char    u_buffer [ 10 ] ;                       /* 変数の宣言 */
        int             flags = MSG_URGENT ;                    /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;      /* ソケットの生成 */

        name.sin_family = AF_INET ;                             /* データ構造体の初期化 */
        name.sin_addr.s_addr = htonl ( 0x0 ) ;                  /* データ構造体の初期化 */
        name.sin_port = htons ( 2048 ) ;                        /* データ構造体の初期化 */

                                                            /* IP アドレス，ポート番号の割り付け */
        bind ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        listen ( sd , backlog ) ;                           /* 受動モードへの移行 */
                                                            /* 接続要求の受け付け */
        accept ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) , &errp ) ;

        buf = u_buffer ;                                        /* 変数の初期化 */

                                                            /* メッセージの受信 */
```

```
        recv ( sd , *buf++ , sizeof ( char ) , flags , &errp ) ;
        ext_tsk ( ) ;                                   /* タスクの終了処理 */
}
```

注意 1　**ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。**

注意 2　**パラメータ *errp* で指定された領域には，以下に示したエラー・コードが格納されます。**

ENOERR	(0x0)	**：正常終了**
EINVAL	(0x16)	**：パラメータの指定が不正です**
EWOULDBLOCK	(0x50)	**：対象ソケットがノンブロッキング・モードです**
ENOTCONN	(0x65)	**：接続が確立していません**
ESHUTDOWN	(0x66)	**：shutdown の発行により接続が遮断されているため，メッセージを送信することができません**

注意 3　**本 API 関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，メッセージの受信処理を実行するうえで必要な条件が整っていなかった場合には，メッセージの受信処理は行わず，戻り値として ERR を，エラー・コードとして EWOULDBLOCK を返しています。**

注意 4　**戻り値に 0x0 が設定された場合は，メッセージの受信処理が完了する以前に対象ソケットの解放処理が実行されたことを意味します。**

- recvfrom

パラメータ *sd* で指定されたソケットと接続が確立している遠隔側ソケットからパラメータ *flags* で指定された方法でメッセージを受信し，パラメータ *buf* で指定された領域に格納します。

なお，パラメータ *name* で指定されたアドレス構造体 sockaddr_in には，遠隔側ソケットの IP アドレス，ポート番号が格納されます。

また，本 API 関数の処理が正常終了した際には，“実際に受信したメッセージのサイズ (単位：バイト) ”が戻り値として返されます。

以下に，本 API 関数の記述例を示します。

なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include        <stdrx85p.h>            /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>               /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>           /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        int             sd , errp ;                             /* 変数の宣言 */
        int             type = SOCK_STREAM ;                    /* 変数の宣言，初期化 */
        struct          sockaddr_in     name ;                  /* データ構造体の宣言 */
        int             backlog = 0x5 ;                         /* 変数の宣言，初期化 */
        char            *buf ;                                  /* 変数の宣言 */
        extern  char    u_buffer [ 10 ] ;                       /* 変数の宣言 */
        int             flags = MSG_URGENT ;                    /* 変数の宣言，初期化 */

        sd = socket ( AF_INET , type , 0x0 , &errp ) ;          /* ソケットの生成 */

        name.sin_family = AF_INET ;                             /* データ構造体の初期化 */
        name.sin_addr.s_addr = htonl ( 0x0 ) ;                  /* データ構造体の初期化 */
        name.sin_port = htons ( 2048 ) ;                        /* データ構造体の初期化 */

                                                                /* IP アドレス，ポート番号の割り付け */
        bind ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) ) ;
        listen ( sd , backlog ) ;                               /* 受動モードへの移行 */
                                                                /* 接続要求の受け付け */
        accept ( sd , ( saddr * ) &name , sizeof ( struct sockaddr_in ) , &errp ) ;

        buf = u_buffer ;                                        /* 変数の初期化 */

                                                                /*メッセージ(送信元アドレス情報を含む)の受信*/
```

```
        recvfrom ( sd , *buf++ , sizeof ( char ) , flags , ( saddr * ) &name , sizeof ( struct sockaddr_in ) , &errp
) ;
        ext_tsk ( ) ;                                   /* タスクの終了処理 */
}
```

注意 1　**ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。**

注意 2　**パラメータ *errp* で指定された領域には，以下に示したエラー・コードが格納されます。**

ENOERR	(0x0)	**：正常終了**
EINVAL	(0x16)	**：パラメータの指定が不正です**
EWOULDBLOCK	(0x50)	**：対象ソケットがノンブロッキング・モードです**
ENOTCONN	(0x65)	**：接続が確立していません**
ESHUTDOWN	(0x66)	**：shutdown の発行により接続が遮断されているため，メッセージを送信することができません**

注意 3　**本 API 関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，メッセージの受信処理を実行するうえで必要な条件が整っていなかった場合には，メッセージの受信処理は行わず，戻り値として ERR を，エラー・コードとして EWOULDBLOCK を返しています。**

注意 4　**戻り値に 0x0 が設定された場合は，メッセージの受信処理が完了する以前に対象ソケットの解放処理が実行されたことを意味します。**

注意 5　**アドレス構造体 sockaddr_in についての詳細は，「5.4.3　インターネット・アドレス用アドレス構造体」を参照してください。**

4.4.8　イベントの選択

イベントの選択は，以下に示した API 関数を処理プログラム (タスク) から発行することにより実現されます。

- nselect

パラメータ *selp* で指定された選択構造体 sel のメンバ se_inflags に設定されているイベント (要求イベントの論理和) のうち，どのイベントが発生していたのかを調べ，実際に発生していたイベント (発生イベントの論理和) をメンバ se_outflags に格納します。

ただし，本 API 関数を発行した際，メンバ se_inflags に設定されているイベントが 1 つも発生していなかった場合には，要求動作の実行が中断 (ブロック) されます。

なお，要求動作の実行再開は，メンバ se_inflags に設定されているイベントが 1 つでも発生した際，または，パラメータ *waitp* で指定された待ち時間が経過した際に行われます。

以下に，プロトコル毎 (TCP プロトコル，UDP プロトコル) に要求イベントとして指定可能な値，および，発生イベントとして格納される値の一覧を示します。

マクロ	意味	TCP プロトコル	UDP プロトコル
READ_NOTIFY	メッセージの受信が可能	○	○
WRITE_NOTIFY	送信キューに空きがある	○	○
ACCEPT_NOTIFY	受動コネクションが確立	○	×
CLOSE_NOTIFY	ソケットの解放処理が正常終了	○	○
CONNECT_NOTIFY	能動コネクションが確立	○	×
EXCEPT_NOTIFY	例外が発生	○	○
RSHUTDOWN_NOTIFY	読み取り方向がピアにより遮断	○	×
TIMEOUT_NOTIFY	要求動作がタイムアウト	○	×
WSHUTDOWN_NOTIFY	書き込み方向が遮断	○	×
URGENT_NOTIFY	緊急データが利用可能	○	×
SENDQEMPTY_NOTIFY	送信キューが空	○	○

以下に，本 API 関数の記述例を示します。

なお，記述例中の ext_tsk は，RX850 Pro が提供しているシステム・コールです。

```
#include        <stdrx85p.h>                    /* RX850 Pro 用標準ヘッダ・ファイルの定義 */
#include        <rxnet.h>                       /* RX-NET(TCP/IP) 用標準ヘッダ・ファイルの定義 */
#include        <fnsconfig.h>                   /* 静的設定情報ヘッダ・ファイルの定義 */


void
func_task ( INT stacd ) {
        sel             selp ;                                          /* データ構造体の宣言 */
        int             cnt = 0x3 ;                                     /* 変数の宣言，初期化 */
        int             waitp = 0x0 ;                                   /* 変数の宣言，初期化 */
        int             errp ;                                          /* 変数の宣言 */

        selp [ 0 ].se_fd = 0x0 ;                                        /* データ構造体の初期化 */
        selp [ 0 ].se_inflags = READ_NOTIFY ;                           /* データ構造体の初期化 */
        selp [ 1 ].se_fd = 0x1 ;                                        /* データ構造体の初期化 */
        selp [ 1 ].se_inflags = READ_NOTIFY | URGENT_NOTIFY ;           /* データ構造体の初期化 */
        selp [ 2 ].se_fd = 0x2 ;                                        /* データ構造体の初期化 */
        selp [ 2 ].se_inflags = READ_NOTIFY ;                           /* データ構造体の初期化 */

                                                                        /* イベントの選択 */
        nselect ( ( sel * ) selp , cnt , ( u32 * ) waitp , usel_nilpfi , ( void * ) 0x0 , &errp ) ;
        ext_tsk ( ) ;                                                   /* タスクの終了処理 */
}
```

注意 1　パラメータ *waitp* に 0x0 (NULL ポインタ)，または，パラメータ *waitp* で指定された領域に -1 を設定した場合，RX-NET(TCP/IP) は " 永久待ち " が指定されたものとして処理を行います。

注意 2　要求動作の実行中断後，メンバ se_inflags に設定されているイベントが発生した際には，パラメータ *waitp* で指定された領域に " 待ち時間が経過するまでの残り時間 (単位：ミリ秒) " が設定されます。

注意 3　パラメータ *errp* で指定された領域には，以下に示したエラー・コードが格納されます。

ENOERR	(0x0)	： 正常終了
EINVAL	(0x16)	： パラメータの指定が不正です
ENETDOWN	(0x5e)	： ネットワークがダウンしています

注意 4　選択構造体 sel についての詳細は，「**5.4.7　選択構造体**」を参照してください。

第 5 章　API 関数

本章では，RX-NET(TCP/IP) が提供しているアプリケーション・プログラム・インタフェース関数 (RX-NET API 関数，および，ソケット API 関数) について解説しています。

5.1　概要

RX-NET(TCP/IP) が提供している API 関数は，ユーザが記述した処理プログラムから RX-NET(TCP/IP) が直接管理している資源 (ソケット，待ちキューなど) を間接的に操作するために用意されたサービス・ルーチンです。
以下に，RX-NET(TCP/IP) が提供している API 関数 (26 種類) を示します。

- RX-NET API 関数 (7 種類)

so_initialize	ll_config	ll_unconfig	ll_route	ll_del_static_route
ping	inet_addr			

- ソケット API 関数 (19 種類)

socket	closesock	setsopt	getsopt	bind
getsockname	getpeername	listen	blocking	nonblocking
connect	shutdown	accept	reject	send
sendto	recv	recvfrom	nselect	

5.2　API 関数の呼び出し

API 関数を C 言語，および，アセンブリ言語で記述された処理プログラムから発行する場合の呼び出し方法を以下に示します。

- C 言語

  API 関数を C 言語で記述された処理プログラムから発行する場合，通常の C 言語関数と同様の方法で呼び出しを行うことにより，API 関数のパラメータは RX-NET(TCP/IP) に引き数として渡され，該当処理が実行されます。

- アセンブリ言語

  API 関数をアセンブリ言語で記述された処理プログラムから発行する場合，ユーザが開発環境として使用する C コンパイラ・パッケージの関数呼び出し規約に従ったパラメータ，および，戻り番地の設定を行ったのち，jarl 命令による呼び出しを行うことにより，API 関数のパラメータは RX-NET(TCP/IP) に引き数として渡され，該当処理が実行されます。

注意　RX-NET(TCP/IP) が提供する API 関数を処理プログラムから発行する場合，以下に示したヘッダ・ファイルの定義 (インクルード処理) を行う必要があります。

stdrx85p.h	： RX850 Pro 用標準ヘッダ・ファイル
rxnet.h	： RX-NET(TCP/IP) 用標準ヘッダ・ファイル
fnsconfig.h	： 静的設定情報ヘッダ・ファイル

なお，stdrx85p.h は，RX850 Pro が提供しています。

5.3　データ・マクロ

RX-NET(TCP/IP) が提供する API 関数を発行する際に使用する各種データ・マクロ (データ・タイプ，バイト順反転用条件付きマクロなど) について以下に示します。

5.3.1　データ・タイプ

表 5-1 に，API 関数を発行する際に指定する各種パラメータのデータ・タイプ一覧を示します。
なお，データ・タイプのマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\ccdep.h，および，std.h で行われています。

表 5-1　データ・タイプ

マクロ	型	意味
u8	unsigned char	汎用 8 ビット整数
u16	unsigned short	汎用 16 ビット整数
u32	unsigned int	汎用 32 ビット整数
a16	u8 要素 2 の配列	汎用 16 ビット整数へのポインタ
a32	u8 要素 4 の配列	汎用 32 ビット整数へのポインタ
a48	u8 要素 6 の配列	MAC アドレス用 48 ビット整数へのポインタ

5.3.2　バイト順反転用条件付きマクロ

表 5-2 に，RX-NET(TCP/IP) が提供しているバイト順反転用条件付きマクロ一覧を示します。
なおバイト順反転用条件付きマクロのマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\bsd_in.h で行われています。

表 5-2　バイト順反転用条件付きマクロ

マクロ	意味
htons (a)	ホスト・バイト・オーダ形式の 16 ビット・データ “ a ” をネットワーク・バイト・オーダ形式の 16 ビット・データに変換
ntohs (a)	ネットワーク・バイト・オーダ形式の 16 ビット・データ “ a ” をホスト・バイト・オーダ形式の 16 ビット・データに変換
htonl (a)	ホスト・バイト・オーダ形式の 32 ビット・データ “ a ” をネットワーク・バイト・オーダ形式の 32 ビット・データに変換
ntohl (a)	ネットワーク・バイト・オーダ形式の 32 ビット・データ “ a ” をホスト・バイト・オーダ形式の 32 ビット・データに変換

5.3.3　アドレス・ファミリ

表 5-3 に，API 関数 socket を発行する際に指定するアドレス・ファミリー覧を示します。
なお，アドレス・ファミリのマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

表 5-3　アドレス・ファミリ

マクロ	数値	意味
AF_INET	0x2	インターネット・アドレス

5.3.4　ソケット・タイプ

表 5-4 に，API 関数 socket を発行する際に指定するソケット・タイプ一覧を示します。
なお，ソケット・タイプのマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

表 5-4　ソケット・タイプ

マクロ	数値	意味
SOCK_STREAM	0x1	バーチャル・サーキット型 (TCP)
SOCK_DGRAM	0x2	データグラム型 (UDP)

5.3.5　プロトコル・レベル

表 5-5 に，API 関数 setsopt，getsopt を発行する際に指定するプロトコル・レベル一覧を示します。
なお，プロトコル・レベルのマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\ip.h，および，socket.h で行われています。

表 5-5　プロトコル・レベル

マクロ	数値	意味
IP_IP	0x0	IP ネットワーク・レベル
SOL_SOCKET	0xffff	ソケット・インタフェース・レベル

5.3.6　ソケット・オプション

表 5-6 に，API 関数 setsopt，getsopt を発行する際に指定するソケット・オプション一覧を示します。
なお，ソケット・オプションのマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\ip.h，および，socket.h で行われています。

表 5-6　ソケット・オプション

マクロ	数値	意味
IP ネットワーク・レベル IP_IP 用ソケット・オプション		
IP_O_TOS	0x1	サービス・タイプ
IP_O_FRAG	0x2	断片化制御タイプ
IP_O_MAXTTL	0x3	生存時間 (TTL：Time To Live) の最大値
IP_O_MINTTL	0x4	生存時間 (TTL：Time To Live) の最小値
IP_O_SECURE	0x5	セキュリティ・オプション
IP_O_LSRR	0x6	Loose Source とルートの記録
IP_O_SSRR	0x7	Strict Source とルートの記録
IP_O_RR	0x8	ルートの記録
IP_O_STREAM	0x9	ストリーム識別子
IP_O_TIME	0xa	インターネット・タイムスタンプ
ソケット・インタフェース・レベル SOL_SOCKET 用ソケット・オプション		
SO_REUSEADDR	0x4	アドレスの再利用
SO_KEEPALIVE	0x8	キープ・アライブ・モード

マクロ	数値	意味
SO_DONTROUTE	0x10	Don't Route オプション
SO_LINGER	0x80	Linger オプション
SO_THROUGHPUT	0x100	Throughput オプション
SO_EXPEDITE	0x200	Expedite オプション

5.3.7　サービス・タイプ

表 5-7 に，API 関数 setsopt を発行する際に指定するサービス・タイプ一覧を示します。
なお，サービス・タイプのマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

表 5-7　サービス・タイプ

マクロ	数値	意味
IP_RELIABLE	0x4	高い信頼性
IP_THROUGHPUT	0x8	最効率のスループット
IP_DELAY	0x10	最短の遅延
IP_PRECEDENCE	0xe0	優先順位のマスク・ビット・パターン
IP_ROUTINE	0x0	ルーチン
IP_PRIORITY	0x20	優先度
IP_IMMED	0x40	即時
IP_FLSHO	0x60	フラッシュを無視
IP_FLASH	0x80	フラッシュ
IP_CRITIC	0xa0	CRITIC/ECP
IP_IC	0xc0	インターネットワーク制御
IP_NC	0xe0	ネットワーク制御

5.3.8　送信制御フラグ

表 5-8 に，API 関数 send，sendto を発行する際に指定する送信制御フラグ一覧を示します。
なお，送信制御フラグのマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

表 5-8　送信制御フラグ

マクロ	数値	意味
MSG_OOB	0x1	緊急データの送信
MSG_DONTROUTE	0x4	ルーティング・テーブル未使用で送信
MSG_FDBROADCAST	0x2000	全二重ブロードキャストで送信
MSG_NONBLOCKING	0x4000	ノンブロッキング・モードで送信
MSG_BLOCKING	0x8000	ブロッキング・モードで送信

5.3.9　受信制御フラグ

表 5-9 に，API 関数 recv，recvfrom を発行する際に指定する受信制御フラグ一覧を示します。
なお，受信制御フラグのマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

表 5-9　受信制御フラグ

マクロ	数値	意味
MSG_PEEK	0x2	データの受信 (読み出し位置の変更なし)
MSG_URGENT	0x800	緊急データの受信
MSG_TRUNCATE	0x1000	受信パケットの切り捨て
MSG_NONBLOCKING	0x4000	ノンブロッキング・モードで受信
MSG_BLOCKING	0x8000	ブロッキング・モードで受信

5.3.10　イベント・フラグ

表 5-10 に，API 関数 nselect を発行する際に指定するイベント・フラグ一覧を示します。
なお，イベント・フラグのマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\select.h で行われています。

表 5-10　イベント・フラグ

マクロ	数値	意味
READ_NOTIFY	((u16) 0x1)	メッセージの受信が可能
WRITE_NOTIFY	((u16) 0x2)	送信キューに空きがある
ACCEPT_NOTIFY	((u16) 0x4)	受動コネクションが確立
CLOSE_NOTIFY	((u16) 0x8)	ソケットの解放処理が正常終了
CONNECT_NOTIFY	((u16) 0x10)	能動コネクションが確立
EXCEPT_NOTIFY	((u16) 0x20)	例外が発生
RSHUTDOWN_NOTIFY	((u16) 0x40)	読み取り方向がピアにより遮断
TIMEOUT_NOTIFY	((u16) 0x80)	要求動作がタイムアウト
WSHUTDOWN_NOTIFY	((u16) 0x100)	書き込み方向が遮断
URGENT_NOTIFY	((u16) 0x200)	緊急データが利用可能
SENDQEMPTY_NOTIFY	((u16) 0x2000)	送信キューが空

5.3.11　戻り値

表 5-11 に，API 関数からの戻り値一覧を示します。
なお，戻り値のマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\nerrno.h，および，std.h で行われています。

表 5-11　戻り値

マクロ	数値	意味
ENOERR	0x0	正常終了
EPERM	0x1	オーナではありません
ENOENT	0x2	ファイル，または，ディレクトリの指定が不正です

マクロ	数値	意味
ENXIO	0x6	ネットワーク・デバイス名，または，アドレスの指定が不正です
EBADF	0x9	ファイル番号の指定が不正です
ENOMEM	0xc	メモリ領域が確保できません
EACCESS	0xd	対象ソケットへのアクセスが拒否されました
EFAULT	0xe	アドレスの指定が不正です
ENODEV	0x13	ネットワーク・デバイス名の指定が不正です
EINVAL	0x16	パラメータの指定が不正です
EMFILE	0x17	最大ソケット生成数を超えています
EWOULDBLOCK	0x50	対象ソケットがノンブロッキング・モードです
EINPROGRESS	0x51	現在，該当処理が実行されています
EALREADY	0x52	既に該当処理が実行されています
ENOTSOCK	0x53	ソケットの指定が不正です
EMSGSIZE	0x55	メッセージのサイズが大きすぎます
EPROTOTYPE	0x56	プロトコルの指定が不正です
ENOPROTOOPT	0x57	ソケット・オプションの指定が不正です
EPROTONOSUPPORT	0x58	プロトコルの指定が不正です
ESOCKTNOSUPPORT	0x59	サービス・タイプの指定が不正です
EOPNOTSUPP	0x5a	ソケット・タイプの指定が不正です
EAFNOSUPPORT	0x5b	アドレス・ファミリの指定が不正です
EADDRINUSE	0x5c	アドレスの指定が不正です
EADDRNOTAVAIL	0x5d	要求アドレスを割り当てることができません
ENETDOWN	0x5e	ネットワークがダウンしています
ENETUNREACH	0x5f	ネットワークに到達できません
ECONNABORTED	0x61	接続が遮断されています
ECONNRESET	0x62	ピアによる接続のリセットが行われました
ENOBUFS	0x63	メッセージのサイズが大きすぎます
EISCONN	0x64	既に能動モード，または，受動モードへの移行が完了しています
ENOTCONN	0x65	接続が確立していません
ESHUTDOWN	0x66	shutdown の発行により接続が遮断されているため，メッセージを送受信することができません
ETIMEDOUT	0x67	接続がタイムアウトしています
ECONNREFUSED	0x68	IP アドレス，ポート番号の割り付け，および，受動モードへの移行が完了していません
EHOSTDOWN	0x6a	ホスト・マシンがダウンしています
EHOSTUNREACH	0x6b	ホスト・マシンを捕捉することができません
ENOURGENTDATA	0x6c	緊急データを受信することができません
ERR	-1	異常終了

5.4　データ構造体

RX-NET(TCP/IP) が提供する API 関数を発行する際に使用する各種データ構造体 (汎用アドレス構造体，選択構造体など) について以下に示します。

5.4.1　汎用アドレス構造体

以下に，汎用アドレス構造体 sockaddr を示します。
なお，汎用アドレス構造体 sockaddr の定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

```
typedef     struct      sockaddr {
            short       sa_type ;           /* アドレス・ファミリ */
            char        sa_data [ 22 ] ;    /* Value Of Address */
} saddr ;
```

5.4.2　インターネット・アドレス用アドレス構造体

以下に，インターネット・アドレス用アドレス構造体 isockaddr を示します。
なお,インターネット・アドレス用アドレス構造体 isockaddr の定義は,標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

```
typedef     struct      isockaddr {
            short       sa_type ;           /* アドレス・ファミリ */
            a16         sa_port ;           /* ポート番号 */
            a32         sa_addr ;           /* Network / Host Address */

} isaddr ;
```

5.4.3　インターネット・アドレス用アドレス構造体

以下に，インターネット・アドレス用アドレス構造体 sockaddr_in を示します。
なお,インターネット・アドレス用アドレス構造体 sockaddr_in の定義は,標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

```
typedef     struct      sockaddr_in {
            i16         sin_family ;                    /* アドレス・ファミリ */
            u16         sin_port ;                      /* ポート番号 */
            struct      in_add          sin_addr ;      /* Network / Host Address */
            char        sin_zero [ 8 ] ;                /* システム予約領域 */
} ;

typedef     struct      in_addr {
            union {
                        struct {
                                        char        s_b1 , s_b2 , s_b3 , s_b4 ;
                        } S_un_b ;
                        struct {
                                        u16         s_w1 , s_w2 ;
                        } S_un_w ;
                        long            S_addr ;
            } S_un ;
} ;
```

5.4.4　IP ソケット・セキュリティ・オプション情報

以下に，IP ソケット・セキュリティ・オプション情報 ipos_t を示します。
なお，IP ソケット・セキュリティ・オプション情報 ipos_t の定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

```
typedef     struct      {
            a16         ipos_security ;         /* セキュリティ */
            a16         ipos_compartments ;     /* コンパートメント */
            a16         ipos_handling ;         /* 処理制限 */
            a32         ipos_tcc ;              /* 伝送制御コード */
} ipos_t ;
```

注意　本情報についての詳細は，「インターネットに関する研究開発機関 IETF (Internet Engineering Task Force) が取りまとめた公開技術文書 RFC」を参照してください。

5.4.5　IP ソケット・タイムスタンプ・オプション情報

以下に，IP ソケット・タイムスタンプ・オプション情報 ipot_t を示します。
なお，IP ソケット・タイムスタンプ・オプション情報 ipot_t の定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

```
typedef     struct      ipot_t {
            a16         ipot_overflow_type ;    /* オーバフロー・フラグ */
            a32         ipot_timestamps [ 9 ] ; /* タイムスタンプ */
} ipot_t ;
```

5.4.6　Linger オプション情報

以下に，Linger オプション情報 lingeropt を示します。
なお，Linger オプション情報 lingeropt の定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\socket.h で行われています。

```
typedef     struct      lingeropt {
            int         linger_on ;             /* Linger オプションの ON / OFF */
            int         linger_time ;           /*  Linger タイムアウト ( 単位：秒 ) */
} lingeropt ;
```

以下に，Linger オプション情報 lingeropt の詳細を示します。

- linger_on

  Linger オプションの ON / OFF を指定します。
  なお，linger_on として指定可能な値は，0x0，0x1 のいずれかに限られます。

  0x0　：OFF
  0x1　：ON

- linger_time

  API 関数 closesock を発行してから実際にソケットの解放処理を行うまでの遅延時間 (Linger タイムアウト，単位：秒) を指定します。

5.4.7　選択構造体

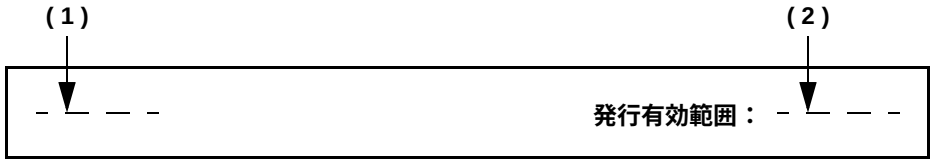
以下に，選択構造体 sel を示します。

なお，選択構造体 sel の定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\select.h で行われています。

```
typedef     struct      sel {
            u16         se_inflags ;        /* 要求イベントの論理和 OR */
            u16         se_outflags ;       /* 発生イベントの論理和 OR */
            i16         se_fd ;             /* Holds Process Local File ID */
            i16         se_1reserved ;      /* システム予約領域 */
            u32         se_user ;           /* ユーザ用拡張領域 */
            u32         se_2reserved ;      /* システム予約領域 */
} sel ;
```

5.5　API 関数解説

次項から RX-NET(TCP/IP) が提供している API 関数について，以下の記述フォーマットに従って解説します。



(3) ⟶ **概要**

(4) ⟶ **C 言語形式**

(5) ⟶ **パラメータ**

I/O	パラメータ	説　　明

(6) ⟶ **機能**

(7) ⟶ **戻り値**

（1） 名称

API 関数の名称を示しています。

（2） 発行有効範囲

API 関数の発行が可能な処理プログラムの種別を示しています。

タスク　　：タスクからのみ発行可能
非タスク　　：非タスクからのみ発行可能
タスク / 非タスク　：タスク，非タスクのどちらかも発行可能

（3） 概要

API 関数の機能概要を示しています。

（4） C 言語形式

API 関数を C 言語で記述された処理プログラムから発行する際の記述形式を示しています。

（5） パラメータ

API 関数のパラメータを以下の形式で示しています。

I/O	パラメータ	説　　明
A	B	C

A ）パラメータの種類

I　…　RX-NET(TCP/IP) への入力パラメータ

O　…　RX-NET(TCP/IP) からの出力パラメータ

B ）パラメータのデータ・タイプ

C ）パラメータの説明

（6） 機能

API 関数の機能詳細を示しています。

（7） 戻り値

API 関数からの戻り値をデータ・マクロ，および，数値で示しています。

5.5.1　RX-NET API 関数

表 5-12 に，RX-NET(TCP/IP) が提供している RX-NET API 関数の一覧を示します。

表 5-12　RX-NET API 関数

API 関数名	機能概要
so_initialize	RX-NET(TCP/IP) の初期化
ll_config	ネットワーク・インタフェースの起動
ll_unconfig	ネットワーク・インタフェースの遮断
ll_route	IP アドレスの登録
ll_del_static_route	IP アドレスの登録解除
ping	ICMP ECHO パケットの送信
inet_addr	IP アドレスの変換

次頁以降に，RX-NET API 関数の外部インタフェース仕様詳細を示します。

so_initialize

発行有効範囲：タスク

概要

RX-NET(TCP/IP) の初期化

C 言語形式

```
int         so_initialize ( void ) ;
```

パラメータ

なし

機能

RX-NET(TCP/IP) が提供する機能を実現するうえで必要となる各種初期化処理を実行します。

なお，RX-NET(TCP/IP) では，RX-NET(TCP/IP) の初期化処理として，RX-NET(TCP/IP) が提供している機能を実現する際に必要となる各種資源の生成，および，デバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されている全てのドライバ関数 devname_init の呼び出しを行っています。

注意 1　デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「6.2.6　デバイス・ドライバ・エントリ・テーブル情報」を参照していください。

注意 2　ドライバ関数 devname_init についての詳細は，「6.4.1　外部インタフェース仕様 devname_init」を参照してください。

戻り値

ENOERR	0x0	正常終了
EALREADY	0x52	既に本 API 関数が発行されています

ll_config

発行有効範囲：タスク

概要

ネットワーク・インタフェースの起動

C 言語形式

int　　ll_config (char *_devname_ , u32 _ipaddress_ , u32 _ipmask_ , unsigned _mtu_ , u32 _ospfarea_) ;

パラメータ

I/O	パラメータ		説　　明
I	char	*_devname_ ;	ネットワーク・デバイス名を格納した領域へのポインタ
I	u32	_ipaddress_ ;	IP アドレス (ネットワーク・バイト・オーダ形式)
I	u32	_ipmask_ ;	ネット・マスク (ネットワーク・バイト・オーダ形式)
I	unsigned	_mtu_ ;	最大転送単位 (単位：バイト)
I	u32	_osfarea_ ;	システム予約領域 (0x0 固定)

機能

ipaddress，_ipmask_，_mtu_ で指定された情報をもとに，_devname_ で指定されたネットワーク・インタフェースを初期化したのち，起動します。

なお，RX-NET(TCP/IP) では，ネットワーク・インタフェースの初期化処理として，RX-NET(TCP/IP) が提供している機能を実現する際に必要となる各種資源の生成，および，パラメータ _devname_ で指定されたデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているドライバ関数 devname_updown の呼び出しを行っています。

注意 1　_devname_ に指定可能なネットワーク・デバイス名は，あらかじめデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているネットワーク・デバイス名に限られます。
なお，デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「6.2.6　デバイス・ドライバ・エントリ・テーブル情報」を参照してください。

注意 2　_mtu_ に 0x0，または，1500 以上の値を指定した場合，RX-NET(TCP/IP) は最大転送単位 (MTU：Maximum Transfer Unit) のディフォールト値 1500 が指定されていたものとして処理を行います。

注意 3　ドライバ関数 devname_updown についての詳細は，「6.4.1　外部インタフェース仕様 devname_updown」を参照してください。

戻り値

ENOERR	0x0	正常終了
ENODEV	0x13	ネットワーク・デバイス名 _devname_ の指定が不正です
EINVAL	0x16	パラメータの指定が不正です
EALREADY	0x52	既に本 API 関数が発行されています
EAFNOSUPPORT	0x5b	IP アドレス _ipaddress_ の指定が不正です

ll_unconfig

発行有効範囲：タスク

概要

ネットワーク・インタフェースの遮断

C 言語形式

```
int        ll_unconfig ( char *devname , u32 ipaddress , u32 ipmask ) ;
```

パラメータ

I/O	パラメータ		説　　明
I	char	*devname ;	ネットワーク・デバイス名を格納した領域へのポインタ
I	u32	ipaddress ;	IP アドレス (ネットワーク・バイト・オーダ形式)
I	u32	ipmask ;	ネット・マスク (ネットワーク・バイト・オーダ形式)

機能

devname，*ipaddress*，*ipmask* で指定されたネットワーク・インタフェースを遮断します。

なお，RX-NET(TCP/IP) では，ネットワーク・インタフェースの遮断処理として，RX-NET(TCP/IP) が提供している機能を実現する際に必要となる各種資源の削除，および，パラメータ *devname* で指定されたデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているドライバ関数 devname_updown の呼び出しを行っています。

注意 1　*devname* に指定可能なネットワーク・デバイス名は，あらかじめデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているネットワーク・デバイス名に限られます。
なお，デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「**6.2.6　デバイス・ドライバ・エントリ・テーブル情報**」を参照してください。

注意 2　ドライバ関数 devname_updown についての詳細は，「**6.4.1　外部インタフェース仕様 devname_updown**」を参照してください。

戻り値

ENOERR	0x0	正常終了
ENODEV	0x13	ネットワーク・デバイス名 *devname* の指定が不正，または，ll_config が発行されていません
EALREADY	0x52	so_initialize が発行されていません

ll_route

発行有効範囲：タスク

概要

IP アドレスの登録

C 言語形式

```
int     ll_route ( char *devname , u32 gateway , u32 ipaddress , u32 ipmask , int hops ) ;
```

パラメータ

I/O	パラメータ		説　　明
I	char	*devname ;	ネットワーク・デバイス名を格納した領域へのポインタ
I	u32	gateway ;	IP アドレス (ネットワーク・バイト・オーダ形式)
I	u32	ipaddress ;	システム予約領域 (0x0 固定)
I	u32	ipmask ;	システム予約領域 (0x0 固定)
I	int	hops ;	システム予約領域 (0x0 固定)

機能

devname で指定されたネットワーク・インタフェースのルーティング・テーブルに *gateway* で指定されたルータの IP アドレスを登録します。

注意　*devname* に指定可能なネットワーク・デバイス名は，あらかじめデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているネットワーク・デバイス名に限られます。
なお，デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「6.2.6　デバイス・ドライバ・エントリ・テーブル情報」を参照してください。

戻り値

ENOERR	0x0	正常終了
ENODEV	0x13	ネットワーク・デバイス名 *devname* の指定が不正，または，so_initialize が発行されていません
EINVAL	0x16	パラメータの指定が不正です
EALREADY	0x52	既に *gateway* で指定された IP アドレスはルーティング・テーブルに登録されています

ll_del_static_route

発行有効範囲：タスク

概要

IP アドレスの登録解除

C 言語形式

int　　ll_del_static_route (char *_devname_ , u32 _gateway_ , u32 _ipaddress_ , u32 _ipmask_) ;

パラメータ

I/O	パラメータ		説　　明
I	char	*_devname_ ;	ネットワーク・デバイス名を格納した領域へのポインタ
I	u32	_gateway_ ;	IP アドレス (ネットワーク・バイト・オーダ形式)
I	u32	_ipaddress_ ;	システム予約領域 (0x0 固定)
I	u32	_ipmask_ ;	システム予約領域 (0x0 固定)

機能

devname で指定されたネットワーク・インタフェースのルーティング・テーブルから _gateway_ で指定されたルータの IP アドレスを削除 (登録解除) します。

注意　_devname_ に指定可能なネットワーク・デバイス名は，あらかじめデバイス・ドライバ・エントリ・テーブル情報 ndevsw に登録されているネットワーク・デバイス名に限られます。
なお，デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「6.2.6　デバイス・ドライバ・エントリ・テーブル情報」を参照してください。

戻り値

ENOERR	0x0	正常終了
ENODEV	0x13	ネットワーク・デバイス名 _devname_ の指定が不正です
EALREADY	0x52	so_initialize が発行されていません

ping

発行有効範囲：タスク

概要

ICMP ECHO パケットの送信

C 言語形式

```
int     ping ( u32 ipaddress , int data_len , int npackets ) ;
```

パラメータ

I/O	パラメータ		説　明
I	u32	*ipaddress* ;	IP アドレス (ネットワーク・バイト・オーダ形式)
I	int	*data_len* ;	ICMP ECHO パケットのデータ部のサイズ (単位：バイト)
I	int	*npackets* ;	送信パケット数

機能

ipaddress で指定されたホスト・マシンに *data_len* ,および ,*npackets* で指定された ICMP ECHO パケットを送信します。

なお，本 API 関数からの復帰処理は，*npackets* で指定された数の ICMP ECHO REPLY パケットを受信した際，または，本 API 関数の発行から 1000 ミリ秒が経過した際に行われます。

戻り値

ENOERR	0x0	正常終了
ETIMEDOUT	0x67	*npackets* で指定された数の ICMP ECHO REPLY パケットを 1000 ミリ秒が経過するまでの間に受信することができなかった

inet_addr　　発行有効範囲：タスク

概要

IP アドレスの変換

C 言語形式

int　　inet_addr (u8 *ipaddress) ;

パラメータ

I/O	パラメータ	説　　明
I	u8　　*ipaddress ;	IP アドレス (ドット形式) を格納した領域へのポインタ

機能

ipaddress で指定されたドット形式の IP アドレスをネットワーク・バイト・オーダ形式の IP アドレスに変換します。
なお，本 API 関数の処理が正常終了した際には，“ ネットワーク・バイト・オーダ形式に変換された IP アドレス ” が戻り値として返されます。

注意　ドット形式の IP アドレスとは，“ 202.247.5.136 ” などといったドット “ . ” で区切られた数字から構成されている IP アドレスを意味しています。

戻り値

正常終了	ネットワーク・バイト・オーダ形式に変換された IP アドレス
異常終了	0xffffffff

5.5.2　ソケット API 関数

表 5-13 に，RX-NET(TCP/IP) が提供しているソケット API 関数の一覧を示します。

表 5-13　ソケット API 関数

API 関数名	機能概要
socket	ソケットの生成
closesock	ソケットの解放
setsopt	ソケット・オプションの設定
getsopt	ソケット・オプションの獲得
bind	IP アドレス，ポート番号の割り付け
getsockname	IP アドレス，ポート番号の獲得
getpeername	遠隔側 IP アドレス，ポート番号の獲得
listen	受動モードへの移行
blocking	ブロッキング・モードへの移行
nonblocking	ノンブロッキング・モードへの移行
connect	接続の確立
shutdown	接続の遮断
accept	接続要求の受け付け
reject	接続要求の削除
send	メッセージの送信
sendto	メッセージ (受信先アドレス情報を含む) の送信
recv	メッセージの受信
recvfrom	メッセージ (送信元アドレス情報を含む) の受信
nselect	イベントの選択

次頁以降に，ソケット API 関数の外部インタフェース仕様詳細を示します。

socket

発行有効範囲：タスク

概要

ソケットの生成

C 言語形式

```
int     socket ( int af , int type , int protocol , int *errp ) ;
```

パラメータ

I/O	パラメータ		説　　明
I	int	*af* ;	アドレス・ファミリ (AF_INET 固定)
I	int	*type* ;	ソケット・タイプ SOCK_STREAM　：バーチャル・サーキット型 (TCP) SOCK_DGRAM　：データグラム型 (UDP)
I	int	*protocol* ;	システム予約領域 (0x0 固定)
O	int	**errp* ;	エラー・コードを格納する領域へのポインタ

機能

type で指定された情報をもとに，ネットワーク通信を実現するうえで必要となるソケット (通信用エンド・ポイント) をブロッキング・モードで生成します。
なお，本 API 関数では，ソケットの生成処理として，以下に示した値でソケット・オプションの初期化を行っています。

- IP ネットワーク・レベル用ソケット・オプション

サービス・タイプ	：通常の信頼性，通常のスループット，通常の遅延，ルーチンを優先
断片化制御タイプ	：OFF
生存時間の最大値	：MAXTTL(単位：秒)
生存時間の最小値	：MINTTL(単位：秒)
セキュリティ・オプション	：OFF
Loose Source とルートの記録	：OFF
Strict Source とルートの記録	：OFF
ルートの記録	：OFF
ストリーム識別子	：OFF
インターネット・タイムスタンプ	：OFF

- ソケット・インタフェース・レベル用ソケット・オプション

アドレスの再利用	：OFF
キープ・アライブ・モード	：OFF
Don't Route オプション	：OFF
Linger オプション	：OFF
Throughput オプション	：OFF
Expedite オプション	：OFF

また，本 API 関数の処理が正常終了した際には，“ 生成したソケットのソケット記述子 ” が戻り値として返されます。

注意 1　*errp* で指定された領域には，以下に示したエラー・コードが格納されます。

ENOERR	(0x0)	：正常終了
EMFILE	(0x17)	：最大ソケット生成数を超えています
EPROTONOSUPPORT	(0x58)	：システム予約領域 *protocol* の指定が不正です

ESOCKTNOSUPPORT	(0x59)	：ソケット・タイプ *type* の指定が不正です
EAFNOSUPPORT	(0x5b)	：アドレス・ファミリ *af* の指定が不正です
ENETDOWN	(0x5e)	：ネットワークがダウンしています

注意 2　**生存時間の最大値** MAXTTL**，および，最小値** MINTTL **についての詳細は，「6.2.7　生存時間情報」を参照してください。**

注意 3　**ソケット・オプションの設定処理についての詳細は「5.5.2　ソケット API 関数 setsopt」を，**IP **アドレス，ポート番号の割り付け処理についての詳細は「5.5.2　ソケット API 関数 bind」を参照してください。**

戻り値

正常終了	**生成したソケットのソケット記述子**
異常終了	ERR　(-1)

closesock

発行有効範囲：タスク

概要

ソケットの解放

C 言語形式

int　　closesock (int *sd*) ;

パラメータ

I/O	パラメータ	説　　明
I	int　　*sd* ;	ソケット記述子

機能

sd で指定されたソケットで確立している接続を遮断 (送受信両方向の接続を遮断) したのち，該当ソケットの解放処理を実行します。
なお，RX-NET(TCP/IP) では，ソケットの解放処理として，ソケット用各種リソースの解放を行っています。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　本 API 関数を発行した際，対象ソケットの Linger オプションが ON であり，かつ，Linger タイムアウトが 0x0 以外の値であった場合には，接続の遮断処理，および，該当ソケットの解放処理は Linger タイムアウトで指定された時間が経過するまで遅延されます。

注意 3　本 API 関数では，正常終了時の後処理として，RX-NET(TCP/IP) に対する CLOSE_NOTIFY イベントの発生通知を行います。

戻り値

ENOERR	0x0	正常終了
EINVAL	0x16	パラメータの指定が不正です
EALREADY	0x52	対象ソケットが生成されていません
ENETDOWN	0x5e	ネットワークがダウンしています

setsopt

発行有効範囲：タスク

概要

ソケット・オプションの設定

C 言語形式

int　　setsopt (int *sd* , int *level* , int *optname* , char **optval* , int *opt_len*) ;

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
I	int	*level* ;	プロトコル・レベル IP_IP　：IP ネットワーク・レベル SOL_SOCKET　：ソケット・インタフェース・レベル
I	int	*optname* ;	ソケット・オプション 【 *level* に IP_IP を指定した場合 】 IP_O_TOS　：サービス・タイプ IP_O_FRAG　：断片化制御タイプ IP_O_MAXTTL　：生存時間の最大値 IP_O_MINTTL　：生存時間の最小値 IP_O_SECURE　：セキュリティ・オプション IP_O_LSRR　：Loose Source とルートの記録 IP_O_SSRR　：Strict Source とルートの記録 IP_O_RR　：ルートの記録 IP_O_STREAM　：ストリーム識別子 IP_O_TIME　：インターネット・タイムスタンプ 【 *level* に SOL_SOCKET を指定した場合 】 SO_REUSEADDR　：アドレスの再利用 ON SO_KEEPALIVE　：キープ・アライブ・モード ON SO_DONTROUTE　：Don't Route オプション ON SO_LINGER　：Linger オプション SO_THROUGHPUT　：Throughput オプション ON SO_EXPEDITE　：Expedite オプション ON ~SO_REUSEADDR　：アドレスの再利用 OFF ~SO_KEEPALIVE　：キープ・アライブ・モード OFF ~SO_DONTROUTE　：Don't Route オプション OFF ~SO_THROUGHPUT　：Throughput オプション OFF ~SO_EXPEDITE　：Expedite オプション OFF
I	char	**optval* ;	設定するオプションの値を格納した領域へのポインタ
I	int	*opt_len* ;	*optval* で指定された領域のサイズ (単位：バイト)

機能

level，*optname*，および，*optval* で指定された情報をもとに，*sd* で指定されたソケットのソケット・オプションを変更します。

以下に，*optname* で指定されたソケット・オプションの種類別 (IP_O_TOS，IP_O_FRAF，IP_O_MAXTTL など) に，*optval* に設定する値の意味を示します。

IP_O_TOS　…サービス・タイプ

【 信頼性 (ビット 2) 】
IP_RELIABLE　：高い信頼性

【 スループット (ビット 3) 】
IP_THROUGHPUT　：最効率のスループット

【 遅延 (ビット 4) 】
IP_DELAY　：最短の遅延

【 優先順位 (ビット 5 〜 7) 】
IP_ROUTINE　：ルーチン
IP_PRIORITY　：優先度
IP_IMMED　：即時
IP_FLSHO　：フラッシュを無視
IP_FLASH　：フラッシュ
IP_CRITIC　：CRITIC/ECP
IP_IC　：インターネットワーク制御
IP_NC　：ネットワーク制御

IP_O_FRAG　…断片化制御タイプ
0x0　：OFF
0x1　：ON

IP_O_MAXTTL　…生存時間の最大値
0x0 〜 0xff　：生存時間 (単位：ミリ秒)

IP_O_MINTTL　…生存時間の最小値
0x0 〜 0xff　：生存時間 (単位：ミリ秒)

IP_O_SECURE　…セキュリティ・オプション
0x0　：OFF
0x1　：ON

IP_O_LSRR　…Loose Source とルートの記録
0x0　：OFF
0x1　：ON

IP_O_SSRR　…Strict Source とルートの記録
0x0　：OFF
0x1　：ON

IP_O_RR　…ルートの記録
0x0　：OFF
0x1　：ON

IP_O_STREAM　…ストリーム識別子
0x0　：OFF
0x1　：ON

IP_O_TIME　…インターネット・タイムスタンプ
0x0　：OFF
0x1　：ON

SO_LINGER　…Linger オプション
0x0　：OFF
0x1　：ON

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　ソケット・オプション *optname* に IP_O_SECURE を指定した場合，*optval* には IP ソケット・セキュリティ・オプション情報 ipos_t へのポインタを設定します。
なお，IP ソケット・セキュリティ・オプション情報 ipos_t についての詳細は，「5.4.4　IP ソケット・セキュリティ・オプション情報」を参照してください。

注意 3　**ソケット・オプション** *optname* **に** IP_O_TIME **を指定した場合，** *optval* **には** IP **ソケット・タイムスタンプ・オプション情報** ipot_t **へのポインタを設定します。**
なお，IP **ソケット・タイムスタンプ・オプション情報** ipot_t **についての詳細は，「5.4.5　IP ソケット・タイムスタンプ・オプション情報」を参照してください。**

注意 4　**ソケット・オプション** *optname* **に** SO_LINGER **を指定した場合，** *optval* **には** Linger **オプション情報** lingeropt **へのポインタを設定します。**
なお，Linger**オプション情報**lingeropt**についての詳細は，「5.4.6　Linger オプション情報」を参照してください。**

注意 5　**ソケット・オプション** *optname* **に以下に示した何れかの値を指定した場合，** *optval* **には** NULL **を，** *opt_len* **には** 0x0 **を設定します。**

SO_REUSEADDR	SO_KEEPALIVE	SO_DONTROUTE
SO_LINGER	SO_THROUGHPUT	SO_EXPEDITE
~SO_REUSEADDR	~SO_KEEPALIVE	~SO_DONTROUTE
~SO_LINGER	~SO_THROUGHPUT	~SO_EXPEDITE

戻り値

ENOERR	0x0	**正常終了**
EINVAL	0x16	**パラメータの指定が不正です**
EPROTOTYPE	0x56	**プロトコル・レベル** *level* **の指定が不正です**
ENOPROTOOPT	0x57	**ソケット・オプション** *optname* **の指定が不正です**

getsopt

発行有効範囲：タスク

概要

ソケット・オプションの獲得

C 言語形式

int　　getsopt (int *sd* , int *level* , int o*ptname* , char **optval* , int **opt_len*) ;

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
I	int	*level* ;	プロトコル・レベル IP_IP　：IP ネットワーク・レベル SOL_SOCKET　：ソケット・インタフェース・レベル
I	int	*optname* ;	ソケット・オプション 【 *level* に IP_IP を指定した場合 】 IP_O_TOS　：サービス・タイプ IP_O_FRAG　：断片化制御タイプ IP_O_MAXTTL　：生存時間の最大値 IP_O_MINTTL　：生存時間の最小値 IP_O_SECURE　：セキュリティ・オプション IP_O_LSRR　：Loose Source とルートの記録 IP_O_SSRR　：Strict Source とルートの記録 IP_O_RR　：ルートの記録 IP_O_STREAM　：ストリーム識別子 IP_O_TIME　：インターネット・タイムスタンプ 【 *level* に SOL_SOCKET を指定した場合 】 SO_REUSEADDR　：アドレスの再利用 SO_KEEPALIVE　：キープ・アライブ・モード SO_DONTROUTE　：Don't Route オプション SO_LINGER　：Linger オプション SO_THROUGHPUT　：Throughput オプション SO_EXPEDITE　：Expedite オプション
O	char	**optval* ;	獲得したソケット・オプションの値を格納する領域へのポインタ
O	int	**opt_len* ;	*optval* で指定された領域のサイズ (単位：バイト) を格納する領域へのポインタ

機能

sd で指定されたソケットに設定されているソケット・オプションの値を *optval* で指定された領域に格納します。
なお，獲得するソケット・オプションの種類については，*level*，および，*optname* の組み合わせで指定します。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　獲得したソケット・オプションの値 *optval* についての詳細は，「5.5.2　ソケット API 関数 setsopt」を参照してください。

戻り値

ENOERR	0x0	**正常終了**
EINVAL	0x16	**パラメータの指定が不正です**
ENOPROTOOPT	0x57	**ソケット・オプション** *optname* **の指定が不正です**
EAFNOSUPPORT	0x5b	**プロトコル・レベル** *level* **の指定が不正です**
ENETDOWN	0x5e	**ネットワークがダウンしています**
ENOTCONN	0x65	**接続が確立していません**

bind

発行有効範囲：タスク

概要

IP アドレス，ポート番号の割り付け

C 言語形式

int　　bind (int *sd* , saddr **name* , int *name_len*) ;

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
I	saddr	**name* ;	アドレス構造体 sockaddr_in へのポインタ
I	int	*name_len* ;	*name* で指定されたアドレス構造体のサイズ (単位:バイト)

機能

sd で指定されたソケットに *name* で指定されたアドレス構造体 sockaddr_in に格納されている IP アドレス，ポート番号を割り付けます。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　アドレス構造体 sockaddr_in についての詳細は，「**5.4.3　インターネット・アドレス用アドレス構造体**」を参照してください。

戻り値

ENOERR	0x0	正常終了
EINVAL	0x16	パラメータの指定が不正です
EAFNOSUPPORT	0x5b	対象ソケットのアドレス・ファミリが AF_INET ではありません
EADDRINUSE	0x5c	IP アドレスの指定が不正です
ENETDOWN	0x5e	ネットワークがダウンしています

getsockname

発行有効範囲：タスク

概要

IP アドレス，ポート番号の獲得

C 言語形式

int　　getsockname (int *sd* , saddr **name* , int **name_len*) ;

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
O	saddr	**name* ;	アドレス構造体 sockaddr_in へのポインタ
I, O	int	**name_len* ;	*name* で指定されたアドレス構造体のサイズ (単位：バイト) を格納した領域へのポインタ

機能

sd で指定されたソケットの IP アドレス,ポート番号を *name* で指定されたアドレス構造体 sockaddr_in に格納します。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　アドレス構造体 sockaddr_in についての詳細は，「**5.4.3　インターネット・アドレス用アドレス構造体**」を参照してください。

戻り値

ENOERR	0x0	正常終了
EINVAL	0x16	パラメータの指定が不正です

getpeername

発行有効範囲：タスク

概要

遠隔側 IP アドレス，ポート番号の獲得

C 言語形式

```
int        getpeername ( int sd , saddr *name , int *name_len ) ;
```

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
O	saddr	**name* ;	アドレス構造体 sockaddr_in へのポインタ
I, O	int	**name_len* ;	*name* で指定されたアドレス構造体のサイズ (単位：バイト) を格納した領域へのポインタ

機能

sd で指定されたソケットと接続が確立している遠隔側ソケット (API 関数 connect，または，accept の発行により接続が確立した遠隔側ソケット) の IP アドレス,ポート番号を *name* で指定されたアドレス構造体 sockaddr_in に格納します。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　アドレス構造体 sockaddr_in についての詳細は，「**5.4.3　インターネット・アドレス用アドレス構造体**」を参照してください。

戻り値

ENOERR	0x0	正常終了
EINVAL	0x16	パラメータの指定が不正です
ENOTCONN	0x65	接続が確立していません

listen　　発行有効範囲：タスク

概要

受動モードへの移行

C 言語形式

int　　listen (int *sd* , int *backlog*) ;

パラメータ

I/O	パラメータ	説　　明
I	int　*sd* ;	ソケット記述子
I	int　*backlog* ;	接続要求の最大保留数 (0x1 〜 0x5)

機能

*sd*で指定されたソケットに*backlog*で指定された接続要求の最大保留数を設定したのち,受動モードへと移行させます。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　*backlog* に指定可能な値は，0x1 〜 0x5 に限られます。なお，*backlog* に 0x5 よりも大きな値を指定した場合，RX-NET(TCP/IP) は 0x5 が指定されていたものとして処理を行います。

注意 3　API 関数 socket の発行により生成されたソケットの状態は，能動モード (クライアントが利用可能な状態)，受動モード (サーバが利用可能な状態) のいずれの状態でもありません。

戻り値

ENOERR	0x0	正常終了
EINVAL	0x16	パラメータの指定が不正です
EOPNOTSUPP	0x5a	対象ソケットのソケット・タイプが SOCK_STREAM ではありません
EISCONN	0x64	既に本 API 関数が発行されています

blocking

発行有効範囲：タスク

概要

ブロッキング・モードへの移行

C 言語形式

```
int         blocking ( int sd ) ;
```

パラメータ

I/O	パラメータ	説　明
I	int *sd* ;	ソケット記述子

機能

sd で指定されたソケットをブロッキング・モードへと移行させます。

これにより，API 関数 (connect，accept，send，sendto，recv，recvfrom など) は，特定の条件が成立するまでの間，要求動作の実行が中断 (ブロック) されます。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　本 API 関数では，ブロッキング・モードへの移行要求がキューイングされません。このため，既に対象ソケットがブロッキング・モードへの移行していた場合には，何も処理は行わず，エラーとしても扱いません。

注意 3　API 関数 socket では，該当ソケットを “ ブロッキング・モード ” で生成しています。

戻り値

ENOERR	0x0	正常終了
EINVAL	0x16	パラメータの指定が不正です
ENETDOWN	0x5e	ネットワークがダウンしています

nonblocking　　発行有効範囲：タスク

概要

ノンブロッキング・モードへの移行

C 言語形式

int　　nonblocking (int *sd*) ;

パラメータ

I/O	パラメータ	説　　明
I	int　　*sd* ;	ソケット記述子

機能

sd で指定されたソケットをノンブロッキング・モードへと移行させます。

これにより，API 関数 (connect，accept，send，sendto，recv，recvfrom など) は，即時に特定の条件が成立しなかった場合，異常終了 (戻り値，または，エラー・コードとして EWOULDBLOCK を返却) となります。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　本 API 関数では，ノンブロッキング・モードへの移行要求がキューイングされません。このため，既に対象ソケットがノンブロッキング・モードへの移行していた場合には，何も処理は行わず，エラーとしても扱いません。

注意 3　API 関数 socket では，該当ソケットを “ ブロッキング・モード ” で生成しています。

戻り値

ENOERR	0x0	正常終了
EINVAL	0x16	パラメータの指定が不正です
ENETDOWN	0x5e	ネットワークがダウンしています

connect　　発行有効範囲：タスク

概要

接続の確立

C 言語形式

int　　connect (int *sd* , saddr **name* , int *name_len*) ;

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
I	saddr	**name* ;	アドレス構造体 sockaddr_in へのポインタ
I	int	*name_len* ;	*name* で指定されたアドレス構造体のサイズ (単位:バイト)

機能

sd で指定されたソケットと *name* で指定された遠隔側ソケットとの接続を確立します。

ただし，本 API 関数を発行した際，遠隔側ソケットにおいて " accept の発行 (接続要求の受付) " が行われていなかった場合には，遠隔側ソケットの待機ソケット・キューに接続要求がキューイングされます。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　本 API 関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，接続の確立処理を実行するうえで必要な条件が整っていなかった場合には，接続の確立処理は行わず，戻り値として EWOULDBLOCK を返しています。

注意 3　アドレス構造体 sockaddr_in についての詳細は，「**5.4.3　インターネット・アドレス用アドレス構造体**」を参照してください。

戻り値

ENOERR	0x0	正常終了
EINVAL	0x16	パラメータの指定が不正です
EWOULDBLOCK	0x50	対象ソケットがノンブロッキング・モードです
EAFNOSUPPORT	0x5b	対象ソケットのアドレス・ファミリが AF_INET ではありません
ETIMEDOUT	0x67	接続がタイムアウトしています
ECONNREFUSED	0x68	listen が発行されていません

shutdown　　発行有効範囲：タスク

概要

接続の遮断

C 言語形式

int　　shutdown (int *sd* , int *direction*) ;

パラメータ

I/O	パラメータ	説　　明
I	int　*sd* ;	ソケット記述子
I	int　*direction* ;	接続の遮断方向 0x0 ： 受信方向の接続を遮断 0x1 ： 送信方向の接続を遮断 0x2 ： 送受信両方向の接続を遮断

機能

sd で指定されたソケット (バーチャル・サーキット型 SOCK_STREAM) で確立している接続のうち，*direction* で指定された方向の接続を遮断します。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　本 API 関数を発行した際，対象ソケットの Linger オプションが ON であり，かつ，Linger タイムアウトが 0x0 以外の値であった場合には，接続の遮断処理は Linger タイムアウトで指定された時間が経過するまで遅延されます。

注意 3　本 API 関数では，ソケットの解放処理 (ソケット用各種リソースの解放) が行われません。このため，対象ソケットを解放する際には，API 関数 closesock の発行が必要となります。

戻り値

ENOERR	0x0	正常終了
EINVAL	0x16	パラメータの指定が不正です
EOPNOTSUPP	0x5a	対象ソケットのソケット・タイプが SOCK_STREAM ではありません

accept　　発行有効範囲：タスク

概要

接続要求の受け付け

C 言語形式

int　　accept (int *sd* , saddr **addrp* , int **addr_len* , int **errp*) ;

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
O	saddr	**name* ;	アドレス構造体 sockaddr_in へのポインタ
I, O	int	**name_len* ;	*name* で指定されたアドレス構造体のサイズ (単位：バイト) を格納した領域へのポインタ
O	int	**errp* ;	エラー・コードを格納する領域へのポインタ

機能

sd で指定されたソケットに対して発行されている遠隔側ソケットからの接続要求 (待機ソケット・キューの先頭にキューイングされている接続要求) を受け付けたのち，*name* で指定されたアドレス構造体 sockaddr_in に遠隔側ソケットの IP アドレス，ポート番号を格納します。

なお，本 API 関数の処理が正常終了した際には，" 接続要求が受け付けられた遠隔側ソケットのソケット記述子 " が戻り値として返されます。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　本 API 関数では，*name*，および，*name_len* に 0x0 (NULL ポインタ) が指定された場合には，接続要求の受け付け処理のみを行い，遠隔側ソケットの IP アドレス，ポート番号の格納処理は行いません。

注意 3　*errp* で指定された領域には，以下に示したエラー・コードが格納されます。

ENOERR	(0x0)	：正常終了
EINVAL	(0x16)	：パラメータの指定が不正です
EMFILE	(0x17)	：最大ソケット生成数を超えています
EWOULDBLOCK	(0x50)	：対象ソケットがノンブロッキング・モードです
EOPNOTSUPP	(0x5a)	：ソケット・タイプの指定が不正です
ENETDOWN	(0x5e)	：ネットワークがダウンしています

注意 4　本 API 関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，接続要求の受け付け処理を実行するうえで必要な条件が整っていなかった場合には，接続要求の受け付け処理は行わず，戻り値として ERR を，エラー・コードとして EWOULDBLOCK を返しています。

注意 5　アドレス構造体 sockaddr_in についての詳細は，「**5.4.3　インターネット・アドレス用アドレス構造体**」を参照してください。

戻り値

正常終了	接続要求が受け付けられた遠隔側ソケットのソケット記述子
異常終了	ERR　(-1)

reject　　発行有効範囲：タスク

概要

接続要求の削除

C 言語形式

int　　reject (int *sd*) ;

パラメータ

I/O	パラメータ	説　　明
I	int　　*sd* ;	ソケット記述子

機能

sd で指定されたソケットの待機ソケット・キューの先頭にキューイングされている接続要求を削除 (接続の確立拒否) します。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　本 API 関数では，削除要求のキューイングが行われません。このため，本 API 関数を発行した際，対象ソケットの待機ソケット・キューに接続要求がキューイングされていなかった場合には，戻り値として EINVAL を返しています。

戻り値

ENOERR	0x0	正常終了
EINVAL	0x16	パラメータの指定が不正です
EOPNOTSUPP	0x5a	対象ソケットのソケット・タイプが SOCK_STREAM ではありません

send　　発行有効範囲：タスク

概要

メッセージの送信

C 言語形式

```
int     send ( int sd , char *buf , int buf_len , int flags , int *errp ) ;
```

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
I	char	**buf* ;	送信するメッセージを格納した領域へのポインタ
I	int	*buf_len* ;	*buf* で指定された領域のサイズ (単位：バイト)
I	int	*flags* ;	送信制御フラグ MSG_OOB　：緊急データの送信 MSG_DONTROUTE　：ルーティング・テーブル未使用で送信 MSG_FDBROADCAST　：全二重ブロードキャストで送信 MSG_NONBLOCKING　：ノンブロッキング・モードで送信 MSG_BLOCKING　：ブロッキング・モードで送信
O	int	**errp* ;	エラー・コードを格納する領域へのポインタ

機能

sd で指定されたソケットと接続が確立している遠隔側ソケット (API 関数 connect，または，accept の発行により接続が確立した遠隔側ソケット) に対して *buf* で指定されたメッセージを *flags* で指定された方法で送信します。

なお，本 API 関数の処理が正常終了した際には，" 実際に送信したメッセージのサイズ (単位：バイト) " が戻り値として返されます。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　*errp* で指定された領域には，以下に示したエラー・コードが格納されます。

ENOERR	(0x0)	：正常終了
EINVAL	(0x16)	：パラメータの指定が不正です
EWOULDBLOCK	(0x50)	：対象ソケットがノンブロッキング・モードです
ENETDOWN	(0x5e)	：ネットワークがダウンしています
ENOBUFS	(0x63)	：メッセージのサイズが大きすぎます
ENOTCONN	(0x65)	：接続が確立していません
ESHUTDOWN	(0x66)	：shutdown の発行により接続が遮断されているため，メッセージを送信することができません

注意 3　本 API 関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，メッセージの送信処理を実行するうえで必要な条件が整っていなかった場合には，メッセージの送信処理は行わず，戻り値として ERR を，エラー・コードとして EWOULDBLOCK を返しています。

注意 4　本 API 関数を発行した際，対象ソケットがデータグラム型 SOCK_DGRAM であった場合には，送信可能なメッセージの最大サイズは 0x5c0 バイトとなります。
このため，メッセージを格納した領域のサイズ *buf_len* に 0x5c0 以上の値を指定した場合には，メッセージの送信処理は行わず，戻り値として ERR を，エラー・コードとして ENOBUFS を返しています。

戻り値

正常終了	**実際に送信したメッセージのサイズ (単位：バイト)**
異常終了	ERR　(-1)

sendto

発行有効範囲：タスク

概要

メッセージ (受信先アドレス情報を含む) の送信

C 言語形式

```
int     sendto ( int sd , char *buf , int buf_len , int flags , saddr *name , int name_len , int *errp ) ;
```

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
I	char	**buf* ;	送信するメッセージを格納した領域へのポインタ
I	int	*buf_len* ;	*buf* で指定された領域のサイズ (単位：バイト)
I	int	*flags* ;	送信制御フラグ MSG_OOB　：緊急データの送信 MSG_DONTROUTE　：ルーティング・テーブル未使用で送信 MSG_FDBROADCAST　：全二重ブロードキャストで送信 MSG_NONBLOCKING　：ノンブロッキング・モードで送信 MSG_BLOCKING　：ブロッキング・モードで送信
I	saddr	**name* ;	アドレス構造体 sockaddr_in へのポインタ
I	int	*name_len* ;	*name* で指定されたアドレス構造体のサイズ (単位:バイト)
O	int	**errp* ;	エラー・コードを格納する領域へのポインタ

機能

sd で指定されたソケットと接続が確立している遠隔側ソケット (*name* で指定された遠隔側ソケット) に対して *buf* で指定されたメッセージを *flags* で指定された方法で送信します。

なお，本 API 関数の処理が正常終了した際には，" 実際に送信したメッセージのサイズ (単位：バイト) " が戻り値として返されます。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　*errp* で指定された領域には，以下に示したエラー・コードが格納されます。

ENOERR	(0x0)	：正常終了
EINVAL	(0x16)	：パラメータの指定が不正です
EWOULDBLOCK	(0x50)	：対象ソケットがノンブロッキング・モードです
ENETDOWN	(0x5e)	：ネットワークがダウンしています
ENOBUFS	(0x63)	：メッセージのサイズが大きすぎます
ENOTCONN	(0x65)	：接続が確立していません
ESHUTDOWN	(0x66)	：shutdown の発行により接続が遮断されているため，メッセージを送信することができません

注意 3　本 API 関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，メッセージの送信処理を実行するうえで必要な条件が整っていなかった場合には，メッセージの送信処理は行わず，戻り値として ERR を，エラー・コードとして EWOULDBLOCK を返しています。

注意 4　本 API 関数を発行した際，対象ソケットがデータグラム型 SOCK_DGRAM であった場合には，送信可能なメッセージの最大サイズは 0x5c0 バイトとなります。

このため，メッセージを格納した領域のサイズ *buf_len* に 0x5c0 以上の値を指定した場合には，メッセージの送信処理は行わず，戻り値として ERR を，エラー・コードとして ENOBUFS を返しています。

注意 5　アドレス構造体 sockaddr_in についての詳細は，「**5.4.3　インターネット・アドレス用アドレス構造体**」を参照してください。

戻り値

正常終了	**実際に送信したメッセージのサイズ (単位：バイト)**
異常終了	ERR　(-1)

recv　　発行有効範囲：タスク

概要

メッセージの受信

C 言語形式

```
int     recv ( int sd , char *buf , int buf_len , int flags , int *errp ) ;
```

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
O	char	**buf* ;	受信したメッセージを格納する領域へのポインタ
I	int	*buf_len* ;	*buf* で指定された領域のサイズ (単位：バイト)
I	int	*flags* ;	受信制御フラグ MSG_PEEK　：データの受信 MSG_URGENT　：緊急データの受信 MSG_TRUNCATE　：受信パケットの切り捨て MSG_NONBLOCKING　：ノンブロッキング・モードで受信 MSG_BLOCKING　：ブロッキング・モードで受信
O	int	**errp* ;	エラー・コードを格納する領域へのポインタ

機能

sd で指定されたソケットと接続が確立している遠隔側ソケット (API 関数 connect，または， accept の発行により接続が確立した遠隔側ソケット) から *flags* で指定された方法でメッセージを受信し，*buf* で指定された領域に格納します。

なお，本 API 関数の処理が正常終了した際には，" 実際に受信したメッセージのサイズ (単位：バイト) " が戻り値として返されます。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　*errp* で指定された領域には，以下に示したエラー・コードが格納されます。

ENOERR	(0x0)	：正常終了
EINVAL	(0x16)	：パラメータの指定が不正です
EWOULDBLOCK	(0x50)	：対象ソケットがノンブロッキング・モードです
ENOTCONN	(0x65)	：接続が確立していません
ESHUTDOWN	(0x66)	：shutdown の発行により接続が遮断されているため，メッセージを受信することができません

注意 3　本 API 関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，メッセージの受信処理を実行するうえで必要な条件が整っていなかった場合には，メッセージの受信処理は行わず，戻り値として ERR を，エラー・コードとして EWOULDBLOCK を返しています。

注意 4　戻り値に 0x0 が設定された場合は，メッセージの受信処理が完了する以前に対象ソケットの解放処理が実行されたことを意味します。

戻り値

正常終了	実際に受信したメッセージのサイズ (単位：バイト)

異常終了	ERR　(-1)

recvfrom

発行有効範囲：タスク

概要

メッセージ (送信元アドレス情報を含む) の受信

C 言語形式

```
int     recvfrom ( int sd , char *buf , int buf_len , int flags , saddr *name , int *name_len , int *errp ) ;
```

パラメータ

I/O	パラメータ		説　　明
I	int	*sd* ;	ソケット記述子
O	char	**buf* ;	受信したメッセージを格納する領域へのポインタ
I	int	*buf_len* ;	*buf* で指定された領域のサイズ (単位：バイト)
I	int	*flags* ;	受信制御フラグ MSG_PEEK　：データの受信 MSG_URGENT　：緊急データの受信 MSG_TRUNCATE　：受信パケットの切り捨て MSG_NONBLOCKING　：ノンブロッキング・モードで受信 MSG_BLOCKING　：ブロッキング・モードで受信
O	saddr	**name* ;	アドレス構造体 sockaddr_in へのポインタ
I, O	int	**name_len* ;	*name* で指定されたアドレス構造体のサイズ (単位：バイト) を格納した領域へのポインタ
O	int	**errp* ;	エラー・コードを格納する領域へのポインタ

機能

sd で指定されたソケットと接続が確立している遠隔側ソケットから *flags* で指定された方法でメッセージを受信し，*buf* で指定された領域に格納します。

なお，*name* で指定されたアドレス構造体 sockaddr_in には，遠隔側ソケットの IP アドレス，ポート番が格納されます。

また，本 API 関数の処理が正常終了した際には，“ 実際に受信したメッセージのサイズ (単位：バイト) ”が戻り値として返されます。

注意 1　ソケット記述子 *sd* には，API 関数 socket の戻り値を設定します。

注意 2　*errp* で指定された領域には，以下に示したエラー・コードが格納されます。

ENOERR	(0x0)	：正常終了
EINVAL	(0x16)	：パラメータの指定が不正です
EWOULDBLOCK	(0x50)	：対象ソケットがノンブロッキング・モードです
ENOTCONN	(0x65)	：接続が確立していません
ESHUTDOWN	(0x66)	：shutdown の発行により接続が遮断されているため，メッセージを送信することができません

注意 3　本 API 関数を発行した際，対象ソケットがノンブロッキング・モードであり，かつ，メッセージの受信処理を実行するうえで必要な条件が整っていなかった場合には，メッセージの受信処理は行わず，戻り値として ERR を，エラー・コードとして EWOULDBLOCK を返しています。

注意 4　戻り値に 0x0 が設定された場合は，メッセージの受信処理が完了する以前に対象ソケットの解放処理が実行されたことを意味します。

注意 5　**アドレス構造体** sockaddr_in **についての詳細は，「5.4.3　インターネット・アドレス用アドレス構造体」を参照してください。**

戻り値

正常終了	**実際に受信したメッセージのサイズ** (**単位：バイト**)
異常終了	ERR　(-1)

nselect　　発行有効範囲：タスク

概要

イベントの選択

C 言語形式

```
int     nselect ( struct sel *selp , int cnt , u32 *waitp , int ( *pfil ) ( struct sel * , int , u32 , void * ) , void *arg , int *errp ) ;
```

パラメータ

I/O	パラメータ		説　　明
O	struct sel	*selp ;	選択構造体 sel の配列へのポインタ
I	int	cnt ;	選択構造体 sel の要素数
I, O	u32	*waitp ;	待ち時間 (単位：ミリ秒) を格納した領域へのポインタ
I	int	(*pfil) (struct sel * , int , u32 , void *) ;	システム予約領域 (usel_nilpfi 固定)
I	void	*arg ;	システム予約領域 (0x0 固定)
O	int	*errp ;	エラー・コードを格納する領域へのポインタ

【 選択構造体 sel の構造 】

```
typedef     struct      sel {
            u16         se_inflags ;        /* 要求イベントの論理和 OR */
            u16         se_outflags ;       /* 発生イベントの論理和 OR */
            i16         se_fd ;             /* Holds Process Local File ID */
            i16         se_1reserved ;      /* システム予約領域 */
            u32         se_user ;           /* ユーザ用拡張領域 */
            u32         se_2reserved ;      /* システム予約領域 */
} sel ;
```

機能

selp で指定された選択構造体 sel のメンバ se_inflags に設定されているイベント (要求イベントの論理和) のうち，どのイベントが発生していたのかを調べ，実際に発生していたイベント (発生イベントの論理和) をメンバ se_outflags に格納します。

ただし，本 API 関数を発行した際，メンバ se_inflags に設定されているイベントが 1 つも発生していなかった場合には，要求動作の実行が中断 (ブロック) されます。

なお，要求動作の実行再開は，メンバ se_inflags に設定されているイベントが 1 つでも発生した際，または，*waitp* で指定された待ち時間が経過した際に行われます。

以下に，プロトコル毎 (TCP プロトコル，UDP プロトコル) に要求イベントとして指定可能な値，および，発生イベントとして格納される値の一覧を示します。

マクロ	意味	TCP プロトコル	UDP プロトコル
READ_NOTIFY	メッセージの受信が可能	○	○
WRITE_NOTIFY	送信キューに空きがある	○	○

マクロ	意味	TCP プロトコル	UDP プロトコル
ACCEPT_NOTIFY	受動コネクションが確立	○	×
CLOSE_NOTIFY	ソケットの解放処理が正常終了	○	○
CONNECT_NOTIFY	能動コネクションが確立	○	×
EXCEPT_NOTIFY	例外が発生	○	○
RSHUTDOWN_NOTIFY	読み取り方向がピアにより遮断	○	×
TIMEOUT_NOTIFY	要求動作がタイムアウト	○	×
WSHUTDOWN_NOTIFY	書き込み方向が遮断	○	×
URGENT_NOTIFY	緊急データが利用可能	○	×
SENDQEMPTY_NOTIFY	送信キューが空	○	○

注意 1　*waitp* **に** 0x0 (NULL **ポインタ**)，**または，***waitp* **で指定された領域に** -1 **を設定した場合，**RX-NET(TCP/IP) **は " 永久待ち " が指定されたものとして処理を行います。**

注意 2　**要求動作の実行中断後，メンバ** se_inflags **に設定されているイベントが発生した際には，***waitp* **で指定された領域に " 待ち時間が経過するまでの残り時間 (単位：ミリ秒) " が設定されます。**

注意 3　*errp* **で指定された領域には，以下に示したエラー・コードが格納されます。**

ENOERR	(0x0)	**：正常終了**
EINVAL	(0x16)	**：パラメータの指定が不正です**
ENETDOWN	(0x5e)	**：ネットワークがダウンしています**

注意 4　**選択構造体** sel **についての詳細は，「5.4.7　選択構造体」を参照してください。**

戻り値

正常終了	**発生していたイベントの総数**
異常終了	ERR　(-1)

第 6 章　RX-NET(TCP/IP) 依存部

本章では，RX-NET(TCP/IP) が提供している RX-NET(TCP/IP) 依存部について解説しています。

6.1　概要

RX-NET(TCP/IP) では，RX-NET(TCP/IP) が提供する機能を実現する際に必要となる基本情報，および，ドライバ関数を RX-NET(TCP/IP) 依存部 (ユーザ・オウン・コーディング部) として切り出し，サンプル・ソース・ファイルを提供しています。
そこで，システム構築時には，これら基本情報，および，ドライバ関数をユーザの実行環境 / アプリケーション・システムにあわせてカスタマイズ化する必要があります。
以下に，RX-NET(TCP/IP) が RX-NET(TCP/IP) 依存部として切り出しているドライバ関数 (5 種類) を示します。

devname_init	devname_updown	devname_start	devname_Thread
devname_intr			

なお，ドライバ関数の呼び出しは，RX-NET(TCP/IP) の初期化処理，ネットワーク・インタフェースの起動処理 / 遮断処理などを実行した際，RX-NET(TCP/IP) により行われます。
したがって，ドライバ関数への入力パラメータに対する設定処理は RX-NET(TCP/IP) が行っています。

6.2　基本情報

RX-NET(TCP/IP) が提供する機能を実現する際に必要となる基本情報 (ヒープ情報，ソケット情報など) について以下に示します。

6.2.1　ヒープ情報

RX-NET(TCP/IP) では，RX-NET(TCP/IP) が提供しているネットワーク通信機能の内部処理で使用する作業領域 (ヒープ領域) のサイズを RX-NET(TCP/IP) 依存部として切り出しています。
このため，ユーザは，規定されたマクロ名を用いてヒープ情報を定義する必要があります。

```
/* ヒープ情報のマクロ定義 */
#define          HEAP_SIZE                    150000
```

注意 1　RX-NET(TCP/IP) では，ヒープ領域のサイズとして，150000 (単位：バイト) を推奨しています。

注意 2　RX-NET(TCP/IP) では，本ヒープ情報のサンプル・ソース・ファイルを提供しています。

【 CA850 対応版の場合 】
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

【 CCV850E 対応版の場合 】
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

6.2.2　ソケット情報

RX-NET(TCP/IP) では，ユーザのアプリケーション・プログラムで使用するソケットの最大生成数を RX-NET(TCP/IP) 依存部として切り出しています。
このため，ユーザは，規定されたマクロ名を用いてソケット情報を定義する必要があります。

```
/* ソケット情報のマクロ定義 */
#define          MAX_SOCKET                   10
```

注意 1　RX-NET(TCP/IP) **では，**1 **個のソケットを使用してネットワーク通信機能を実現しています。このため，ユーザが** API **関数**　socket **を発行して生成可能なソケットの総数は，“** MAX_SOCKET - 1 **”から算出された数となります。**

注意 2　RX-NET(TCP/IP) **では，本ソケット情報のサンプル・ソース・ファイルを提供しています。**

【 CA850 **対応版の場合 】**
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

【 CCV850E **対応版の場合 】**
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

6.2.3　イベント・テーブル情報

RX-NET(TCP/IP) **では，**RX-NET(TCP/IP) **が提供しているネットワーク通信機能の内部処理で使用する同期制御用状態保持テーブル (イベント・テーブル) の最大数を** RX-NET(TCP/IP) **依存部として切り出しています。**
このため，ユーザは，規定されたマクロ名を用いてイベント・テーブル情報を定義する必要があります。

```
/* イベント・テーブル情報のマクロ定義 */
#define         MAX_EVENT                   ( MAX_SOCKET + 15 )
```

注意 1　RX-NET(TCP/IP) **では，イベント・テーブルの最大数として，“** MAX_SOCKET + **ネットワーク通信を行うタスクの総数 ” から算出された値を推奨しています。**

注意 2　RX-NET(TCP/IP) **では，本イベント・テーブル情報のサンプル・ソース・ファイルを提供しています。**

【 CA850 **対応版の場合 】**
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

【 CCV850E **対応版の場合 】**
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

6.2.4　周期起動ハンドラ情報

RX-NET(TCP/IP) **では，**RX-NET(TCP/IP) **が提供しているネットワーク通信機能を実現する際に必要となる周期起動ハンドラの指定番号を** RX-NET(TCP/IP) **依存部として切り出しています。**
このため，ユーザは，規定されたマクロ名を用いて周期起動ハンドラ情報を定義する必要があります。

```
/* 周期起動ハンドラ情報のマクロ定義 */
#define         OS_CYC_NO                   0x1
```

注意　RX-NET(TCP/IP) **では，本周期起動ハンドラ情報のサンプル・ソース・ファイルを提供しています。**

【 CA850 **対応版の場合 】**
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

【 CCV850E **対応版の場合 】**
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

6.2.5　タスク情報

RX-NET(TCP/IP) **では，**RX-NET(TCP/IP) **が提供しているネットワーク通信機能を実現する際に必要となるタスクの優先度を** RX-NET(TCP/IP) **依存部として切り出しています。**
このため，ユーザは，規定されたマクロ名を用いてタスク情報を定義する必要があります。

```
/* タスク情報のマクロ定義 */
#define         OS_RCVTSK_PRI               0x1
```

注意 1　**タスクの優先度** OS_RCVTSK_PRI **には，システム内で最高優先度となる値を指定する必要があります。**

注意 2　RX-NET(TCP/IP) **では，本タスク情報のサンプル・ソース・ファイルを提供しています。**

【 CA850 対応版の場合 】
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

【 CCV850E 対応版の場合 】
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

6.2.6　デバイス・ドライバ・エントリ・テーブル情報

RX-NET(TCP/IP) では，RX-NET(TCP/IP) が提供しているネットワーク通信機能を実現する際に必要となるドライバ関数の登録処理を RX-NET(TCP/IP) 依存部として切り出しています。
このため，ユーザは，規定された変数名を用いてデバイス・ドライバ・エントリ・テーブル情報を定義する必要があります。

```
/* デバイス・ドライバ・エントリ・テーブル情報の定義 */
netdev          ndevsw [ ] = {
/*
                ネットワーク・デバイス名 ,
                システム予約領域 (0x0 固定 ) ,          システム予約領域 (0x0 固定 ) ,
                システム予約領域 (0x0 固定 ) ,          システム予約領域 (0x0 固定 ) ,
                初期化関数用第 1 パラメータ ,           初期化関数用第 2 パラメータ ,
                初期化関数用第 3 パラメータ ,           初期化関数用第 4 パラメータ ,
                システム予約領域 ,                      システム予約領域 ({ 0x0 } 固定 ) ,
                初期化関数名 ,                          開始 / 終了関数名 ,
                システム予約領域 ,                      送信関数名 ,
                システム予約領域
*/
        /************
        第 1 要素
        ************/
        {
                "im" ,
                0x0 ,                                   0x0 ,
                0x0 ,                                   0x0 ,
                0x0 ,                                   0x0 ,
                0x0 ,                                   0x0 ,
                { 0x0 } ,                               { 0x0 } ,
                noent_init ,                            noent_updown ,
                im_send ,                               noent_start ,
                noent_ioctl
        } ,
        /************
        第 2 要素
        ************/
        {
                "devname" ,
                0x0 ,                                   0x0 ,
                0x0 ,                                   0x0 ,
                devname_init_P0 ,                       devname_init_P1 ,
                devname_init_P2 ,                       devname_init_P3 ,
                { AF_ETHER } ,                          { 0x0 } ,
                devname_init ,                          devname_updown ,
                en_scomm ,                              devname_start ,
                devname_ioctl
        }
} ;
```

注意 1　第 1 要素については，RX-NET(TCP/IP) が提供しているネットワーク通信機能を実現する際に必要となる内部処理関数の登録処理が記述されています。このため，第 1 要素の内容を変更することは禁止されています。

注意 2　RX-NET(TCP/IP) では，本タスク情報のサンプル・ソース・ファイルを提供しています。

【 CA850 対応版の場合 】
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

【 CCV850E 対応版の場合 】
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

6.2.7　生存時間情報

RX-NET(TCP/IP) では，API 関数　socket で行われる " ソケット・オプション (生存時間の最大値，および，最小値) の初期化 " で用いられる値を RX-NET(TCP/IP) 依存部として切り出しています。
このため，ユーザは，規定されたマクロ名を用いて生存時間情報を定義する必要があります。

```
/* 生存時間情報のマクロ定義 */
#define         MAXTTL                  0xff
#define         MINTTL                  0x1
```

注意 1　RX-NET(TCP/IP) では,生存時間の最大値,および,最小値として指定可能な値を 0x0 ～ 0xff に限定しています。

注意 2　RX-NET(TCP/IP) では，本生存時間情報のサンプル・ソース・ファイルを提供しています。

【 CA850 対応版の場合 】
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

【 CCV850E 対応版の場合 】
nectools32\src\netconf\<*conf_name*>\src\rxnetconf.c

6.3　データ・マクロ

RX-NET(TCP/IP) が RX-NET(TCP/IP) 依存部として切り出しているドライバ関数を発行する際に使用する各種データ・マクロ (データ・タイプ，戻り値など) について以下に示します。

6.3.1　データ・タイプ

表 6-1 に，ドライバ関数を発行する際に指定する各種パラメータのデータ・タイプ一覧を示します。
なお，データ・タイプのマクロ定義は，RX850 Pro が提供している標準ヘッダ・ファイル nectools32\inc850\stdrx85p.h から呼び出されるヘッダ・ファイル nectools32\inc850\types.h，または，types.inc で行われています。

表 6-1　データ・タイプ

マクロ	型	意味
INT	int	符号付き 32 ビット整数
ID	short	管理オブジェクトの ID 番号

6.3.2　戻り値

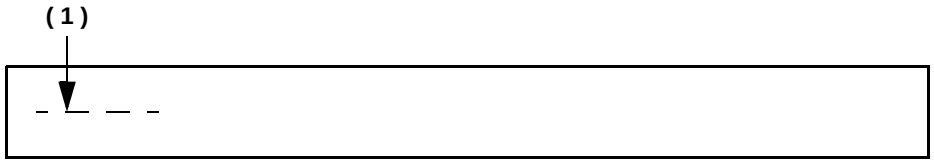
表 6-2 に，ドライバ関数からの戻り値一覧を示します。
なお，戻り値のマクロ定義は，標準ヘッダ・ファイル nectools32\inc850\rxnet.h から呼び出されるヘッダ・ファイル nectools32\inc850\rxnet\nerrno.h，および，RX850 Pro が提供している標準ヘッダ・ファイル nectools32\inc850\stdrx85p.h から呼び出されるヘッダ・ファイル nectools32\inc850\option.h，または，option.inc で行われています。

表 6-2　戻り値

マクロ	数値	意味
ENOERR	0x0	正常終了
TSK_NULL	0x0	正常終了

6.4　ドライバ関数解説

次項から RX-NET(TCP/IP) が提供しているドライバ関数について，以下の記述フォーマットに従って解説します。



(2) → 概要

(3) → C 言語形式

(4) → パラメータ

I/O	パラメータ	説　　明

(5) → 機能

(6) → 戻り値

（1） 名称

ドライバ関数の名称を示しています。

（2） 概要

ドライバ関数の機能概要を示しています。

（3） C 言語形式

ドライバ関数を RX-NET(TCP/IP) **から呼び出す際の記述形式を示しています。**

（4） パラメータ

ドライバ関数のパラメータを以下の形式で示しています。

I/O	パラメータ	説　　明
A	B	C

A ）**パラメータの種類**

I　…　**ドライバ関数への入力パラメータ**

O　…　**ドライバ関数からの出力パラメータ**

B ）**パラメータのデータ・タイプ**

C ）**パラメータの説明**

（5） 機能

ドライバ関数の機能詳細を示しています。

（6） 戻り値

ドライバ関数からの戻り値をデータ・マクロ，および，数値で示しています。

6.4.1　外部インタフェース仕様

表 6-3 に，RX-NET(TCP/IP) が提供しているドライバ関数の一覧を示します。

表 6-3　ドライバ関数

ドライバ関数名	機能概要
devname_init	so_initialize を発行した際に呼び出される初期化関数
devname_updown	ll_config，または，ll_unconfig を発行した際に呼び出される開始 / 終了関数
devname_start	RX-NET(TCP/IP) が送信処理を実行した際に呼び出される送信関数
devname_Thread	ネットワーク割り込みハンドラから呼び出されるネットワーク・タスク
devname_intr	受信割り込み，送信完了割り込み，エラー割り込みが発生した際に呼び出されるネットワーク割り込みハンドラ

次頁以降に，各種ドライバ関数の外部インタフェース仕様詳細を示します。

*devname*_init

概要

so_initialize を発行した際に呼び出される初期化関数

C 言語形式

int　　*devname*_init (netdev **ndp*) ;

パラメータ

I/O	パラメータ	説　　明
I	netdev　　**ndp* ;	デバイス・ドライバ・エントリ・テーブル情報を格納した領域へのポインタ

機能

RX-NET(TCP/IP) が提供している API 関数 so_initialize を発行した際に呼び出される初期化関数です。
以下に，本ドライバ関数で実行すべき処理を示します。

- ネットワーク・コントローラが存在しているか否かの確認
- MAC アドレスをネットワーク・コントローラから獲得
- 獲得した MAC アドレスを RX-NET(TCP/IP) に通知
- ネットワーク・タスク devname_Thread の生成 / 起動
- 該当処理が正常終了したか否かを本ドライバ関数の戻り値として設定

注意 1　デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「**6.2.6　デバイス・ドライバ・エントリ・テーブル情報**」を参照してください。

注意 2　RX-NET(TCP/IP) では，本ドライバ関数のサンプル・ソース・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\src\rxnet\drivers\<*driver_name*>*devname*api.c

【CCV850E 対応版の場合】
nectools32\src\rxnet\drivers\<*driver_name*>*devname*api.c

戻り値

ENOERR	0x0	正常終了
その他		異常終了

*devname*_updown

概要

ll_config，または，ll_unconfig を発行した際に呼び出される開始 / 終了関数

C 言語形式

int　　*devname*_updown (netdev **ndp* , u16 *flags*) ;

パラメータ

I/O	パラメータ		説　　明
I	netdev	**ndp* ;	デバイス・ドライバ・エントリ・テーブル情報を格納した領域へのポインタ
I	u16	*flags* ;	呼び出し元判定フラグ 0x0 ： ll_config からの呼び出し 0x1 ： ll_unconfig からの呼び出し

機能

RX-NET(TCP/IP)が提供しているAPI関数ll_config，または，ll_unconfigを発行した際に呼び出される開始/終了関数です。以下に，本ドライバ関数で実行すべき処理を示します。

【 ll_config から呼び出された場合 】

- ネットワーク・コントローラの初期化 (動作開始)
- ネットワーク割り込みハンドラ devname_intr の登録
- マスカブル割り込みの許可
- 該当処理が正常終了したか否かを本ドライバ関数の戻り値として設定

【 ll_unconfig から呼び出された場合 】

- マスカブル割り込みの禁止
- ネットワーク・コントローラの動作停止
- ネットワーク割り込みハンドラ devname_intr の登録解除
- 該当処理が正常終了したか否かを本ドライバ関数の戻り値として設定

注意 1　デバイス・ドライバ・エントリ・テーブル情報 ndevsw についての詳細は，「**6.2.6　デバイス・ドライバ・エントリ・テーブル情報**」を参照してください。

注意 2　RX-NET(TCP/IP) では，本ドライバ関数のサンプル・ソース・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\src\rxnet\drivers\<*driver_name*>*devname*api.c

【CCV850E 対応版の場合】
nectools32\src\rxnet\drivers\<*driver_name*>*devname*api.c

戻り値

ENOERR	0x0	正常終了
その他		異常終了

*devname*_start

概要

RX-NET(TCP/IP) が送信処理を実行した際に呼び出される送信関数

C 言語形式

int　　*devname*_start (struct m **mp*) ;

パラメータ

I/O	パラメータ	説　　明
I	struct　m　　**mp* ;	送信データ構造体へのポインタ

機能

RX-NET(TCP/IP) が送信処理を実行した際に呼び出される送信関数です。
以下に，本ドライバ関数で実行すべき処理を示します。

- *mp* で指定されたパケットの送信
- ENOERR を本ドライバ関数の戻り値として設定

注意　RX-NET(TCP/IP) では，本ドライバ関数のサンプル・ソース・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\src\rxnet\drivers\<*driver_name*>*devname*api.c

【CCV850E 対応版の場合】
nectools32\src\rxnet\drivers\<*driver_name*>*devname*api.c

戻り値

ENOERR	0x0	正常終了

*devname*_Thread

概要

ネットワーク割り込みハンドラから呼び出されるネットワーク・タスク

C 言語形式

void　*devname*_Thread (INT *stacd*) ;

パラメータ

I/O	パラメータ	説　明
I	INT *stacd* ;	システム予約領域

機能

ネットワーク割り込みハンドラ　devname_intr から呼び出されるネットワーク・タスクです。
以下に，本ドライバ関数で実行すべき処理を示します。

【 受信割り込み発生時に呼び出された場合 】

- 受信したパケットをネットワーク・コントローラから獲得
- 獲得したパケットを RX-NET(TCP/IP) に通知
- ネットワーク・タスク (自タスク) を wait 状態 (起床待ち状態) へと遷移

【 送信完了割り込み発生時に呼び出された場合 】

- RX-NET(TCP/IP) が提供しているネットワーク通信機能の内部処理 (送信処理) の呼び出し
- 後続の送信を催促
- ネットワーク・タスク (自タスク) を wait 状態 (起床待ち状態) へと遷移

【 エラー割り込み発生時に呼び出された場合 】

- エラーの発生要因に対応した処理
- ネットワーク・タスク (自タスク) を wait 状態 (起床待ち状態) へと遷移

注意　RX-NET(TCP/IP) では，本ドライバ関数のサンプル・ソース・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\src\rxnet\drivers\<*driver_name*>*devname*isrsub.c

【CCV850E 対応版の場合】
nectools32\src\rxnet\drivers\<*driver_name*>*devname*isrsub.c

戻り値

なし

*devname*_intr

概要

受信割り込み，送信完了割り込み，エラー割り込みが発生した際に呼び出されるネットワーク割り込みハンドラ

C 言語形式

ID　*devname*_intr (void) ;

パラメータ

なし

機能

受信割り込み，送信完了割り込み，エラー割り込みが発生した際に呼び出されるネットワーク割り込みハンドラです。
以下に，本ドライバ関数で実行すべき処理を示します。

- ネットワーク・コントローラ内の割り込みマスク操作 (禁止)
- ネットワーク・タスク devname_Thread に対する起床要求の発行
- プロセッサの割り込み終了処理
- TSK_NULL を本ドライバ関数の戻り値として設定

注意　RX-NET(TCP/IP) では，本ドライバ関数のサンプル・ソース・ファイルを提供しています。

【CA850 対応版の場合】
nectools32\src\rxnet\drivers\<*driver_name*>*devname*api.c

【CCV850E 対応版の場合】
nectools32\src\rxnet\drivers\<*driver_name*>*devname*api.c

戻り値

TSK_NULL	0x0	正常終了

索引

S

T

あ

い

え

お

か

き

こ

さ

し

せ

そ

た

ち

て

と

ね

は

ひ

ふ

め

も

り

ろ

わ

お問い合わせ先

【技術的なお問い合わせ先】

ＮＥＣ半導体テクニカルホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）

電　話　：044−435−9494
FAX　：044−435−9608
E-mail　：info@lsi.nec.co.jp

【営業関係お問い合わせ先】

システムLSI第一営業事業部
東　京　(03)3798−6106, 6107, 6108, 6155
大　阪　(06)6945−3178, 3200, 3208
名古屋　(052)222−2375
仙　台　(022)267−8740
水　戸　(029)226−1702
広　島　(082)242−5504
鳥　取　(0857)27−5313
松　山　(089)945−4149

システムLSI第二営業事業部
東　京　(03)3798−6110, 6111, 6112, 6151, 6156
名古屋　(052)222−2170, 2190
松　本　(0263)35−1662
前　橋　(027)243−6060
立　川　(042)526−5981
静　岡　(054)254−4794
金　沢　(076)232−7303
福　岡　(092)261−2806

【資料の請求先】

上記営業関係お問い合わせ先またはNEC特約店へお申しつけください。

【NECエレクトロンデバイス　ホームページ】

NECエレクトロンデバイスの情報がインターネットでご覧になれます。　URL（アドレス）　http://www.ic.nec.co.jp/

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］RX-NET ネットワーク・ライブラリ TCP/IP Ver.1.30 ユーザーズ・マニュアル
（U15083JJ4V0UM00（第4版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）（　　）
ご住所（　　）
お電話番号（　　）
お仕事の内容（　　）
お名前（　　）

1．ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（　　）					
（　　）					

2．わかりやすい所（第　章，第　章，第　章，第　章，その他　　）

理由［　　］

3．わかりにくい所（第　章，第　章，第　章，第　章，その他　　）

理由［　　］

4．ご意見，ご要望

5．このドキュメントをお届けしたのは
NEC販売員，特約店販売員，その他（　　）

ご協力ありがとうございました。
下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

日本電気(株) NEC エレクトロンデバイス
半導体テクニカルホットライン
FAX：（044）435-9608

2000.6

キリトリ