NEC

ユーザーズ・マニュアル

RX850V4 Ver.4.41

リアルタイム・オペレーティング・システム

コーディング編（CubeSuite Ver.1.20）

対象ツール

RX850V4 Ver.4.41

資料番号　U20044JJ1V0UM00（第1版）
発行年月　February 2010

〔メ　モ〕

目次要約

WindowsおよびWindows Vista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。

TRON は，"The Real-time Operating system Nucleus" の略称です。

ITRON は，"Industrial TRON" の略称です。

μITRON は，"Micro Industrial TRON" の略称です。

TRON, ITRON, およびμITRON は，特定の商品ないし商品群を指すものではありません。

μITRON4.0 仕様は，(社)トロン協会が策定したオープンなリアルタイムカーネル仕様です。

μITRON4.0 仕様の仕様書は，(社)トロン協会 Web サイト（http://www.assoc.tron.org/）から入手が可能です。

μITRON仕様の著作権は(社)トロン協会に属しています。

・本資料に記載されている内容は2010年2月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。

・文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。

・当社は，本資料に記載された当社製品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，一切その責を負いません。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。

・本資料に記載された回路，ソフトウエ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，ソフトウエ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，一切その責を負いません。

・当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。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。当社製品をお客様の機器にご使用の際には，当社製品の不具合の結果として，生命，身体および財産に対する損害や社会的損害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を行ってください。

・当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「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。

「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

「特別水準」：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

「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，生命維持のための装置またはシステム等

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。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

注1. 本事項において使用されている「当社」とは，NECエレクトロニクス株式会社およびNEC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う。

注2. 本事項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，製造製品をいう。

（M8E0909J）

〔メ　モ〕

はじめに

対 象 者　このマニュアルはV850マイクロコントローラの各製品の応用システムを設計，開発するユーザを対象としています。

目　　的　このマニュアルは，RX850V4のコーディング機能とその操作方法を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
- システム構築
- タスク管理機能
- タスク付属同期機能
- タスク例外処理機能
- 同期通信機能
- 拡張同期通信機能
- メモリ・プール管理機能
- 時間管理機能
- システム状態管理機能
- 割り込み管理機能
- サービス・コール管理機能
- システム構成管理機能
- スケジューリング機能
- システム初期化処理
- データ・タイプとマクロ
- サービス・コール
- システム・コンフィギュレーション・ファイル
- コンフィギュレータCF850V4

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，アセンブラの一般知識を必要とします。

V850マイクロコントローラのハードウエア機能を知りたいとき
→各製品の**ユーザーズ・マニュアル　ハードウエア編**を参照してください。

V850マイクロコントローラの命令機能を知りたいとき
→V850ES　**ユーザーズ・マニュアル　アーキテクチャ編**（U15943J）または
　V850E1　**ユーザーズ・マニュアル　アーキテクチャ編**（U14559J）を参照してください。

凡　　例

データ表記の重み	：左が上位桁，右が下位桁
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XXXXB
	10進数…XXXX
	16進数…0xXXXX

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）　$2^{20} = 1024^2$

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）

資　料　名		資料番号	
		和　文	英　文
RXシリーズ	起動編（CubeSuite Ver.1.20）	U20041J	U20041E
	メッセージ編（CubeSuite Ver.1.20）	U20042J	U20042E
RX850V4 Ver.4.41	コーディング編（CubeSuite Ver.1.20）	このマニュアル	U20044E
	デバッグ編（CubeSuite Ver.1.20）	U20045J	U20045E
	解析編（CubeSuite）	U19439J	U19439E
	内部構造編（CubeSuite Ver.1.20）	U20046J	U20046E
CubeSuite統合開発環境	起動編	U19809J	U19809E
	解析編	U19816J	U19816E
	プログラミング編	U19390J	U19390E
	メッセージ編	U19810J	U19810E
	コーディング編（CXコンパイラ）	U19811J	U19811E
	ビルド編（CXコンパイラ）	U19812J	U19812E
	V850　コーディング編	U19383J	U19383E
	V850　ビルド編	U19386J	U19386E
	V850　デバッグ編	U19815J	U19815E
	V850　設計編	U20184J	U20184E

注意　上記関連資料は，予告なしに内容を変更することがあります。設計などには，必ず最新の資料を使用してください。

目　　次

図の目次

表の目次

第 1 章　概　　説

1.1　概　　要

RX850V4 は，効率のよいリアルタイム処理環境，および，マルチタスク処理環境を提供するとともに，対象 CPU の制御機器分野における応用範囲を拡大することを目的として開発された“リアルタイム・マルチタスク OS”です。

また，実行環境に組み込んで使用することを前提として開発されているため，ROM 化を意識し，コンパクトな設計が行われています。

1.1.1　リアルタイム OS

制御機器分野におけるシステムでは，内外の事象変化に対するリアルタイム性が要求されます。しかし，従来のシステムでは，このような要求をユーザが用意した単純な割り込み処理で対処してきたため，制御機器が高性能化，多様化するにつれ，単純な割り込み処理だけでの対処が困難になってきています。

つまり，処理プログラム量の増大，システムの複雑化により，内外の事象変化に対する処理を“どのような順序で実行させるのか”を管理することが煩雑になってきたといえます。

そこで，このような問題を解決するために考えられたのが“リアルタイム OS”です。

リアルタイム OS は，内外の事象変化に対するリアルタイム性を保証するとともに，最適な処理プログラムを最適な順序で実行させることを主な目的（仕事）としています。

1.1.2　マルチタスク OS

OS の世界では，OS の管理下で実行する処理プログラムを“タスク”，1 つのプロセッサ上で複数のタスクを同時実行させることを“マルチタスキング”と呼んでいます。

しかし，厳密にはプロセッサ自体は一度に 1 つのタスク（命令）しか実行することができないため，タスクの実行を何らかの基準（きっかけ）を利用して非常に短い間隔で切り替えることにより，疑似的に複数のタスクが同時実行しているかのように見せています。

このように，システム内で規定されている何らかの基準を利用してタスクを切り替え，タスクの並列処理を可能としたのが“マルチタスク OS”です。

マルチタスク OS は，複数のタスクを並列実行させることにより，システム全体の処理能力を向上させることを主な目的（仕事）としています。

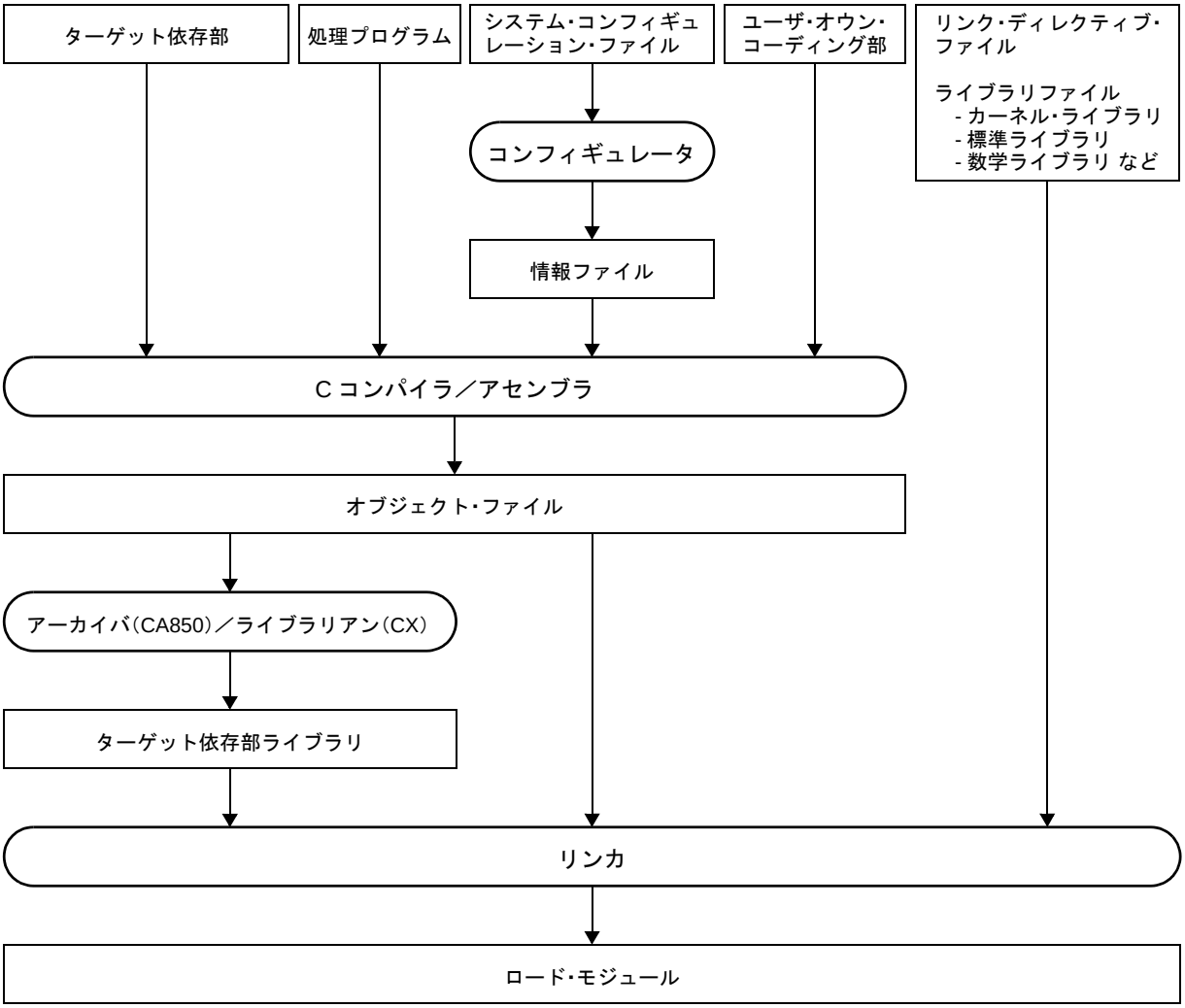
第 2 章　システム構築

本章では，RX850V4 が提供している機能を利用したシステム（ロード・モジュール）の構築手順について解説しています。

2.1　概　　要

システム構築とは，RX850V4 の提供媒体からユーザの開発環境（ホスト・マシン）上にインストールされたファイル群（カーネル・ライブラリなど）を用いてロード・モジュールを生成することです。
以下に，システム構築の手順を示します。

図２－１　システム構築の手順



RX850V4 では，ロード・モジュールを生成する際に必要となるファイル群のサンプル・プログラムを提供しています。
サンプル・プログラムは，以下のフォルダに格納されています。

<rx_sample>=<CubeSuite_root>¥SampleProjects¥V850¥ *デバイス名*△*種別*△ (*コンパイラ名*) △ V*x.xx*¥appli

- <CubeSuite_root>

  CubeSuite のインストール・フォルダを表しています。
  デフォルトでは，“ C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite”となります。

- SampleProjects

  CubeSuite のサンプル・プロジェクト・フォルダを表しています。

- V850

  V850 のサンプル・プロジェクト・フォルダを表しています。

- *デバイス名*△*種別*△ (*コンパイラ名*) △ V*x.xx*

  RX850V4 のサンプル・プロジェクト・フォルダを表しています。

  デバイス名 :　サンプルを提供しているデバイス名を表しています。
  ただし，フォルダ名に“/”を使用することができないため，デバイス名に“/”が含まれている場合は，“_”に置き換えられています。

  △ :　スペースを表しています。

  種別 :　サンプル・プログラムの種別を表しています。

  コンパイラ名 :　コンパイラ・パッケージ名（CA850，または CX）を表しています。

  V*x.xx* :　RX850V4 のサンプル・プロジェクトのバージョンを表しています。

- appli

  RX850V4 が提供しているサンプル・プログラムが格納されているフォルダを表しています。

2.2　ターゲット依存部の記述

RX850V4 では，さまざまな実行環境に対応するために，RX850V4 が処理を実行するうえで必要となるハードウエア依存処理をターゲット依存部として切り出しています。これにより，さまざまな実行環境への移植性を向上させるとともに，カスタマイズを容易なものとしています。

以下に，機能別に切り出されているターゲット依存部の一覧を示します。

- タスク管理機能
  - オーバフロー後処理
    オーバフロー後処理は，RX850V4，および，処理プログラムが処理を実行するうえで必要となるスタックがオーバフローした際の後処理を実行するためにターゲット依存部として切り出された後処理専用ルーチン（関数名：_kernel_stk_overflow）であり，スタックがオーバフローした際に RX850V4 から呼び出されます。

- 割り込み管理機能
  - サービス・コール dis_int
    マスカブル割り込みの受け付けを許可状態から禁止状態へと変更するためにターゲット依存部として切り出されたマスカブル割り込みの受け付け操作処理専用ルーチン（関数名：_kernel_usr_dis_int）であり，処理プログラムからサービス・コール dis_int が発行された際に呼び出されます。
  - サービス・コール ena_int
    マスカブル割り込みの受け付けを禁止状態から許可状態へと変更するためにターゲット依存部として切り出されたマスカブル割り込みの受け付け操作処理専用ルーチン（関数名：_kernel_usr_ena_int）であり，処理プログラムからサービス・コール ena_int が発行された際に呼び出されます。
  - 割り込みマスク設定処理（上書き設定）
    割り込みマスク設定処理（上書き設定）は，該当ユーザ・オウン関数のパラメータで指定された割り込みマスク・パターンを割り込み制御レジスタ xxICn，または，割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に設定するためにターゲット依存部として切り出された割り込みマスク・パターン設定処理専用ルーチン（関数名：_kernel_usr_set_intmsk）であり，処理プログラムからサービス・コール unl_cpu，iunl_cpu，chg_ims，ichg_ims が発行された際に呼び出されます。
  - 割り込みマスク設定処理（OR 設定）
    割り込みマスク設定処理（OR 設定）は，該当ユーザ・オウン関数のパラメータで指定された割り込みマスク・パターンと CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または，割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）の論理和 OR をとり，その結果を対象レジスタの割り込みマスク・フラグ xxMKn に設定するためにターゲット依存部として切り出された割り込みマスク・パターン設定処理専用ルーチン（関数名：_kernel_usr_msk_intmsk）であり，処理プログラムからサービス・コール loc_cpu，iloc_cpu が発行された際に呼び出されます。
  - 割り込みマスク獲得処理
    割り込みマスク獲得処理は，該当ユーザ・オウン関数のパラメータに CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または，割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）を格納するためにターゲット依存部として切り出された割り込みマスク・パターンの獲得処理専用ルーチン（関数名：_kernel_usr_get_intmsk）であり，処理プログラムからサービス・コール loc_cpu，iloc_cpu，get_ims，iget_ims が発行された際に呼び出されます。

備考 1　ターゲット依存部についての詳細は，「第 3 章　タスク管理機能」，「第 11 章　割り込み管理機能」を参照してください。

備考 2　RX850V4 では，ターゲット依存部のサンプル・ソース・ファイルを提供しています。

<rx_sample>¥src¥usr_disint.c，usr_enaint.c，usr_getmsk.c，usr_intmsk.c，usr_setmsk.c，usr_stkover.s

2.2.1　ターゲット依存部ライブラリの生成

「2.2　ターゲット依存部の記述」で作成された C 言語ソース・ファイル，および，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／アーカイバ（CA850），または C コンパイラ／アセンブラ／ライブラリアン（CX）を実行し，ライブラリ・ファイル（ターゲット依存部ライブラリ）を生成します。
以下に，ターゲット依存部ライブラリを生成する際に必要となるファイル群の一覧を示します。

- オーバフロー後処理
- サービス・コール dis_int
- サービス・コール ena_int
- 割り込みマスク設定処理（上書き設定）
- 割り込みマスク設定処理（OR 設定）
- 割り込みマスク獲得処理

備考　C コンパイラ／アセンブラ／アーカイバ（CA850）についての詳細は「CubeSuite V850 ビルド編」を，C コンパイラ／アセンブラ／ライブラリアン（CX）についての詳細は「CubuSuite ビルド編（CX コンパイラ）」のユーザーズ・マニュアルを参照してください。

2.3　処理プログラムの記述

システムとして実現すべき処理を記述します。
なお，RX850V4 では，処理プログラムを実現すべき処理の種類，および，用途にあわせて以下に示した 7 種類に分類しています。

- タ ス ク
  他処理プログラム（周期ハンドラ，割り込みハンドラなど）とは異なり，RX850V4 が提供するサービス・コールを使用して明示的に操作しないかぎり実行されることのない処理プログラムです。

- タスク例外処理ルーチン
  タスク例外処理要求が発行された際に起動されるタスク例外処理専用ルーチンです。
  なお，RX850V4 では，タスク例外処理ルーチンを“タスク例外処理要求を発行されたタスクの延長線”として位置づけています。このため，タスク例外処理ルーチンの起動タイミングは，タスク例外処理要求を発行されたタスクが RUNNING 状態へと遷移した際となります。

- 周期ハンドラ
  一定の時間（初期起動位相，または起動周期）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，RX850V4 では，周期ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。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処理を実行中であっても，その処理は中断され，周期ハンドラに制御が移ります。

- 割り込みハンドラ
  割り込みが発生した際に起動される割り込み処理専用ルーチンです。
  なお，RX850V4 では，割り込みハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。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処理を実行中であっても，その処理は中断され，割り込みハンドラに制御が移ります。
  なお，割り込みハンドラは，“直接起動割り込みハンドラ”とは異なり，割り込みが発生した際，RX850V4 が提供している割り込み前処理（レジスタの退避／復帰，スタックの切り替えなど）を経由して処理が実行されます。このため，“直接起動割り込みハンドラ”と比較して，その処理内容を簡素化することができます。

- 直接起動割り込みハンドラ
  割り込みが発生した際に起動される割り込み処理専用ルーチンです。
  なお，RX850V4 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）”として位置づけています。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処理を実行中であっても，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。
  なお，直接起動割り込みハンドラは，“割り込みハンドラ”とは異なり，割り込みが発生した際，RX850V4 が介在することなく CPU が強制的に制御を移すハンドラ・アドレスから呼び出されます。このため，ハードウエアの限界に近い応答性を期待することができます。

- 拡張サービス・コール・ルーチン
  ユーザ定義の関数を RX850V4 に登録したものであり，RX850V4 が提供するサービス・コールを使用して明示的に呼び出さないかぎり実行されることのない処理プログラムです。
  なお，RX850V4 では，拡張サービス・コール・ルーチンを“拡張サービス・コール・ルーチンを呼び出した処理プログラムの延長線”として位置づけています。

- CPU 例外ハンドラ
  CPU 例外が発生した際に起動される CPU 例外処理専用ルーチンです。
  なお，RX850V4 では，CPU 例外ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため，CPU 例外が発生した際には，システム内で最高優先度を持つタスクが処理を実行中であっても，その処理は中断され，CPU 例外ハンドラに制御が移ります。

備考 1　処理プログラムについての詳細は，「第 3 章　タスク管理機能」，「第 5 章　タスク例外処理機能」，「第 9 章　時間管理機能」，「第 11 章　割り込み管理機能」，「第 12 章　サービス・コール管理機能」，「第 13 章　システム構成管理機能」を参照してください。

備考 2　RX850V4 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。

<rx_sample>¥src¥task.c，texrtn.c，cychdr.c，inthdr.c，exchdr.c

2.4　システム・コンフィギュレーション・ファイルの記述

RX850V4 に提供するデータを保持した情報ファイル（システム情報テーブル・ファイル，システム情報ヘッダ・ファイル，エントリ・ファイル）を生成する際に必要となるシステム・コンフィギュレーション・ファイルを記述します。

備考 1　システム・コンフィギュレーション・ファイルについての詳細は，「第 18 章　システム・コンフィギュレーション・ファイル」を参照してください。

備考 2　RX850V4 では，システム･コンフィギュレーション･ファイルのサンプル･ソース･ファイルを提供しています。

<rx_sample>¥src¥sys.cfg

2.5　ユーザ・オウン・コーディング部の記述

RX850V4 では，さまざまな実行環境に対応するために，RX850V4 が処理を実行するうえで必要となるハードウエア依存処理をユーザ・オウン・コーディング部として切り出しています。これにより，さまざまな実行環境への移植性を向上させるとともに，カスタマイズを容易なものとしています。

以下に，機能別に切り出されているユーザ・オウン・コーディング部の一覧を示します。

- 割り込み管理機能
  - 割り込みエントリ処理
    割り込みが発生した際に CPU 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（割り込み前処理，直接起動割り込みハンドラなど）への分岐処理を割り付けるためにユーザ・オウン・コーディング部として切り出されたエントリ処理専用ルーチンです。
    なお，コンフィギュレーション時に割り込みハンドラ情報として定義された割り込みハンドラの割り込みエントリ処理は，コンフィギュレーション時に作成したシステム・コンフィギュレーション・ファイルに対してコンフィギュレータを実行することにより生成されるエントリ・ファイルに内包されています。したがって，割り込みエントリ処理をカスタマイズする必要がない場合には，該当エントリ・ファイルを利用することにより，割り込みエントリ処理の記述が不要となります。

- システム構成管理機能
  - CPU 例外エントリ処理
    CPU 例外が発生した際に CPU 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（CPU 例外前処理，ブート処理など）への分岐処理を割り付けるためにユーザ・オウン・コーディング部として切り出されたエントリ処理専用ルーチンです。
    なお，コンフィギュレーション時に CPU 例外ハンドラ情報として定義された CPU 例外ハンドラの CPU 例外エントリ処理は，コンフィギュレーション時に作成したシステム・コンフィギュレーション・ファイルに対してコンフィギュレータを実行することにより生成されるエントリ・ファイルに内包されています。したがって，CPU 例外エントリ処理をカスタマイズする必要がない場合には，該当エントリ・ファイルを利用することにより，CPU 例外エントリ処理の記述が不要となります。
  - 初期化ルーチン
    ユーザの実行環境に依存したハードウエア（周辺コントローラなど）を初期化するためにユーザ・オウン・コーディング部として切り出された初期化処理専用ルーチンであり，カーネル初期化部から呼び出されます。

- スケジューリング機能
  - アイドル・ルーチン
    CPU が提供しているスタンバイ機能を有効活用（低消費電力システムの実現）するためにユーザ・オウン・コーディング部として切り出されたアイドル処理専用ルーチンであり，RX850V4 のスケジューリング対象となるタスク（RUNNING 状態，または READY 状態のタスク）がシステム内に 1 つも存在しなくなった際にスケジューラから呼び出されます。

- システム初期化処理
  - ブート処理
    RX850V4 が処理を実行するうえで必要となる最低限のハードウエアを初期化するためにユーザ・オウン・コーディング部として切り出された初期化処理専用ルーチンであり，CPU 例外エントリ処理から呼び出されます。

備考 1　ユーザ・オウン・コーディング部についての詳細は，「第 11 章　割り込み管理機能」，「第 13 章　システム構成管理機能」，「第 14 章　スケジューリング機能」，「第 15 章　システム初期化処理」を参照してください。

備考 2　RX850V4 では，ユーザ・オウン・コーディング部のサンプル・ソース・ファイルを提供しています。

<rx_sample>¥src¥inirtn.c，idlrtn.s，boot.s

2.6　リンク・ディレクティブ・ファイルの記述

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

備考 1　リンク・ディレクティブ・ファイルについての詳細は，「CubeSuite V850 コーディング編」，または「CubeSuite コーディング編（CX コンパイラ）」のユーザーズ・マニュアルを参照してください。

備考 2　RX850V4 では，機能単位にモジュール化されたオブジェクトの割り付け先（セクション名）を規定しています。したがって，規定されたセクション名をリンク・ディレクティブ・ファイルに定義することは必須となります。以下に，RX850V4 が規定しているセクション名一覧を示します。

セクション名	属性	タイプ	ROM/RAM	意味
.rx_text	RX	PROGBITS	ROM/RAM	カーネル共通部モジュール，カーネル・モジュール
.rx_info	R	PROGBITS	ROM/RAM	OS 資源（基本クロック周期，最大タスク優先度など）に関するデータ
.rx_memory	RW	NOBITS	RAM	システム・スタック，タスク・スタック，データ・キュー領域，固定長メモリ・プール，可変長メモリ・プール
.rx_text	RX	PROGBITS	ROM/RAM	カーネル共通部モジュール，カーネル・モジュール
.rx_control	RW	NOBITS	RAM	管理オブジェクト（システム管理テーブル，基本タスク管理ブロックなど）

備考 3　RX850V4 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。

<rx_sample>¥src¥sample.dir

2.7　ロード・モジュールの生成

「2.2　ターゲット依存部の記述」から「2.6　リンク・ディレクティブ・ファイルの記述」で作成されたファイル群，および，RX850V4，C コンパイラ・パッケージが提供しているライブラリ・ファイルに対して，CubeSuite 上でビルドを実行し，ロード・モジュールを生成します。

1）プロジェクトの作成／読み込み

プロジェクトの新規作成，または既存のプロジェクトの読み込みを行います。

備考 1　プロジェクトの新規作成，および既存のプロジェクトの読み込みについての詳細は，「RX シリーズ 起動編」，および「CubeSuite 起動編」のユーザーズ・マニュアルを参照してください。

備考 2　複数バージョンの　RX850V4　をインストールしている環境でプロジェクトを作成する場合，使用する RX850V4 は最新バージョンとなります。
使用する RX850V4 のバージョンの変更は，「RX850V4 のバージョンの設定」で行います。

2）ビルド対象プロジェクトの設定

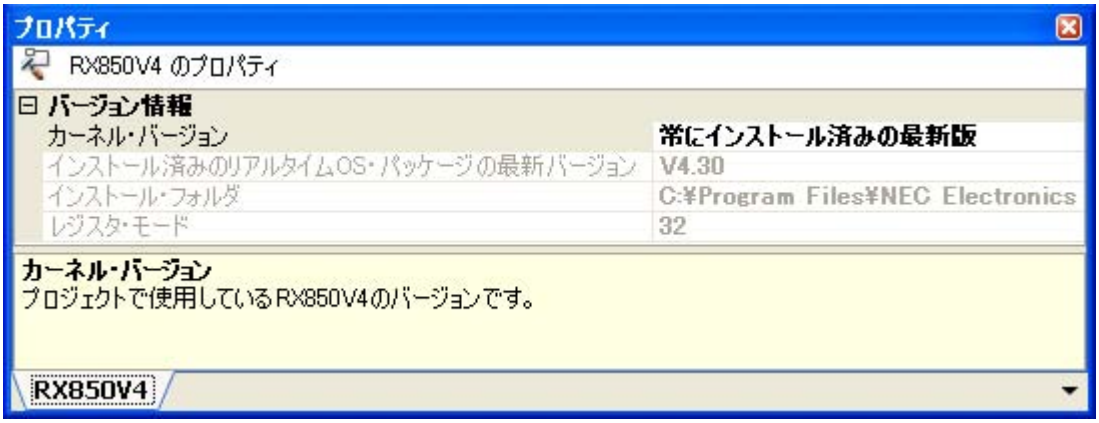
ビルドの設定や実行を行う場合は，アクティブ・プロジェクトを設定します。
なお，サブプロジェクトがない場合，プロジェクトは常にアクティブになります。

備考　アクティブ・プロジェクトの設定についての詳細は，「CubeSuite V850 ビルド編」，または「CubuSuite ビルド編（CX コンパイラ）」のユーザーズ・マニュアルを参照してください。

3）RX850V4 のバージョンの設定

プロジェクト・ツリーでリアルタイム OS ノードを選択し，プロパティ パネルをオープンします。
［RX850V4］タブの［カーネル・バージョン］プロパティにおいて，使用する RX850V4 のバージョンを選択します。
プロジェクト作成時には，デフォルトで［常にインストール済みの最新版］が選択されます。

図 2 － 2　プロパティ パネル：［RX850V4］タブ



備考　V850E2M のデバイスに対応した RX850V4 は V4.30 以降となります。
V850E2M のデバイスを指定したプロジェクトにおいて，V4.30 より前のバージョンを選択した場合は，選択前のバージョンに戻ります。

4）ビルド対象ファイルの設定

プロジェクトへのビルド対象ファイルの追加／削除，依存関係の更新などを行います。

備考　プロジェクトへのビルド対象ファイルの追加／削除，依存関係の更新についての詳細は，「CubeSuite V850 ビルド編」，または「CubuSuite ビルド編（CX コンパイラ）」のユーザーズ・マニュアルを参照してください。

以下に，ロード・モジュールを生成する際に必要となるファイル群の一覧を示します。

- 「2.2.1　ターゲット依存部ライブラリの生成」で作成されたライブラリ・ファイル
  - ターゲット依存部ライブラリ
- 「2.3　処理プログラムの記述」で作成された C 言語／アセンブリ言語ソース・ファイル
  - 処理プログラム（タスク，タスク例外処理ルーチン，周期ハンドラ，割り込みハンドラ，直接起動割り込みハンドラ，拡張サービス・コール・ルーチン，CPU 例外ハンドラ）
- 「2.4　システム・コンフィギュレーション・ファイルの記述」で作成されたシステム・コンフィギュレーション・ファイル
  - システム・コンフィギュレーション・ファイル

  備考　システム・コンフィギュレーション・ファイル名の拡張子は，“cfg”を指定してください。
  拡張子が異なる場合は，“cfg”が自動的に付加されます（例えば，ファイル名に“aaa.c”を指定した場合，“aaa.c.cfg”となります）。
- 「2.5　ユーザ・オウン・コーディング部の記述」で作成された C 言語／アセンブリ言語ソース・ファイル
  - ユーザ・オウン・コーディング部（初期化ルーチン，アイドル・ルーチン，ブート処理）
- 「2.6　リンク・ディレクティブ・ファイルの記述」で作成されたリンク・ディレクティブ・ファイル
  - リンク・ディレクティブ・ファイル
- RX850V4 が提供しているライブラリ・ファイル
  - カーネル・ライブラリ
- C コンパイラ・パッケージが提供しているライブラリ・ファイル
  - 標準ライブラリ，数学ライブラリなど

備考 1　プロジェクト・ツリー パネルにシステム・コンフィギュレーション・ファイルを追加すると，リアルタイム OS 生成ファイル・ノードが表示されます。
リアルタイム OS 生成ファイル・ノードには，以下の情報ファイルが表示されます。ただし，この時点では，これらのファイルは生成されません。

- システム情報テーブル・ファイル
- システム情報ヘッダ・ファイル
- エントリ・ファイル

図 2 － 3　プロジェクト・ツリー パネル（sys.cfg 追加後）

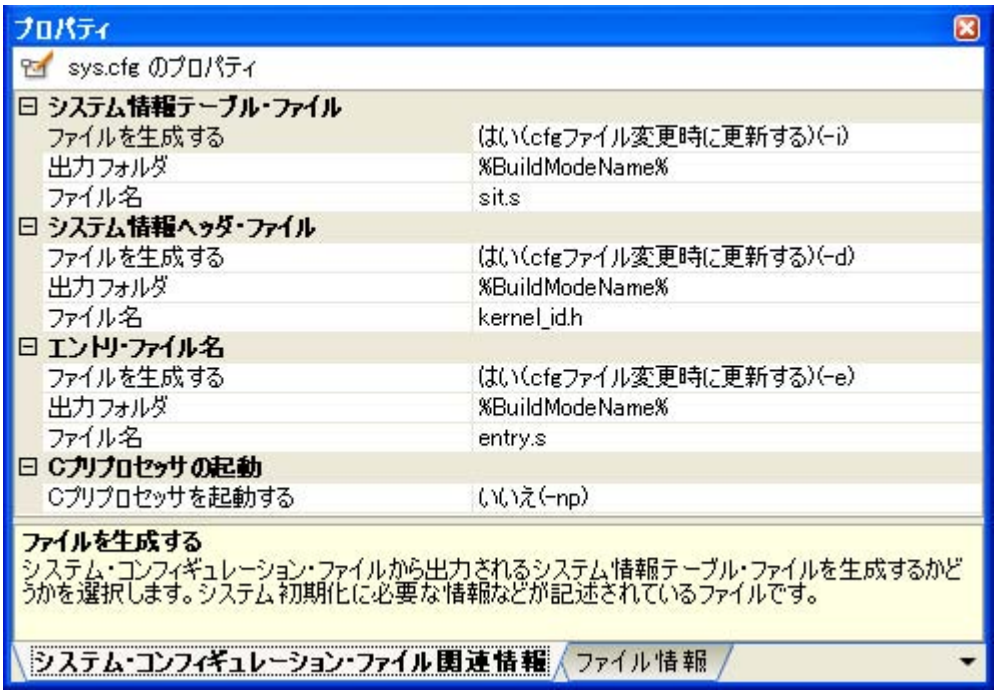
備考 2　システム・コンフィギュレーション・ファイルを差し替える場合は，追加しているシステム・コンフィギュレーション・ファイルを一旦プロジェクトから外したのち，再度ファイルを追加してください。

備考 3　システム・コンフィギュレーション・ファイルは，プロジェクトに複数追加することができますが，有効となるのは最初に追加したファイルです。有効なファイルをプロジェクトから外しても，追加済みのファイルは有効にならないため，再度ファイルを追加してください。

5） 情報ファイルの出力指定

プロジェクト・ツリーでシステム・コンフィギュレーション・ファイルを選択し，プロパティ パネルをオープンします。
［システム・コンフィギュレーション・ファイル関連情報］タブにおいて，情報ファイル（システム情報テーブル・ファイル，システム情報ヘッダ・ファイル，エントリ・ファイル）を出力することを設定します。

図２－４　プロパティ パネル：［システム・コンフィギュレーション・ファイル情報］タブ



6） ロード・モジュール・ファイルの出力指定

ビルドの生成物として，ロード・モジュール・ファイルを出力することを設定します。

備考　　ロード・モジュールの出力指定についての詳細は，「CubeSuite V850 ビルド編」，または「CubuSuite ビルド編（CX コンパイラ）」のユーザーズ・マニュアルを参照してください。

7） ビルド・オプションの設定

コンパイラ，アセンブラ，リンカなどに対するオプションを設定します。

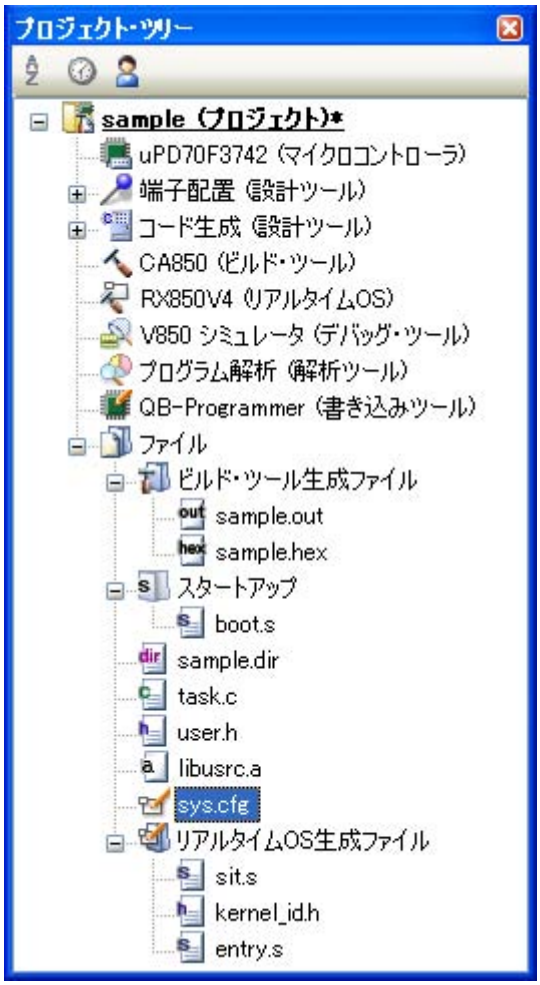
備考　　ビルド・オプションの設定についての詳細は，「CubeSuite V850 ビルド編」，または「CubuSuite ビルド編（CX コンパイラ）」のユーザーズ・マニュアルを参照してください。

8） ビルドの実行

ビルドを実行し，ロード・モジュール・ファイルを生成します。

備考　ビルドの実行についての詳細は，「CubeSuite V850 ビルド編」，または「CubuSuite ビルド編（CX コンパイラ）」のユーザーズ・マニュアルを参照してください。

図２－５　プロジェクト・ツリー パネル（ビルド実行後）



9） プロジェクトの保存

プロジェクトの設定情報をプロジェクト・ファイルに保存します。

備考　プロジェクトの保存についての詳細は，「CubeSuite 起動編」のユーザーズ・マニュアルを参照してください。

第３章　タスク管理機能

本章では，RX850V4 が提供しているタスク管理機能について解説しています。

3.1　概　　要

RX850V4 におけるタスク管理機能では，タスクの生成／起動／終了などといったタスクの状態を操作する機能のほかに，優先度の参照，タスク詳細情報の参照などといったタスクの状態を参照する機能も提供しています。

3.2　タ ス ク

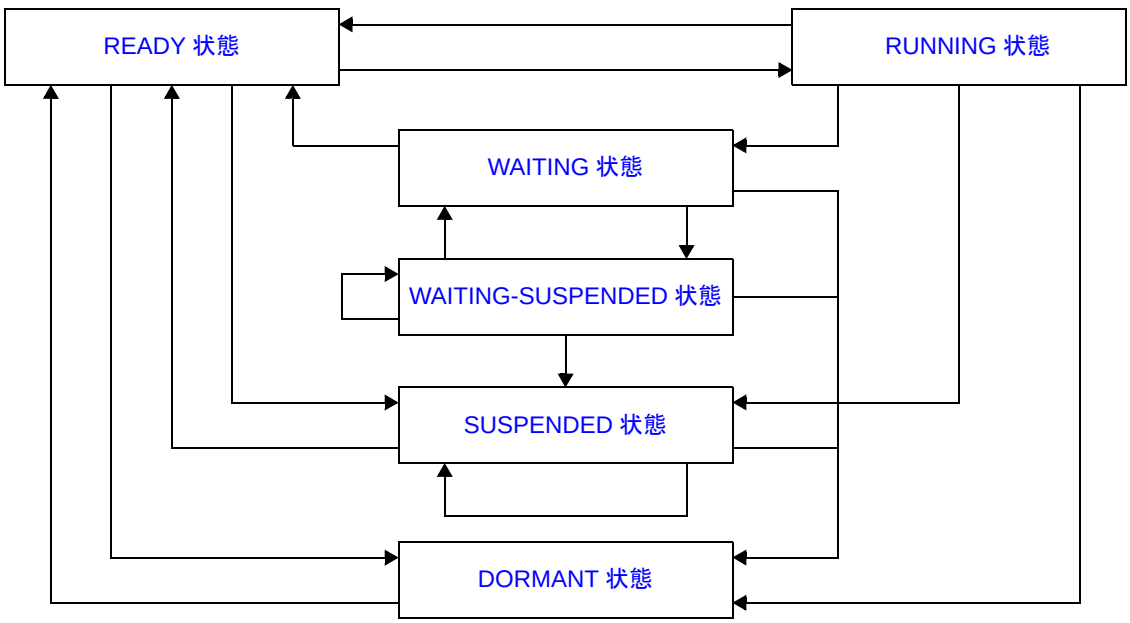
タスクは，他処理プログラム（周期ハンドラ，割り込みハンドラなど）とは異なり，RX850V4 が提供するサービス・コールを使用して明示的に操作しないかぎり実行されることのない処理プログラムです。
なお，RX850V4 では，タスクを管理するに当たり，タスクと一対一に対応した管理オブジェクト（タスク管理ブロック）を用いることにより，タスクが取り得る状態の管理，および，タスク自体の管理を行っています。

備考　タスクが処理を実行するうえで必要となるプログラム・カウンタ，汎用レジスタなどの実行環境情報は，“タスク・コンテキスト”と呼ばれ，タスクの実行が切り替わる際には，現在実行中のタスクのタスク・コンテキストがセーブされ，次に実行されるタスクのタスク・コンテキストがロードされます。

3.2.1　タスクの状態

タスクは，処理を実行するうえで必要となる資源の獲得状況，および，事象発生の有無などにより，さまざまな状態へと遷移していきます。そこで，RX850V4 では，各タスクが現在どのような状態にあるかを認識し，管理する必要があります。
なお，RX850V4 では，タスクが取り得る状態を以下に示した 6 種類に分類し，管理しています。

図３－１　タスクの状態遷移



1) DORMANT 状態

タスクとして起動されていない状態，またはタスクとしての処理を終了した際に遷移する状態です。
なお，DORMANT 状態のタスクは，RX850V4 の管理下にありながらも，RX850V4 のスケジューリング対象からは除外されています。

2) READY 状態

処理を実行するうえで必要となる準備は整っているが，より高い優先度（同一優先度の場合もある）を持つタスクが処理を実行中のため，CPU の利用権が割り当てられるのを待っている状態です。

3) RUNNING 状態

CPU の利用権が割り当てられ，処理を実行中の状態です。
なお，RUNNING 状態のタスクは，システム全体を通して同時に複数存在することはありません。

4) WAITING 状態

処理を実行するうえで必要となる条件が整わないため，処理の実行が中断した状態です。
なお，WAITING 状態からの処理再開は，処理の実行が中断した箇所からとなります。したがって，処理を再開するうえで必要となる情報（タスク・コンテキスト：プログラム・カウンタ，汎用レジスタなど）は，中断直前の値が復元されます。
また，RX850V4 では，要求条件の種類により，WAITING 状態を以下に示す 10 状態に細分化し，管理しています。

表 3 － 1　WAITING 状態の種類

状態種別	意味
起床待ち状態	slp_tsk，または tslp_tsk を発行した際，自タスクの起床要求カウンタ（起床要求の発行回数を保持）が 0x0 の場合に遷移する状態です。
時間経過待ち状態	dly_tsk を発行した際に遷移する状態です。
資源獲得待ち状態	wai_sem，または twai_sem を発行した際，対象セマフォから資源を獲得することができなかった場合に遷移する状態です。
イベントフラグ待ち状態	wai_flg，または twai_flg を発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合に遷移する状態です。
データ送信待ち状態	snd_dtq，または tsnd_dtq を発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むことができなかった場合に遷移する状態です。
データ受信待ち状態	rcv_dtq，または trcv_dtq を発行した際，対象データ・キューからデータを受信することができなかった場合に遷移する状態です。
メッセージ受信待ち状態	rcv_mbx，または trcv_mbx を発行した際，対象メールボックスからメッセージを受信することができなかった場合に遷移する状態です。
ミューテックス待ち状態	loc_mtx，または tloc_mtx を発行した際，対象ミューテックスをロックすることができなかった場合に遷移する状態です。
固定長メモリ・ブロック待ち状態	get_mpf，または tget_mpf を発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった場合に遷移する状態です。
可変長メモリ・ブロック待ち状態	get_mpl，または tget_mpl を発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった場合に遷移する状態です。

5) SUSPENDED 状態

強制的に処理の実行を中断させられた状態です。
なお，SUSPENDED 状態からの処理再開は，処理の実行が中断した箇所からの再開となります。したがって，処理を再開するうえで必要となる情報（タスク・コンテキスト：プログラム・カウンタ，汎用レジスタなど）は，中断直前の値が復元されます。

6) WAITING-SUSPENDED 状態

WAITING 状態と SUSPENDED 状態が複合した状態です。
なお，WAITING 状態が解除された際には SUSPENDED 状態へ，SUSPENDED 状態が解除された際には WAITING 状態へと遷移します。

3.2.2　タスクの優先度

タスクには，処理を実行するうえでの優先順位を決定する優先度が付けられています。そこで，RX850V4 のスケジューラでは，実行可能な状態（RUNNING 状態，および，READY 状態）にあるタスクの優先度を参照し，その中から最も高い優先度（最高優先度）を持つタスクを選び出し，CPU の利用権を与えています。
なお，RX850V4 では，“タスクの優先度”を以下に示した 2 種類に分類し，管理しています。

- 初期優先度
  タスクの生成時に設定される優先度です。したがって，タスクが DORMANT 状態から READY 状態へと遷移した直後の優先度（スケジューラが参照する優先度）は，初期優先度となります。

- 現在優先度
  タスクを起動後，RX850V4 が各種操作（タスクのスケジューリング，優先度順の待ちキューへのキューイング，タスクの優先度継承）を行う際に参照する優先度です。

備考 1　RX850V4 におけるタスクの優先度は，その値が小さいほど，高い優先度であることを意味します。

備考 2　システム内で利用可能な優先度の範囲については，システム・コンフィギュレーション・ファイル作成時に基本情報（*最大タスク優先度 maxtpri*）を定義することにより実現されます。

3.2.3　タスクの基本型

タスクを記述する場合，VP_INT 型の引き数を 1 つ持った void 型の関数として記述します。
なお，引き数 *exinf* には“タスクの拡張情報”が設定されます。
以下に，タスクを C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        .............
        .............

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

備考 1　sta_tsk，または ista_tsk の発行により DORMANT 状態から READY 状態へと遷移した場合，引き数 *exinf* には“sta_tsk，または ista_tsk 発行時に指定した拡張情報”が設定されます。

備考 2　タスク内で return 命令が発行された場合，ext_tsk と同等の処理が実行されます。

備考 3　タスクの拡張情報についての詳細は，「3.4　タスクの起動」を参照してください。

3.2.4　タスク内での処理

RX850V4 では，タスクを切り替える際，独自のスケジューリング処理を行っています。このため，タスクを記述する際には，以下に示す注意点があります。

- 記述方法
  C 言語，またはアセンブリ言語で記述します。
  C 言語で記述するときは通常の関数と同様に記述することができます。
  アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。

- スタックの切り替え
  RX850V4 では，タスクを切り替える際，タスク情報において指定されたタスク・スタックへの切り替え処理を行っています。

- サービス・コールの発行
  タスクでは，“タスク内から発行可能なサービス・コール”のみが発行可能となります。

  備考　各サービス・コールの発行有効範囲についての詳細は，表 17 － 1 ～表 17 － 14 を参照してください。

- マスカブル割り込みの受け付け状態（PSW の ID フラグ）
  処理開始時（DORMANT 状態のタスクが RUNNING 状態になりタスク処理に制御が移る時），マスカブル割り込みの受け付け状態はタスク情報の属性で設定される初期割り込み状態によって変わります。
  処理内では，マスカブル割り込みの受け付け状態を変更することができます。変更した状態は，処理終了後（RUNNING 状態のタスクが DORMANT 状態になった後）に制御が移った処理プログラムへは継承されません。
  処理再開時（RUNNING 状態のタスクが READY 状態，WAITING 状態，WAITING-SUSPENDED 状態，SUSPENDED 状態のいずれかに遷移した後，再び RUNNING 状態になりタスク処理に制御が移る時），マスカブル割り込みの受け付け状態は処理中断前の状態に戻ります。

3.3　タスクの生成

RX850V4 では，タスクの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

タスクの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“CRE_TSK”を使用してタスクを定義することをいいます。

静的 API“CRE_TSK”の詳細は，「18.5.1　タスク情報」を参照してください。

3.4　タスクの起動

RX850V4 では，タスクの起動において，“起動要求をキューイングする”，“起動要求をキューイングしない”の 2 種類のインタフェースを用意しています。

なお，RX850V4 では，コンフィギュレーション時にタスク情報で指定した拡張情報，および，サービス・コール sta_tsk，ista_tsk 発行時に第 2 パラメータ *stacd* で指定した値を“タスクの拡張情報”と呼んでいます。

3.4.1　起動要求をキューイングする起動

タスクの起動（起動要求をキューイングする）は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- act_tsk，iact_tsk
  パラメータ *tskid* で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち，初期優先度のレディ・キューの最後尾にキューイングします。これにより，対象タスクは，RX850V4 のスケジューリング対象となります。
  ただし，本サービス・コールを発行した際，対象タスクが DORMANT 状態以外の場合には，対象タスクのキューイング処理，および，状態操作処理は行わず，対象タスクに起動要求をキューイング（起動要求カウンタに 0x1 を加算）しています。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

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

        act_tsk ( tskid );              /* タスクの起動（起動要求をキューイングする）*/

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

備考 1　RX850V4 が管理する起動要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールでは，起動要求数が 127 回を越えるような場合には，起動要求の発行起動要求の発行（起動要求カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

備考 2　本サービス・コールの発行により起動されたタスクには，拡張情報として“タスク情報で指定した拡張情報”が渡されます。

3.4.2　起動要求をキューイングしない起動

タスクの起動（起動要求をキューイングしない）は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sta_tsk，ista_tsk

  パラメータ *tskid* で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち，初期優先度のレディ・キューの最後尾にキューイングします。これにより，対象タスクは，RX850V4 のスケジューリング対象となります。ただし，本サービス・コールでは，起動要求のキューイングが行われません。このため，対象タスクが DORMANT 状態以外の場合には，対象タスクの状態操作処理は行わず，戻り値として E_OBJ を返します。
  なお，パラメータ *stacd* には，対象タスクに引き渡す拡張情報を指定します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        VP_INT  stacd = 123;            /* 変数の宣言，初期化 */

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

        sta_tsk ( tskid, stacd );       /* タスクの起動（起動要求をキューイングしない）*/

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

3.5　起動要求のキューイング解除

起動要求のキューイング解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- can_act，ican_act
  パラメータ *tskid* で指定されたタスクにキューイングされている起動要求をすべて解除（起動要求カウンタに 0x0 を設定）します。
  なお，正常終了時は戻り値として本サービス・コールの発行により解除した起動要求数を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER_UINT ercd;                   /* 変数の宣言 */
        ID      tskid = 8;              /* 変数の宣言，初期化 */

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

        ercd = can_act ( tskid );       /* 起動要求のキューイング解除 */

        if ( ercd >= 0x0 ) {
                .............           /* 正常終了処理 */
                .............
        }

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

備考　本サービス・コールでは，状態操作処理は行わず，起動要求カウンタの設定処理のみを行います。
したがって，READY 状態などから DORMANT 状態に遷移することはありません。

3.6　タスクの終了

3.6.1　自タスクの終了

自タスクの終了，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ext_tsk
  自タスクを RUNNING 状態から DORMANT 状態へと遷移させ，レディ・キューから外します。これにより，自タスクは，RX850V4 のスケジューリング対象から除外されます。
  ただし，本サービス・コールを発行した際，自タスクの起動要求がキューイングされていた（起動要求カウンタが 0x0 以外の値であった）場合には，自タスクの状態操作（DORMANT 状態への状態遷移処理）を行ったのち，自タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        .............
        .............

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

備考 1　本サービス・コールでは，自タスクの状態操作（DORMANT 状態への状態遷移処理）を行う際に，

- 優先度（現在優先度）
- 起床要求数
- サスペンド要求数
- 割り込み状態

といった情報をタスク生成時に設定される値で初期化しています。
また，自タスクがミューテックスをロックしていた場合には，ロック状態の解除（unl_mtx と同等の処理）もあわせて行われます。

備考 2　タスク内で return 命令が発行された場合，本サービス・コールと同等の処理が実行されます。

3.6.2　タスクの強制終了

タスクの強制終了，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ter_tsk
  パラメータ *tskid* で指定されたタスクを強制的に DORMANT 状態へと遷移させます。これにより，対象タスクは，RX850V4 のスケジューリング対象から除外されます。
  ただし，本サービス・コールを発行した際，対象タスクの起動要求がキューイングされていた（起動要求カウンタが 0x0 以外の値であった）場合には，対象タスクの状態操作（DORMANT 状態への状態遷移処理）を行ったのち，対象タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

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

        ter_tsk ( tskid );              /* タスクの強制終了 */

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

備考　本サービス・コールでは，対象タスクの状態操作（DORMANT 状態への状態遷移処理）を行う際に，

- 優先度（現在優先度）
- 起床要求数
- サスペンド要求数
- 割り込み状態

といった情報をタスク生成時に設定される値で初期化しています。
また，対象タスクがミューテックスをロックしていた場合には，ロック状態の解除（unl_mtx と同等の処理）もあわせて行われます。

3.7　タスク優先度の変更

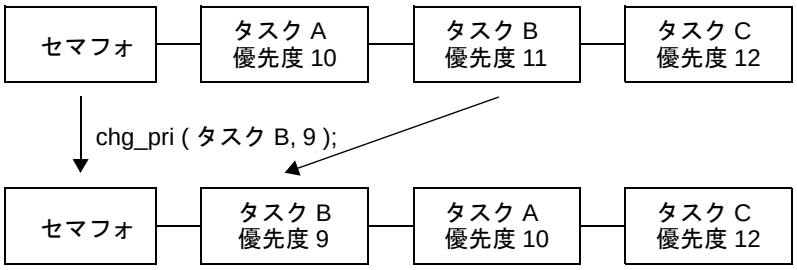
タスク優先度の変更は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- chg_pri，ichg_pri
  パラメータ *tskid* で指定されたタスクの優先度（現在優先度）をパラメータ *tskpri* で指定された値に変更します。
  対象タスクが RUNNING 状態，または READY 状態であった場合には，優先度を変更したのち，対象タスクをパラメータ *tskpri* で指定された優先度に応じたレディ・キューの最後尾につなぎかえます。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        PRI     tskpri = 9;             /* 変数の宣言，初期化 */

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

        chg_pri ( tskid, tskpri );      /* タスク優先度の変更 */

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

備考　対象タスクが何らかの待ちキューに優先度順でキューイングされていた場合，本サービス・コールの発行により，待ち順序が変わることがあります。

例　セマフォの待ちキューに 3 つのタスク（タスク A：優先度 10，タスク B：優先度 11，タスク C：優先度 12）が優先度順でキューイングされているとき，タスク B の優先度を 11 から 9 に変更した場合，待ちキューの待ち順序は，以下のように変更されます。



3.8　タスク優先度の参照

タスク優先度の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- get_pri，iget_pri
  パラメータ *tskid* で指定されたタスクの現在優先度をパラメータ *p_tskpri* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        PRI     p_tskpri;               /* 変数の宣言 */

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

        get_pri ( tskid, &p_tskpri );   /* タスク優先度の参照 */

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

3.9　タスク状態の参照

3.9.1　タスク詳細情報の参照

タスク詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_tsk，iref_tsk
  パラメータ *tskid* で指定されたタスクのタスク詳細情報（現在状態，現在優先度など）をパラメータ *pk_rtsk* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        T_RTSK  pk_rtsk;                /* データ構造体の宣言 */
        STAT    tskstat;                /* 変数の宣言 */
        PRI     tskpri;                 /* 変数の宣言 */
        STAT    tskwait;                /* 変数の宣言 */
        ID      wobjid;                 /* 変数の宣言 */
        TMO     lefttmo;                /* 変数の宣言 */
        UINT    actcnt;                 /* 変数の宣言 */
        UINT    wupcnt;                 /* 変数の宣言 */
        UINT    suscnt;                 /* 変数の宣言 */
        ATR     tskatr;                 /* 変数の宣言 */
        PRI     itskpri;                /* 変数の宣言 */

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

        ref_tsk ( tskid, &pk_rtsk );    /* タスク詳細情報の参照 */

        tskstat = pk_rtsk.tskstat;      /* 現在状態の獲得 */
        tskpri = pk_rtsk.tskpri;        /* 現在優先度の獲得 */
        tskwait = pk_rtsk.tskwait;      /* 待ち要因の獲得 */
        wobjid = pk_rtsk.wobjid;        /* 管理オブジェクトの ID の獲得 */
        lefttmo = pk_rtsk.lefttmo;      /* 残り時間の獲得 */
        actcnt = pk_rtsk.actcnt;        /* 起動要求数の獲得 */
        wupcnt = pk_rtsk.wupcnt;        /* 起床要求数の獲得 */
        suscnt = pk_rtsk.suscnt;        /* サスペンド要求数の獲得 */
        tskatr = pk_rtsk.tskatr;        /* 属性の獲得 */
        itskpri = pk_rtsk.itskpri;      /* 初期優先度の獲得 */

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

備考　タスク詳細情報 T_RTSK についての詳細は，「16.2.1　タスク詳細情報」を参照してください。

3.9.2　タスク基本情報の参照

タスク基本情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_tst，iref_tst
  パラメータ *tskid* で指定されたタスクのタスク基本情報（現在状態，待ち要因）をパラメータ *pk_rtst* で指定された領域に格納します。
  タスク情報のうち，現在状態，待ち要因のみを参照したい場合に使用します。
  取得する情報が少ないので ref_tsk，iref_tsk より高速に応答します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        T_RTST  pk_rtst;                /* データ構造体の宣言 */
        STAT    tskstat;                /* 変数の宣言 */
        STAT    tskwait;                /* 変数の宣言 */

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

        ref_tst ( tskid, &pk_rtst );    /* タスク基本情報の参照 */

        tskstat = pk_rtst.tskstat;      /* 現在状態の獲得 */
        tskwait = pk_rtst.tskwait;      /* 待ち要因の獲得 */

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

備考　タスク基本情報 T_RTST についての詳細は，「16.2.2　タスク基本情報」を参照してください。

3.10　ターゲット依存部

RX850V4 では，さまざまな実行環境に対応するために，メモリ・プール管理機能のうち，RX850V4，および，処理プログラムが処理を実行するうえで必要となるスタックがオーバフローした際の後処理（オーバフロー後処理）をターゲット依存部として切り出しています。これにより，スタックのオーバフローによるシステムの暴走を防ぐことが可能となります。

備考　RX850V4 によるスタックのオーバフロー・チェックは，コンフィギュレーション時に基本情報において“オーバフロー・チェックを行う”が定義された場合にかぎり行われます。

3.10.1　オーバフロー後処理

オーバフロー後処理は，RX850V4，および，処理プログラムが処理を実行するうえで必要となるスタックがオーバフローした際の後処理を実行するためにターゲット依存部として切り出された後処理専用ルーチンであり，スタックがオーバフローした際に RX850V4 から呼び出されます。

- オーバフロー後処理の基本型
  オーバフロー後処理を記述する場合，INT 型の引き数を 2 つ持った void 型の関数（関数名：_kernel_stk_overflow）として記述します。
  なお，引き数 *r6* には“スタックのオーバフローが検出された時点のスタック・ポインタ sp の値”が，r7 には“スタックのオーバフローが検出された時点のプログラム・カウンタ pc の値”が設定されます。
  以下に，オーバフロー後処理をアセンブリ言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

        .text
        .align  0x4
        .globl  __kernel_stk_overflow
__kernel_stk_overflow :

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

.halt_loop :
        jbr     .halt_loop
```

- オーバフロー後処理内での処理
  オーバフロー後処理は，RX850V4，および，処理プログラムが処理を実行するうえで必要となるスタックがオーバフローした際の後処理を実行するためにターゲット依存部として切り出された後処理専用ルーチンです。このため，オーバフロー後処理を記述する際には，以下に示す注意点があります。
  - 記述方法
    C 言語，またはアセンブリ言語で記述します。
    C 言語で記述するときは通常の関数と同様に記述することができます。
    アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。
  - スタックの切り替え
    RX850V4 では，オーバフロー後処理に制御を移す際に，スタックの切り替えに関する処理を行っていません。したがって，基本情報において指定されたシステム・スタックを使用する際には，オーバフロー後処理内でスタックの切り替えに関する記述を行う必要があります。
  - サービス・コールの発行
    オーバフロー後処理では，正常な動作を保証することができないため，サービス・コールの発行が禁止されています。

  以下に，オーバフロー後処理として実行すべき処理の一覧を示します。
  - スタックのオーバフローに対応した後処理

  備考　オーバフロー後処理として記述すべき処理内容（リセットなど）については，ユーザのシステムに依存しています。

3.11　省メモリ化

RX850V4 では，タスクが処理を実行する際に必要となるタスク・スタックのサイズを削減する方法として，2 種類（制約タスク，プリエンプト禁止）の手段を提供しています。

3.11.1　制約タスク

通常，タスク・スタックのサイズを見積もる際には，最大消費サイズを該当サイズとして見積もり，確保します。

しかし，実際に最大消費サイズを使用しているとき以外は，未使用の領域が存在していることになります。この未使用領域を有効活用したものが制約タスクです。

システム・コンフィギュレーション・ファイル作成時にタスク情報として「属性：制約タスク TA_RSTR」が定義されたタスクでは，タスク・スタックの未使用領域の合計サイズ分のサイズを動的に削減することが可能となります。

3.11.2　プリエンプト禁止

RX850V4 では，システム・コンフィギュレーション・ファイル作成時にタスク情報として「属性：プリエンプトの受け付け状態 TA_DISPREEMPT」を定義することができます。

この属性が定義されたタスクでは,「非タスクから発行されたスケジューリング要求を無視して処理を続行する」といった動作を行うため，1 タスク当たり 24 ～ 44 バイトの管理領域を削減することが可能となります。

第 4 章　タスク付属同期機能

本章では，RX850V4 が提供しているタスク付属同期機能について解説しています。

4.1　概　　要

RX850V4 におけるタスク付属同期機能では，タスクに従属した同期機能を提供しています。

4.2　起床待ち状態への移行

4.2.1　永久待ち

起床待ち状態への移行（永久待ち）は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- slp_tsk
  自タスクを RUNNING 状態から WAITING 状態（起床待ち状態）へと遷移させます。
  ただし，本サービス・コールを発行した際，自タスクの起床要求がキューイングされていた（起床要求カウンタが 0x0 以外）場合には，状態操作処理は行わず，起床要求カウンタから 0x1 を減算します。
  なお，起床待ち状態の解除は，以下の場合に行われ，起床待ち状態から READY 状態へと遷移します。

起床待ち状態の解除操作	エラー・コード
wup_tsk の発行により，起床要求が発行された	E_OK
iwup_tsk の発行により，起床要求が発行された	E_OK
rel_wai の発行により，起床待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，起床待ち状態を強制的に解除された	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */

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

        ercd = slp_tsk ( );             /* 起床待ち状態への移行（永久待ち）*/

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }
```

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

4.2.2　タイムアウト付き

起床待ち状態への移行（タイムアウト付き）は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- tslp_tsk
  自タスクを RUNNING 状態からタイムアウト付きの WAITING 状態（起床待ち状態）へと遷移させます。
  ただし，本サービス・コールを発行した際，自タスクの起床要求がキューイングされていた（起床要求カウンタが 0x0 以外）場合には，状態操作処理は行わず，起床要求カウンタから 0x1 を減算します。
  なお，起床待ち状態の解除は，以下の場合に行われ，起床待ち状態から READY 状態へと遷移します。

起床待ち状態の解除操作	エラー・コード
wup_tsk の発行により，起床要求が発行された	E_OK
iwup_tsk の発行により，起床要求が発行された	E_OK
rel_wai の発行により，起床待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，起床待ち状態を強制的に解除された	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>          /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task        /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;               /* 変数の宣言 */
        TMO     tmout = 3600;       /* 変数の宣言，初期化 */

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

        ercd = tslp_tsk ( tmout );  /* 起床待ち状態への移行（タイムアウト付き）*/

        if ( ercd == E_OK ) {
                .............       /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............       /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............       /* タイムアウト処理 */
                .............
        }

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

備考　待ち時間 *tmout* に TMO_FEVR が指定された際には“slp_tsk と同等の処理”を実行します。

4.3　タスクの起床

タスクの起床は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- wup_tsk，iwup_tsk
  パラメータ *tskid* で指定されたタスクを WAITING 状態（起床待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移させます。
  ただし，本サービス・コールを発行した際，対象タスクが起床待ち状態以外の場合には，状態操作処理は行わず，起床要求カウンタに 0x1 を加算します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

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

        wup_tsk ( tskid );              /* タスクの起床 */

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

備考　RX850V4 が管理する起床要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールでは，起床要求数が 127 回を越えるような場合には，起床要求のキューイング（起床要求カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

4.4　起床要求の解除

起床要求の解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- can_wup，ican_wup
  パラメータ *tskid* で指定されたタスクにキューイングされている起床要求をすべて解除（起床要求カウンタに 0x0 を設定）します。
  なお，本サービス・コールは戻り値として解除した起床要求数を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER_UINT ercd;                   /* 変数の宣言 */
        ID      tskid = 8;              /* 変数の宣言，初期化 */

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

        ercd = can_wup ( tskid );       /* 起床要求の解除 */

        if ( ercd >= 0x0 ) {
                .............           /* 正常終了処理 */
                .............
        }

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

4.5　WAITING 状態の強制解除

WAITING 状態の強制解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rel_wai，irel_wai
  パラメータ *tskid* で指定されたタスクの WAITING 状態を強制的に解除します。これにより，対象タスクは待ちキューから外れ，WAITING 状態から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  なお，本サービス・コールの発行により WAITING 状態を解除されたタスクには，WAITING 状態へと遷移するきっかけとなったサービス・コール（slp_tsk，wai_sem など）の戻り値として E_RLWAI を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

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

        rel_wai ( tskid );              /*WAITING 状態の強制解除 */

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

備考 1　本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが WAITING 状態，または WAITING-SUSPENDED 状態以外の場合には，戻り値として E_OBJ を返します。

備考 2　本サービス・コールでは，SUSPENDED 状態の解除は行われません。

4.6　SUSPENDED 状態への移行

SUSPENDED 状態への移行は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sus_tsk，isus_tsk
  パラメータ *tskid* で指定されたタスクを RUNNING 状態から SUSPENDED 状態へ，READY 状態から SUSPENDED 状態へ，または WAITING 状態から WAITING-SUSPENDED 状態へと遷移させます。
  ただし，本サービス・コールを発行した際，対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態へと遷移していた場合には，状態操作処理は行わず，サスペンド要求のキューイング（カウンタの加算処理）のみを行います。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

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

        sus_tsk ( tskid );              /*SUSPENDED 状態への移行 */

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

備考　RX850V4 が管理するサスペンド要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールでは，サスペンド要求数が 127 回を越えるような場合には，サスペンド要求の発行（サスペンド要求カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

4.7　SUSPENDED 状態の解除

4.7.1　SUSPENDED 状態の解除

SUSPENDED 状態の解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rsm_tsk，irsm_tsk
  パラメータ *tskid* で指定されたタスクを SUSPENDED 状態から READY 状態へ，または WAITING-SUSPENDED 状態から WAITING 状態へと遷移させます。
  ただし，本サービス・コールを発行した際，サスペンド要求がキューイングされていた場合には，状態操作処理は行わず，サスペンド要求カウンタの減算処理のみを行います。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

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

        rsm_tsk ( tskid );              /*SUSPENDED 状態の解除 */

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

備考　本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態以外の場合には，戻り値として E_OBJ を返します。

4.7.2　SUSPENDED 状態の強制解除

SUSPENDED 状態の強制解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- frsm_tsk，ifrsm_tsk
  パラメータ *tskid* で指定されたタスクに発行されているサスペンド要求をすべて解除（サスペンド要求カウンタに 0x0 を設定）します。これにより，対象タスクは SUSPENDED 状態から READY 状態へ，または WAITING-SUSPENDED 状態から WAITING 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */

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

        frsm_tsk ( tskid );             /*SUSPENDED 状態の強制解除 */

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

備考　本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態以外の場合には，戻り値として E_OBJ を返します。

4.8　時間経過待ち状態への移行

時間経過待ち状態への移行は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- dly_tsk
  自タスクをパラメータ *dlytim* で指定された遅延時間が経過するまでの間，RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させます。
  なお，時間経過待ち状態の解除は，以下の場合に行われ，時間経過待ち状態から READY 状態へと遷移します。

時間経過待ち状態の解除操作	エラー・コード
パラメータ *dlytim* で指定された遅延時間が経過した	E_OK
rel_wai の発行により，時間経過待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，時間経過待ち状態を強制的に解除された	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        RELTIM  dlytim = 3600;          /* 変数の宣言，初期化 */

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

        ercd = dly_tsk ( dlytim );      /* 時間経過待ち状態への移行 */

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

4.9　タイムアウト付き起床待ちと時間経過待ちの違い

タイムアウト付き起床待ちと時間経過待ちには，以下にのような違いがあります。

表 4 － 1　タイムアウト付き起床待ちと時間経過待ちの違い

	タイムアウト付き起床待ち	時間経過待ち
状態遷移するサービス・コール	tslp_tsk	dly_tsk
タイムアウト時の戻り値	E_TMOUT	E_OK
wup_tsk，iwup_tsk が発行された際の動作	起床	起床要求をキューイング（時間経過待ちの解除は行われない）

第 5 章　タスク例外処理機能

本章では，RX850V4 が提供しているタスク例外処理機能について解説しています。

5.1　概　　要

RX850V4 におけるタスク例外処理機能では，タスク例外処理要求が発行された際に起動するタスク例外処理ルーチンに関連した機能（タスク例外処理ルーチンの状態を操作する機能，タスク例外処理ルーチンの状態を参照する機能など）を提供しています。

5.2　タスク例外処理ルーチン

タスク例外処理ルーチンは，タスク例外処理要求が発行された際に起動されるタスク例外処理専用ルーチンです。

なお，RX850V4 では，タスク例外処理ルーチンを“タスク例外処理要求を発行されたタスクの延長線”として位置づけています。このため，タスク例外処理ルーチンの起動タイミングは，タスク例外処理要求を発行されたタスクが RUNNING 状態へと遷移した際となります。

また，RX850V4 では，タスク例外処理ルーチンを管理するに当たり，タスク例外処理ルーチンと一対一に対応した管理オブジェクト（タスク管理ブロックに内包されているタスク例外処理ルーチン管理ブロック）を用いることにより，タスク例外処理ルーチンが取り得る状態の管理，および，タスク例外処理ルーチン自体の管理を行っています。

備考　　タスク例外処理ルーチンが起動した際の禁止状態は，タスク例外処理許可状態となります。

5.2.1　タスク例外処理ルーチンの基本型

タスク例外処理ルーチンを記述する場合，TEXPTN 型の引き数を 1 つ，VP_INT 型の引き数を 1 つ持った void 型の関数として記述します。

なお，引き数 *rasptn* には“タスク例外処理要求（ras_tex，または iras_tex）の発行時に指定したタスク例外要因”が，*exinf* には“タスク情報で指定した拡張情報”が設定されます。

以下に，タスク例外処理ルーチンを C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
texrtn ( TEXPTN rasptn, VP_INT exinf )
{
        .............
        .............

        return;                         /* タスク例外処理ルーチンの終了 */
}
```

備考　　タスク例外処理ルーチンの起動タイミングは，タスク例外処理要求を発行されたタスクが RUNNING 状態へと遷移した際に行われます。このため，1 回目のタスク例外処理要求の発行から実際に該当タスク例外処理ルーチンが起動するまでの間には，複数回のタスク例外処理要求の発行が行われる可能性があります。
このような場合，RX850V4 では，1 回目のタスク例外処理要求から該当タスク例外処理ルーチンが起動するまでの間に発行された“すべてのタスク例外要因の論理和 OR”をタスク例外処理ルーチンの引き数 *rasptn* として設定しています。

5.2.2　タスク例外処理ルーチン内での処理

RX850V4 では，タスク例外処理要求を発行されたタスクからタスク例外処理ルーチンに制御を移す際，独自のタスク例外前処理を行っています。また，タスク例外処理ルーチンからタスク例外処理要求を発行されたタスクに制御を戻す際にも，独自のタスク例外後処理を行っています。このため，タスク例外処理ルーチンを記述する際には，以下に示す注意点があります。

- 記述方法
  C 言語，またはアセンブリ言語で記述します。
  C 言語で記述するときは通常の関数と同様に記述することができます。
  アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。

- スタックの切り替え
  RX850V4 では，タスク例外処理ルーチンを“タスク例外処理要求を発行されたタスクの延長線”として位置づけています。したがって，タスク例外処理ルーチンに制御を移す際，スタックの切り替え処理は行われません。

- サービス・コールの発行
  RX850V4 では，タスク例外処理ルーチンを“タスク例外処理要求を発行されたタスクの延長線”として位置づけています。このため，タスク例外処理ルーチンでは，“タスク内から発行可能なサービス・コール”のみが発行可能となります。

  備考　　各サービス・コールの発行有効範囲についての詳細は，表 17 － 1 ～表 17 － 14 を参照してください。

- マスカブル割り込みの受け付け状態（PSW の ID フラグ）
  処理開始時，マスカブル割り込みの受け付け状態は，該当タスク例外処理ルーチンに対応するタスクの状態を継承します。
  処理内では，マスカブル割り込みの受け付け状態を変更することができます。変更した状態は，該当タスク例外処理ルーチンに対応するタスクへ継承されます。

5.3　タスク例外処理ルーチンの登録

RX850V4 では，タスク例外処理ルーチンの静的な登録のみサポートしています。処理プログラムからサービス・コールを発行して動的に登録することはできません。

タスク例外処理ルーチンの静的登録とは，システム・コンフィギュレーション・ファイルで静的 API“DEF_TEX”を使用してタスク例外処理ルーチンを定義することをいいます。

静的 API“DEF_TEX”の詳細は，「18.5.2　タスク例外処理ルーチン情報」を参照してください。

備考　　制約タスクにタスク例外処理ルーチンを登録することはできません。

5.4　タスク例外処理ルーチンの起動

タスク例外処理ルーチンの起動は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ras_tex，iras_tex
  パラメータ *tskid* で指定されたタスクにタスク例外処理要求を発行します。これにより，対象タスクが RUNNING 状態へと遷移した際には，対象タスクに登録されているタスク例外処理ルーチンが起動します。
  なお，パラメータ *rasptn* には，対象タスク例外処理ルーチンに引き渡すタスク例外要因を指定します。対象タスク例外処理ルーチンは，タスク例外要因を関数パラメータと同様に取り扱うことで操作可能となります。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        TEXPTN  rasptn = 123;           /* 変数の宣言，初期化 */

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

        ras_tex ( tskid, rasptn );      /* タスク例外処理ルーチンの起動 */

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

備考　本サービス・コールでは，タスク例外処理要求のキューイングが行われません。このため，タスク例外処理ルーチンの起動以前（タスク例外処理要求の発行後，対象タスクが RUNNING 状態へと遷移するまでの間）に複数回のタスク例外処理要求が発行された場合には，2 回目以降の本サービス・コールの発行時には，タスク例外処理要求の発行は行わず，タスク例外要因の保留（タスク例外要因の論理和 OR）のみを行います。

5.5　タスク例外処理ルーチンの起動禁止，起動許可

タスク例外処理ルーチンの起動禁止，起動許可は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- dis_tex
  自タスクに登録されているタスク例外処理ルーチンの禁止状態をタスク例外処理許可状態からタスク例外処理禁止状態へと遷移させます。これにより，本サービス・コールの発行から ena_tex が発行されるまでの間，対象タスク例外処理ルーチンは RX850V4 の起動対象から除外されます。
  なお, RX850V4 では, 本サービス・コールの発行から ena_tex が発行されるまでの間にタスク例外処理要求（ras_tex, または iras_tex）が発行された場合には，タスク例外処理要求の受け付けなどといった処理を行うだけであり，実際の起動処理は対象タスク例外処理ルーチンがタスク例外処理許可状態へと遷移するまで遅延されます。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        .............
        .............

        dis_tex ( );                    /* タスク例外処理ルーチンの起動禁止 */

        .............                   /* タスク例外処理禁止状態 */
        .............

        ena_tex ( );                    /* タスク例外処理ルーチンの起動許可 */

        .............                   /* タスク例外処理許可状態 */
        .............
}
```

備考 1　本サービス・コールでは，禁止要求のキューイングが行われません。このため，対象タスク例外処理ルーチンがタスク例外処理禁止状態に遷移していた場合には，何も処理は行わず，エラーとしても扱いません。

備考 2　RX850V4 では，タスクの起動時，タスク例外処理禁止状態となります。

- ena_tex

自タスクに登録されているタスク例外処理ルーチンの禁止状態をタスク例外処理禁止状態からタスク例外処理許可状態へと遷移させます。これにより，対象タスク例外処理ルーチンは RX850V4 の起動対象となります。
なお，RX850V4 では，dis_tex の発行から本サービス・コールが発行されるまでの間にタスク例外処理要求（ras_tex，または iras_tex）が発行された場合には，タスク例外処理要求の受け付けなどといった処理を行うだけであり，実際の起動処理は対象タスク例外処理ルーチンがタスク例外処理許可状態へと遷移するまで遅延されます。
以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        .............
        .............

        dis_tex ( );                    /* タスク例外処理ルーチンの起動禁止 */

        .............                   /* タスク例外処理禁止状態 */
        .............

        ena_tex ( );                    /* タスク例外処理ルーチンの起動許可 */

        .............                   /* タスク例外処理許可状態 */
        .............
}
```

備考　本サービス・コールでは，起動要求のキューイングが行われません。このため，対象タスク例外処理ルーチンがタスク例外処理許可状態に遷移していた場合には，何も処理は行わず，エラーとしても扱いません。

5.6　タスク例外処理禁止状態の参照

タスク例外処理禁止状態の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sns_tex
  本サービス・コールを発行した際に RUNNING 状態であるタスクに登録されているタスク例外処理ルーチンの状態（タスク例外処理禁止状態，タスク例外処理許可状態）を獲得します。
  戻り値としてタスク例外処理ルーチンの状態を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        BOOL    ercd;                   /* 変数の宣言 */

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

        ercd = sns_tex ( );             /* タスク例外処理禁止状態の参照 */

        if ( ercd == TRUE ) {
                .............           /* タスク例外処理禁止状態 */
                .............
        } else if ( ercd == FALSE ) {
                .............           /* タスク例外処理許可状態 */
                .............
        }

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

5.7　タスク例外処理ルーチン詳細情報の参照

タスク例外処理ルーチン詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_tex，iref_tex
  パラメータ *tskid* で指定されたタスクに登録されているタスク例外処理ルーチンのタスク例外処理ルーチン詳細情報（現在状態，保留例外要因など）をパラメータ *pk_rtex* で指定された領域に格納します。
  指定されたタスクにタスク例外処理ルーチンが登録されていなかった場合は，戻り値として E_OBJ を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      tskid = 8;              /* 変数の宣言，初期化 */
        T_RTEX  pk_rtex;                /* データ構造体の宣言 */
        STAT    texstat;                /* 変数の宣言 */
        TEXPTN  pndptn;                 /* 変数の宣言 */
        ATR     texatr;                 /* 変数の宣言 */

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

        ref_tex ( tskid, &pk_rtex );    /* タスク例外処理ルーチン詳細情報の参照 */

        texstat = pk_rtex.texstat;      /* 現在状態の獲得 */
        pndptn = pk_rtex.pndptn;        /* 保留例外要因の獲得 */
        texatr = pk_rtex.texatr;        /* 属性の獲得 */

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

備考　タスク例外処理ルーチン詳細情報 T_RTEX についての詳細は，「16.2.3　タスク例外処理ルーチン詳細情報」を参照してください。

第 6 章　同期通信機能

本章では，RX850V4 が提供している同期通信機能について解説しています。

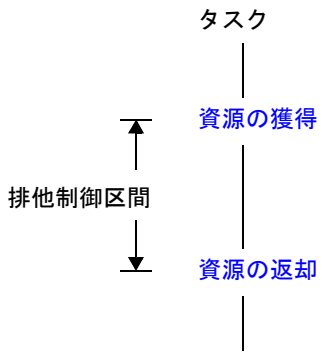
6.1　概　　要

RX850V4 における同期通信機能では，タスク間の排他制御，同期，通信を実現する手段としてセマフォ，イベントフラグ，データ・キュー，メールボックスを提供しています。

6.2　セマフォ

マルチタスク処理では，並行に動作するタスクが限られた数の資源（A/D コンバータ，コプロセッサ，ファイルなど）を同時に使用するといった資源使用の競合を防ぐ機能（排他制御機能）が必要となります。そこで，RX850V4 では，このような資源使用の競合を防ぐ機能として“非負数の計数型セマフォ”を提供しています。
以下に，セマフォを利用した場合の処理の流れを示します。

図 6 － 1　処理の流れ（セマフォ）



6.2.1　セマフォの生成

RX850V4 では，セマフォの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。
セマフォの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“CRE_SEM”を使用してセマフォを定義することをいいます。
静的 API“CRE_SEM”の詳細は，「18.5.3　セマフォ情報」を参照してください。

6.2.2　資源の獲得

資源の獲得は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- wai_sem
  パラメータ *semid* で指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。
  ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，自タスクを対象セマフォの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（資源獲得待ち状態）へと遷移させます。
  なお，資源獲得待ち状態の解除は，以下の場合に行われ，資源獲得待ち状態から READY 状態へと遷移します。

資源獲得待ち状態の解除操作	エラー・コード
sig_sem の発行により，対象セマフォに資源が返却された	E_OK
isig_sem の発行により，対象セマフォに資源が返却された	E_OK
rel_wai の発行により，資源獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，資源獲得待ち状態を強制的に解除された	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      semid = 1;              /* 変数の宣言，初期化 */

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

        ercd = wai_sem ( semid );       /* 資源の獲得 */

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

備考　自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

- pol_sem，ipol_sem

  パラメータ *semid* で指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。
  ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，戻り値として E_TMOUT を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      semid = 1;              /* 変数の宣言，初期化 */

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

        ercd = pol_sem ( semid );       /* 資源の獲得（ポーリング）*/

        if ( ercd == E_OK ) {
                ............            /* ポーリング成功処理 */
                ............
        } else if ( ercd == E_TMOUT ) {
                ............            /* ポーリング失敗処理 */
                ............
        }

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

- twai_sem

パラメータ *semid* で指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。
ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，自タスクを対象セマフォの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（資源獲得待ち状態）へと遷移させます。
なお，資源獲得待ち状態の解除は，以下の場合に行われ，資源獲得待ち状態から READY 状態へと遷移します。

資源獲得待ち状態の解除操作	エラー・コード
sig_sem の発行により，対象セマフォに資源が返却された	E_OK
isig_sem の発行により，対象セマフォに資源が返却された	E_OK
rel_wai の発行により，資源獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，資源獲得待ち状態を強制的に解除された	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      semid = 1;              /* 変数の宣言，初期化 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

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

                                        /* 資源の獲得（タイムアウト付き）*/
        ercd = twai_sem ( semid, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }

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

備考 1　自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　待ち時間 *tmout* に TMO_FEVR が指定された際には“wai_sem と同等の処理”を，TMO_POL が指定された際には“pol_sem，ipol_sem と同等の処理”を実行します。

6.2.3　資源の返却

資源の返却は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sig_sem，isig_sem
  パラメータ *semid* で指定されたセマフォに資源を返却（セマフォ・カウンタに 0x1 を加算）します。
  ただし，本サービス・コールを発行した際，対象セマフォの待ちキューにタスクがキューイングされていた場合には，資源の返却（セマフォ・カウンタの加算処理）は行わず，該当タスク（待ちキューの先頭タスク）に資源を渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（資源獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      semid = 1;              /* 変数の宣言，初期化 */

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

        sig_sem ( semid );              /* 資源の返却 */

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

備考　RX850V4 では，セマフォの資源数として取り得る最大値（最大資源数）をコンフィギュレーション時に定義させています。このため，本サービス・コールでは，資源数が最大資源数を越えるような場合には，資源の返却（セマフォ・カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

6.2.4　セマフォ詳細情報の参照

セマフォ詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_sem，iref_sem
  パラメータ *semid* で指定されたセマフォのセマフォ詳細情報（待ちタスクの有無，現在資源数など）をパラメータ *pk_rsem* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      semid = 1;              /* 変数の宣言，初期化 */
        T_RSEM  pk_rsem;                /* データ構造体の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        UINT    semcnt;                 /* 変数の宣言 */
        ATR     sematr;                 /* 変数の宣言 */
        UINT    maxsem;                 /* 変数の宣言 */

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

        ref_sem ( semid, &pk_rsem );    /* セマフォ詳細情報の参照 */

        wtskid = pk_rsem.wtskid;        /* 待ちタスクの有無の獲得 */
        semcnt = pk_rsem.semcnt;        /* 現在資源数の獲得 */
        sematr = pk_rsem.sematr;        /* 属性の獲得 */
        maxsem = pk_rsem.maxsem;        /* 最大資源数の獲得 */

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

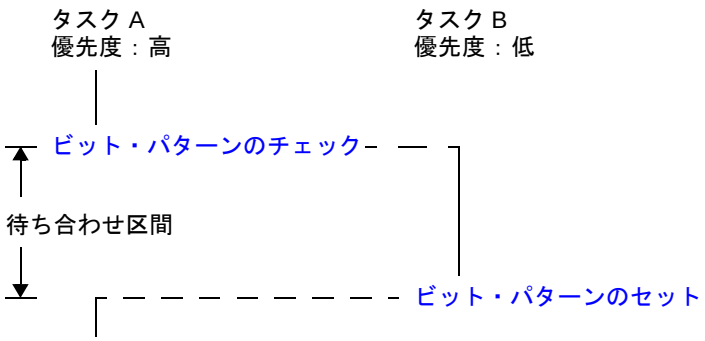
備考　セマフォ詳細情報 T_RSEM についての詳細は，「16.2.4　セマフォ詳細情報」を参照してください。

6.3　イベントフラグ

マルチタスク処理では，あるタスクの処理結果が出るまでの間，他タスクが処理の実行を待つといったタスク間の待ち合わせ機能（事象の発生有無を判断できる機能）が必要となります。そこで，RX850V4 では，このようなタスク間の待ち合わせ機能として“32 ビット幅のイベントフラグ”を提供しています。

以下に，イベントフラグを利用した場合の処理の流れを示します。

図 6 － 2　処理の流れ（イベントフラグ）



6.3.1　イベントフラグの生成

RX850V4 では，イベントフラグの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

イベントフラグの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“CRE_FLG”を使用してイベントフラグを定義することをいいます。

静的 API“CRE_FLG”の詳細は，「18.5.4　イベントフラグ情報」を参照してください。

6.3.2　ビット・パターンのセット

ビット・パターンのセットは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- set_flg，iset_flg

  パラメータ *flgid* で指定されたイベントフラグのビット・パターンとパラメータ *setptn* で指定されたビット・パターンの論理和 OR をとり，その結果を対象イベントフラグにセットします。
  なお，本サービス・コールを発行した際，対象イベントフラグの待ちキューにキューイングされているタスクの要求条件を満足した場合には，ビット・パターンのセット（論理和 OR の設定処理）とともに，該当タスクを待ちキューから外します。これにより，該当タスクは，WAITING 状態（イベントフラグ待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        FLGPTN  setptn = 10;            /* 変数の宣言，初期化 */

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

        set_flg ( flgid, setptn );      /* ビット・パターンのセット */

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

備考　本サービス・コールを発行した際，対象イベントフラグのビット・パターンが B'1100，パラメータ *setptn* で指定されたビット・パターンが B'1010 であった場合には，対象イベントフラグのビット・パターンは B'1110 となります。

6.3.3　ビット・パターンのクリア

ビット・パターンのクリアは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- clr_flg，iclr_flg

  パラメータ *flgid* で指定されたイベントフラグのビット・パターンとパラメータ *clrptn* で指定されたビット・パターンの論理積 AND をとり，その結果を対象イベントフラグに設定します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        FLGPTN  clrptn = 10;            /* 変数の宣言，初期化 */

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

        clr_flg ( flgid, clrptn );      /* ビット・パターンのクリア */

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

備考　本サービス・コールを発行した際，対象イベントフラグのビット・パターンが B'1100，パラメータ *clrptn* で指定されたビット・パターンが B'1010 であった場合には，対象イベントフラグのビット・パターンは B'1000 となります。

6.3.4　ビット・パターンのチェック

ビット・パターンのチェックは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- wai_flg
  パラメータ *waiptn* で指定された要求ビット・パターンとパラメータ *wfmode* で指定された要求条件を満足するビット・パターンがパラメータ *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。
  なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンをパラメータ *p_flgptn* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，自タスクを対象イベントフラグの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（イベントフラグ待ち状態）へと遷移させます。
  なお，イベントフラグ待ち状態の解除は，以下の場合に行われ，イベントフラグ待ち状態から READY 状態へと遷移します。

イベントフラグ待ち状態の解除操作	エラー・コード
set_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された	E_OK
iset_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された	E_OK
rel_wai の発行により，イベントフラグ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，イベントフラグ待ち状態を強制的に解除された	E_RLWAI

以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* = TWF_ANDW
  waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。
- *wfmode* = TWF_ORW
  waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task           task        /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        FLGPTN  waiptn = 14;            /* 変数の宣言，初期化 */
        MODE    wfmode = TWF_ANDW;      /* 変数の宣言，初期化 */
        FLGPTN  p_flgptn;               /* 変数の宣言 */

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

                                        /* ビット・パターンのチェック */
        ercd = wai_flg ( flgid, waiptn, wfmode, &p_flgptn );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
```

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

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

備考 1　RX850V4 では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した場合，RX850V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　自タスクを対象イベントフラグ（TA_WMUL　属性）の待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 3　対象イベントフラグ（TA_CLR　属性）の要求条件が満足した際，RX850V4　はビット・パターンのクリア（0x0 の設定）を行います。

備考 4　rel_wai，または irel_wai の発行によりイベントフラグ待ち状態を解除された場合，パラメータ *p_flgptn* で指定された領域の内容は不定となります。

- pol_flg，ipol_flg
  パラメータ *waiptn* で指定された要求ビット・パターンとパラメータ *wfmode* で指定された要求条件を満足するビット・パターンがパラメータ *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。
  なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンをパラメータ *p_flgptn* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，戻り値として E_TMOUT を返します。
  以下に，要求条件 *wfmode* の指定形式を示します。

  - *wfmode* = TWF_ANDW
    waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。
  - *wfmode* = TWF_ORW
    waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        FLGPTN  waiptn = 14;            /* 変数の宣言，初期化 */
        MODE    wfmode = TWF_ANDW;      /* 変数の宣言，初期化 */
        FLGPTN  p_flgptn;               /* 変数の宣言 */

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

                                        /* ビット・パターンのチェック（ポーリング）*/
        ercd = pol_flg ( flgid, waiptn, wfmode, &p_flgptn );

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

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

備考 1　RX850V4 では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した場合，RX850V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際，RX850V4 はビット・パターンのクリア（0x0 の設定）を行います。

備考 3　本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合，パラメータ *p_flgptn* で指定された領域の内容は不定となります。

- twai_flg

パラメータ *waiptn* で指定された要求ビット・パターンとパラメータ *wfmode* で指定された要求条件を満足するビット・パターンがパラメータ *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。
なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンをパラメータ *p_flgptn* で指定された領域に格納します。
ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，自タスクを対象イベントフラグの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（イベントフラグ待ち状態）へと遷移させます。
なお，イベントフラグ待ち状態の解除は，以下の場合に行われ，イベントフラグ待ち状態から READY 状態へと遷移します。

イベントフラグ待ち状態の解除操作	エラー・コード
set_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された	E_OK
iset_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された	E_OK
rel_wai の発行により，イベントフラグ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，イベントフラグ待ち状態を強制的に解除された	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* = TWF_ANDW
  waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。
- *wfmode* = TWF_ORW
  waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        FLGPTN  waiptn = 14;            /* 変数の宣言，初期化 */
        MODE    wfmode = TWF_ANDW;      /* 変数の宣言，初期化 */
        FLGPTN  p_flgptn;               /* 変数の宣言 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

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

                                        /* ビット・パターンのチェック（タイムアウト付き）*/
        ercd = twai_flg ( flgid, waiptn, wfmode, &p_flgptn, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
```

```
                .............
        } else if ( ercd == E_TMOUT ) {
                .............              /* タイムアウト処理 */
                .............
        }

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

備考 1　RX850V4 では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した場合，RX850V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　自タスクを対象イベントフラグ（TA_WMUL　属性）の待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 3　対象イベントフラグ（TA_CLR　属性）の要求条件が満足した際，RX850V4　はビット・パターンのクリア（0x0 の設定）を行います。

備考 4　rel_wai，または irel_wai の発行，または待ち時間の経過によりイベントフラグ待ち状態を解除された場合，パラメータ *p_flgptn* で指定された領域の内容は不定となります。

備考 5　待ち時間 *tmout* に TMO_FEVR が指定された際には“wai_flg と同等の処理”を，TMO_POL が指定された際には“pol_flg，ipol_flg と同等の処理”を実行します。

6.3.5　イベントフラグ詳細情報の参照

イベントフラグ詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_flg，iref_flg
  パラメータ *flgid* で指定されたイベントフラグのイベントフラグ詳細情報（待ちタスクの有無，現在ビット・パターンなど）をパラメータ *pk_rflg* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      flgid = 1;              /* 変数の宣言，初期化 */
        T_RFLG  pk_rflg;                /* データ構造体の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        FLGPTN  flgptn;                 /* 変数の宣言 */
        ATR     flgatr;                 /* 変数の宣言 */

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

        ref_flg ( flgid, &pk_rflg );    /* イベントフラグ詳細情報の参照 */

        wtskid = pk_rflg.wtskid;        /* 待ちタスクの有無の獲得 */
        flgptn = pk_rflg.flgptn;        /* 現在ビット・パターンの獲得 */
        flgatr = pk_rflg.flgatr;        /* 属性の獲得 */

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

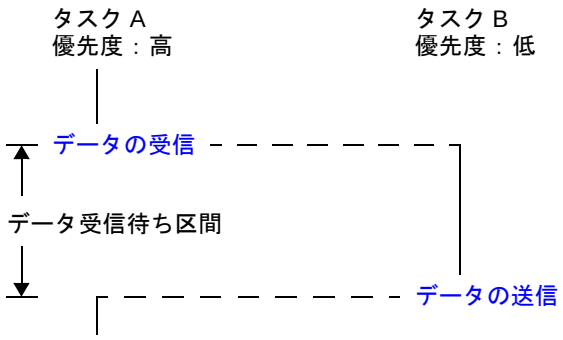
備考　イベントフラグ詳細情報 T_RFLG についての詳細は，「16.2.5　イベントフラグ詳細情報」を参照してください。

6.4　データ・キュー

マルチタスク処理では，あるタスクの処理結果を他タスクに通知するといったタスク間の通信機能（データの受け渡し機能）が必要となります。そこで，RX850V4 では，規定されたデータ・サイズの通信機能として“データの書き込み／読み出しが可能なデータ・キュー領域を有するデータ・キュー”を提供しています。

以下に，データ・キューを利用した場合の処理の流れを示します。

図 6 － 3　処理の流れ（データ・キュー）



備考　1 回のデータ送信／受信処理で送信／受信するデータのサイズは，4 バイトです。

6.4.1　データ・キューの生成

RX850V4 では，データ・キューの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

データ・キューの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“CRE_DTQ”を使用してデータ・キューを定義することをいいます。

静的 API“CRE_DTQ”の詳細は，「18.5.5　データ・キュー情報」を参照してください。

6.4.2　データの送信

データの送信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- snd_dtq
  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域にパラメータ *data* で指定されたデータを書き込みます。
  ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，自タスクを対象データ・キューの送信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（データ送信待ち状態）へと遷移させます。
  なお，データ送信待ち状態の解除は，以下の場合に行われ，データ送信待ち状態から READY 状態へと遷移します。

データ送信待ち状態の解除操作	エラー・コード
rcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
prcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
iprcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
trcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
rel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI

また，本サービス・コールを発行した際，対象データ・キューの受信待ちキューにタスクがキューイングされていた場合には，データの書き込みは行わず，該当タスクにデータを渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  data = 123;             /* 変数の宣言，初期化 */

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

        ercd = snd_dtq ( dtqid, data ); /* データの送信 */

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

備考 1　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

備考 2　自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

- psnd_dtq，ipsnd_dtq

  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域にパラメータ *data* で指定されたデータを書き込みます。
  ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，戻り値として E_TMOUT を返します。
  また，本サービス・コールを発行した際，対象データ・キューの受信待ちキューにタスクがキューイングされていた場合には，データの書き込みは行わず，該当タスクにデータを渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  data = 123;             /* 変数の宣言，初期化 */

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

        ercd = psnd_dtq ( dtqid, data );/* データの送信（ポーリング）*/

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

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

備考　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

- tsnd_dtq

パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域にパラメータ *data* で指定されたデータを書き込みます。
ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，自タスクを対象データ・キューの送信待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（データ送信待ち状態）へと遷移させます。
なお，データ送信待ち状態の解除は，以下の場合に行われ，データ送信待ち状態から READY 状態へと遷移します。

データ送信待ち状態の解除操作	エラー・コード
rcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
prcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
iprcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
trcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
rel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

また，本サービス・コールを発行した際，対象データ・キューの受信待ちキューにタスクがキューイングされていた場合には，データの書き込みは行わず，該当タスクにデータを渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  data = 123;             /* 変数の宣言，初期化 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

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

                                        /* データの送信（タイムアウト付き）*/
        ercd = tsnd_dtq ( dtqid, data, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }

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

備考 1　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

備考 2　自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には“snd_dtq と同等の処理”を，TMO_POL が指定された際には“psnd_dtq，ipsnd_dtq と同等の処理”を実行します。

6.4.3　データの強制送信

データの強制送信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- fsnd_dtq，ifsnd_dtq

  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域にパラメータ *data* で指定されたデータを書き込みます。
  ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，書き込まれてから最も時間が経過しているデータの領域に該当データを上書きします。
  また，本サービス・コールを発行した際，対象データ・キューの受信待ちキューにタスクがキューイングされていた場合には，データの書き込みは行わず，該当タスクにデータを渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  data = 123;             /* 変数の宣言，初期化 */

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

        fsnd_dtq ( dtqid, data );       /* データの強制送信 */

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

6.4.4　データの受信

データの受信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rcv_dtq
  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域からデータを読み込み，パラメータ *p_data* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，自タスクを対象データ・キューの受信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（データ受信待ち状態）へと遷移させます。
  なお，データ受信待ち状態の解除は，以下の場合に行われ，データ受信待ち状態から READY 状態へと遷移します。

データ受信待ち状態の解除操作	エラー・コード
snd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
psnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ipsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
tsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
fsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ifsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
rel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  p_data;                 /* 変数の宣言 */

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

                                        /* データの受信 */
        ercd = rcv_dtq ( dtqid, &p_data );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

備考 1　自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は，データの受信要求を行った順に行われます。

備考 2　rel_wai，または irel_wai の発行によりデータ受信待ち状態を解除された場合，パラメータ *p_data* で指定された領域の内容は不定となります。

- prcv_dtq，iprcv_dtq

  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域からデータを読み込み，パラメータ *p_data* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，戻り値として E_TMOUT を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  p_data;                 /* 変数の宣言 */

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

                                        /* データの受信（ポーリング）*/
        ercd = prcv_dtq ( dtqid, &p_data );

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

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

備考　本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合，パラメータ *p_data* で指定された領域の内容は不定となります。

- trcv_dtq

  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー領域からデータを読み込み，パラメータ *p_data* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，自タスクを対象データ・キューの受信待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（データ受信待ち状態）へと遷移させます。
  なお，データ受信待ち状態の解除は，以下の場合に行われ，データ受信待ち状態から READY 状態へと遷移します。

データ受信待ち状態の解除操作	エラー・コード
snd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
psnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ipsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
tsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
fsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ifsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
rel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        VP_INT  p_data;                 /* 変数の宣言 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

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

                                        /* データの受信（タイムアウト付き）*/
        ercd = trcv_dtq ( dtqid, &p_data, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }

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

備考 1　自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は，データの受信要求を行った順に行われます。

備考 2　rel_wai，または irel_wai の発行，または待ち時間の経過によりデータ受信待ち状態を解除された場合，パラメータ *p_data* で指定された領域の内容は不定となります。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には“rcv_dtq と同等の処理”を，TMO_POL が指定された際には“prcv_dtq，iprcv_dtq と同等の処理”を実行します。

6.4.5　データ・キュー詳細情報の参照

データ・キュー詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_dtq，iref_dtq
  パラメータ *dtqid* で指定されたデータ・キューのデータ・キュー詳細情報（待ちタスクの有無，未受信データの総数など）をパラメータ *pk_rdtq* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

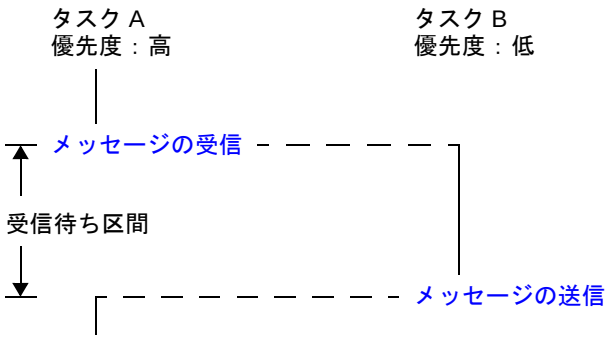
```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      dtqid = 1;              /* 変数の宣言，初期化 */
        T_RDTQ  pk_rdtq;                /* データ構造体の宣言 */
        ID      stskid;                 /* 変数の宣言 */
        ID      rtskid;                 /* 変数の宣言 */
        UINT    sdtqcnt;                /* 変数の宣言 */
        ATR     dtqatr;                 /* 変数の宣言 */
        UINT    dtqcnt;                 /* 変数の宣言 */

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

        ref_dtq ( dtqid, &pk_rdtq );    /* データ・キュー詳細情報の参照 */

        stskid = pk_rdtq.stskid;        /* データ送信待ちタスクの有無の獲得 */
        rtskid = pk_rdtq.rtskid;        /* データ受信待ちタスクの有無の獲得 */
        sdtqcnt = pk_rdtq.sdtqcnt;      /* 未受信データの総数の獲得 */
        dtqatr = pk_rdtq.dtqatr;        /* 属性の獲得 */
        dtqcnt = pk_rdtq.dtqcnt;        /* データ数の獲得 */

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

備考　データ・キュー詳細情報 T_RDTQ についての詳細は，「16.2.6　データ・キュー詳細情報」を参照してください。

6.5　メールボックス

マルチタスク処理では，あるタスクの処理結果を他タスクに通知するといったタスク間の通信機能（メッセージの受け渡し機能）が必要となります。そこで，RX850V4 では，共有されているメモリ領域に書き込まれたメッセージの先頭アドレス受け渡し機能として“メールボックス”を提供しています。
以下に，メールボックスを利用した場合の処理の流れを示します。

図 6 － 4　処理の流れ（メールボックス）



6.5.1　メッセージ

RX850V4 では，処理プログラム間でメールボックスを介してやり取りされる情報を“メッセージ”と呼んでいます。
なお，メッセージは，メールボックスを介することにより任意の処理プログラムに対して送信することができますが，RX850V4 における同期通信機能（メールボックス）では，メッセージの先頭アドレスを受信側処理プログラムに渡すだけであり，メッセージの内容が他領域にコピーされるわけではないので注意が必要です。

- メモリ領域の確保
  RX850V4 では，get_mpf，get_mpl などといったサービス・コールを発行して確保したメモリ領域をメッセージ用に使用することを推奨しています。

  備考　RX850V4 では，メッセージの先頭領域をメールボックスのメッセージ用待ちキューにキューイングする際のリンク領域として使用します。このため，メッセージ用のメモリ領域を RX850V4 が管理しているメモリ領域以外から確保する場合は，4 バイト・アラインされたアドレスから確保する必要があります。

- メッセージの基本型
  RX850V4 では，メッセージの内容，長さについては，利用するメールボックスの属性により，以下の規定を設けています。

  - TA_MFIFO 属性のメールボックスを利用する場合
    RX850V4 では，メッセージの先頭 4 バイト（システム予約領域 msgnext）以降の内容，長さについては特に規定していません。
    したがって，先頭 4 バイト以降の内容，長さについては，TA_MFIFO 属性のメールボックスを利用して情報のやり取りを行う処理プログラム間で規定することになります。
    以下に，TA_MFIFO 属性用メッセージを C 言語で記述する場合の基本型を示します。

    【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  t_msg {
        struct  t_msg   *msgnext;       /* システム予約領域 */
} T_MSG;
```

  - TA_MPRI 属性のメールボックスを利用する場合
    RX850V4 では，メッセージの先頭 8 バイト（システム予約領域 msgque，優先度 msgpri）以降の内容，長さについては特に規定していません。
    したがって，先頭 8 バイト以降の内容，長さについては，TA_MPRI 属性のメールボックスを利用して情報のやり取りを行う処理プログラム間で規定することになります。
    以下に，TA_MPRI 属性用メッセージを C 言語で記述する場合の基本型を示します。

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  t_msg_pri {
        struct  t_msg   msgque;         /* システム予約領域 */
        PRI     msgpri;                 /* 優先度 */
} T_MSG_PRI;
```

備考 1　RX850V4 におけるメッセージの優先度は，その値が小さいほど，高い優先度であることを意味します。

備考 2　メッセージの優先度として指定可能な値は，システム・コンフィギュレーション・ファイル作成時に*メールボックス情報*（*最大メッセージ優先度 maxmpri*）で定義された値域に限定されます。

備考 3　メッセージ T_MSG，T_MSG_PRI についての詳細は，「16.2.7　メッセージ」を参照してください。

6.5.2　メールボックスの生成

RX850V4 では，メールボックスの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

メールボックスの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“CRE_MBX”を使用してメールボックスを定義することをいいます。

静的 API“CRE_MBX”の詳細は，「18.5.6　メールボックス情報」を参照してください。

6.5.3　メッセージの送信

メッセージの送信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- snd_mbx，isnd_mbx
  パラメータ *mbxid* で指定されたメールボックスにパラメータ *pk_msg* で指定されたメッセージを送信します。
  ただし，本サービス・コールを発行した際，対象メールボックスの待ちキューにタスクがキューイングされていた場合には，メッセージの送信（メッセージのキューイング処理）は行わず，該当タスクにメッセージを渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（メッセージ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      mbxid = 1;              /* 変数の宣言，初期化 */
        T_MSG_PRI       *pk_msg;        /* データ構造体の宣言 */

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

        .............                   /* メモリ領域（メッセージ用）の確保 /
        .............

        .............                   /* 本体（内容）の作成 */
        .............

        pk_msg->msgpri = 8;             /* データ構造体の初期化 */

                                        /* メッセージの送信 */
        snd_mbx ( mbxid, ( T_MSG * ) pk_msg );

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

備考 1　メッセージを対象メールボックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　RX850V4 のメールボックスは，メッセージの先頭アドレスを受信側処理プログラムに渡すだけであり，メッセージの内容が他領域にコピーされるわけではありません。したがって，本サービス・コールの発行後であってもメッセージの内容を書き換えることができます。

備考 3　メッセージ T_MSG，T_MSG_PRI についての詳細は，「16.2.7　メッセージ」を参照してください。

6.5.4　メッセージの受信

メッセージの受信は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rcv_mbx
  パラメータ *mbxid* で指定されたメールボックスからメッセージを受信し，その先頭アドレスをパラメータ *ppk_msg* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，自タスクを対象メールボックスの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（メッセージ受信待ち状態）へと遷移させます。
  なお，メッセージ受信待ち状態の解除は，以下の場合に行われ，メッセージ受信待ち状態から READY 状態へと遷移します。

メッセージ受信待ち状態の解除操作	エラー・コード
snd_mbx の発行により，対象メールボックスにメッセージが送信された	E_OK
isnd_mbx の発行により，対象メールボックスにメッセージが送信された	E_OK
rel_wai の発行により，メッセージ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メッセージ受信待ち状態を強制的に解除された	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbxid = 1;              /* 変数の宣言，初期化 */
        T_MSG   *ppk_msg;               /* データ構造体の宣言 */

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

                                        /* メッセージの受信 */
        ercd = rcv_mbx ( mbxid, &ppk_msg );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

備考 1　自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　rel_wai，または irel_wai の発行によりメッセージ受信待ち状態を解除された場合，パラメータ *ppk_msg* で指定された領域の内容は不定となります。

備考 3　メッセージ T_MSG，T_MSG_PRI についての詳細は，「16.2.7　メッセージ」を参照してください。

- prcv_mbx，iprcv_mbx

  パラメータ *mbxid* で指定されたメールボックスからメッセージを受信し，その先頭アドレスをパラメータ *ppk_msg* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，戻り値として E_TMOUT を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbxid = 1;              /* 変数の宣言，初期化 */
        T_MSG   *ppk_msg;               /* データ構造体の宣言 */

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

                                        /* メッセージの受信（ポーリング）*/
        ercd = prcv_mbx ( mbxid, &ppk_msg );

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

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

備考 1　本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合，パラメータ *ppk_msg* で指定された領域の内容は不定となります。

備考 2　メッセージ T_MSG，T_MSG_PRI についての詳細は，「16.2.7　メッセージ」を参照してください。

- trcv_mbx

パラメータ *mbxid* で指定されたメールボックスからメッセージを受信し，その先頭アドレスをパラメータ *ppk_msg* で指定された領域に格納します。
ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，自タスクを対象メールボックスの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（メッセージ受信待ち状態）へと遷移させます。
なお，メッセージ受信待ち状態の解除は，以下の場合に行われ，メッセージ受信待ち状態から READY 状態へと遷移します。

メッセージ受信待ち状態の解除操作	エラー・コード
snd_mbx の発行により，対象メールボックスにメッセージが送信された	E_OK
isnd_mbx の発行により，対象メールボックスにメッセージが送信された	E_OK
rel_wai の発行により，メッセージ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メッセージ受信待ち状態を強制的に解除された	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mbxid = 1;              /* 変数の宣言，初期化 */
        T_MSG   *ppk_msg;               /* データ構造体の宣言 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

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

                                        /* メッセージの受信（タイムアウト付き）*/
        ercd = trcv_mbx ( mbxid, &ppk_msg, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }

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

備考 1　自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　rel_wai，または irel_wai の発行，または待ち時間の経過によりメッセージ受信待ち状態を解除された場合，パラメータ *ppk_msg* で指定された領域の内容は不定となります。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には“rcv_mbx と同等の処理”を，TMO_POL が指定された際には“prcv_mbx，iprcv_mbx と同等の処理”を実行します。

備考 4　メッセージ T_MSG，T_MSG_PRI についての詳細は，「16.2.7　メッセージ」を参照してください。

6.5.5　メールボックス詳細情報の参照

メールボックス詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_mbx，iref_mbx
  パラメータ *mbxid* で指定されたメールボックスのメールボックス詳細情報（待ちタスクの有無，待ちメッセージの有無など）をパラメータ *pk_rmbx* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      mbxid = 1;              /* 変数の宣言，初期化 */
        T_RMBX  pk_rmbx;                /* データ構造体の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        T_MSG   *pk_msg;                /* データ構造体の宣言 */
        ATR     mbxatr;                 /* 変数の宣言 */

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

        ref_mbx ( mbxid, &pk_rmbx );    /* メールボックス詳細情報の参照 */

        wtskid = pk_rmbx.wtskid;        /* 待ちタスクの有無の獲得 */
        pk_msg = pk_rmbx.pk_msg;        /* 待ちメッセージの有無の獲得 */
        mbxatr = pk_rmbx.mbxatr;        /* 属性の獲得 */

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

備考　メールボックス詳細情報 T_RMBX についての詳細は，「16.2.8　メールボックス詳細情報」を参照してください。

第 7 章　拡張同期通信機能

本章では，RX850V4 が提供している拡張同期通信機能について解説しています。

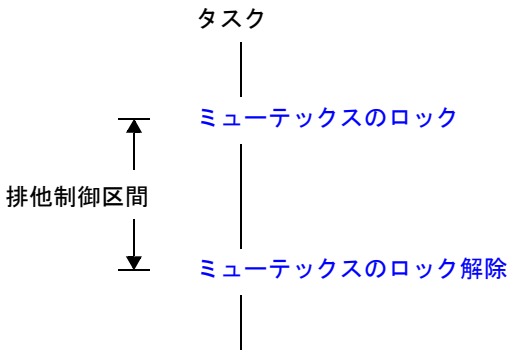
7.1　概　　要

RX850V4 における拡張同期通信機能では，タスク間の排他制御を実現する手段としてミューテックスを提供しています。

7.2　ミューテックス

マルチタスク処理では，並行に動作するタスクが限られた数の資源（A/D コンバータ，コプロセッサ，ファイルなど）を同時に使用するといった資源使用の競合を防ぐ機能（排他制御機能）が必要となります。そこで，RX850V4 では，このような資源使用の競合を防ぐ機能として“ミューテックス”を提供しています。
以下に，ミューテックスを利用した場合の処理の流れを示します。
RX850V4 のミューテックスは優先度継承プロトコル，優先度上限プロトコルをサポートしていません。FIFO 順，優先度順のみサポートしています。

図 7 － 1　処理の流れ（ミューテックス）



7.2.1　セマフォとの相違点

RX850V4 のミューテックスは優先度継承プロトコル，優先度上限プロトコルをサポートしていないので最大資源数が 1 つのセマフォ（バイナリ・セマフォ）と似た動作をしますが，以下のような違いがあります。

- ミューテックスのロック解除（資源返却に相当）できるのはミューテックスをロックしたタスクのみ
  →　セマフォはどのタスク／ハンドラからでも資源の返却が可能

- ミューテックスをロックしているタスクを終了する（ext_tsk，ter_tsk）際に，自動的にロック解除処理が行われる
  →　セマフォは自動的に資源の返却を行わないので資源を獲得したまま終了してしまう

- セマフォは複数の資源を管理できる（最大資源数を指摘できる）がミューテックスの資源最大数に相当する値は 1 固定

7.2.2　ミューテックスの生成

RX850V4 では，ミューテックスの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。
ミューテックスの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“CRE_MTX”を使用してミューテックスを定義することをいいます。
静的 API“CRE_MTX”の詳細は，「18.5.7　ミューテックス情報」を参照してください。

7.2.3　ミューテックスのロック

ミューテックスのロックは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- loc_mtx
  パラメータ *mtxid* で指定されたミューテックスをロックします。
  ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，自タスクを対象ミューテックスの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（ミューテックス待ち状態）へと遷移させます。
  なお，ミューテックス待ち状態の解除は，以下の場合に行われ，ミューテックス待ち状態から READY 状態へと遷移します。

ミューテックス待ち状態の解除操作	エラー・コード
unl_mtx の発行により，対象ミューテックスのロック状態が解除された	E_OK
ext_tsk の発行により，対象ミューテックスのロック状態が解除された	E_OK
ter_tsk の発行により，対象ミューテックスのロック状態が解除された	E_OK
rel_wai の発行により，ミューテックス待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，ミューテックス待ち状態を強制的に解除された	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mtxid = 8;              /* 変数の宣言，初期化 */

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

        ercd = loc_mtx ( mtxid );       /* ミューテックスのロック */

        if ( ercd == E_OK ) {
                .............           /* ロック状態 */
                .............

                unl_mtx ( mtxid );      /* ミューテックスのロック解除 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

備考 1　自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　RX850V4 では，自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

- ploc_mtx
  パラメータ *mtxid* で指定されたミューテックスをロックします。
  ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，戻り値として E_TMOUT を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mtxid = 8;              /* 変数の宣言，初期化 */

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

        ercd = ploc_mtx ( mtxid );      /* ミューテックスのロック（ポーリング）*/

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............

                unl_mtx ( mtxid );      /* ミューテックスのロック解除 */
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

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

備考　RX850V4 では，自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

- tloc_mtx
  パラメータ *mtxid* で指定されたミューテックスをロックします。
  ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，自タスクを対象ミューテックスの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（ミューテックス待ち状態）へと遷移させます。
  なお，ミューテックス待ち状態の解除は，以下の場合に行われ，ミューテックス待ち状態から READY 状態へと遷移します。

ミューテックス待ち状態の解除操作	エラー・コード
unl_mtx の発行により，対象ミューテックスのロック状態が解除された	E_OK
ext_tsk の発行により，対象ミューテックスのロック状態が解除された	E_OK
ter_tsk の発行により，対象ミューテックスのロック状態が解除された	E_OK
rel_wai の発行により，ミューテックス待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，ミューテックス待ち状態を強制的に解除された	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mtxid = 8;              /* 変数の宣言，初期化 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

        .............
        .............
                                        /* ミューテックスのロック（タイムアウト付き）*/
        ercd = tloc_mtx ( mtxid, tmout );

        if ( ercd == E_OK ) {
                .............           /* ロック状態 */
                .............

                unl_mtx ( mtxid );      /* ミューテックスのロック解除 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }

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

備考 1　自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　RX850V4 では，自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には“loc_mtx と同等の処理”を，TMO_POL が指定された際には“ploc_mtx と同等の処理”を実行します。

7.2.4　ミューテックスのロック解除

ミューテックスのロック解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- unl_mtx
  パラメータ *mtxid* で指定されたミューテックスのロック状態を解除します。
  ただし，本サービス・コールを発行した際，対象ミューテックスの待ちキューにタスクがキューイングされていた場合には，ミューテックスのロック解除処理後，ただちに該当タスク（待ちキューの先頭タスク）によるミューテックスのロック処理が行われます。
  このとき，該当タスクは，待ちキューから外れ，WAITING 状態（ミューテックス待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mtxid = 8;              /* 変数の宣言，初期化 */

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

        ercd = loc_mtx ( mtxid );       /* ミューテックスのロック */

        if ( ercd == E_OK ) {
                .............           /* ロック状態 */
                .............

                unl_mtx ( mtxid );      /* ミューテックスのロック解除 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

備考　ミューテックスのロック解除が可能なタスクは“対象ミューテックスをロックしたタスク”に限られます。このため，自タスクがロックしていないミューテックスに対して本サービス・コールを発行した場合には，何も処理は行わず，戻り値として E_ILUSE を返します。

7.2.5　ミューテックス詳細情報の参照

ミューテックス詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_mtx，iref_mtx
  パラメータ *mtxid* で指定されたミューテックスのミューテックス詳細情報（ロックの有無，待ちタスクの有無など）をパラメータ *pk_rmtx* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      mtxid = 1;              /* 変数の宣言，初期化 */
        T_RMTX  pk_rmtx;                /* データ構造体の宣言 */
        ID      htskid;                 /* 変数の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        ATR     mtxatr;                 /* 変数の宣言 */

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

        ref_mtx ( mbxid, &pk_rmtx );    /* ミューテックス詳細情報の参照 */

        htskid = pk_rmtx.htskid;        /* ロックの有無の獲得 */
        wtskid = pk_rmtx.wtskid;        /* 待ちタスクの有無の獲得 */
        mtxatr = pk_rmtx.mtxatr;        /* 属性の獲得 */

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

備考　ミューテックス詳細情報 T_RMTX についての詳細は，「16.2.9　ミューテックス詳細情報」を参照してください。

第 8 章　メモリ・プール管理機能

本章では，RX850V4 が提供しているメモリ・プール管理機能について解説しています。

8.1　概　　要

RX850V4 におけるメモリ・プール管理機能では，カーネル初期化部において静的に確保／初期化されたメモリ領域を管理対象としています。

なお，RX850V4 では，静的に確保／初期化されたメモリ領域の一部を“固定長メモリ・プール”，または“可変長メモリ・プール”として解放することにより，固定長メモリ・ブロックの獲得／解放，可変長メモリ・ブロックの獲得／解放などといったメモリ領域を動的に操作する機能のほかに，固定長メモリ・プール詳細情報の参照，可変長メモリ・プール詳細情報の参照などといったメモリ領域の状態を参照する機能も提供しています。

8.2　固定長メモリ・プール

RX850V4 では，処理プログラムから動的なメモリ操作要求が行われた際に利用するメモリ領域として“固定長メモリ・プール”を提供しています。

なお，固定長メモリ・プールに対する動的なメモリ操作は，固定サイズの固定長メモリ・ブロックを単位として行われます。

8.2.1　固定長メモリ・プールの生成

RX850V4 では，固定長メモリ・プールの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

固定長メモリ・プールの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“CRE_MPF”を使用して固定長メモリ・プールを定義することをいいます。

静的 API“CRE_MPF”の詳細は，「18.5.8　固定長メモリ・プール情報」を参照してください。

8.2.2　固定長メモリ・ブロックの獲得

固定長メモリ・ブロックの獲得は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- get_mpf
  パラメータ *mpfid* で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。
  なお，固定長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われ，固定長メモリ・ブロック獲得待ち状態から READY 状態へと遷移します。

固定長メモリ・ブロック獲得待ち状態の解除操作	エラー・コード
rel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された	E_OK
irel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された	E_OK
rel_wai の発行により，固定長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，固定長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mpfid = 1;              /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */

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

                                        /* 固定長メモリ・ブロックの獲得 */
        ercd = get_mpf ( mpfid, &p_blk );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
                                        /* 固定長メモリ・ブロックの返却 */
                rel_mpf ( mpfid, p_blk );
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

備考 1　自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　rel_wai，または irel_wai の発行により固定長メモリ・ブロック獲得待ち状態を解除された場合，パラメータ *p_blk* で指定された領域の内容は不定となります。

- pget_mpf，ipget_mpf
  パラメータ *mpfid* で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，戻り値として E_TMOUT を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mpfid = 1;              /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */

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

                                        /* 固定長メモリ・ブロックの獲得（ポーリング）*/
        ercd = pget_mpf ( mpfid, &p_blk );

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............

                                        /* 固定長メモリ・ブロックの返却 */
                rel_mpf ( mpfid, p_blk );
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

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

備考　本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合，パラメータ *p_blk* で指定された領域の内容は不定となります。

- tget_mpf

パラメータ *mpfid* で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。
なお，固定長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われ，固定長メモリ・ブロック獲得待ち状態から READY 状態へと遷移します。

固定長メモリ・ブロック獲得待ち状態の解除操作	エラー・コード
rel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された	E_OK
irel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された	E_OK
rel_wai の発行により，固定長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，固定長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mpfid = 1;              /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

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

                                        /* 固定長メモリ･ブロックの獲得(タイムアウト付き)*/
        ercd = tget_mpf ( mpfid, &p_blk, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............

                                        /* 固定長メモリ・ブロックの返却 */
                rel_mpf ( mpfid, p_blk );
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }

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

備考 1　自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　rel_wai，または irel_wai の発行，または待ち時間の経過により固定長メモリ・ブロック獲得待ち状態を解除された場合，パラメータ *p_blk* で指定された領域の内容は不定となります。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には“get_mpf と同等の処理”を，TMO_POL が指定された際には“pget_mpf，ipget_mpf と同等の処理”を実行します。

8.2.3　固定長メモリ・ブロックの返却

固定長メモリ・ブロックの返却は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rel_mpf，irel_mpf
  パラメータ *mpfid* で指定された固定長メモリ・プールにパラメータ *blk* で指定された固定長メモリ・ブロックを返却します。
  ただし，本サービス・コールを発行した際，対象固定長メモリ・プールの待ちキューにタスクがキューイングされていた場合には，固定長メモリ・ブロックの返却は行わず，該当タスク（待ちキューの先頭タスク）に固定長メモリ・ブロックを渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（固定長メモリ・ブロック獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mpfid = 1;              /* 変数の宣言，初期化 */
        VP      blk;                    /* 変数の宣言 */

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

        ercd = get_mpf ( mpfid, &blk ); /* 固定長メモリ・ブロックの獲得 */

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............

                rel_mpf ( mpfid, blk ); /* 固定長メモリ・ブロックの返却 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

備考 1　RX850V4 では，固定長メモリ・ブロックを返却する際，メモリ・クリア処理を行っていません。したがって，返却された固定長メモリ・ブロックの内容は不定となります。

備考 2　固定長メモリ・ブロックを返却する際は，必ず獲得した固定長メモリ・プールに対して本サービス・コールを発行してください。異なる固定長メモリ・プールに対して本サービス・コールを発行してもエラーにはなりませんが，以後の動作は保証されません。

8.2.4　固定長メモリ・プール詳細情報の参照

固定長メモリ・プール詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_mpf，iref_mpf
  パラメータ *mpfid* で指定された固定長メモリ・プールの固定長メモリ・プール詳細情報（待ちタスクの有無，空き固定長メモリ・ブロックの総数など）をパラメータ *pk_rmpf* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      mpfid = 1;              /* 変数の宣言，初期化 */
        T_RMPF  pk_rmpf;                /* データ構造体の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        UINT    fblkcnt;                /* 変数の宣言 */
        ATR     mpfatr;                 /* 変数の宣言 */

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

        ref_mpf ( mpfid, &pk_rmpf );    /* 固定長メモリ・プール詳細情報の参照 */

        wtskid = pk_rmpf.wtskid;        /* 待ちタスクの有無の獲得 */
        fblkcnt = pk_rmpf.fblkcnt;      /* 空き固定長メモリ・ブロックの総数の獲得 */
        mpfatr = pk_rmpf.mpfatr;        /* 属性の獲得 */

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

備考　固定長メモリ・プール詳細情報 T_RMPF についての詳細は，「16.2.10　固定長メモリ・プール詳細情報」を参照してください。

8.3　可変長メモリ・プール

RX850V4 では，処理プログラムから動的なメモリ操作要求が行われた際に利用するメモリ領域として“可変長メモリ・プール”を提供しています。

なお，可変長メモリ・プールに対する動的なメモリ操作は，任意サイズの可変長メモリ・ブロックを単位として行われます。

8.3.1　可変長メモリ・プールの生成

RX850V4 では，可変長メモリ・プールの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

可変長メモリ・プールの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“CRE_MPL”を使用して可変長メモリ・プールを定義することをいいます。

静的 API“CRE_MPL”の詳細は，「18.5.9　可変長メモリ・プール情報」を参照してください。

8.3.2　可変長メモリ・ブロックの獲得

可変長メモリ・ブロックの獲得は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- get_mpl
  パラメータ *mplid* で指定された可変長メモリ・プールからパラメータ *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（可変長メモリ・ブロック獲得待ち状態）へと遷移させます。
  なお，可変長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われ，可変長メモリ・ブロック獲得待ち状態から READY 状態へと遷移します。

可変長メモリ・ブロック獲得待ち状態の解除操作	エラー・コード
rel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された	E_OK
irel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された	E_OK
rel_wai の発行により，可変長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，可変長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mplid = 1;              /* 変数の宣言，初期化 */
        UINT    blksz = 256;            /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */

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

                                        /* 可変長メモリ・ブロックの獲得 */
        ercd = get_mpl ( mplid, blksz, &p_blk );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............
                                        /* 可変長メモリ・ブロックの返却 */
                rel_mpl ( mplid, p_blk );
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

備考 1　RX850V4 では，可変長メモリ・ブロックの獲得処理を“4 の整数倍値”を単位として行います。したがって，パラメータ *blksz* に 4 の整数倍値以外の値が指定された場合には，4 の整数倍値に繰り上げられます。

備考 2　自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 3　rel_wai，または irel_wai の発行により可変長メモリ・ブロック獲得待ち状態を解除された場合，パラメータ *p_blk* で指定された領域の内容は不定となります。

- pget_mpl，ipget_mpl
  パラメータ *mplid* で指定された可変長メモリ・プールからパラメータ *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
  ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，戻り値として E_TMOUT を返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mplid = 1;              /* 変数の宣言，初期化 */
        UINT    blksz = 256;            /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */

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

                                        /* 可変長メモリ・ブロックの獲得（ポーリング）*/
        ercd = pget_mpl ( mplid, blksz, &p_blk );

        if ( ercd == E_OK ) {
                .............           /* ポーリング成功処理 */
                .............

                                        /* 可変長メモリ・ブロックの返却 */
                rel_mpl ( mplid, p_blk );
        } else if ( ercd == E_TMOUT ) {
                .............           /* ポーリング失敗処理 */
                .............
        }

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

備考 1　RX850V4 では，可変長メモリ・ブロックの獲得処理を“4 の整数倍値”を単位として行います。したがって，パラメータ *blksz* に 4 の整数倍値以外の値が指定された場合には，4 の整数倍値に繰り上げられます。

備考 2　本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合，パラメータ *p_blk* で指定された領域の内容は不定となります。

- tget_mpl

パラメータ *mplid* で指定された可変長メモリ・プールからパラメータ *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスをパラメータ *p_blk* で指定された領域に格納します。
ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（可変長メモリ・ブロック獲得待ち状態）へと遷移させます。
なお，可変長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われ，可変長メモリ・ブロック獲得待ち状態から READY 状態へと遷移します。

可変長メモリ・ブロック獲得待ち状態の解除操作	エラー・コード
rel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された	E_OK
irel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された	E_OK
rel_wai の発行により，可変長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，可変長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
パラメータ *tmout* で指定された待ち時間が経過した	E_TMOUT

以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mplid = 1;              /* 変数の宣言，初期化 */
        UINT    blksz = 256;            /* 変数の宣言，初期化 */
        VP      p_blk;                  /* 変数の宣言 */
        TMO     tmout = 3600;           /* 変数の宣言，初期化 */

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

                                        /* 可変長メモリ･ブロックの獲得(タイムアウト付き)*/
        ercd = tget_mpl ( mplid, blksz, &p_blk, tmout );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............

                                        /* 可変長メモリ・ブロックの返却 */
                rel_mpl ( mplid, p_blk );
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        } else if ( ercd == E_TMOUT ) {
                .............           /* タイムアウト処理 */
                .............
        }

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

備考 1　RX850V4 では，可変長メモリ・ブロックの獲得処理を“4 の整数倍値”を単位として行います。したがって，パラメータ *blksz* に 4 の整数倍値以外の値が指定された場合には，4 の整数倍値に繰り上げられます。

備考 2　自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 3　rel_wai，または irel_wai の発行，または待ち時間の経過により可変長メモリ・ブロック獲得待ち状態を解除された場合，パラメータ *p_blk* で指定された領域の内容は不定となります。

備考 4　待ち時間 *tmout* に TMO_FEVR が指定された際には“get_mpl と同等の処理”を，TMO_POL が指定された際には“pget_mpl，ipget_mpl と同等の処理”を実行します。

8.3.3　可変長メモリ・ブロックの返却

可変長メモリ・ブロックの返却は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rel_mpl，irel_mpl

  パラメータ *mplid* で指定された可変長メモリ・プールにパラメータ *blk* で指定された可変長メモリ・ブロックを返却します。
  可変長メモリ・ブロックを返却したあと，対象可変長メモリ・プールの待ちキューにキューイングされているタスクをキューの先頭から調べていき，待ちタスクが要求するサイズのメモリを割り当てられる場合はメモリを割り当てます。この動作を待ちキューにタスクがなくなるか，メモリが割り当てられなくなるまで繰り返します。これにより，メモリを獲得できたタスクは，待ちキューから外れ，WAITING 状態（可変長メモリ・ブロック獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER      ercd;                   /* 変数の宣言 */
        ID      mplid = 1;              /* 変数の宣言，初期化 */
        UINT    blksz = 256;            /* 変数の宣言，初期化 */
        VP      blk;                    /* 変数の宣言 */

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

                                        /* 可変長メモリ・ブロックの獲得 */
        ercd = get_mpl ( mplid, blksz, &blk );

        if ( ercd == E_OK ) {
                .............           /* 正常終了処理 */
                .............

                rel_mpl ( mplid, blk ); /* 可変長メモリ・ブロックの返却 */
        } else if ( ercd == E_RLWAI ) {
                .............           /* 強制終了処理 */
                .............
        }

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

備考 1　RX850V4 では，可変長メモリ・ブロックを返却する際，メモリ・クリア処理を行っていません。したがって，返却された可変長メモリ・ブロックの内容は不定となります。

備考 2　可変長メモリ・ブロックを返却する際は，必ず獲得した可変長メモリ・プールに対して本サービス・コールを発行してください。異なる可変長メモリ・プールに対して本サービス・コールを発行してもエラーにはなりませんが，以後の動作は保証されません。

8.3.4　可変長メモリ・プール詳細情報の参照

可変長メモリ・プール詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_mpl，iref_mpl
  パラメータ *mplid* で指定された可変長メモリ・プールの可変長メモリ・プール詳細情報（待ちタスクの有無，空き可変長メモリ・ブロックの合計サイズなど）をパラメータ *pk_rmpl* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      mplid = 1;              /* 変数の宣言，初期化 */
        T_RMPL  pk_rmpl;                /* データ構造体の宣言 */
        ID      wtskid;                 /* 変数の宣言 */
        SIZE    fmplsz;                 /* 変数の宣言 */
        UINT    fblksz;                 /* 変数の宣言 */
        ATR     mplatr;                 /* 変数の宣言 */

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

        ref_mpl ( mplid, &pk_rmpl );    /* 可変定長メモリ・プール詳細情報の参照 */

        wtskid = pk_rmpl.wtskid;        /* 待ちタスクの有無の獲得 */
        fmplsz = pk_rmpl.fmplsz;        /* 空き可変長メモリ・ブロックの合計サイズの獲得 */
        fblksz = pk_rmpl.fblksz;        /* 空き可変長メモリ・ブロックの最大サイズの獲得 */
        mplatr = pk_rmpl.mplatr;        /* 属性の獲得 */

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

備考　可変長メモリ・プール詳細情報 T_RMPL についての詳細は，「16.2.11　可変長メモリ・プール詳細情報」を参照してください。

第 9 章　時間管理機能

本章では，RX850V4 が提供している時間管理機能について解説しています。

9.1　概　　要

RX850V4 における時間管理機能では，一定周期で発生する基本クロック用タイマ割り込みを利用してシステム時刻を操作／参照する機能のほかに，時間に依存した処理を実現する手段（タイマ・オペレーション機能：遅延起床，タイムアウト，周期ハンドラ）を提供しています。

9.2　システム時刻

システム時刻とは，RX850V4 が時間管理を行う際に使用する“時間（単位：ミリ秒）”です。

なお，システム時刻は，カーネル初期化部において，初期化されたのち，システム・コンフィギュレーション・ファイル作成時に基本情報（*基本クロック周期 tim_base*）で定義された基本クロック周期ごとに更新されます。

9.2.1　基本クロック用タイマ割り込み

RX850V4 では，時間管理機能を実現するために，一定周期で発生する割り込み（基本クロック用タイマ割り込み）を使用します。

基本クロック用タイマ割り込みが発生した際は，RX850V4 の時間に関連した処理（システム時刻の更新，タスクのタイムアウト／遅延，周期ハンドラの起動など）が実行されます。

基本クロック用タイマ割り込みの割り込み要因は，システム・コンフィギュレーション・ファイルのシステム情報 CLK_INTNO で指定します。

システム情報 CLK_INTNO の詳細については，「18.4.2　基本情報」を参照してください。

基本クロック用タイマ割り込みを発生するためのハードウエアの初期化はRX850V4は行いませんのでユーザが行う必要があります。

ブート処理，または初期化ルーチンで使用するハードウエアの初期化，割り込みマスクの解除を行ってください。

備考　基本クロック用タイマ割り込みに対応した処理は，基本クロック用タイマ割り込みの発生をトリガに開始されますが，該当処理内の ISPRn（基本クロック用タイマ割り込みの優先順位 n に該当するビット）は 0 に設定されます。したがって，基本クロック用タイマ割り込みに対応した処理中に，基本クロック用タイマ割り込み自身や基本クロック用タイマ割り込みより優先順位の低い割り込みが発生した際には，該当割り込みは受け付けられます。

9.2.2　基本クロック周期

RX850V4 のサービス・コールでは時間指定パラメータの単位は「ミリ秒」になっています。

基本クロック用タイマ割り込みの発生周期は 1 ミリ秒とするのが望ましいですが，ターゲット・システムの性質上（処理能力，必要とする時間分解能など）1 ミリ秒とすることが困難な場合があります。

この場合，システム・コンフィギュレーション・ファイルのシステム情報 DEF_TIM で基本クロック用タイマ割り込みの発生周期を指定することができます。

システム情報 DEF_TIM の詳細については，「18.4.2　基本情報」を参照してください。

基本クロック周期を指定すると 1 回の基本クロック用タイマ割り込みで基本クロック周期分の時間が経過したとして処理されます。

基本クロック周期は 1 以上の整数のみ指定可能です。「2.5」のような小数は指定できません。

9.3　タイマ・オペレーション機能

RX850V4 におけるタイマ・オペレーション機能では，時間に依存した処理を実現する手段として“遅延起床，タイムアウト，周期ハンドラ”を提供しています。

9.3.1　遅延起床

遅延起床とは，一定の時間が経過するまでの間，自タスクを RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させ，一定の時間が経過した際には，該当タスクを WAITING 状態から READY 状態へと遷移させるものです。
なお，遅延起床は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

dly_tsk

9.3.2　タイムアウト

タイムアウトとは，タスクから発行された要求条件が即時成立しなかった場合，一定の時間が経過するまでの間，該当タスクを RUNNING 状態から WAITING 状態（起床待ち状態，資源獲得待ち状態，イベントフラグ待ち状態など）へと遷移させ，一定の時間が経過した際には，要求条件の成立／不成立を問わず，該当タスクを WAITING 状態から READY 状態へと遷移させるものです。
なお，タイムアウトは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

tslp_tsk	twai_sem	twai_flg	tsnd_dtq
trcv_dtq	trcv_mbx	tloc_mtx	tget_mpf
tget_mpl			

9.3.3　周期ハンドラ

周期ハンドラは，一定の時間ごとに周期的に起動される周期処理専用ルーチンです。
なお，RX850V4 では，周期ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。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処理を実行中であっても，その処理は中断され，周期ハンドラに制御が移ります。
また，RX850V4 では，周期ハンドラを管理するに当たり，周期ハンドラと一対一に対応した管理オブジェクト（周期ハンドラ管理ブロック）を用いることにより，周期ハンドラが取り得る状態の管理，および，周期ハンドラ自体の管理を行っています。

- 周期ハンドラの基本型
  周期ハンドラを記述する場合，VP_INT 型の引き数を 1 つ持った void 型の関数として記述します。
  なお，引き数 *exinf* には“周期ハンドラ情報で指定した拡張情報”が設定されます。
  以下に，周期ハンドラを C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
cychdr ( VP_INT exinf )
{
        .............
        .............

        return;                         /* 周期ハンドラの終了 */
}
```

- 記述方法
  C 言語，またはアセンブリ言語で記述します。
  C 言語で記述する際は通常の void 型関数と同様に記述することができます。
  アセンブリ言語で記述する際は使用するコンパイラの呼び出し規約にのっとって作成してください。

- スタックの切り替え
  RX850V4 では，周期ハンドラに制御を移す際，基本情報において指定されたシステム・スタックへの切り替え処理

を，基本クロック用タイマ割り込みが発生した処理プログラムに制御を戻す際に該当スタックへの切り替え処理を行っています。したがって，周期ハンドラ中はシステム・スタックが使用されます。

- サービス・コールの発行
RX850V4 では，周期ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため，周期ハンドラでは，“非タスク内から発行可能なサービス・コール”のみが発行可能となります。

備考 1　RX850V4 では，周期ハンドラ内の処理を高速に終了させる目的から，周期ハンドラ内の処理が完了するまでの間，スケジューラの起動を遅延しています。したがって，周期ハンドラ内でディスパッチ処理（タスクのスケジューリング処理）を伴うサービス・コール（isig_sem，iset_flg など）が発行された際には，キュー操作などといった処理が行われるだけであり，実際のディスパッチ処理の実行は“周期ハンドラからの復帰命令（return 命令の発行）”が発行されるまで遅延され，一括して行うようにしています。

備考 2　各サービス・コールの発行有効範囲についての詳細は，表 17 － 1 ～表 17 － 14 を参照してください。

- マスカブル割り込みの受け付け状態（PSW の ID フラグ）
処理開始時，マスカブル割り込みの受け付け許可状態（PSW の ID フラグは 0）となります。
処理内では，マスカブル割り込みの受け付け状態を変更することができます。変更した状態は，処理終了後に制御が移った処理プログラムへは継承されません。

備考 1　周期ハンドラは，基本クロック用タイマ割り込みの発生をトリガに開始されますが，該当処理内の ISPRn（基本クロック用タイマ割り込みの優先順位 n に該当するビット）は 0 に設定されます。したがって，周期ハンドラ内で基本クロック用タイマ割り込み自身や基本クロック用タイマ割り込みより優先順位の低い割り込みが発生した際には，該当割り込みは受け付けられます。

備考 2　周期ハンドラ内で基本クロック用タイマ割り込みが受け付けられ，更に該当周期ハンドラの周期時間に達した際には，周期ハンドラが多重起動することになります。

備考 3　周期ハンドラ内でマスカブル割り込みの受け付けを完全に禁止することはできません。周期ハンドラの開始後に PSW の ID フラグを 1 に設定することで，マスカブル割り込みの受け付けを禁止することは可能ですが，周期ハンドラの開始からマスカブル割り込みの受け付けを禁止するまでの区間に，マスカブル割り込みが受け付けられる可能性があります。

9.3.4　周期ハンドラの生成

RX850V4 では，周期ハンドラの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。

周期ハンドラの静的生成とは，システム・コンフィギュレーション・ファイルで静的 API“CRE_CYC”を使用して周期ハンドラを定義することをいいます。

静的 API“CRE_CYC”の詳細は，「18.5.10　周期ハンドラ情報」を参照してください。

9.4　システム時刻の設定

システム時刻の設定は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- set_tim，iset_tim
  RX850V4 のシステム時刻（単位：ミリ秒）をパラメータ *p_systim* で指定された時間に変更します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        SYSTIM  p_systim;               /* データ構造体の宣言 */

        p_systim.ltime = 3600;          /* データ構造体の初期化 */
        p_systim.utime = 0;             /* データ構造体の初期化 */

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

        set_tim ( &p_systim );          /* システム時刻の設定 */

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

備考　　システム時刻情報 SYSTIM についての詳細は，「16.2.12　システム時刻情報」を参照してください。

9.5　システム時刻の参照

システム時刻の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- get_tim，iget_tim
  RX850V4 のシステム時刻（単位：ミリ秒）をパラメータ *p_systim* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        SYSTIM  p_systim;               /* データ構造体の宣言 */
        UW      ltime;                  /* 変数の宣言 */
        UH      utime;                  /* 変数の宣言 */

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

        get_tim ( &p_systim );          /* システム時刻の参照 */

        ltime = p_systim.ltime;         /* システム時刻（下位 32 ビット）の獲得 */
        utime = p_systim.utime;         /* システム時刻（上位 16 ビット）の獲得 */

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

備考 1　RX850V4 では，システム時刻として表現できない数値（48 ビット幅からオーバフローした数値）については無視しています。

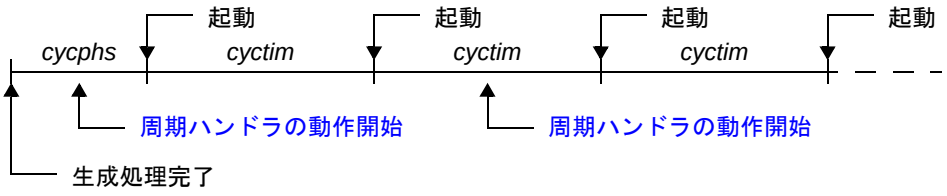
備考 2　システム時刻情報 SYSTIM についての詳細は，「16.2.12　システム時刻情報」を参照してください。

9.6　周期ハンドラの動作開始

周期ハンドラの動作開始は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

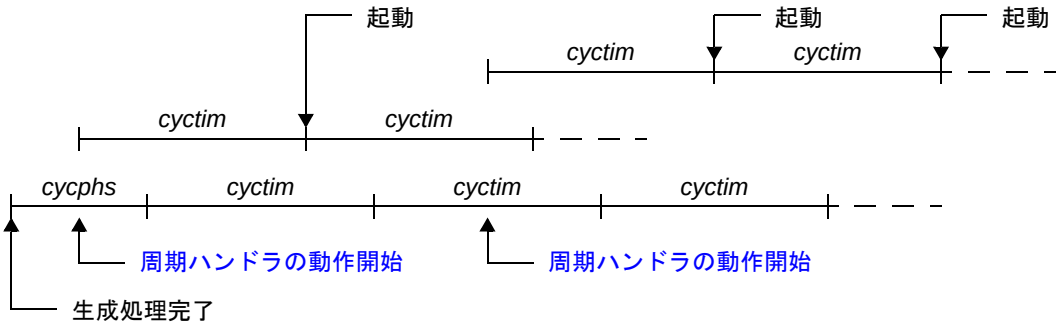
- sta_cyc，ista_cyc
  パラメータ *cycid* で指定された周期ハンドラの動作状態を停止状態（STP 状態）から動作状態（STA 状態）へと遷移させます。これにより，対象周期ハンドラは，RX850V4 の起動対象となります。
  なお，本サービス・コールの発行から 1 回目の起動要求が発行されるまでの相対時間間隔は，コンフィギュレーション時に対象周期ハンドラに対して TA_PHS 属性を指定しているか否かにより異なります。
  - “指定あり”の場合
    コンフィギュレーション時に定義した起動位相（初期起動位相 *cycphs*，起動周期 *cyctim*）で対象周期ハンドラに対する起動タイミング設定処理が行われます。
    ただし，対象周期ハンドラの動作状態が開始状態の場合には，本サービス・コールを発行しても何も処理は行わず，エラーとしても扱いません。
    図 9 － 1 に，周期ハンドラの起動タイミング・イメージを示します。

図 9 － 1　TA_PHS 属性：指定あり



  - “指定なし”の場合
    本サービス・コールの発行を基準点とした起動位相（起動周期 *cyctim*）で対象周期ハンドラに対する起動タイミング設定処理が行われます。
    なお，起動タイミング設定処理については，対象周期ハンドラの動作状態に関係なく実行されます。
    図 9 － 2 に，周期ハンドラの起動タイミング・イメージを示します。

図 9 － 2　TA_PHS 属性：指定なし



以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      cycid = 1;              /* 変数の宣言，初期化 */

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

```
        sta_cyc ( cycid );     /* 周期ハンドラの動作開始 */

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

9.7　周期ハンドラの動作停止

周期ハンドラの動作停止は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- stp_cyc，istp_cyc
  パラメータ *cycid* で指定された周期ハンドラの動作状態を動作状態（STA 状態）から停止状態（STP 状態）へと遷移させます。これにより，本サービス・コールの発行から sta_cyc，または ista_cyc が発行されるまでの間，対象周期ハンドラは，RX850V4 の起動対象から除外されます。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      cycid = 1;              /* 変数の宣言，初期化 */

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

        stp_cyc ( cycid );              /* 周期ハンドラの動作停止 */

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

備考　動作状態が停止状態（STP 状態）へと遷移している周期ハンドラに対し本サービス・コールを発行しても何も処理は行わず，エラーとしても扱いません。

9.8　周期ハンドラ詳細情報の参照

周期ハンドラ詳細情報の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ref_cyc，iref_cyc
  パラメータ *cycid* で指定された周期ハンドラの周期ハンドラ詳細情報（現在状態，残り時間など）をパラメータ *pk_rcyc* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ID      cycid = 1;              /* 変数の宣言，初期化 */
        T_RCYC  pk_rcyc;                /* データ構造体の宣言 */
        STAT    cycstat;                /* 変数の宣言 */
        RELTIM  lefttim;                /* 変数の宣言 */
        ATR     cycatr;                 /* 変数の宣言 */
        RELTIM  cyctim;                 /* 変数の宣言 */
        RELTIM  cycphs;                 /* 変数の宣言 */

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

        ref_cyc ( cycid, &pk_rcyc );    /* 周期ハンドラ詳細情報の参照 */

        cycstat = pk_rcyc.cycstat;      /* 現在状態の獲得 */
        lefttim = pk_rcyc.lefttim;      /* 残り時間の獲得 */
        cycatr = pk_rcyc.cycatr;        /* 属性の獲得 */
        cyctim = pk_rcyc.cyctim;        /* 起動周期の獲得 */
        cycphs = pk_rcyc.cycphs;        /* 初期起動位相の獲得 */

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

備考　周期ハンドラ詳細情報T_RCYCについての詳細は，「16.2.13　周期ハンドラ詳細情報」を参照してください。

第 10 章　システム状態管理機能

本章では，RX850V4 が提供しているシステム状態管理機能について解説しています。

10.1　概　　要

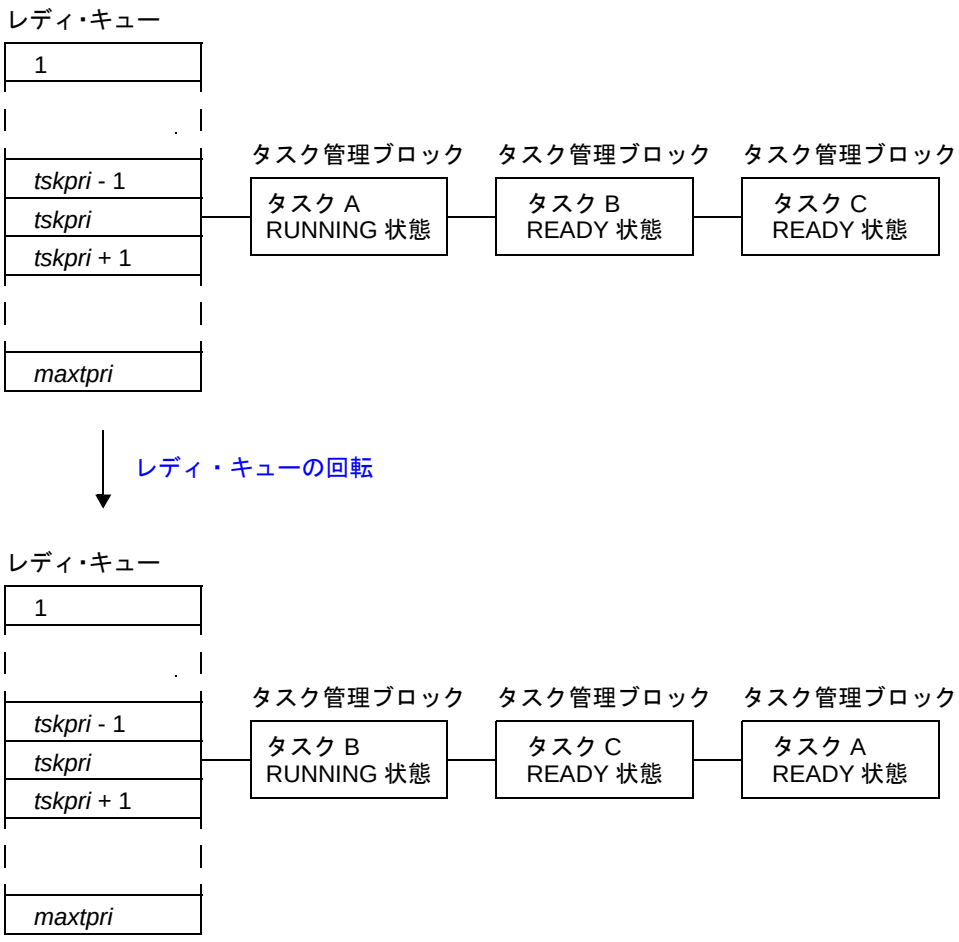
RX850V4 におけるシステム状態管理機能では，レディ・キューの回転，スケジューラの起動などといったシステムの状態を操作する機能のほかに，コンテキスト種別の参照，CPU ロック状態の参照などといったシステムの状態を参照する機能も提供しています。

10.2　レディ・キューの回転

レディ・キューの回転は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- rot_rdq，irot_rdq
  パラメータ *tskpri* で指定された優先度に対応したレディ・キューの先頭タスクを最後尾につなぎかえ，タスクの実行順序を明示的に変更します。
  以下に，“レディ・キューの回転”を利用した際の状態変化を示します。

図 10 － 1　レディ・キューの回転



以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
cychdr ( VP_INT exinf )
{
        PRI     tskpri = 8;             /* 変数の宣言，初期化 */

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

        irot_rdq ( tskpri );            /* レディ・キューの回転 */

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

        return;                         /* 周期ハンドラの終了 */
}
```

備考 1　本サービス・コールでは，レディ・キューの対象優先度にタスクが 1 つもキューイングされていなかった場合には，何も処理は行わず，エラーとしても扱いません。

備考 2　本サービス・コールを周期ハンドラなどから一定周期で発行することにより，ラウンドロビン・スケジューリングを実現することができます。

備考 3　RX850V4　におけるレディ・キューは，優先度をキーとしたハッシュ・テーブルであり，実行可能な状態（RUNNING 状態，または READY 状態）へと遷移したタスクの管理ブロック（タスク管理ブロック）が FIFO 順でキューイングされます。このため，スケジューラは，起動された際にレディ・キューの優先度高位から検出処理を実行し，キューイングされているタスクを検出した場合には，該当優先度の先頭タスクに CPU の利用権を与えることにより，RX850V4 のスケジューリング方式（優先度方式，FCFS 方式）を実現しています。

10.3　スケジューラの強制起動

スケジューラの強制起動は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- vsta_sch
  RX850V4 のスケジューラを明示的に強制起動します。このため，スケジューリング要求が保留されていた際には，“タスクの切り替え”が発生する場合があります。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{

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

        vsta_sch (  );                  /* スケジューラの強制起動 */

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

備考　RX850V4 では，本サービス・コールを“コンフィギュレーション時にプリエンプトの受け付け状態を禁止状態と定義したタスクから RX850V4 のスケジューラを起動するための機能”として提供しています。

10.4　RUNNING 状態のタスクの参照

RUNNING 状態のタスクの参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- get_tid，iget_tid
  RUNNING 状態のタスクの ID をパラメータ *p_tskid* で指定された領域に格納します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
inthdr ( void )
{
        ID      p_tskid;                /* 変数の宣言 */

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

        iget_tid ( &p_tskid );          /*RUNNING 状態のタスク ID の参照 */

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

        return;                         /* 割り込みハンドラの終了 */
}
```

備考　本サービス・コールでは，RUNNING 状態へと遷移しているタスクが存在しなかった場合（IDLE 状態）には，パラメータ *p_tskid* で指定された領域に TSK_NONE を格納しています。

10.5　CPU ロック状態への移行

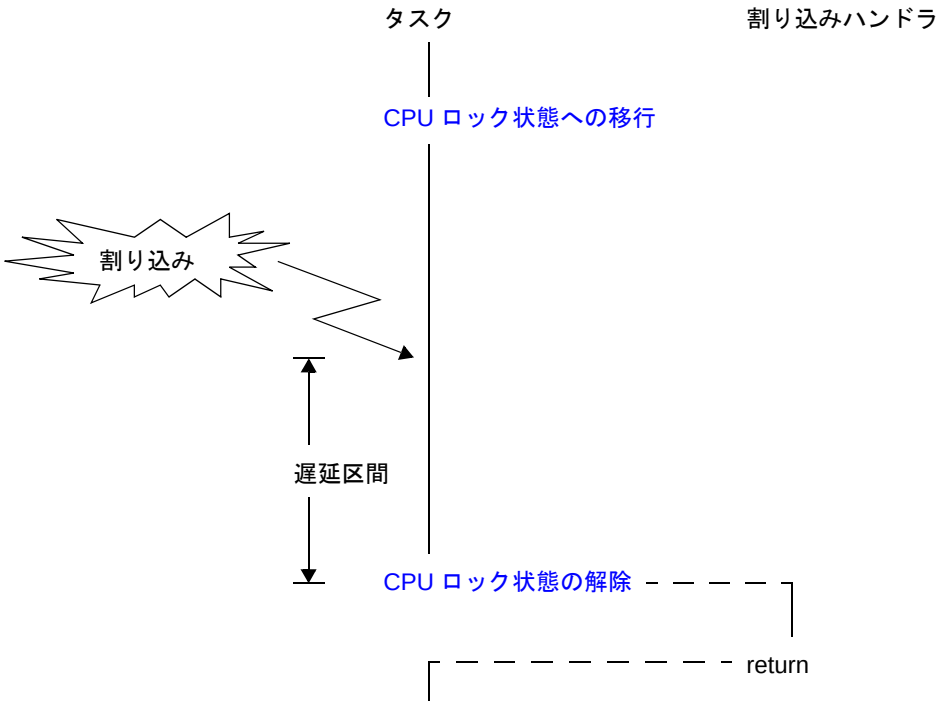
CPU ロック状態への移行は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- loc_cpu，iloc_cpu
  システム状態種別を非 CPU ロック状態から CPU ロック状態へと変更します。これにより，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間，“マスカブル割り込みの受け付け処理”，および，一部のサービス・コールを除き，サービス・コールの発行が禁止されます。

発行可能なサービス・コール	機能概要
sns_tex	タスク例外処理禁止状態の参照
loc_cpu，iloc_cpu	CPU ロック状態への移行
unl_cpu，iunl_cpu	CPU ロック状態の解除
sns_loc	CPU ロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_ctx	コンテキスト種別の参照
sns_dpn	ディスパッチ保留状態の参照

なお，RX850V4 では，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間にマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を unl_cpu，または iunl_cpu が発行されるまで遅延しています。
以下に，“CPU ロック状態への移行”を利用した際の処理の流れを示します。

図 10 － 2　CPU ロック状態への移行



以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ............
        ............

        loc_cpu ( );                    /*CPU ロック状態への移行 */

        ............                    /*CPU ロック状態 */
        ............

        unl_cpu ( );                    /*CPU ロック状態の解除 */

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

備考 1　本サービス・コールの内部処理（割り込みマスク設定処理，割り込みマスク獲得処理）は，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。
なお，サンプル・ソース・ファイルでは，割り込みマスク設定処理，割り込みマスク獲得処理として“割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に対する操作”を行っています。

<rx_sample>¥src¥usr_getmsk.c，usr_intmsk.c

備考 2　本サービス・コールの発行により変更した CPU ロック状態の解除は，本サービス・コールを発行した処理プログラムが終了する以前に行う必要があります。

備考 3　本サービス・コールでは，ロック要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別が CPU ロック状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 4　RX850V4 では，一定周期で発生する基本クロック用タイマ割り込みを利用して時間管理機能を実現しています。このため，本サービス・コールの発行により，該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には，時間管理機能が正常に動作しなくなる場合があります。

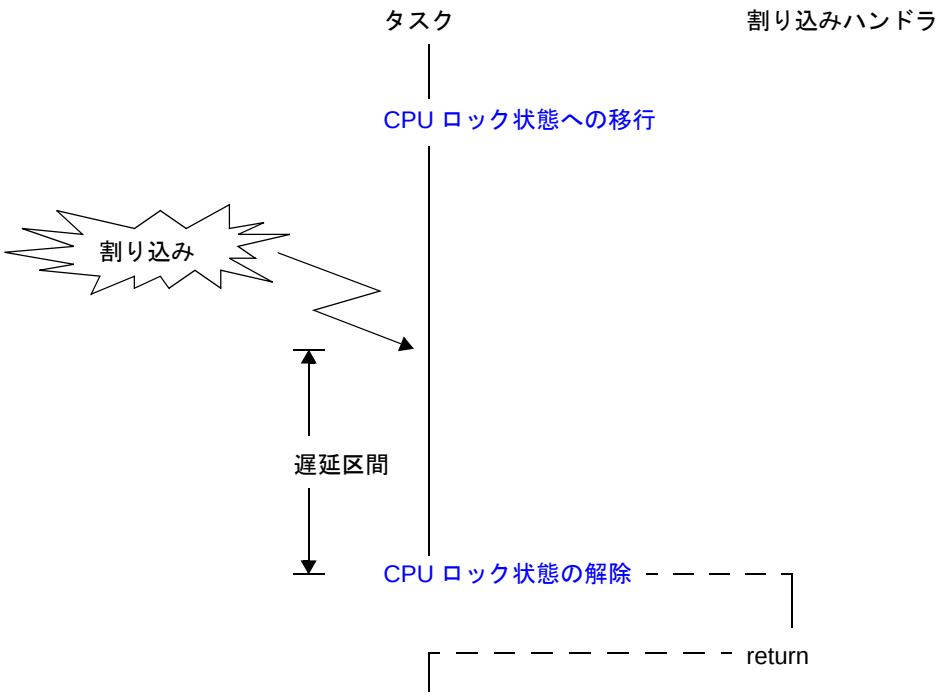
備考 5　RX850V4 では，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間に“本サービス・コール，または sns_*xxx* 以外のサービス・コール”を発行した場合には，戻り値として E_CTX を返します。

10.6　CPU ロック状態の解除

CPU ロック状態の解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- unl_cpu，iunl_cpu
  システム状態種別を CPU ロック状態から非 CPU ロック状態へと変更します。これにより，loc_cpu，または iloc_cpu の発行により抑制（禁止）されていた“マスカブル割り込みの受け付け処理”，および，サービス・コールの発行が許可されます。
  なお，RX850V4 では，loc_cpu，または iloc_cpu の発行から本サービス・コールが発行されるまでの間にマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を本サービス・コールが発行されるまで遅延しています。
  以下に，“CPU ロック状態の解除”を利用した際の処理の流れを示します。

図 10 － 3　CPU ロック状態の解除



以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        .............
        .............

        loc_cpu ( );                    /*CPU ロック状態への移行 */

        .............                   /*CPU ロック状態 */
        .............

        unl_cpu ( );                    /*CPU ロック状態の解除 */

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

備考 1　本サービス・コールの内部処理（割り込みマスク設定処理）は，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。
なお，サンプル・ソース・ファイルでは，割り込みマスク設定処理として“割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に対する操作”を行っています。

<rx_sample>¥src¥usr_setmsk.c

備考 2　本サービス・コールでは，解除要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別が非 CPU ロック状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 3　本サービス・コールでは，dis_dsp の発行により変更されたディスパッチ禁止状態の解除処理は行われません。したがって，CPU ロック状態以前のシステム状態種別がディスパッチ禁止状態であった場合には，本サービス・コール発行後のシステム状態種別は，ディスパッチ禁止状態となります。

備考 4　本サービス・コールでは，dis_int の発行により禁止されたマスカブル割り込みの受け付けの許可処理は行われません。したがって，CPU ロック状態以前のマスカブル割り込みの受け付けが禁止状態であった場合には，本サービス・コール発行後のマスカブル割り込みの受け付けは，禁止状態となります。

備考 5　RX850V4 では，loc_cpu，または iloc_cpu の発行から本サービス・コールが発行されるまでの間に“loc_cpu，iloc_cpu，または sns_*xxx* 以外のサービス・コール”を発行した場合には，戻り値として E_CTX を返します。

10.7　CPU ロック状態の参照

CPU ロック状態の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sns_loc
  本サービス・コールを発行した際のシステム状態種別（CPU ロック状態，非 CPU ロック状態）を獲得します。
  なお，本サービス・コールの発行により獲得したシステム状態種別については，戻り値として返します。
  以下に，本サービス・コールの記述例を示します。

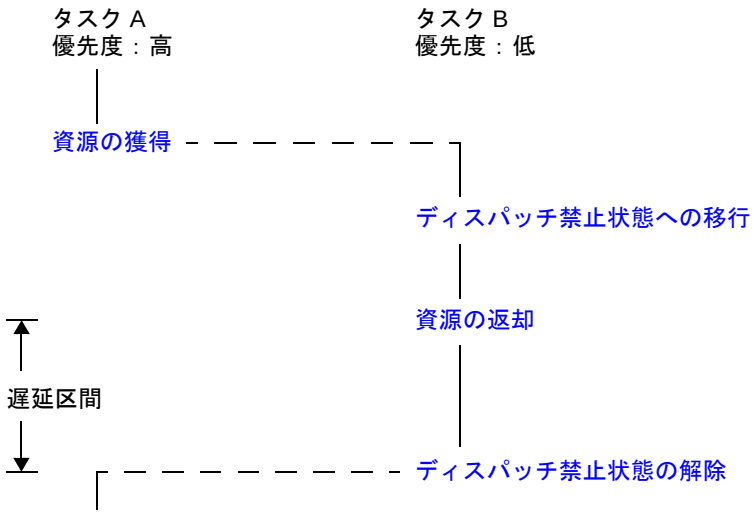
```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        BOOL ercd;                      /* 変数の宣言 */

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

        ercd = sns_loc ( );             /*CPU ロック状態の参照 */

        if ( ercd == TRUE ) {
                .............           /*CPU ロック状態 */
                .............
        } else if ( ercd == FALSE ) {
                .............           /* 非 CPU ロック状態 */
                .............
        }

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

10.8　ディスパッチ禁止状態への移行

ディスパッチ禁止状態への移行は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- dis_dsp
  システム状態種別をディスパッチ許可状態からディスパッチ禁止状態へと変更します。これにより，本サービス・コールの発行から ena_dsp が発行されるまでの間，“タスクのディスパッチ処理”が抑制（禁止）されます。
  なお，RX850V4 では，本サービス・コールの発行から ena_dsp が発行されるまでの間に“タスクのディスパッチ処理”を伴うサービス・コール（chg_pri，sig_sem など）が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際の“タスクのディスパッチ処理”は，ena_dsp が発行されるまで遅延され，一括して行うようにしています。
  以下に，“ディスパッチ禁止状態への移行”を利用した際の処理の流れを示します。

図 10 － 4　ディスパッチ禁止状態への移行



以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        .............
        .............

        dis_dsp ( );                    /* ディスパッチ禁止状態への移行 */

        .............                   /* ディスパッチ禁止状態 */
        .............

        ena_dsp ( );                    /* ディスパッチ禁止状態の解除 */

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

備考 1　本サービス・コールの発行により変更したディスパッチ禁止状態の解除は，本サービス・コールを発行したタスクが DORMANT 状態へと遷移する以前に行う必要があります。

備考 2　本サービス・コールでは，禁止要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別がディスパッチ禁止状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

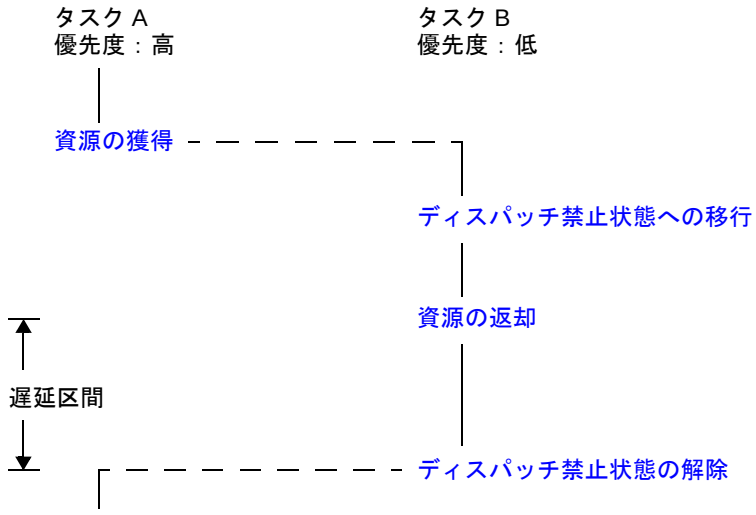
備考 3　RX850V4 では，本サービス・コールの発行から ena_dsp が発行されるまでの間に“自タスクを状態遷移させる可能性のあるサービス・コール（wai_sem，wai_flg など）”を発行した場合には，要求条件の即時成立／不成立を問わず，戻り値として E_CTX を返します。

10.9　ディスパッチ禁止状態の解除

ディスパッチ禁止状態の解除は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ena_dsp
  システム状態種別をディスパッチ禁止状態からディスパッチ許可状態へと変更します。これにより，dis_dsp の発行により抑制（禁止）されていた“タスクのディスパッチ処理”が許可されます。
  なお，RX850V4 では，dis_dsp の発行から本サービス・コールが発行されるまでの間に“タスクのディスパッチ処理”を伴うサービス・コール（chg_pri，sig_sem など）が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際の“タスクのディスパッチ処理”は，本サービス・コールが発行されるまで遅延され，一括して行うようにしています。
  以下に，“ディスパッチ禁止状態の解除”を利用した際の処理の流れを示します。

図 10 － 5　ディスパッチ禁止状態の解除



以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        .............
        .............

        dis_dsp ( );                    /* ディスパッチ禁止状態への移行 */

        .............                   /* ディスパッチ禁止状態 */
        .............

        ena_dsp ( );                    /* ディスパッチ禁止状態の解除 */

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

備考 1　本サービス・コールでは，許可要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別がディスパッチ許可状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 2　RX850V4 では，dis_dsp の発行から本サービス・コールが発行されるまでの間に“自タスクを状態遷移させる可能性のあるサービス・コール（wai_sem，wai_flg など）”を発行した場合には，要求条件の即時成立／不成立を問わず，戻り値として E_CTX を返します。

10.10 ディスパッチ禁止状態の参照

ディスパッチ禁止状態の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sns_dsp
  本サービス・コールを発行した際のシステム状態種別（ディスパッチ禁止状態，ディスパッチ許可状態）を獲得します。
  なお，本サービス・コールの発行により獲得したシステム状態種別については，戻り値として返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        BOOL ercd;                      /* 変数の宣言 */

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

        ercd = sns_dsp ( );             /* ディスパッチ禁止状態の参照 */

        if ( ercd == TRUE ) {
                .............           /* ディスパッチ禁止状態 */
                .............
        } else if ( ercd == FALSE ) {
                .............           /* ディスパッチ許可状態 */
                .............
        }

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

10.11 コンテキスト種別の参照

コンテキスト種別の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sns_ctx
  本サービス・コールを発行した処理プログラムのコンテキスト種別（非タスク・コンテキスト，タスク・コンテキスト）を獲得します。
  なお，本サービス・コールの発行により獲得したコンテキスト種別については，戻り値として返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        BOOL ercd;                      /* 変数の宣言 */

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

        ercd = sns_ctx ( );             /* コンテキスト種別の参照 */

        if ( ercd == TRUE ) {
                .............           /* 非タスク・コンテキスト処理 */
                .............
        } else if ( ercd == FALSE ) {
                .............           /* タスク・コンテキスト処理 */
                .............
        }

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

10.12 ディスパッチ保留状態の参照

ディスパッチ保留状態の参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- sns_dpn
  本サービス・コールを発行した際のシステム状態種別（ディスパッチ保留状態であるか否か）を獲得します。
  なお，本サービス・コールの発行により獲得したシステム状態種別については，戻り値として返します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        BOOL ercd;                      /* 変数の宣言 */

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

        ercd = sns_dpn ( );             /* ディスパッチ保留状態の参照 */

        if ( ercd == TRUE ) {
                .............           /* ディスパッチ保留状態 */
                .............
        } else if ( ercd == FALSE ) {
                .............           /* 非ディスパッチ保留状態 */
                .............
        }

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

第 11 章　割り込み管理機能

本章では，RX850V4 が提供している割り込み管理機能について解説しています。

11.1　概　　要

RX850V4 における割り込み管理機能では，割り込みが発生した際に起動する割り込みハンドラに関連した機能を提供しています。

11.2　ターゲット依存部

RX850V4 では，さまざまな実行環境に対応するために，割り込み管理機能のうち，RX850V4 が処理を実行するうえで必要となるハードウエア依存処理（サービス・コール dis_int，サービス・コール ena_int，割り込みマスク設定処理（上書き設定），割り込みマスク設定処理（OR 設定），割り込みマスク獲得処理）をターゲット依存部として切り出しています。これにより，さまざまな実行環境への移植性を向上させるとともに，カスタマイズを容易なものとしています。

11.2.1　サービス・コール dis_int

サービス・コール dis_int は，マスカブル割り込みの受け付けを許可状態から禁止状態へと変更するためにターゲット依存部として切り出されたマスカブル割り込みの受け付け操作処理専用ルーチンであり，処理プログラムからサービス・コール dis_int が発行された際に呼び出されます。

- サービス・コール dis_int の基本型
  サービス・コール dis_int を記述する場合，INTNO 型の引き数を 1 つ持った void 型の関数（関数名：_kernel_usr_dis_int）として記述します。
  なお，引き数 *intno* には“受け付けを許可状態から禁止状態へと変更するマスカブル割り込みに対応した例外コード番号”が設定されます。
  以下に，サービス・コール dis_int を C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
_kernel_usr_dis_int ( INTNO intno )
{
        .............
        .............

        return;                         /* サービス・コール dis_int の終了 */
}
```

- サービス・コール dis_int 内での処理
  サービス・コール dis_int は，マスカブル割り込みの受け付けを許可状態から禁止状態へと変更するためにターゲット依存部として切り出されたマスカブル割り込みの受け付け操作処理専用ルーチンです。このため，サービス・コール dis_int を記述する際には，以下に示す注意点があります。
  - 記述方法
    C 言語，またはアセンブリ言語で記述します。
    C 言語で記述するときは通常の関数と同様に記述することができます。
    アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。
  - スタックの切り替え
    RX850V4 では，サービス・コール dis_int に制御を移す際に，スタックの切り替えに関する処理を行っていません。したがって，基本情報において指定されたシステム・スタックを使用する際には，サービス・コール dis_int 内でスタックの切り替えに関する記述を行う必要があります。

- サービス・コールの発行
  サービス・コール dis_int では，マスカブル割り込みの受け付け操作処理を高速に終了する目的から，サービス・コールの発行が禁止されています。

以下に，サービス・コール dis_int として実行すべき処理の一覧を示します。

- 割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn を操作し, 指定された例外コード番号に対応したマスカブル割り込みの受け付けを許可状態から禁止状態へと変更
- サービス・コール dis_int を発行した処理プログラムに制御を戻す

11.2.2　サービス・コール ena_int

サービス・コール ena_int は，マスカブル割り込みの受け付けを禁止状態から許可状態へと変更するためにターゲット依存部として切り出されたマスカブル割り込みの受け付け操作処理専用ルーチンであり，処理プログラムからサービス・コール ena_int が発行された際に呼び出されます。

- サービス・コール ena_int の基本型
  サービス・コールena_intを記述する場合，INTNO型の引き数を1つ持ったvoid型の関数（関数名：_kernel_usr_ena_int）として記述します。
  なお，引き数 *intno* には“受け付けを禁止状態から許可状態へと変更するマスカブル割り込みに対応した例外コード番号”が設定されます。
  以下に，サービス・コール ena_int を C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
_kernel_usr_ena_int ( INTNO intno )
{
        ............
        ............

        return;                         /* サービス・コール ena_int の終了 */
}
```

- サービス・コール ena_int 内での処理
  サービス・コール ena_int は，マスカブル割り込みの受け付けを禁止状態から許可状態へと変更するためにターゲット依存部として切り出されたマスカブル割り込みの受け付け操作処理専用ルーチンです。このため，サービス・コール ena_int を記述する際には，以下に示す注意点があります。
  - 記述方法
    C 言語，またはアセンブリ言語で記述します。
    C 言語で記述するときは通常の関数と同様に記述することができます。
    アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。
  - スタックの切り替え
    RX850V4 では，サービス・コール ena_int に制御を移す際に，スタックの切り替えに関する処理を行っていません。したがって，基本情報において指定されたシステム・スタックを使用する際には，サービス・コール ena_int 内でスタックの切り替えに関する記述を行う必要があります。
  - サービス・コールの発行
    サービス・コール ena_int では，マスカブル割り込みの受け付け操作処理を高速に終了する目的から，サービス・コールの発行が禁止されています。

  以下に，サービス・コール ena_int として実行すべき処理の一覧を示します。

  - 割り込み制御レジスタ xxICn，または，割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn を操作し，指定された例外コード番号に対応したマスカブル割り込みの受け付けを禁止状態から許可状態へと変更
  - サービス・コールを発行した処理プログラムに制御を戻す

11.2.3　割り込みマスク設定処理（上書き設定）

割り込みマスク設定処理（上書き設定）は，該当ユーザ・オウン関数のパラメータで指定された割り込みマスク・パターンを割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に設定するためにターゲット依存部として切り出された割り込みマスク・パターン設定処理専用ルーチンであり，処理プログラムからサービス・コール unl_cpu，iunl_cpu，chg_ims，ichg_ims が発行された際に呼び出されます。

- 割り込みマスク設定処理（上書き設定）の基本型
  割り込みマスク設定処理（上書き設定）を記述する場合，VP 型の引き数を 1 つ持った void 型の関数（関数名：_kernel_usr_set_intmsk）として記述します。
  なお，引き数 *p_intms* には“設定する割り込みマスク・パターン格納した領域へのポインタ”が設定されます。
  以下に，割り込みマスク設定処理（上書き設定）を C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
_kernel_usr_set_intmsk ( VP p_intms )
{
        .............
        .............

        return;                         /* 割り込みマスク設定処理（上書き設定）*/
}
```

- 割り込みマスク設定処理（上書き設定）内での処理
  割り込みマスク設定処理（上書き設定）は，パラメータで指定された割り込みマスク・パターンを割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に設定するためにターゲット依存部として切り出された割り込みマスク・パターン設定処理専用ルーチンです。このため，割り込みマスク設定処理（上書き設定）を記述する際には，以下に示す注意点があります。

  - 記述方法
    C 言語，またはアセンブリ言語で記述します。
    C 言語で記述するときは通常の関数と同様に記述することができます。
    アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。
  - スタックの切り替え
    RX850V4 では，割り込みマスク設定処理（上書き設定）に制御を移す際に，スタックの切り替えに関する処理を行っていません。したがって，基本情報において指定されたシステム・スタックを使用する際には，割り込みマスク設定処理（上書き設定）内でスタックの切り替えに関する記述を行う必要があります。
  - サービス・コールの発行
    割り込みマスク設定処理（上書き設定）では，割り込みマスク・パターンの設定処理を高速に終了する目的から，サービス・コールの発行が禁止されています。

  以下に，割り込みマスク設定処理（上書き設定）として実行すべき処理の一覧を示します。

  - パラメータで指定された割り込みマスク・パターンを割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に設定するためにターゲット依存部として切り出された割り込みマスク・パターン設定
  - 割り込みマスク設定処理（上書き設定）を呼び出した処理プログラムに制御を戻す

11.2.4　割り込みマスク設定処理（OR 設定）

割り込みマスク設定処理（OR 設定）は，該当ユーザ・オウン関数のパラメータで指定された割り込みマスク・パターンと CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）の論理和 OR をとり，その結果を対象レジスタの割り込みマスク・フラグ xxMKn に設定するためにターゲット依存部として切り出された割り込みマスク・パターン設定処理専用ルーチンであり，処理プログラムからサービス・コール loc_cpu，iloc_cpu が発行された際に呼び出されます。

- 割り込みマスク設定処理（OR 設定）の基本型
  割り込みマスク設定処理（OR 設定）を記述する場合，VP 型の引き数を 1 つ持った void 型の関数（関数名：_kernel_usr_msk_intmsk）として記述します。
  なお，引き数 *p_intms* には“設定する割り込みマスク・パターン格納した領域へのポインタ”が設定されます。
  以下に，割り込みマスク設定処理（上書き設定）を C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
_kernel_usr_msk_intmsk ( VP p_intms )
{
        .............
        .............

        return;                         /* 割り込みマスク設定処理（OR 設定）*/
}
```

- 割り込みマスク設定処理（OR 設定）内での処理
  割り込みマスク設定処理（OR 設定）は，パラメータで指定された割り込みマスク・パターンと CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）の論理和 OR をとり，その結果を対象レジスタの割り込みマスク・フラグ xxMKn に設定するためにターゲット依存部として切り出された割り込みマスク・パターン設定処理専用ルーチンです。このため，割り込みマスク設定処理（OR 設定）を記述する際には，以下に示す注意点があります。

  - 記述方法
    C 言語，またはアセンブリ言語で記述します。
    C 言語で記述するときは通常の関数と同様に記述することができます。
    アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。
  - スタックの切り替え
    RX850V4 では，割り込みマスク設定処理（OR 設定）に制御を移す際に，スタックの切り替えに関する処理を行っていません。したがって，基本情報において指定されたシステム・スタックを使用する際には，割り込みマスク設定処理（OR 設定）内でスタックの切り替えに関する記述を行う必要があります。
  - サービス・コールの発行
    割り込みマスク設定処理（OR 設定）では，割り込みマスク・パターンの設定処理を高速に終了する目的から，サービス・コールの発行が禁止されています。

  以下に，割り込みマスク設定処理（OR 設定）として実行すべき処理の一覧を示します。

  - パラメータで指定された割り込みマスク・パターンと CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）の論理和 OR をとり，その結果を対象レジスタの割り込みマスク・フラグ xxMKn に設定
  - 割り込みマスク設定処理（OR 設定）を呼び出した処理プログラムに制御を戻す

11.2.5　割り込みマスク獲得処理

割り込みマスク獲得処理は，該当ユーザ・オウン関数のパラメータで指定された領域に CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）を格納するためにターゲット依存部として切り出された割り込みマスク・パターンの獲得処理専用ルーチンであり，処理プログラムからサービス・コール loc_cpu，iloc_cpu，get_ims，iget_ims が発行された際に呼び出されます。

- 割り込みマスク獲得処理の基本型
  割り込みマスク獲得処理を記述する場合，VP型の引き数を1つ持ったvoid型の関数（関数名：_kernel_usr_get_intmsk）として記述します。
  なお，引き数 *p_intms* には“獲得した割り込みマスク・パターン格納する領域へのポインタ”が設定されます。
  以下に，割り込みマスク獲得処理を C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
_kernel_usr_get_intmsk ( VP p_intms )
{
        .............
        .............

        return;                         /* 割り込みマスク獲得処理 */
}
```

- 割り込みマスク獲得処理内での処理
  割り込みマスク獲得処理は，パラメータで指定された領域に CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）を格納するためにターゲット依存部として切り出された割り込みマスク・パターンの獲得処理専用ルーチンです。このため，割り込みマスク獲得処理を記述する際には，以下に示す注意点があります。

  - 記述方法
    C 言語，またはアセンブリ言語で記述します。
    C 言語で記述するときは通常の関数と同様に記述することができます。
    アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。
  - スタックの切り替え
    RX850V4 では，割り込みマスク獲得処理に制御を移す際に，スタックの切り替えに関する処理を行っていません。したがって，基本情報において指定されたシステム・スタックを使用する際には，割り込みマスク獲得処理内でスタックの切り替えに関する記述を行う必要があります。
  - サービス・コールの発行
    割り込みマスク獲得処理では，割り込みマスク・パターンの獲得処理を高速に終了する目的から，サービス・コールの発行が禁止されています。

  以下に，割り込みマスク獲得処理として実行すべき処理の一覧を示します。

  - パラメータで指定された領域に CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）を格納
  - 割り込みマスク獲得処理を呼び出した処理プログラムに制御を戻す

11.3　ユーザ・オウン・コーディング部

RX850V4 では，さまざまな実行環境に対応するために，割り込み管理機能のうち，RX850V4 が処理を実行するうえで必要となるハードウエア依存処理（割り込みエントリ処理）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。これにより，さまざまな実行環境への移植性を向上させるとともに，カスタマイズを容易なものとしています。

11.3.1　割り込みエントリ処理

割り込みエントリ処理は，割り込みが発生した際に CPU 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（割り込み前処理，直接起動割り込みハンドラなど）への分岐処理を割り付けるためにユーザ・オウン・コーディング部として切り出されたエントリ処理専用ルーチンです。

なお，コンフィギュレーション時に割り込みハンドラ情報として定義された割り込みハンドラの割り込みエントリ処理は，コンフィギュレーション時に作成したシステム・コンフィギュレーション・ファイルに対してコンフィギュレータを実行することにより生成されるエントリ・ファイルに内包されています。したがって，割り込みエントリ処理をカスタマイズする必要がない場合には，該当エントリ・ファイルを利用することにより，割り込みエントリ処理の記述が不要となります。

- 割り込みエントリ処理の基本型
  割り込みエントリ処理を記述する場合，ハンドラ・アドレスに対して該当処理（割り込み前処理，直接起動割り込みハンドラなど）への分岐処理を割り付けます。
  以下に，割り込みエントリ処理をアセンブリ言語で記述する場合の基本型を示します。

```
        -- 割り込み前処理への分岐処理
        .section        "sec_nam"       -- ハンドラ・アドレスの設定
        jr      __kernel_int_entry      -- 割り込み前処理への分岐

        -- 直接起動割り込みハンドラへの分岐処理
        .section        "sec_nam"       -- ハンドラ・アドレスの設定
        jr      _inthdr                 -- 直接起動割り込みハンドラへの分岐
```

- 割り込みエントリ処理内での処理
  割り込みエントリ処理は，割り込みが発生した際に RX850V4 を介在させることなく呼び出されるエントリ処理専用ルーチンです。このため，割り込みエントリ処理を記述する際には，以下に示す注意点があります。
  - 記述方法
    使用するコンパイラの呼び出し規約にのっとって，アセンブリ言語で記述します。
  - スタックの切り替え
    割り込みエントリ処理を実行するうえで切り替えを必要とするスタックは存在しません。したがって，割り込みエントリ処理内でスタックの切り替えに関する記述を行う必要がありません。
  - サービス・コールの発行
    割り込みエントリ処理では，発生した割り込みに対応した処理（割り込みハンドラ，直接起動割り込みハンドラなど）が実行されるまでの応答性を高速化する目的から，サービス・コールの発行が禁止されています。

  以下に，割り込みエントリ処理として実行すべき処理の一覧を示します。
  - ハンドラ・アドレスの設定
  - 該当処理（割り込み前処理，直接起動割り込みハンドラなど）に制御を移す

11.4　割り込みハンドラ

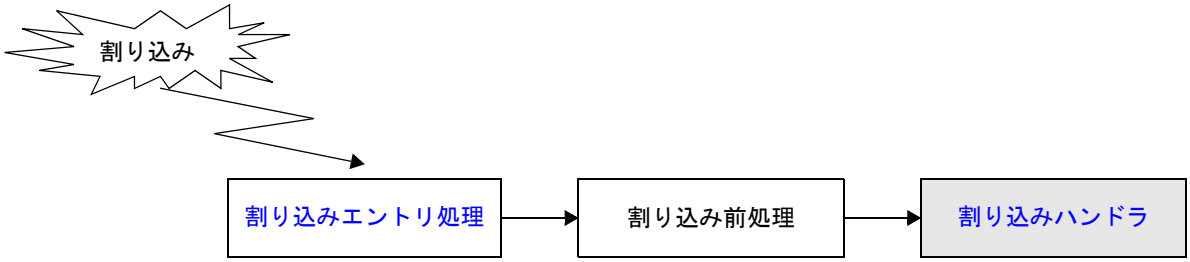
割り込みハンドラは，割り込みが発生した際に起動される割り込み処理専用ルーチンです。

なお，RX850V4 では，割り込みハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。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処理を実行中であっても，その処理は中断され，割り込みハンドラに制御が移ります。

また，RX850V4 では，割り込みハンドラを管理するに当たり，割り込みハンドラと一対一に対応した管理オブジェクト（割り込みハンドラ管理ブロック）を用いることにより，割り込みハンドラ自体の管理を行っています。

以下に，割り込みの発生から割り込みハンドラに制御が移るまでに実行される処理の流れを示します。

図 11 － 1　処理の流れ（割り込みハンドラ）



11.4.1　割り込みハンドラの基本型

割り込みハンドラを記述する場合，引き数を持たない void 型の関数として記述します。

以下に，割り込みハンドラを C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
inthdr ( void )
{
        .............
        .............

        return;                         /* 割り込みハンドラの終了 */
}
```

11.4.2　割り込みハンドラ内での処理

RX850V4 では，割り込みが発生した処理プログラムから割り込みハンドラに制御を移す際，独自の割り込み前処理を行っています。また，割り込みハンドラから割り込みが発生した処理プログラムに制御を戻す際にも，独自の割り込み後処理を行っています。このため，割り込みハンドラを記述する際には，以下に示す注意点があります。

- 記述方法
  C 言語，またはアセンブリ言語で記述します。
  C 言語で記述するときは通常の関数と同様に記述することができます。
  アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。

- スタックの切り替え
  RX850V4 では，割り込みハンドラに制御を移す際，基本情報において指定されたシステム・スタックへの切り替え処理を，割り込みが発生した処理プログラムに制御を戻す際に該当スタックへの切り替え処理を行っています。したがって，割り込みハンドラ内でスタックの切り替えに関する記述を行う必要がありません。

- サービス・コールの発行
  RX850V4 では，割り込みハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため，割り込みハンドラでは，“非タスク内から発行可能なサービス・コール”のみが発行可能となります。

  備考 1　RX850V4 では，割り込みハンドラ内の処理を高速に終了させる目的から，割り込みハンドラ内の処理が完了するまでの間，スケジューラの起動を遅延しています。したがって，割り込みハンドラ内でディスパッチ処理（タスクのスケジューリング処理）を伴うサービス・コール（isig_sem，iset_flg など）が発行された際には，キュー操作などといった処理が行われるだけであり，実際のディスパッチ処理の実行は“割り込みハンドラからの復帰命令（return 命令の発行）”が発行されるまで遅延され，一括して行うようにしています。

  備考 2　各サービス・コールの発行有効範囲についての詳細は，表 17 － 1 ～表 17 － 14 を参照してください。

- マスカブル割り込みの受け付け状態（PSW の ID フラグ）
  処理開始時，マスカブル割り込みの受け付け禁止状態（PSW の ID フラグは 1）となります。
  処理内では，マスカブル割り込みの受け付け状態を変更することができます。変更した状態は，処理終了後に制御が移った処理プログラムへは継承されません。

  備考　処理開始時，ISPRn（該当割り込みの優先順位 n に該当するビット）は 1 となります。
  処理終了時，ISPRn は 0 にクリアされます。

11.4.3　割り込みハンドラの登録

RX850V4 では，割り込みハンドラの静的な登録のみサポートしています。処理プログラムからサービス・コールを発行して動的に登録することはできません。

割り込みハンドラの静的登録とは，システム・コンフィギュレーション・ファイルで静的 API“DEF_INH”を使用して割り込みハンドラを定義することをいいます。

静的 API“DEF_INH”の詳細は，「18.5.11　割り込みハンドラ情報」を参照してください。

11.5　直接起動割り込みハンドラ

RX850V4 は直接起動割り込みハンドラの動作に関与しません。

直接起動割り込みハンドラの使用方法は RX850V4 等のリアルタイム OS を使用しない場合の割り込みの使用方法と同じです。

直接起動割り込みハンドラからはすべてのサービス・コールを発行できません。

直接起動割り込みハンドラ起動時はスタックの切り替えが行われませんので，割り込みが発生した時点で使用していたスタックをそのまま使用します。

したがって，すべてのタスク・スタック，システム・スタックのサイズを決定する際は直接起動割り込みハンドラが使用するサイズを考慮しておく必要があります。

11.6　処理プログラム内におけるマスカブル割り込み受け付け状態

V850 マイクロコントローラのマスカブル割り込みの受け付け状態は，PSW.ID，xxMKn，ISPRn の値で変わります。詳細はハードウエアのマニュアルを参照してください。

- PSW.ID
  プログラム・ステータス・ワード・レジスタ（PSW）の ID フラグです。
  全マスカブル割り込みの受け付け状態を保持します。
  0 が全マスカブル割り込みの受け付け許可状態，1 が全マスカブル割り込みの受け付け禁止状態です。
  処理プログラムごとに初期状態が決まっています。その詳細は，表 11 － 1 を参照してください。
  RX850V4 の処理プログラム内で，EI 命令や DI 命令などで変更することができます。

表 11 － 1　処理プログラム起動時のマスカブル割り込み受け付け状態

処理プログラム名	PSW.ID
タスク	ユーザ設定状態
タスク例外処理ルーチン	起動前の状態を継承
周期ハンドラ	0
割り込みハンドラ	1
直接起動割り込みハンドラ	1
拡張サービス・コール・ルーチン	起動前の状態を継承
CPU 例外ハンドラ	1
初期化ルーチン	1
アイドル・ルーチン	0

備考　タスク起動時の PSW.ID にあるユーザ設定状態とは，タスク情報の属性で設定される初期割り込み状態です。マスカブル割り込み許可ならば 0，マスカブル割り込み禁止ならば 1 となります。

- xxMKn
  各割り込みごとに割り当てられている割り込み制御レジスタ（xxICn）の割り込みマスク・フラグ（xxMKn）の値です。各マスカブル割り込みの受け付け状態を保持します。
  0 がマスカブル割り込みの受け付け許可状態，1 がマスカブル割り込みの受け付け禁止状態です。
  RX850V4 の処理プログラム内で，サービス・コール dis_int，ena_int，chg_ims，loc_cpu，unl_cpu の発行などで変更することができます。
  初期状態の設定はシステム初期化処理（ブート処理，初期化ルーチンなど）で記述する必要があります。処理プログラム起動時に xxMKn の値は操作されません。

- ISPRn
  インサービス・プライオリティ・レジスタ（ISPR）の割り込み優先順位レベル n に対応するビットの値です。受け付け中のマスカブル割り込みの優先順位レベルを保持します。
  0 が優先順位 n の割り込み要求信号を受け付けていない状態，1 が優先順位 n の割り込み要求信号を受け付け中の状態です。
  マスカブル割り込みをトリガに起動する処理プログラム（割り込みハンドラ，直接起動割り込みハンドラ）のみ割り込み優先順位レベル n に対応するビットの値が 1 となります。処理プログラム内で値を変更することはできません。

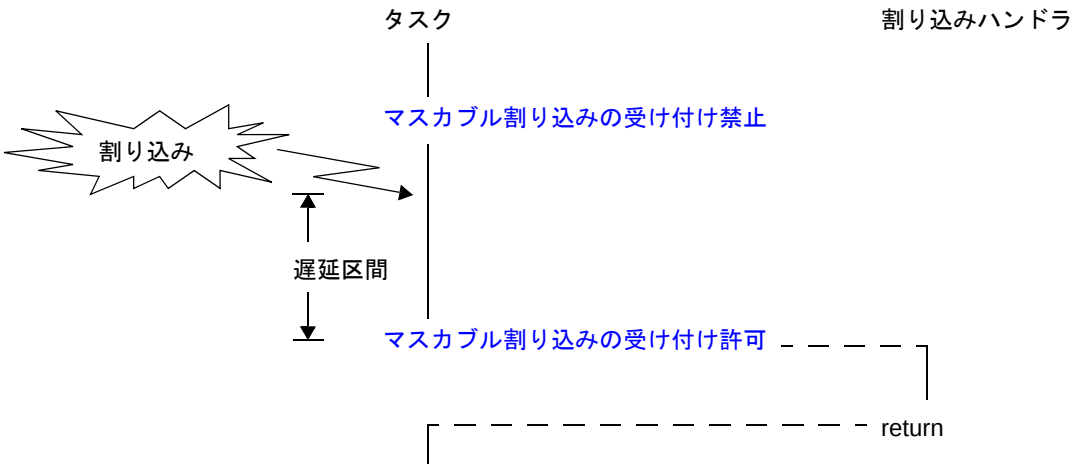
備考　周期ハンドラは，基本クロック用タイマ割り込みの発生をトリガに開始されますが，該当処理内の ISPRn（基本クロック用タイマ割り込みの優先順位 n に該当するビット）は 0 に設定されます。したがって，周期ハンドラ内で基本クロック用タイマ割り込み自身や基本クロック用タイマ割り込みより優先順位の低い割り込みが発生した際には，該当割り込みは受け付けられます。

11.7　マスカブル割り込みの受け付け禁止

マスカブル割り込みの受け付け禁止は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- dis_int
  パラメータ *intno* で指定された例外コード番号に対応したマスカブル割り込みの受け付けを許可状態から禁止状態へと変更します。
  なお，RX850V4 では，本サービス・コールの発行から ena_int が発行されるまでの間にパラメータ *intno* で指定された例外コード番号に対応したマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を ena_int が発行されるまで遅延しています。
  以下に，“マスカブル割り込みの受け付け禁止”を利用した際の処理の流れを示します。

図 11 － 2　マスカブル割り込みの受け付け禁止



以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        INTNO   intno = 0x80;           /* 変数の宣言，初期化 */

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

        dis_int ( intno );              /* マスカブル割り込みの受け付け禁止 */

        .............                   /* 受け付け禁止状態 */
        .............

        ena_int ( intno );              /* マスカブル割り込みの受け付け許可 */

        .............                   /* 受け付け許可状態 */
        .............
}
```

備考 1　本サービス・コールは，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。
なお，サンプル・ソース・ファイルでは，マスカブル割り込みの受け付け禁止処理として“割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に対する操作”を行っています。

<rx_sample>¥src¥usr_disint.c

備考 2　本サービス・コールでは，禁止要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，対応するマスカブル割り込みの受け付けが禁止状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

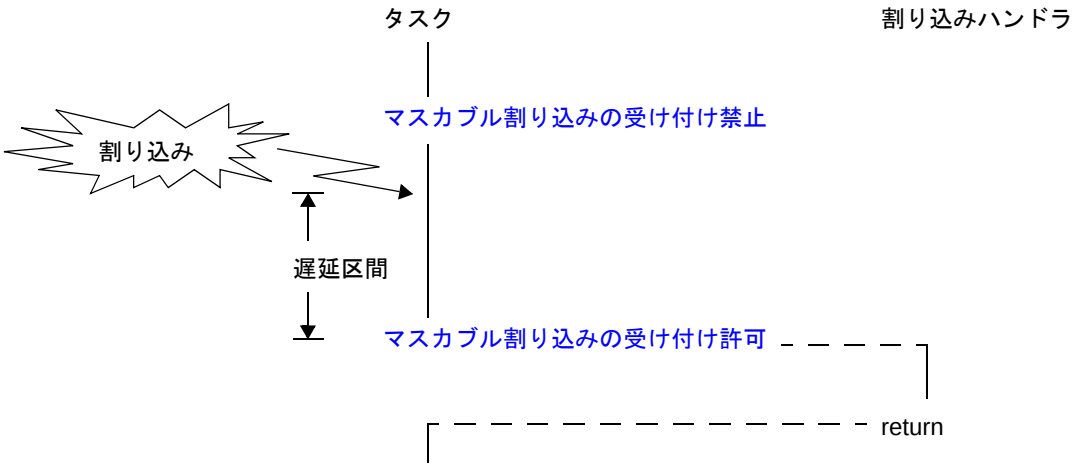
備考 3　RX850V4 では，一定周期で発生する基本クロック用タイマ割り込みを利用して時間管理機能を実現しています。このため，本サービス・コールの発行により，該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には，時間管理機能が正常に動作しなくなる場合があります。

11.8　マスカブル割り込みの受け付け許可

マスカブル割り込みの受け付け許可は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- ena_int
  パラメータ *intno* で指定された例外コード番号に対応したマスカブル割り込みの受け付けを禁止状態から許可状態へと変更します。
  なお，RX850V4 では，dis_int の発行から本サービス・コールが発行されるまでの間にパラメータ *intno* で指定された例外コード番号に対応したマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を本サービス・コールが発行されるまで遅延しています。
  以下に，“マスカブル割り込みの受け付け許可”を利用した際の処理の流れを示します。

図 11 － 3　マスカブル割り込みの受け付け許可



以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        INTNO   intno = 0x80;           /* 変数の宣言，初期化 */

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

        dis_int ( intno );              /* マスカブル割り込みの受け付け禁止 */

        ............                    /* 受け付け禁止状態 */
        ............

        ena_int ( intno );              /* マスカブル割り込みの受け付け許可 */

        ............                    /* 受け付け許可状態 */
        ............
}
```

備考 1　本サービス・コールは，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。
なお，サンプル・ソース・ファイルでは，マスカブル割り込みの受け付け許可処理として“割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に対する操作”を行っています。

<rx_sample>¥src¥usr_enaint.c

備考 2　本サービス・コールでは，許可要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，対応するマスカブル割り込みの受け付けが許可状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

11.9　割り込みマスク・パターンの変更

割り込みマスク・パターンの変更は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- chg_ims，ichg_ims
  CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または，割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）をパラメータ *p_intms* で指定された状態へと変更します。
  以下に，*p_intms* で指定された領域に設定する値（割り込みマスク・フラグ）の意味を示します。

  0：　マスカブル割り込みの受け付けは許可状態
  1：　マスカブル割り込みの受け付けは禁止状態

  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        UH      intms[0x3];             /* 変数の宣言 */
        UH      *p_intms;               /* 変数の宣言 */

        intms[0x0] = 0x0000;            /* 変数の初期化 */
        intms[0x1] = 0x1014;            /* 変数の初期化 */
        intms[0x2] = 0x0021;            /* 変数の初期化 */
        p_intms = intms;                /* 変数の初期化 */

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

        chg_ims ( p_intms );            /* 割り込みマスク・パターンの変更 */

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

備考 1　本サービス・コールの内部処理（割り込みマスク設定処理）は，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。

<rx_sample>¥src¥usr_setmsk.c

備考 2　RX850V4 では，一定周期で発生する基本クロック用タイマ割り込みを利用して時間管理機能を実現しています。このため，本サービス・コールの発行により，該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には，時間管理機能が正常に動作しなくなる場合があります。

11.10 割り込みマスク・パターンの参照

割り込みマスク・パターンの参照は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- get_ims，iget_ims
  CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または，割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）をパラメータ *p_intms* で指定された領域に格納します。
  以下に，*p_intms* で指定された領域に格納される値（割り込みマスク・フラグ）の意味を示します。

  0：　マスカブル割り込みの受け付けは許可状態
  1：　マスカブル割り込みの受け付けは禁止状態

  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        UH      p_intms[0x3];           /* 変数の宣言 */

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

        get_ims ( p_intms );            /* 割り込みマスク・パターンの参照 */

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

備考　本サービス・コールの内部処理（割り込みマスク獲得処理）は，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。

<rx_sample>¥src¥usr_getmsk.c

11.11 ノンマスカブル割り込み

ノンマスカブル割り込みは，割り込み優先順位の対象外となっており，識別可能なすべての割り込みに優先して受け付けられます。また，ノンマスカブル割り込みは，CPU を割り込み禁止状態（プログラム・ステータス・ワード PSW の ID フラグに 1 を設定）へと変更しても受け付けられる割り込みです。したがって，RX850V4 を CPU ロック状態，またはマスカブル割り込みの受け付け禁止状態へと変更してもノンマスカブル割り込みは受け付けられることになります。

備考　RX850V4 では，ノンマスカブル割り込みに対応した割り込みハンドラを RX850V4 の管理対象からは除外しています。このため，ノンマスカブル割り込みに対応した割り込みハンドラ内では，サービス・コールの発行が禁止されています。

11.12 基本クロック用タイマ割り込み

RX850V4 では，一定周期で発生する基本クロック用タイマ割り込みを利用して時間管理機能を実現しています。

したがって，基本クロック用タイマ割り込みが発生した際には，RX850V4 が提供している時間管理用割り込みハンドラが起動し，時間に関連した処理（システム時刻の更新，タスクの遅延起床／タイムアウト，周期ハンドラの起動など）が実行されます。

備考　loc_cpu，iloc_cpu，dis_int の発行により，該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には，時間管理機能が正常に動作しなくなる場合があります。

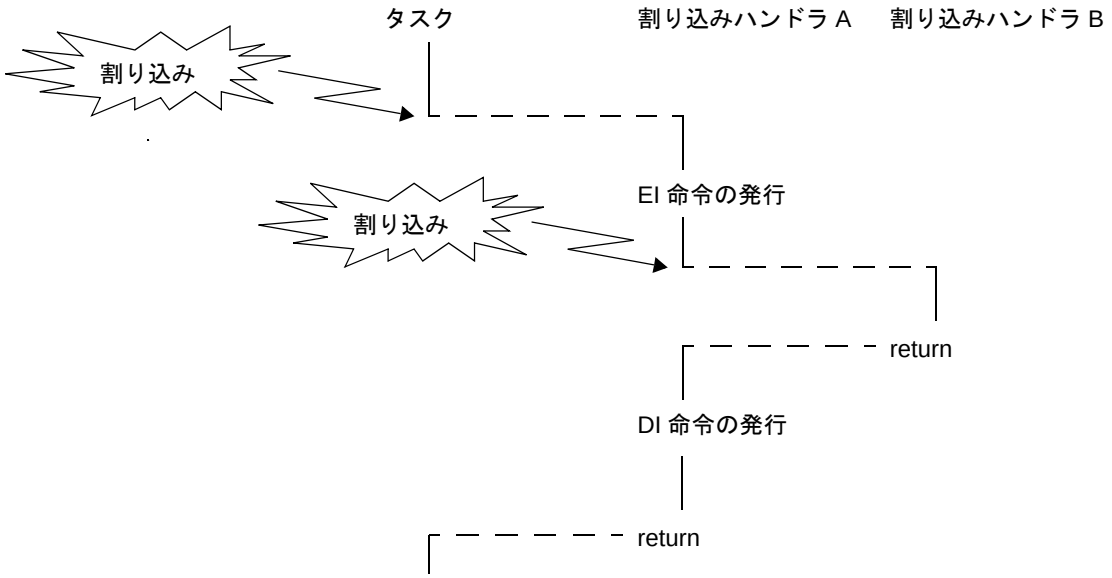
11.13 多重割り込み

RX850V4 では，割り込みハンドラ内で再び割り込みが発生することを“多重割り込み”と呼んでいます。

ただし，割り込みハンドラは，割り込み禁止状態（プログラム・ステータス・ワード PSW の ID フラグに 1 を設定）で該当処理の実行が開始されるため，多重割り込みを発生させるためには，割り込みハンドラ内で明示的に割り込み禁止状態の解除処理（ EI 命令の発行など）を記述する必要があります。

以下に，多重割り込みが発生した際の処理の流れを示します。

図 11 － 4　多重割り込み



第 12 章　サービス・コール管理機能

本章では，RX850V4 が提供しているサービス・コール管理機能について解説しています。

12.1　概　　要

RX850V4 におけるサービス・コール管理機能では，拡張サービス・コール・ルーチンの登録／呼び出しなどといった拡張サービス・コール・ルーチンの状態を操作する機能を提供しています。

12.2　拡張サービス・コール・ルーチン

拡張サービス・コール・ルーチンは，ユーザ定義の関数を RX850V4 に登録したものであり，RX850V4 が提供するサービス・コールを使用して明示的に呼び出さないかぎり実行されることのない処理プログラムです。

なお，RX850V4 では，拡張サービス・コール・ルーチンを“拡張サービス・コール・ルーチンを呼び出した処理プログラムの延長線”として位置づけています。

また，RX850V4 では，拡張サービス・コール・ルーチンを管理するに当たり，拡張サービス・コール・ルーチンと一対一に対応した管理オブジェクト（拡張サービス・コール・ルーチン管理ブロック）を用いることにより，拡張サービス・コール・ルーチン自体の管理を行っています。

12.2.1　拡張サービス・コール・ルーチンの基本型

拡張サービス･コール･ルーチンを記述する場合, VP_INT型の引き数を3つ持ったER_UINT型の関数として記述します。

なお，引き数 *par1*，*par2*，*par3* には“呼び出し要求（cal_svc，または ical_svc）の発行時に指定された引き継ぎデータ”が設定されます。

以下に，拡張サービス・コール・ルーチンを C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

ER_UINT
svcrtn ( VP_INT par1, VP_INT par2, VP_INT par3  )
{
        .............
        .............

        return ( ER_UINT ercd );        /* 拡張サービス・コール・ルーチンの終了 */
}
```

12.2.2　拡張サービス・コール・ルーチン内での処理

RX850V4 では，呼び出し要求を発行した処理プログラムから拡張サービス・コール・ルーチンに制御を移す際，独自の拡張サービス・コール・ルーチン前処理を行っています。また，拡張サービス・コール・ルーチンから呼び出し要求を発行した処理プログラムに制御を戻す際にも，独自の拡張サービス・コール・ルーチン後処理を行っています。このため，拡張サービス・コール・ルーチンを記述する際には，以下に示す注意点があります。

- 記述方法
  C 言語，またはアセンブリ言語で記述します。
  C 言語で記述するときは通常の関数と同様に記述することができます。
  アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。

- スタックの切り替え
  RX850V4 では，拡張サービス・コール・ルーチンを“拡張サービス・コール・ルーチンを呼び出した処理プログラムの延長線”として位置づけています。したがって，拡張サービス・コール・ルーチンに制御を移す際，スタックの切り替え処理は行われません。

- サービス・コールの発行
  RX850V4 では，拡張サービス・コール・ルーチンを“拡張サービス・コール・ルーチンを呼び出した処理プログラムの延長線”として位置づけています。したがって，拡張サービス・コール・ルーチン内で発行可能なサービス・コールは，拡張サービス・コール・ルーチンを呼び出した処理プログラムの種類（タスク，非タスク）に依存します。

  備考　　各サービス・コールの発行有効範囲についての詳細は，表 17 － 1 ～表 17 － 14 を参照してください。

- マスカブル割り込みの受け付け状態（PSW の ID フラグ）
  マスカブル割り込みの受け付け状態は拡張サービス・コール・ルーチン呼び出し元の処理プログラムに依存しています。
  処理開始時，マスカブル割り込みの受け付け状態は拡張サービス・コール・ルーチンを呼び出した処理プログラムの状態を継承します。
  処理内では，マスカブル割り込みの受け付け状態を変更することができます。変更した状態は，処理終了後に拡張サービス・コール・ルーチン呼び出し元の処理プログラムに制御が戻る際に継承されます。

12.3　拡張サービス・コール・ルーチンの登録

RX850V4 では，拡張サービス・コール・ルーチンの静的な登録のみサポートしています。処理プログラムからサービス・コールを発行して動的に登録することはできません。

拡張サービス・コール・ルーチンの静的登録とは，システム・コンフィギュレーション・ファイルで静的 API“DEF_SVC”を使用して拡張サービス・コール・ルーチンを定義することをいいます。

静的 API“DEF_SVC”の詳細は，「18.5.13　拡張サービス・コール・ルーチン情報」を参照してください。

12.4　拡張サービス・コール・ルーチンの呼び出し

拡張サービス・コール・ルーチンの呼び出しは，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

- cal_svc，ical_svc
  パラメータ *fncd* で指定された拡張サービス・コール・ルーチンを呼び出します。
  以下に，本サービス・コールの記述例を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

#pragma rtos_task       task            /* プラグマ指令の定義 */

void
task ( VP_INT exinf )
{
        ER_UINT ercd;                   /* 変数の宣言 */
        FN      fncd = 1;               /* 変数の宣言，初期化 */
        VP_INT  par1 = 123;             /* 変数の宣言，初期化 */
        VP_INT  par2 = 456;             /* 変数の宣言，初期化 */
        VP_INT  par3 = 789;             /* 変数の宣言，初期化 */

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

                                        /* 拡張サービス・コール・ルーチンの呼び出し */
        ercd = cal_svc ( fncd, par1, par2, par3 );

        if ( ercd != E_RSFN ) {
                .............           /* 正常終了処理 */
                .............
        }

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

備考　本サービス・コールを使用して呼び出すことができる拡張サービス・コール・ルーチンは，総引き継ぎデータ数が 4 つ未満のものに限られます。

第 13 章　システム構成管理機能

本章では，RX850V4 が提供しているシステム構成管理機能について解説しています。

13.1　概　　要

RX850V4 におけるシステム構成管理機能では，CPU 例外が発生した際に起動する CPU 例外ハンドラに関連した機能を提供しています。

13.2　ユーザ・オウン・コーディング部

RX850V4 では，さまざまな実行環境に対応するために，システム構成管理機能のうち，RX850V4 が処理を実行するうえで必要となるハードウエア依存処理（CPU 例外エントリ処理，初期化ルーチン）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。これにより，さまざまな実行環境への移植性を向上させるとともに，カスタマイズを容易なものとしています。

13.2.1　CPU 例外エントリ処理

CPU 例外エントリ処理は，CPU 例外が発生した際に CPU 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（CPU 例外前処理，ブート処理など）への分岐処理を割り付けるためにユーザ・オウン・コーディング部として切り出されたエントリ処理専用ルーチンです。

なお，コンフィギュレーション時に CPU 例外ハンドラ情報として定義された CPU 例外ハンドラの CPU 例外エントリ処理は，コンフィギュレーション時に作成したシステム・コンフィギュレーション・ファイルに対してコンフィギュレータを実行することにより生成されるエントリ・ファイルに内包されています。したがって，CPU 例外エントリ処理をカスタマイズする必要がない場合には，該当エントリ・ファイルを利用することにより，CPU 例外エントリ処理の記述が不要となります。

- CPU 例外エントリ処理の基本型
  CPU 例外エントリ処理を記述する場合，ハンドラ・アドレスに対して該当処理（CPU 例外前処理，ブート処理など）への分岐処理を割り付けます。
  以下に，CPU 例外エントリ処理をアセンブリ言語で記述する場合の基本型を示します。

```
        -- CPU 例外前処理への分岐処理
        .section        "sec_nam"       -- ハンドラ・アドレスの設定
        jr      __kernel_exc_entry      --CPU 例外前処理への分岐

        -- ブート処理への分岐処理
        .section        "sec_nam"       -- ハンドラ・アドレスの設定
        jr      __boot                  -- ブート処理への分岐
```

- CPU 例外エントリ処理内での処理
  CPU 例外エントリ処理は，CPU 例外が発生した際に RX850V4 を介在させることなく呼び出されるエントリ処理専用ルーチンです。このため，CPU 例外エントリ処理を記述する際には，以下に示す注意点があります。
  - 記述方法
    使用するコンパイラの呼び出し規約にのっとって，アセンブリ言語で記述します。
  - スタックの切り替え
    CPU 例外エントリ処理を実行するうえで切り替えを必要とするスタックは存在しません。したがって，CPU 例外エントリ処理内でスタックの切り替えに関する記述を行う必要がありません。
  - サービス・コールの発行
    CPU 例外エントリ処理では，発生した CPU 例外に対応した処理（ブート処理，CPU 例外ハンドラなど）が実行されるまでの応答性を高速化する目的から，サービス・コールの発行が禁止されています。

以下に，CPU 例外エントリ処理として実行すべき処理の一覧を示します。

- ハンドラ・アドレスの設定
- ラベルの外部宣言
- 該当処理（ブート処理，CPU 例外ハンドラなど）に制御を移す

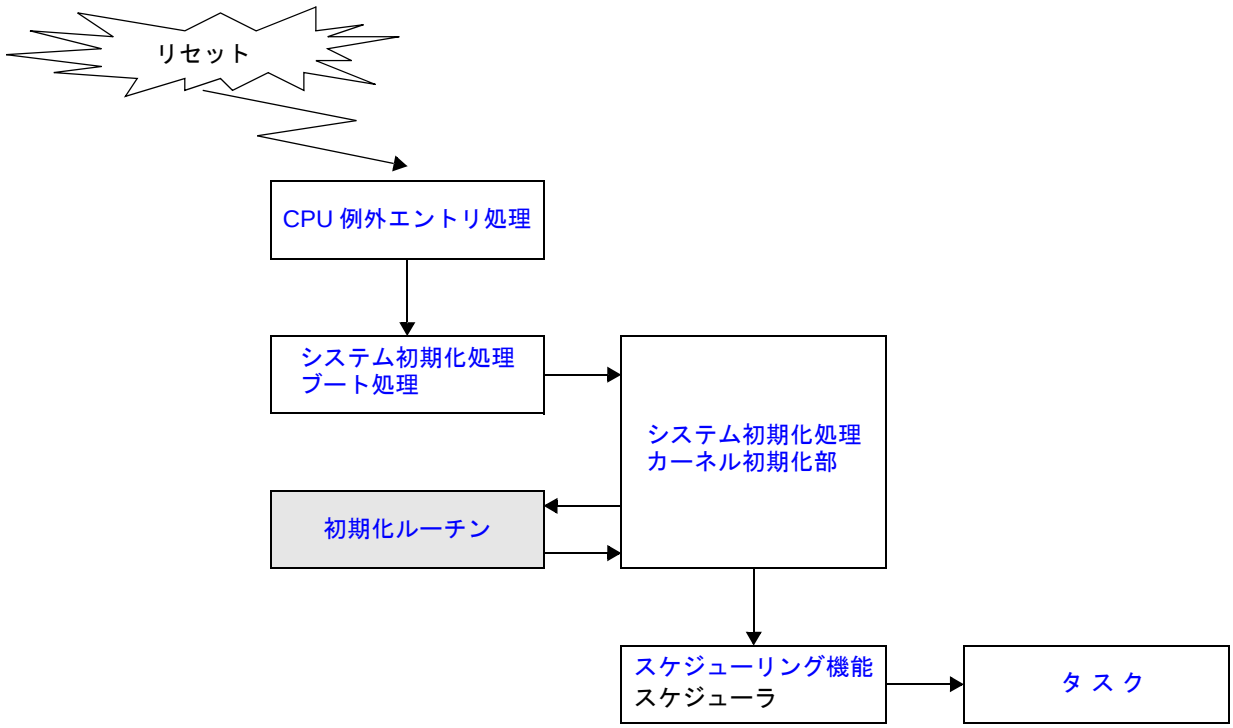
13.2.2　初期化ルーチン

初期化ルーチンは，ユーザの実行環境に依存したハードウエア（周辺コントローラなど）を初期化するためにユーザ・オウン・コーディング部として切り出された初期化処理専用ルーチンであり，カーネル初期化部から呼び出されます。

なお，RX850V4 では，初期化ルーチンを管理するに当たり，初期化ルーチンと一対一に対応した管理オブジェクト（初期化ルーチン管理ブロック）を用いることにより，初期化ルーチンが取り得る状態の管理，および，初期化ルーチン自体の管理を行っています。

以下に，リセットの発生からタスクに制御が移るまでに実行される処理の流れを示します。

図 13 － 1　処理の流れ（初期化ルーチン）



- 初期化ルーチンの基本型

  初期化ルーチンを記述する場合，VP_INT 型の引き数を 1 つ持った void 型の関数として記述します。

  なお，引き数 *exinf* には“初期化ルーチン情報で指定した拡張情報”が設定されます。

  以下に，初期化ルーチンを C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
inirtn ( VP_INT exinf )
{
        .............
        .............

        return;                         /* 初期化ルーチンの終了 */
}
```

- 初期化ルーチン内での処理
  RX850V4 では，カーネル初期化部から初期化ルーチンに制御を移す際，独自の初期化前処理を行っています。また，初期化ルーチンからカーネル初期化部に制御を戻す際にも，独自の初期化後処理を行っています。このため，初期化ルーチンを記述する際には，以下に示す注意点があります。

  - 記述方法
    C 言語，またはアセンブリ言語で記述します。
    C 言語で記述するときは通常の関数と同様に記述することができます。
    アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。
  - スタックの切り替え
    RX850V4 では，初期化ルーチンに制御を移す際，基本情報において指定されたシステム・スタックへの切り替え処理を，カーネル初期化部に制御を戻す際に該当スタックへの切り替え処理を行っています。したがって，初期化ルーチン内でスタックの切り替えに関する記述を行う必要がありません。
  - サービス・コールの発行
    RX850V4 では，初期化ルーチンを“タスク”として位置づけています。このため，初期化ルーチンでは，“状態遷移を引き起こす可能性があるサービス・コールを除いたタスク内から発行可能なサービス・コール”のみが発行可能となります。

    備考　各サービス・コールの発行有効範囲についての詳細は，表 17 － 1 ～表 17 － 14 を参照してください。

  以下に，初期化ルーチンとして実行すべき処理の一覧を示します。

  - 内部ユニットの初期化
  - 周辺コントローラの初期化
  - ROM 領域のデータを RAM 領域にコピー
  - カーネル初期化部に制御を戻す

  備考　RX850V4 が時間管理用に利用するハードウエア（タイマ・コントローラなど）を初期化する際には，システム・コンフィギュレーション・ファイル作成時に基本情報で定義した*基本クロック周期 tim_base* で基本クロック用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。

- マスカブル割り込みの受け付け状態（PSW の ID フラグ）
  処理開始時，マスカブル割り込みの受け付け禁止状態（PSW の ID フラグは 1）となります。
  処理内では，マスカブル割り込みの受け付けを変更することができません。変更した場合，以後の動作は保証されません。

13.2.3　初期化ルーチンの登録

RX850V4 では，初期化ルーチンの静的な登録のみサポートしています。処理プログラムからサービス・コールを発行して動的に登録することはできません。

初期化ルーチンの静的登録とは，システム・コンフィギュレーション・ファイルで静的 API“ATT_INI”を使用して初期化ルーチンを定義することをいいます。

静的 API“ATT_INI”の詳細は，「18.5.14　初期化ルーチン情報」を参照してください。

13.3　CPU 例外ハンドラ

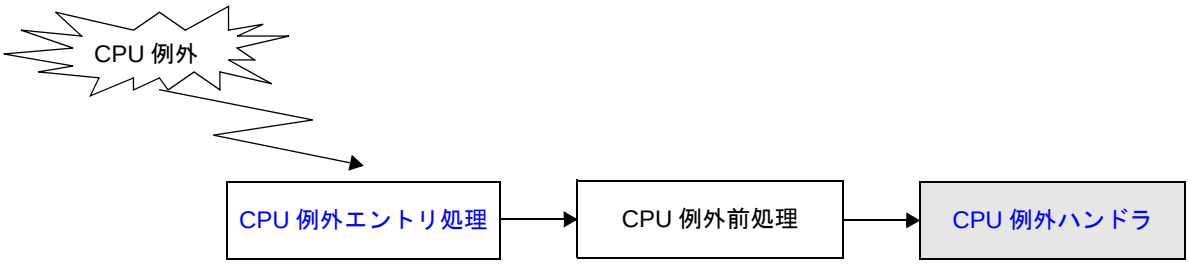
CPU 例外ハンドラは，CPU 例外が発生した際に起動される CPU 例外処理専用ルーチンです。

なお，RX850V4 では，CPU 例外ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため，CPU 例外が発生した際には，システム内で最高優先度を持つタスクが処理を実行中であっても，その処理は中断され，CPU 例外ハンドラに制御が移ります。

また，RX850V4 では，CPU 例外ハンドラを管理するに当たり，CPU 例外ハンドラと一対一に対応した管理オブジェクト（CPU 例外ハンドラ管理ブロック）を用いることにより，CPU 例外ハンドラ自体の管理を行っています。

以下に，CPU 例外の発生から CPU 例外ハンドラに制御が移るまでに実行される処理の流れを示します。

図 13 － 2　処理の流れ（CPU 例外ハンドラ）



13.3.1　CPU 例外ハンドラの基本型

CPU 例外ハンドラを記述する場合，引き数を持たない void 型の関数として記述します。

以下に，CPU 例外ハンドラを C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
exchdr ( void )
{
        .............
        .............

        return;                         /*CPU 例外ハンドラの終了 */
}
```

13.3.2　CPU 例外ハンドラ内での処理

RX850V4 では，CPU 例外が発生した処理プログラムから CPU 例外ハンドラに制御を移す際，独自の CPU 例外前処理を行っています。また，CPU 例外ハンドラから CPU 例外が発生した処理プログラムに制御を戻す際にも，独自の CPU 例外後処理を行っています。このため，CPU 例外ハンドラを記述する際には，以下に示す注意点があります。

- 記述方法
  C 言語，またはアセンブリ言語で記述します。
  C 言語で記述するときは通常の関数と同様に記述することができます。
  アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。

- スタックの切り替え
  RX850V4 では，CPU 例外ハンドラに制御を移す際，基本情報において指定されたシステム・スタックへの切り替え処理を，CPU 例外が発生した処理プログラムに制御を戻す際に該当スタックへの切り替え処理を行っています。したがって，CPU 例外ハンドラ内でスタックの切り替えに関する記述を行う必要がありません。

- サービス・コールの発行
  RX850V4 では，CPU 例外ハンドラを“タスクとは独立したもの（非タスク）”として位置づけています。このため，CPU 例外ハンドラでは，“非タスク内から発行可能なサービス・コール”のみが発行可能となります。

  備考 1　RX850V4 では，CPU 例外ハンドラ内の処理を高速に終了させる目的から，CPU 例外ハンドラ内の処理が完了するまでの間，スケジューラの起動を遅延しています。したがって，CPU 例外ハンドラ内でディスパッチ処理（タスクのスケジューリング処理）を伴うサービス・コール（isig_sem，iset_flg など）が発行された際には，キュー操作などといった処理が行われるだけであり，実際のディスパッチ処理の実行は“CPU 例外ハンドラからの復帰命令（return 命令の発行）”が発行されるまで遅延され，一括して行うようにしています。
  備考 2　各サービス・コールの発行有効範囲についての詳細は，表 17 － 1 ～表 17 － 14 を参照してください。

- マスカブル割り込みの受け付け状態（PSW の ID フラグ）
  処理開始時，マスカブル割り込みの受け付け禁止状態（PSW の ID フラグは 1）となります。
  処理内では，マスカブル割り込みの受け付け状態を変更することはできません。

13.4　CPU 例外ハンドラの登録

RX850V4 では，CPU 例外ハンドラの静的な登録のみサポートしています。処理プログラムからサービス・コールを発行して動的に登録することはできません。

CPU 例外ハンドラの静的登録とは，システム・コンフィギュレーション・ファイルで静的 API“DEF_EXC”を使用して CPU 例外ハンドラを定義することをいいます。

静的 API“DEF_EXC”の詳細は，「18.5.12　CPU 例外ハンドラ情報」を参照してください。

第 14 章　スケジューリング機能

本章では，RX850V4 が提供しているスケジューリング機能について解説しています。

14.1　概　　要

RX850V4 におけるスケジューリング機能では，動的に変化していくタスクの状態を直接参照することにより，タスクの実行順序を管理／決定し，最適なタスクに CPU の利用権を与える機能を提供しています。

14.2　駆動方式

RX850V4 では，スケジューラの駆動方式として何らかの事象（きっかけ）が発生した際に起動する“事象駆動方式”を採用しています。

- 事象駆動方式

  RX850V4 における事象駆動方式では，以下に示した事象が発生した場合にスケジューラを起動し，タスクのスケジューリング処理を実行します。

  - タスクの状態遷移を引き起こす可能性があるサービス・コールの発行
  - 非タスク（周期ハンドラ，割り込みハンドラなど）からの復帰命令の発行
  - 時間管理を行う際に使用している基本クロック用タイマ割り込みの発生
  - vsta_sch の発行

14.3　スケジューリング方式

RX850V4 では，タスクのスケジューリング方式として各タスクに定義されている優先度を利用した“優先度方式”，および，RX850V4 のスケジューリング対象となってからの経過時間を利用した“FCFS 方式”を採用しています。

- 優先度方式

  RX850V4 における優先度方式では，実行可能な状態（RUNNING 状態，または READY 状態）にある全タスクの中から“最も高い優先度を持つタスク”を選び出し，CPU の利用権を与えます。

- FCFS 方式

  RX850V4 における FCFS 方式（First Come First Served 方式）では，実行可能な状態（READY 状態）へと遷移してから“最も時間が経過しているタスク”を選び出し，CPU の利用権を与えます。
  ただし，FCFS 方式によるタスクのスケジューリングは，優先度方式による選び出し基準である“最も高い優先度を持つタスク”が複数存在した場合にかぎり実行されます。

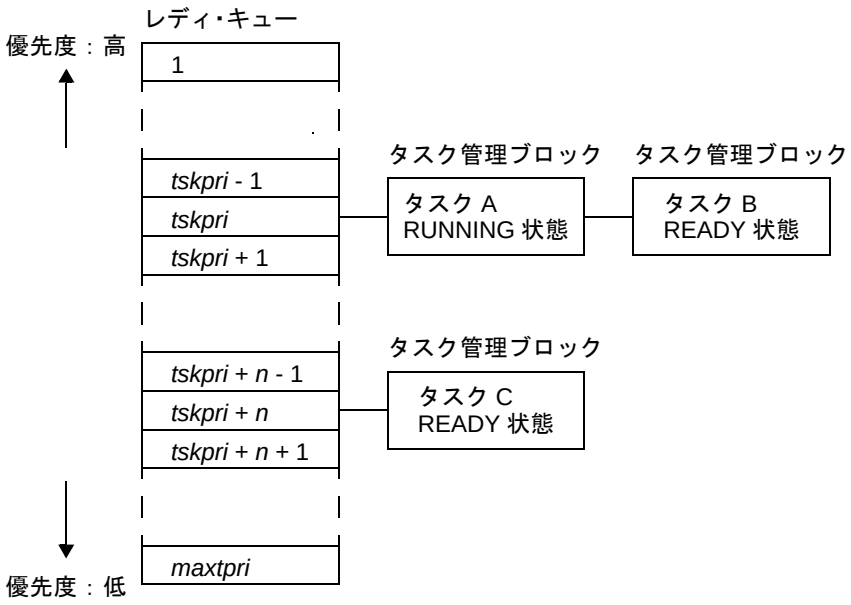
14.3.1　レディ・キュー

RX850V4 では，タスクのスケジューリング方式を実現する手段として“レディ・キュー”を提供しています。

なお, RX850V4 におけるレディ･キューは, 優先度をキーとしたハッシュ･テーブルであり, 実行可能な状態（RUNNING 状態，または READY 状態）へと遷移したタスクの管理ブロック（タスク管理ブロック）が FIFO 順でキューイングされます。このため，スケジューラは，起動された際にレディ・キューの優先度高位から検出処理を実行し，キューイングされているタスクを検出した場合には，該当優先度の先頭タスクに CPU の利用権を与えることにより，RX850V4 のスケジューリング方式（優先度方式，FCFS 方式）を実現しています。

以下に，複数のタスクがレディ・キューにキューイングされている場合を示します。

図 14 － 1　スケジューリング方式（優先度方式，FCFS 方式）の実現

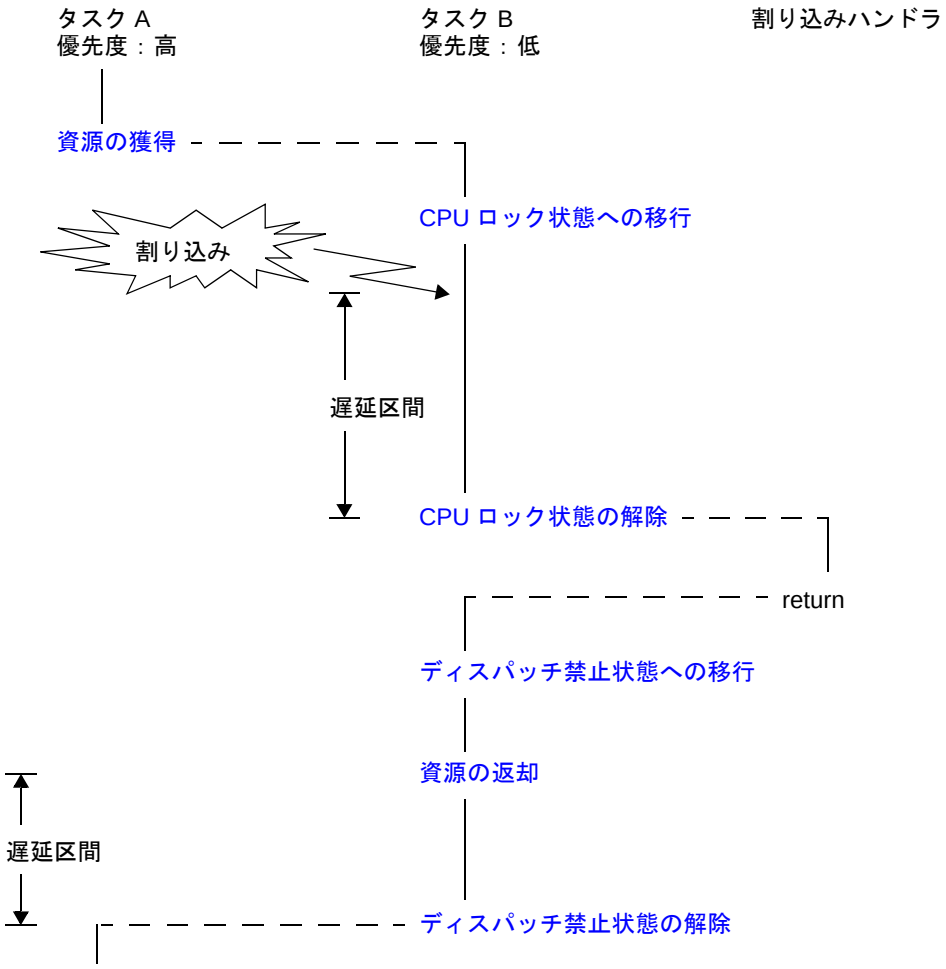


- レディ・キューの生成
  RX850V4 では，レディ・キューの静的な生成のみサポートしています。処理プログラムからサービス・コールを発行して動的に生成することはできません。
  レディ・キューの静的生成とは，システム・コンフィギュレーション・ファイルでシステム情報“MAX_PRI”を使用して最大タスク優先度を定義することをいいます。
  システム情報“MAX_PRI”の詳細は，「18.4.2　基本情報」を参照してください。

14.4　スケジューリングのロック機能

RX850V4 では，処理プログラムからスケジューラの状態を明示的に操作し，ディスパッチ処理を禁止／許可する機能（スケジューリングのロック機能）を提供しています。
以下に，スケジューリングのロック機能を利用した際の処理の流れを示します。

図 14 － 2　スケジューリングのロック機能



なお，スケジューリングのロック機能は，以下に示したサービス・コールを処理プログラムから発行することにより実現されます。

loc_cpu	iloc_cpu	unl_cpu	iunl_cpu
dis_dsp	ena_dsp		

14.5　アイドル・ルーチン

アイドル・ルーチンは，CPU が提供しているスタンバイ機能を有効活用（低消費電力システムの実現）するためにユーザ・オウン・コーディング部として切り出されたアイドル処理専用ルーチンであり，RX850V4 のスケジューリング対象となるタスク（RUNNING 状態，または READY 状態のタスク）がシステム内に 1 つも存在しなくなった際にスケジューラから呼び出されます。

なお，RX850V4 では，アイドル・ルーチンを管理するに当たり，アイドル・ルーチンと一対一に対応した管理オブジェクト（アイドル・ルーチン管理ブロック）を用いることにより，アイドル・ルーチンが取り得る状態の管理，および，アイドル・ルーチン自体の管理を行っています。

14.5.1　アイドル・ルーチンの基本型

アイドル・ルーチンを記述する場合，引き数を持たない void 型の関数として記述します。
以下に，アイドル・ルーチンを C 言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

void
idlrtn ( void )
{
        .............
        .............

        return;                         /* アイドル・ルーチンの終了 */
}
```

14.5.2　アイドル・ルーチン内での処理

RX850V4 では，スケジューラからアイドル・ルーチンに制御を移す際，独自のアイドル前処理を行っています。また，アイドル・ルーチンからスケジューラに制御を戻す際にも，独自のアイドル後処理を行っています。このため，アイドル・ルーチンを記述する際には，以下に示す注意点があります。

- 記述方法
  C 言語，またはアセンブリ言語で記述します。
  C 言語で記述するときは通常の関数と同様に記述することができます。
  アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。

- スタックの切り替え
  RX850V4 では，アイドル・ルーチンに制御を移す際，基本情報において指定されたシステム・スタックへの切り替え処理を行っています。したがって，アイドル・ルーチン内でスタックの切り替えに関する記述を行う必要がありません。

- サービス・コールの発行
  RX850V4 では，アイドル・ルーチン内でサービス・コールの発行を制限（禁止）しています。

- マスカブル割り込みの受け付け状態（PSW の ID フラグ）
  処理開始時，マスカブル割り込みの受け付け許可状態（PSW の ID フラグは 0）となります。
  処理内では，マスカブル割り込みの受け付け状態を変更することができます。
  処理終了後，マスカブル割り込みの受け付け状態は他の処理プログラムに継承されません。

- 処理のループ
  アイドル・ルーチンの処理終了後は，アイドル・ルーチンの処理が先頭から再開されます。再開された際，前に動作していたアイドル・ルーチンの状態（スタック・ポインタ，マスカブル割り込み受け付け状態）は継承されません。

以下に，アイドル・ルーチンとして実行すべき処理の一覧を示します。

- CPU が提供しているスタンバイ機能の有効活用

14.6　アイドル・ルーチンの登録

RX850V4 では，アイドル・ルーチンの静的な登録のみサポートしています。処理プログラムからサービス・コールを発行して動的に登録することはできません。

アイドル・ルーチンの静的登録とは，システム・コンフィギュレーション・ファイルで静的 API“VATT_IDL”を使用してアイドル・ルーチンを定義することをいいます。

静的 API“VATT_IDL”の詳細は，「18.5.15　アイドル・ルーチン情報」を参照してください。

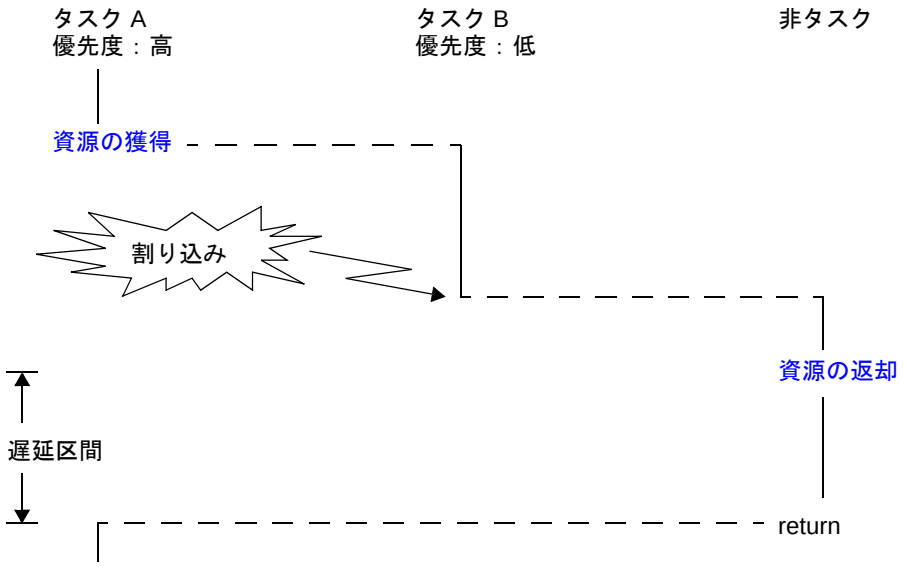
備考　アイドル・ルーチン情報が未定義の場合には，コンフィギュレーション時にデフォルト・アイドル・ルーチン（関数名：_kernel_default_idlrtn）の登録処理を行っています。

14.7　非タスク内におけるスケジューリング処理

RX850V4 では，非タスク（周期ハンドラ，割り込みハンドラなど）内の処理を高速に終了させる目的から，非タスク内の処理が完了するまでの間，スケジューラの起動を遅延しています。したがって，非タスク内でディスパッチ処理（タスクのスケジューリング処理）を伴うサービス・コール（isig_sem，iset_flg など）が発行された際には，キュー操作などといった処理が行われるだけであり，実際のディスパッチ処理の実行は“非タスクからの復帰命令（return 命令の発行）”が発行されるまで遅延され，一括して行うようにしています。

以下に，非タスク内でディスパッチ処理を伴うサービス・コールを発行した際の処理の流れを示します。

図 14 － 3　非タスク内におけるスケジューリング処理



第 15 章　システム初期化処理

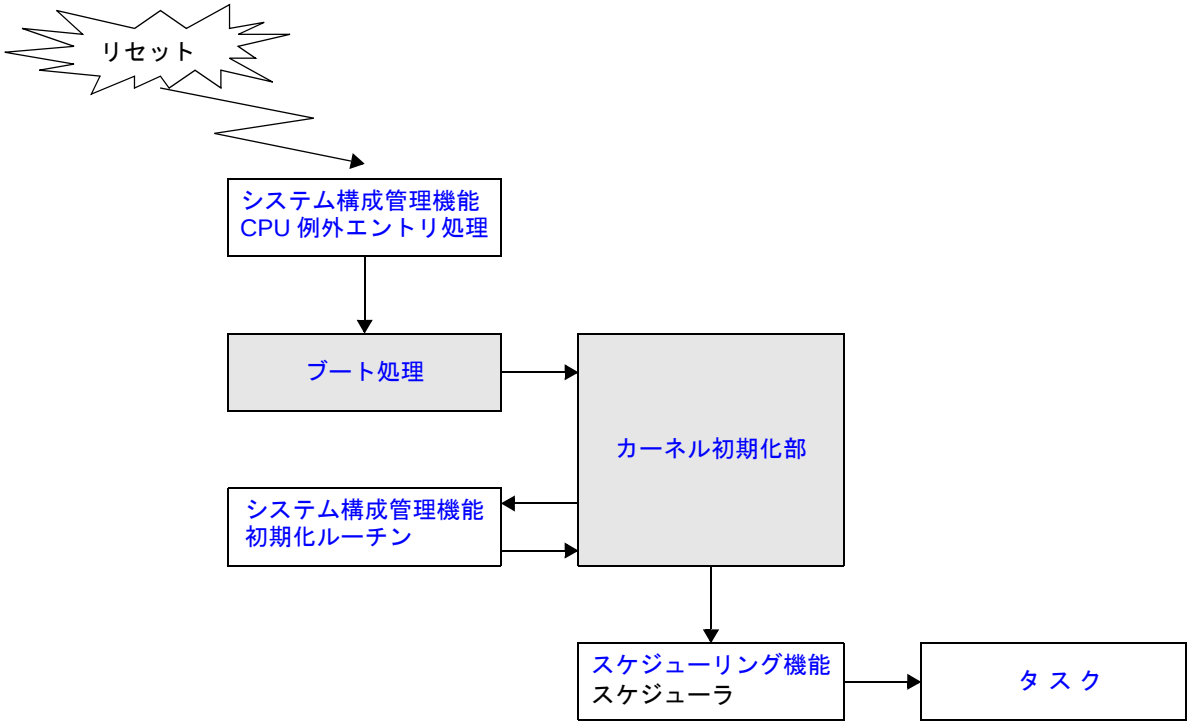
本章では，RX850V4 が提供しているシステム初期化処理について解説しています。

15.1　概　　要

RX850V4 におけるシステム初期化処理では，リセットの発生から処理プログラムに制御を移すまでに必要となる“RX850V4 が処理を実行するうえで必要となるハードウエア，および，ソフトウエアの初期化処理”を提供しています。

以下に，リセットの発生から処理プログラム（タスク）に制御が移るまでに実行される処理の流れを示します。

図 15 － 1　処理の流れ（システム初期化処理）



15.2　ユーザ・オウン・コーディング部

RX850V4 では，さまざまな実行環境に対応するために，システム初期化処理のうち，RX850V4 が処理を実行するうえで必要となるハードウエア依存処理（ブート処理）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。これにより，さまざまな実行環境への移植性を向上させるとともに，カスタマイズを容易なものとしています。

15.2.1　ブート処理

ブート処理は，RX850V4 が処理を実行するうえで必要となる最低限のハードウエアを初期化するためにユーザ・オウン・コーディング部として切り出された初期化処理専用ルーチンであり，CPU 例外エントリ処理から呼び出されます。

- ブート処理の基本型
  ブート処理を記述する場合，引き数を持たない void 型の関数として記述します。
  以下に，ブート処理をアセンブリ言語で記述する場合の基本型を示します。

```
#include        <kernel.h>              /* 標準ヘッダ・ファイルの定義 */

        .text
        .align  0x4
        .globl  __boot
__boot :
        .extern __kernel_sit

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

        mov     #__kernel_sit, r6       /*SIT の先頭アドレスの設定 */
        jarl    __kernel_start, lp      /* カーネル初期化部に制御を移す */
```

- ブート処理内での処理
  ブート処理は，CPU 例外エントリ処理から RX850V4 を介在させることなく呼び出される初期化処理専用ルーチンです。このため，ブート処理を記述する際には，以下に示す注意点があります。
  - 記述方法
    C 言語，またはアセンブリ言語で記述します。
    C 言語で記述するときは通常の関数と同様に記述することができます。
    アセンブリ言語で記述するときは使用するコンパイラの呼び出し規約にのっとって作成してください。
  - スタックの切り替え
    ブート処理が開始された時点では“スタックの設定処理”が行われていません。したがって，ブート処理用スタックを使用する際には，ブート処理の開始部分でスタックの設定処理（スタック・ポインタ sp の設定）を記述する必要があります。
  - サービス・コールの発行
    ブート処理が開始された時点では“カーネル初期化部の実行”が行われていません。したがって，ブート処理では，サービス・コールの発行が禁止されています。

  以下に，ブート処理として実行すべき処理の一覧を示します。
  - グローバル・ポインタ gp，テキスト・ポインタ tp の設定
  - エレメント・ポインタ ep の設定
  - スタック・ポインタ sp の設定
  - CPU の内部ユニット，周辺コントローラの初期化
  - 初期値なしメモリ領域（bss セクションなど）の初期化
  - システム情報テーブル（SIT）の先頭アドレスを r6 に設定
  - カーネル初期化部 _kernel_start に制御を移す

備考 1　グローバル・ポインタ gp，テキスト・ポインタ tp，エレメント・ポインタ ep の設定は，ブート処理の開始部分で行う必要があります。なお，スタック・ポインタ sp については，ブート処理内でブート処理用スタックを使用する場合にかぎり，設定が必要となります。

備考 2　エレメント・ポインタ ep にデータ・セクションのベース・アドレスを設定します。

15.3　カーネル初期化部

カーネル初期化部は，RX850V4 が処理を実行するうえで必要となる最低限のソフトウエアを初期化するために用意された初期化処理専用ルーチンであり，ブート処理から呼び出されます。
なお，カーネル初期化部では，以下に示した処理が実行されます。

- 管理領域の確保／初期化
  - 管理オブジェクト
    システム情報テーブル
    システム管理テーブル
    レディ・キュー
    割り込みマスク情報配列
    割り込みマスク管理配列
    カーネル初期化ルーチン情報配列
    カーネル共通ルーチン情報ブロック
    バージョン情報ブロック
    タスク情報ブロック配列
    基本タスク管理ブロック配列
    拡張タスク管理ブロック配列
    タスク例外管理ブロック配列
    セマフォ情報ブロック配列
    セマフォ管理ブロック配列
    イベントフラグ情報ブロック配列
    イベントフラグ管理ブロック配列
    データ・キュー情報ブロック配列
    データ・キュー管理ブロック配列
    メールボックス情報ブロック配列
    メールボックス管理ブロック配列
    ミューテックス情報ブロック配列
    ミューテックス管理ブロック配列
    固定長メモリ・プール情報ブロック配列
    固定長メモリ・プール管理ブロック配列
    可変長メモリ・プール情報ブロック配列
    可変長メモリ・プール管理ブロック配列
    周期ハンドラ情報ブロック配列
    周期ハンドラ管理ブロック配列
    拡張サービス・コール・ルーチン情報ブロック配列
    割り込みハンドラ /CPU 例外ハンドラ情報ブロック配列
    割り込みハンドラ /CPU 例外ハンドラ ID 配列
    初期化ルーチン情報ブロック配列
    アイドル・ルーチン情報ブロック
  - スタック
    システム・スタック
    タスク・スタック
  - バッファ
    データ・キュー領域
  - メモリ・プール
    固定長メモリ・プール
    可変長メモリ・プール
- システム時刻の初期化
- 時間管理用割り込みハンドラの登録
- 初期化ルーチンの登録
- アイドル・ルーチンの登録
- 初期化ルーチンの呼び出し
- スケジューラに制御を移す

備考　カーネル初期化部は，RX850V4 が提供しているシステム初期化処理に内包されています。したがって，ユーザがカーネル初期化部を記述する必要はありません。
なお，カーネル初期化部が異常終了した場合，以下に示した値がレジスタ lp に設定されます。

マクロ	数値	意味
E_CFG_VER	1	リアルタイム OS のバージョン番号が不正である
E_CFG_CPU	2	CPU 種別が不正である
E_CFG_CC	3	C コンパイラ・パッケージの種類が不正である
E_CFG_REG	4	レジスタ・モード種別が不正である
E_CFG_NOMEM	5	メモリが足りません

第 16 章　データ・タイプとマクロ

本章では，RX850V4 が提供するサービス・コールを発行する際に使用するデータ・タイプ，データ構造体，マクロについて解説しています。

なお，マクロ，およびデータ構造体の定義は，<rx_root>¥inc850 に格納されている各ヘッダ・ファイルで行われています。

備考　<rx_root> は，RX850V4 のインストール・フォルダを表しています。
デフォルトでは，“ C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥RX850V4¥V*x.xx*”となります。

16.1　データ・タイプ

以下に，サービス・コール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，ITRON 仕様共通マクロ定義ファイル <rx_root>¥inc850¥itron.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥types.h で行われています。

表 16 － 1　データ・タイプ

マクロ	型	意味
B	signed char	符号付き 8 ビット整数
H	signed short	符号付き 16 ビット整数
W	signed long	符号付き 32 ビット整数
UB	unsigned char	符号なし 8 ビット整数
UH	unsigned short	符号なし 16 ビット整数
UW	unsigned long	符号なし 32 ビット整数
VB	signed char	データ・タイプが一定しない値（8 ビット）
VH	signed short	データ・タイプが一定しない値（16 ビット）
VW	signed long	データ・タイプが一定しない値（32 ビット）
VP	void *	データ・タイプが一定しない値（ポインタ）
FP	void (*)	処理プログラムの起動アドレス（ポインタ）
INT	signed int	符号付き 32 ビット整数
UINT	unsigned int	符号なし 32 ビット整数
BOOL	signed long	真偽値（TRUE，または FALSE）
FN	signed short	拡張サービス・コール・ルーチンの機能コード番号
ER	signed long	サービス・コールからの戻り値
ID	signed short	管理オブジェクトの ID
ATR	unsigned short	管理オブジェクトの属性
STAT	unsigned short	管理オブジェクトの状態
MODE	unsigned short	サービス・コールの動作モード
PRI	signed short	タスク，またはメッセージの優先度
SIZE	unsigned long	領域のサイズ（単位：バイト）
TMO	signed long	タスクの待ち時間（単位：ミリ秒）
RELTIM	unsigned long	相対時間（単位：ミリ秒）
VP_INT	signed int	データ・タイプが一定しない値（ポインタ），または符号付き 32 ビット整数

マクロ	型	意味
ER_BOOL	signed long	真偽値（TRUE，または FALSE）
ER_ID	signed long	管理オブジェクトの ID
ER_UINT	signed int	符号なし 32 ビット整数
TEXPTN	unsigned int	タスク例外要因，保留例外要因
FLGPTN	unsigned int	イベントフラグのビット・パターン
INTNO	unsigned short	例外コード番号
EXCNO	unsigned short	例外コード番号

16.2　データ構造体

RX850V4 が提供するサービス・コールを発行する際に使用するデータ構造体（タスク詳細情報，タスク基本情報など）について以下に示します。

16.2.1　タスク詳細情報

以下に，ref_tsk，および，iref_tsk を発行する際に使用するタスク詳細情報 T_RTSK を示します。

なお，タスク詳細情報 T_RTSK の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rtsk {
        STAT    tskstat;        /* 現在状態 */
        PRI     tskpri;         /* 現在優先度 */
        PRI     tskbpri;        /* システム予約領域 */
        STAT    tskwait;        /* 待ち要因 */
        ID      wobjid;         /* 管理オブジェクトの ID*/
        TMO     lefttmo;        /* 残り時間 */
        UINT    actcnt;         /* 起動要求数 */
        UINT    wupcnt;         /* 起床要求数 */
        UINT    suscnt;         /* サスペンド要求数 */
        ATR     tskatr;         /* 属性 */
        PRI     itskpri;        /* 初期優先度 */
        ID      memid;          /* システム予約領域 */
} T_RTSK;
```

以下に，タスク詳細情報 T_RTSK の詳細を示します。

- tskstat
  タスクの現在状態が格納されます。

TTS_RUN：	RUNNING 状態
TTS_RDY：	READY 状態
TTS_WAI：	WAITING 状態
TTS_SUS：	SUSPENDED 状態
TTS_WAS：	WAITING-SUSPENDED 状態
TTS_DMT：	DORMANT 状態

- tskpri
  タスクの現在優先度が格納されます。

- tskbpri
  システム予約領域です。

- tskwait
  タスクの待ち要因（WAITING 状態の種類）が格納されます。

TTW_SLP：	slp_tsk，または tslp_tsk による起床待ち状態
TTW_DLY：	dly_tsk による時間経過待ち状態
TTW_SEM：	wai_sem，または twai_sem による資源獲得待ち状態
TTW_FLG：	wai_flg，または twai_flg によるイベントフラグ待ち状態
TTW_SDTQ：	snd_dtq，または tsnd_dtq によるデータ送信待ち状態
TTW_RDTQ：	rcv_dtq，または trcv_dtq によるデータ受信待ち状態
TTW_MBX：	rcv_mbx，または trcv_mbx によるメッセージ受信待ち状態
TTW_MTX：	loc_mtx，または tloc_mtx によるミューテックス待ち状態
TTW_MPF：	get_mpf，または tget_mpf による固定長メモリ・ブロック獲得待ち状態
TTW_MPL：	get_mpl，または tget_mpl による可変長メモリ・ブロック獲得待ち状態

- wobjid
  タスクが WAITING 状態へと遷移するきっかけとなった管理オブジェクト（セマフォ，イベントフラグなど）の ID が格納されます。

- lefttmo
  タスクの時間経過待ち状態が解除されるまでの残り時間（単位：ミリ秒）が格納されます。

- actcnt
  タスクの起動要求数が格納されます。

- wupcnt
  タスクの起床要求数が格納されます。

- suscnt
  タスクのサスペンド要求数が格納されます。

- tskatr
  タスクの属性（記述言語，初期起動状態など）が格納されます。

  タスクの記述言語（ビット 0）
  TA_HLNG：　C 言語
  TA_ASM：　アセンブリ言語

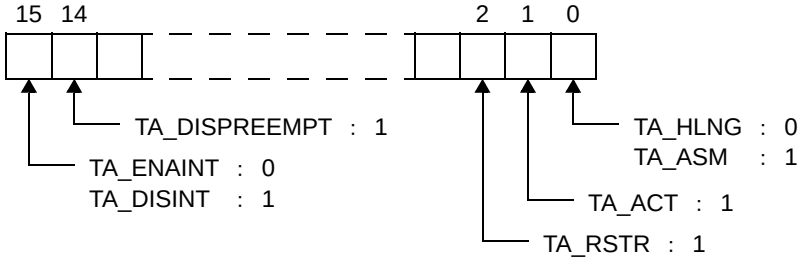
  タスクの初期起動状態（ビット 1）
  TA_ACT：　READY 状態

  タスクの種別（ビット 2）
  TA_RSTR：　制約タスク

  プリエンプトの受け付け状態（ビット 14）
  TA_DISPREEMPT：　DORMANT 状態から READY 状態へと遷移した際には，受け付け禁止状態

  初期割り込み状態（ビット 15）
  TA_ENAINT：　DORMANT 状態から READY 状態へと遷移した際には，受け付け許可状態
  TA_DISINT：　DORMANT 状態から READY 状態へと遷移した際には，受け付け禁止状態

  【 tskatr の構造 】



- itskpri
  タスクの初期優先度が格納されます。

- memid
  システム予約領域です。

16.2.2　タスク基本情報

以下に，ref_tst，および，iref_tst を発行する際に使用するタスク基本情報 T_RTST を示します。
なお，タスク基本情報 T_RTST の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rtst {
        STAT    tskstat;        /* 現在状態 */
        STAT    tskwait;        /* 待ち要因 */
} T_RTST;
```

以下に，タスク基本情報 T_RTST の詳細を示します。

- tskstat
  タスクの現在状態が格納されます。

TTS_RUN：	RUNNING 状態
TTS_RDY：	READY 状態
TTS_WAI：	WAITING 状態
TTS_SUS：	SUSPENDED 状態
TTS_WAS：	WAITING-SUSPENDED 状態
TTS_DMT：	DORMANT 状態

- tskwait
  タスクの待ち要因（WAITING 状態の種類）が格納されます。

TTW_SLP：	slp_tsk，または tslp_tsk による起床待ち状態
TTW_DLY：	dly_tsk による時間経過待ち状態
TTW_SEM：	wai_sem，または twai_sem による資源獲得待ち状態
TTW_FLG：	wai_flg，または twai_flg によるイベントフラグ待ち状態
TTW_SDTQ：	snd_dtq，または tsnd_dtq によるデータ送信待ち状態
TTW_RDTQ：	rcv_dtq，または trcv_dtq によるデータ受信待ち状態
TTW_MBX：	rcv_mbx，または trcv_mbx によるメッセージ受信待ち状態
TTW_MTX：	loc_mtx，または tloc_mtx によるミューテックス待ち状態
TTW_MPF：	get_mpf，または tget_mpf による固定長メモリ・ブロック獲得待ち状態
TTW_MPL：	get_mpl，または tget_mpl による可変長メモリ・ブロック獲得待ち状態

16.2.3　タスク例外処理ルーチン詳細情報

以下に，ref_tex，および，iref_tex を発行する際に使用するタスク例外処理ルーチン詳細情報 T_RTEX を示します。
なお，タスク例外処理ルーチン詳細情報 T_RTEX の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rtex {
        STAT    texstat;        /* 現在状態 */
        TEXPTN  pndptn;         /* 保留例外要因 */
        ATR     texatr;         /* 属性 */
} T_RTEX;
```

以下に，タスク例外処理ルーチン詳細情報 T_RTEX の詳細を示します。

- texstat
  タスク例外処理ルーチンの現在状態が格納されます。

  TTEX_ENA：　　タスク例外処理許可状態
  TTEX_DIS：　　タスク例外処理禁止状態

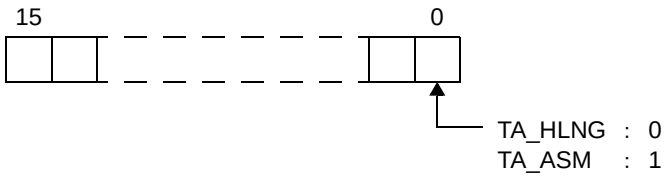
- pndptn
  タスク例外処理ルーチンの保留例外要因が格納されます。
  なお，保留例外要因とは，ras_tex，または iras_tex による例外処理要求発行後，対象タスクが RUNNING 状態へと遷移するまでの間に複数回のタスク例外処理要求が発行された場合に行われる保留処理の結果（タスク例外要因の論理和 OR）を意味します。

  備考　　ras_tex，または iras_tex による例外処理要求が未発行の場合，0x0 が格納されます。

- texatr
  タスク例外処理ルーチンの属性（記述言語）が格納されます。

  タスク例外処理ルーチンの記述言語（ビット 0）
  TA_HLNG：　　C 言語
  TA_ASM：　　アセンブリ言語

  【 texatr の構造 】



16.2.4　セマフォ詳細情報

以下に，ref_sem，および，iref_sem を発行する際に使用するセマフォ詳細情報 T_RSEM を示します。
なお，セマフォ詳細情報 T_RSEM の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rsem {
        ID      wtskid;         /* 待ちタスクの有無 */
        UINT    semcnt;         /* 現在資源数 */
        ATR     sematr;         /* 属性 */
        UINT    maxsem;         /* 最大資源数 */
} T_RSEM;
```

以下に，セマフォ詳細情報 T_RSEM の詳細を示します。

- wtskid
  セマフォの待ちキューにタスクがキューイングされているか否かが格納されます。

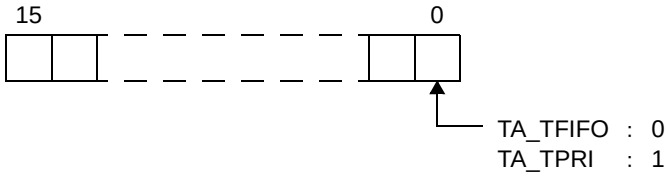
  TSK_NONE：　　待ちキューにタスクはキューイングされていない
  その他：　　待ちキューの先頭にキューイングされているタスクの ID

- semcnt
  セマフォの現在資源数が格納されます。

- sematr
  セマフォの属性（キューイング方式）が格納されます。

  タスク・キューイング方式（ビット 0）
  TA_TFIFO：　　資源の獲得要求を行った順
  TA_TPRI：　　タスクの優先度順

  【 sematr の構造 】



- maxsem
  セマフォの最大資源数が格納されます。

16.2.5　イベントフラグ詳細情報

以下に，ref_flg，および，iref_flg を発行する際に使用するイベントフラグ詳細情報 T_RFLG を示します。
なお，イベントフラグ詳細情報 T_RFLG の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rflg {
        ID      wtskid;         /* 待ちタスクの有無 */
        FLGPTN  flgptn;         /* 現在ビット・パターン */
        ATR     flgatr;         /* 属性 */
} T_RFLG;
```

以下に，イベントフラグ詳細情報 T_RFLG の詳細を示します。

- wtskid
  イベントフラグの待ちキューにタスクがキューイングされているか否かが格納されます。

  TSK_NONE：　　待ちキューにタスクはキューイングされていない
  その他：　　待ちキューの先頭にキューイングされているタスクの ID

- flgptn
  イベントフラグの現在ビット・パターンが格納されます。

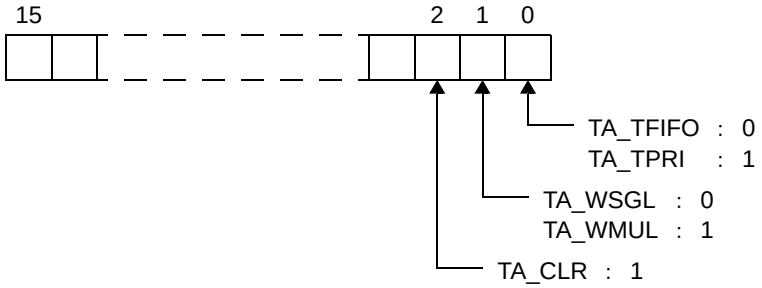
- flgatr
  イベントフラグの属性（キューイング方式，キューイング可能なタスクの最大数など）が格納されます。

  タスク・キューイング方式（ビット 0）
  TA_TFIFO：　　ビット・パターンのチェックを行った順
  TA_TPRI：　　タスクの優先度順

  キューイング可能なタスクの最大数（ビット 1）
  TA_WSGL：　　1 個
  TA_WMUL：　　複数

  ビット・パターンのクリア（ビット 2）
  TA_CLR：　　要求条件が満足した際，ビット・パターンをクリア（0x0 の設定）

  【 flgatr の構造 】



16.2.6　データ・キュー詳細情報

以下に，ref_dtq，および，iref_dtq を発行する際に使用するデータ・キュー詳細情報 T_RDTQ を示します。
なお，データ・キュー詳細情報 T_RDTQ の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rdtq {
        ID      stskid;         /* データ送信待ちタスクの有無 */
        ID      rtskid;         /* データ受信待ちタスクの有無 */
        UINT    sdtqcnt;        /* 未受信データの総数 */
        ATR     dtqatr;         /* 属性 */
        UINT    dtqcnt;         /* データ数 */
        ID      memid;          /* システム予約領域 */
} T_RDTQ;
```

以下に，データ・キュー詳細情報 T_RDTQ の詳細を示します。

- stskid
  データ・キューの送信待ちキューにタスクがキューイングされているか否かが格納されます。

  TSK_NONE：　送信待ちキューにタスクはキューイングされていない
  その他：　送信待ちキューの先頭にキューイングされているタスクの ID

- rtskid
  データ・キューの受信待ちキューにタスクがキューイングされているか否かが格納されます。

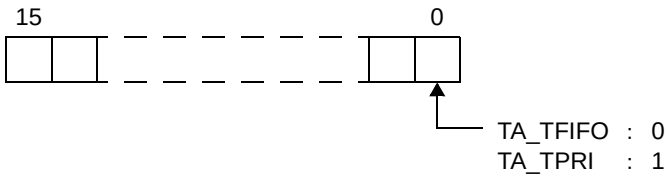
  TSK_NONE：　受信待ちキューにタスクはキューイングされていない
  その他：　受信待ちキューの先頭にキューイングされているタスクの ID

- sdtqcnt
  データ・キューのデータ・キュー領域にキューイングされている未受信データの総数が格納されます。

- dtqatr
  データ・キューの属性（キューイング方式）が格納されます。

  タスク・キューイング方式（ビット 0）
  TA_TFIFO：　データの送信要求を行った順
  TA_TPRI：　タスクの優先度順

  【 dtqatr の構造 】



- dtqcnt
  データ・キューのデータ・キュー領域にキューイング可能なデータの最大数が格納されます。

- memid
  システム予約領域です。

16.2.7　メッセージ

以下に，snd_mbx，isnd_mbx，rcv_mbx，prcv_mbx，iprcv_mbx，および，trcv_mbx を発行する際に使用するメッセージ T_MSG，T_MSG_PRI を示します。

なお，メッセージ T_MSG，T_MSG_PRI の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥types.h で行われています。

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  t_msg {
        struct  t_msg   *msgnext;       /* システム予約領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  t_msg_pri {
        struct  t_msg   msgque;         /* システム予約領域 */
        PRI     msgpri;                 /* 優先度 */
} T_MSG_PRI;
```

以下に，メッセージ T_MSG，T_MSG_PRI の詳細を示します。

- msgnext，msgque
  システム予約領域です。

- msgpri
  メッセージの優先度が格納されます。

  備考 1　RX850V4 におけるメッセージの優先度は，その値が小さいほど，高い優先度であることを意味します。

  備考 2　メッセージの優先度として指定可能な値は，システム・コンフィギュレーション・ファイル作成時にメールボックス情報（*最大メッセージ優先度 maxmpri*）で定義された値域に限定されます。

16.2.8　メールボックス詳細情報

以下に，ref_mbx，および，iref_mbx を発行する際に使用するメールボックス詳細情報 T_RMBX を示します。
なお，メールボックス詳細情報 T_RMBX の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rmbx {
        ID      wtskid;         /* 待ちタスクの有無 */
        T_MSG   *pk_msg;        /* 待ちメッセージの有無 */
        ATR     mbxatr;         /* 属性 */
} T_RMBX;
```

以下に，メールボックス詳細情報 T_RMBX の詳細を示します。

- wtskid
  メールボックスの待ちキューにタスクがキューイングされているか否かが格納されます。

  TSK_NONE：　　待ちキューにタスクはキューイングされていない
  その他：　　待ちキューの先頭にキューイングされているタスクの ID

- pk_msg
  メールボックスの待ちキューにメッセージがキューイングされているか否かが格納されます。

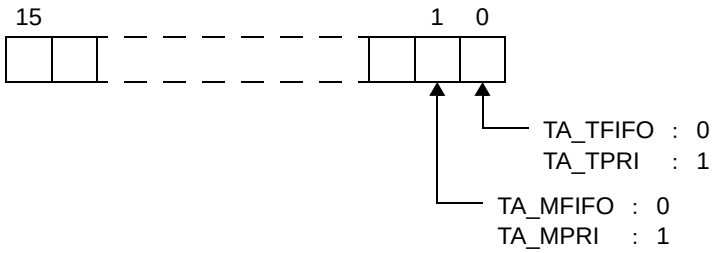
  NULL：　　待ちキューにメッセージはキューイングされていない
  その他：　　待ちキューの先頭にキューイングされているメッセージの先頭アドレス

- mbxatr
  メールボックスの属性（キューイング方式）が格納されます。

  タスク・キューイング方式（ビット 0）
  TA_TFIFO：　　メッセージの受信要求を行った順
  TA_TPRI：　　タスクの優先度順

  メッセージ・キューイング方式（ビット 1）
  TA_MFIFO：　　メッセージの送信要求を行った順
  TA_MPRI：　　メッセージの優先度順

  【 mbxatr の構造 】



16.2.9　ミューテックス詳細情報

以下に，ref_mtx，および，iref_mtx を発行する際に使用するミューテックス詳細情報 T_RMTX を示します。
なお，ミューテックス詳細情報 T_RMTX の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rmtx {
        ID      htskid;         /* ロックの有無 */
        ID      wtskid;         /* 待ちタスクの有無 */
        ATR     mtxatr;         /* 属性 */
        PRI     ceilpri;        /* システム予約領域 */
} T_RMTX;
```

以下に，ミューテックス詳細情報 T_RMTX の詳細を示します。

- htskid
  ミューテックスをロックしているタスクが存在しているか否かが格納されます。

  TSK_NONE：　　ロックしているタスクは存在しない
  その他：　　ロックしているタスクの ID

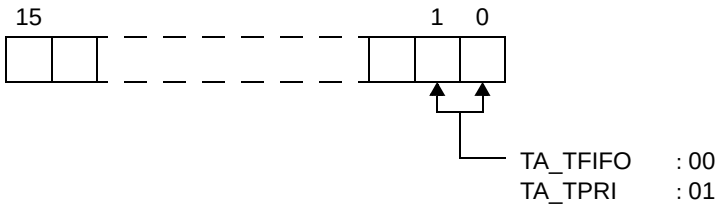
- wtskid
  ミューテックスの待ちキューにタスクがキューイングされているか否かが格納されます。

  TSK_NONE：　　待ちキューにタスクはキューイングされていない
  その他：　　待ちキューの先頭にキューイングされているタスクの ID

- mtxatr
  ミューテックスの属性（キューイング方式）が格納されます。

  タスク・キューイング方式（ビット 0 ～ 1）
  TA_TFIFO：　　ミューテックスのロック要求を行った順
  TA_TPRI：　　タスクの優先度順

  【 mtxatr の構造 】



- ceilpri
  システム予約領域です。

16.2.10 固定長メモリ・プール詳細情報

以下に，ref_mpf，および，iref_mpf を発行する際に使用する固定長メモリ・プール詳細情報 T_RMPF を示します。
なお，固定長メモリ・プール詳細情報 T_RMPF の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rmpf {
        ID      wtskid;         /* 待ちタスクの有無 */
        UINT    fblkcnt;        /* 空き固定長メモリ・ブロックの総数 */
        ATR     mpfatr;         /* 属性 */
        ID      memid;          /* システム予約領域 */
} T_RMPF;
```

以下に，固定長メモリ・プール詳細情報 T_RMPF の詳細を示します。

- wtskid
  固定長メモリ・プールの待ちキューにタスクがキューイングされているか否かが格納されます。

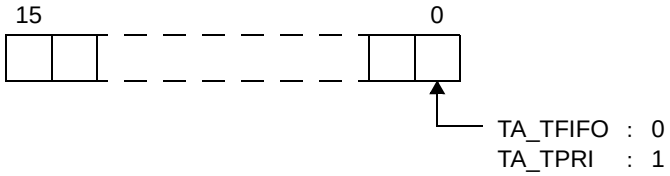
  TSK_NONE：　待ちキューにタスクはキューイングされていない
  その他：　待ちキューの先頭にキューイングされているタスクの ID

- fblkcnt
  固定長メモリ・プールから獲得可能な空き固定長メモリ・ブロックの総数が格納されます。

- mpfatr
  固定長メモリ・プールの属性（キューイング方式）が格納されます。

  タスク・キューイング方式（ビット 0）
  TA_TFIFO：　固定長メモリ・ブロックの獲得要求を行った順
  TA_TPRI：　タスクの優先度順

  【 mpfatr の構造 】



- memid
  システム予約領域です。

16.2.11 可変長メモリ・プール詳細情報

以下に，ref_mpl，および，iref_mpl を発行する際に使用する可変長メモリ・プール詳細情報 T_RMPL を示します。
なお，可変長メモリ・プール詳細情報 T_RMPL の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rmpl {
        ID      wtskid;         /* 待ちタスクの有無 */
        SIZE    fmplsz;         /* 空き可変長メモリ・ブロックの合計サイズ */
        UINT    fblksz;         /* 空き可変長メモリ・ブロックの最大サイズ */
        ATR     mplatr;         /* 属性 */
        ID      memid;          /* システム予約領域 */
} T_RMPL;
```

以下に，可変長メモリ・プール詳細情報 T_RMPL の詳細を示します。

- wtskid
  可変長メモリ・プールの待ちキューにタスクがキューイングされているか否かが格納されます。

  TSK_NONE：　　待ちキューにタスクはキューイングされていない
  その他：　　待ちキューの先頭にキューイングされているタスクの ID

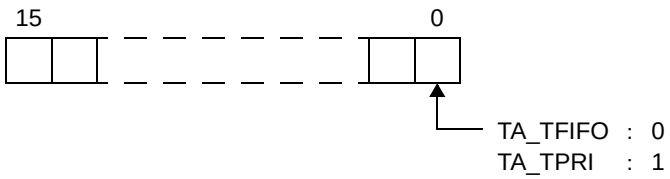
- fmplsz
  可変長メモリ・プールから獲得可能な空き可変長メモリ・ブロックの合計サイズ（単位：バイト）が格納されます。

- fblksz
  可変長メモリ・プールから獲得可能な空き可変長メモリ・ブロックの最大サイズ（単位：バイト）が格納されます。

- mplatr
  可変長メモリ・プールの属性（キューイング方式）が格納されます。

  タスク・キューイング方式（ビット 0）
  TA_TFIFO：　　可変長メモリ・ブロックの獲得要求を行った順
  TA_TPRI：　　タスクの優先度順

  【 mplatr の構造 】



- memid
  システム予約領域です。

16.2.12 システム時刻情報

以下に，set_tim，iset_tim，get_tim，および，iget_tim を発行する際に使用するシステム時刻情報 SYSTIM を示します。
なお，システム時刻情報 SYSTIM の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_systim {
        UW      ltime;          /* システム時刻（下位 32 ビット）*/
        UH      utime;          /* システム時刻（上位 16 ビット）*/
} SYSTIM;
```

以下に，システム時刻情報 SYSTIM の詳細を示します。

- ltime
  システム時刻（単位：ミリ秒）の下位 32 ビットが格納されます。

- utime
  システム時刻（単位：ミリ秒）の上位 16 ビットが格納されます。

16.2.13 周期ハンドラ詳細情報

以下に，ref_cyc，および，iref_cyc を発行する際に使用する周期ハンドラ詳細情報 T_RCYC を示します。
なお，周期ハンドラ詳細情報 T_RCYC の定義は，標準ヘッダ・ファイル <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥packet.h で行われています。

```
typedef struct  t_rcyc {
        STAT    cycstat;        /* 現在状態 */
        RELTIM  lefttim;        /* 残り時間 */
        ATR     cycatr;         /* 属性 */
        RELTIM  cyctim;         /* 起動周期 */
        RELTIM  cycphs;         /* 初期起動位相 */
} T_RCYC;
```

以下に，周期ハンドラ詳細情報 T_TCYC の詳細を示します。

- cycstat
  周期ハンドラの現在状態が格納されます。

  TCYC_STP：　　停止状態（STP 状態）
  TCYC_STA：　　動作状態（STA 状態）

- lefttim
  周期ハンドラが次に起動するまでの残り時間（単位：ミリ秒）が格納されます。

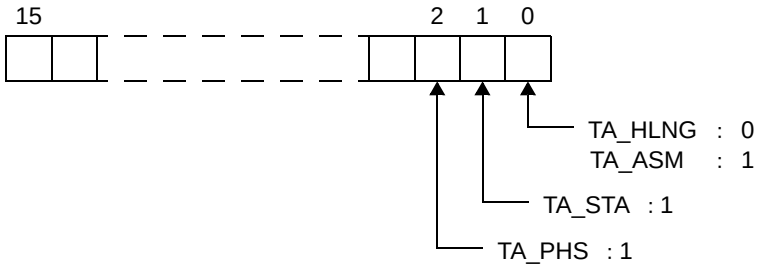
- cycatr
  周期ハンドラの属性（記述言語，初期起動状態など）が格納されます。

  周期ハンドラの記述言語（ビット 0）
  TA_HLNG：　　C 言語
  TA_ASM：　　アセンブリ言語

  周期ハンドラの初期起動状態（ビット 1）
  TA_STA：　　動作状態（STA 状態）

  起動位相保存の有無（ビット 2）
  TA_PHS：　　保存

  【 cycatr の構造 】



- cyctim
  周期ハンドラの起動周期（単位：ミリ秒）が格納されます。

- cycphs
  周期ハンドラの初期起動位相（単位：ミリ秒）が格納されます。
  なお，RX850V4 における初期起動位相は，周期ハンドラの生成処理が完了してから 1 回目の起動要求が発行されるまでの相対時間間隔を意味しています。

16.3　マ ク ロ

RX850V4 が提供するサービス・コールを発行する際に使用するマクロ（管理オブジェクトの現在状態，処理プログラムの属性など）について以下に示します。

16.3.1　管理オブジェクトの現在状態

以下に，サービス・コール（ref_tsk，ref_sem など）の発行により獲得される各種管理オブジェクトの現在状態一覧を示します。

なお，現在状態のマクロ定義は，ITRON 仕様共通マクロ定義ファイル <rx_root>¥inc850¥itron.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥option.h で行われています。

表 16 － 2　管理オブジェクトの現在状態

マクロ	値	意味
TTS_RUN	0x01	RUNNING 状態
TTS_RDY	0x02	READY 状態
TTS_WAI	0x04	WAITING 状態
TTS_SUS	0x08	SUSPENDED 状態
TTS_WAS	0x0c	WAITING-SUSPENDED 状態
TTS_DMT	0x10	DORMANT 状態
TTEX_ENA	0x00	タスク例外処理許可状態
TTEX_DIS	0x01	タスク例外処理禁止状態
TCYC_STP	0x00	停止状態（STP 状態）
TCYC_STA	0x01	動作状態（STA 状態）
TTW_SLP	0x0001	slp_tsk，または tslp_tsk による起床待ち状態
TTW_DLY	0x0002	dly_tsk による時間経過待ち状態
TTW_SEM	0x0004	wai_sem，または twai_sem による資源獲得待ち状態
TTW_FLG	0x0008	wai_flg，または twai_flg によるイベントフラグ待ち状態
TTW_SDTQ	0x0010	snd_dtq，または tsnd_dtq によるデータ送信待ち状態
TTW_RDTQ	0x0020	rcv_dtq，または trcv_dtq によるデータ受信待ち状態
TTW_MBX	0x0040	rcv_mbx，または trcv_mbx によるメッセージ受信待ち状態
TTW_MTX	0x0080	loc_mtx，または tloc_mtx によるミューテックス待ち状態
TTW_MPF	0x2000	get_mpf，または tget_mpf による固定長メモリ・ブロック獲得待ち状態
TTW_MPL	0x4000	get_mpl，または tget_mpl による可変長メモリ・ブロック獲得待ち状態
TSK_NONE	0	タスクはキューイングされていない

16.3.2　処理プログラムの属性

以下に，サービス・コール（ref_tsk，ref_cyc など）の発行により獲得される各種処理プログラムの属性一覧を示します。
なお，属性のマクロ定義は，ITRON 仕様共通マクロ定義ファイル <rx_root>¥inc850¥itron.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥option.h で行われています。

表 16 － 3　処理プログラムの属性

マクロ	値	意味
TA_HLNG	0x0000	C 言語
TA_ASM	0x0001	アセンブリ言語
TA_ACT	0x0002	READY 状態
TA_RSTR	0x0004	制約タスク
TA_DISPREEMPT	0x4000	DORMANT 状態から READY 状態へと遷移した際には，受け付け禁止状態
TA_ENAINT	0x0000	DORMANT 状態から READY 状態へと遷移した際には，受け付け禁止状態
TA_DISINT	0x8000	DORMANT 状態から READY 状態へと遷移した際には，受け付け禁止状態
TA_STA	0x0002	動作状態（STA 状態）
TA_PHS	0x0004	保存

16.3.3　管理オブジェクトの属性

以下に，サービス・コール（ref_sem，ref_flg など）の発行により獲得される各種管理オブジェクトの属性一覧を示します。
なお，属性のマクロ定義は，ITRON 仕様共通マクロ定義ファイル <rx_root>¥inc850¥itron.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥option.h で行われています。

表 16 － 4　管理オブジェクトの属性

マクロ	値	意味
TA_TFIFO	0x0000	要求を行った順
TA_TPRI	0x0001	タスクの優先度順
TA_WSGL	0x0000	1 個
TA_WMUL	0x0002	複数
TA_CLR	0x0004	ビット・パターンをクリア
TA_MFIFO	0x0000	要求を行った順
TA_MPRI	0x0002	メッセージの優先度順

16.3.4　サービス・コールの動作モード

以下に，サービス・コール（act_tsk，wup_tsk など）を発行する際に使用する各種サービス・コールの動作モード一覧を示します。

なお，動作モードのマクロ定義は，ITRON 仕様共通マクロ定義ファイル <rx_root>¥inc850¥itron.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥option.h で行われています。

表 16 － 5　サービス・コールの動作モード

マクロ	値	意味
TSK_SELF	0	自タスク
TPRI_INI	0	初期優先度
TMO_FEVR	-1	永久待ち
TMO_POL	0	ポーリング
TWF_ANDW	0x00	AND 待ち
TWF_ORW	0x01	OR 待ち
TPRI_SELF	0	自タスクの現在優先度

16.3.5　戻 り 値

以下に，サービス・コール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 <rx_root>¥inc850¥kernel.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥errcd.h，および，option.h で行われています。

表 16 － 6　戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	未サポートの機能である
E_RSFN	-10	コンフィギュレーション時に使用するサービス･コールとして定義されていない
E_RSATR	-11	属性の指定が不正である
E_PAR	-17	パラメータの指定が不正である
E_ID	-18	ID の指定が不正である
E_CTX	-25	サービス・コールの発行状態が不正（コンテキスト・エラー）である
E_ILUSE	-28	サービス・コールの使用方法が不正である
E_NOMEM	-33	メモリ領域が確保できない
E_OBJ	-41	管理オブジェクトの状態が不正である
E_NOEXS	-42	管理オブジェクトが生成されていない
E_QOVR	-43	カウンタがオーバフローした
E_RLWAI	-49	rel_wai，または irel_wai の発行により，WAITING 状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した
FALSE	0	偽
TRUE	1	真

16.3.6　構成定数

以下に，構成定数の一覧を示します。

なお，構成定数のマクロ定義は，ITRON 仕様共通マクロ定義ファイル <rx_root>¥inc850¥itron.h から呼び出されるヘッダ・ファイル <rx_root>¥inc850¥rx850v4¥component.h で行われています。ただし，一部数値が可変なマクロ定義に関しては，システム・コンフィギュレーション・ファイルの設定内容にしたがい，システム情報ヘッダ・ファイルで行なわれています。

表 16 － 7　優先度範囲

マクロ	数値	意味
TMIN_TPRI	1	タスク優先度の最小値
TMAX_TPRI	可変	タスク優先度の最大値
TMIN_MPRI	1	メッセージ優先度の最小値
TMAX_MPRI	0x7ffff(32767)	メッセージ優先度の最大値

表 16 － 8　バージョン情報

マクロ	数値	意味
TKERNEL_MAKER	0x0117	カーネルのメーカー・コード
TKERNEL_PRID	0x2230	カーネルの識別番号
TKERNEL_SPVER	0x5401	ITRON 仕様のバージョン番号
TKERNEL_PRVER	0x0430	カーネルのバージョン番号

表 16 － 9　キューイング数の最大値

マクロ	数値	意味
TMAX_ACTCNT	127	タスク起動要求キューイング数の最大値
TMAX_WUPCNT	127	タスク起床要求キューイング数の最大値
TMAX_SUSCNT	127	サスペンド要求キューイング数の最大値

表 16 － 10　ビットパターンのビット数

マクロ	数値	意味
TBIT_TEXPTN	32	タスク例外要因のビット数
TBIT_FLGPTN	32	イベントフラグのビット数

表 16 － 11　基本クロック周期

マクロ	数値	意味
TIC_NUME	可変	基本クロック周期（単位：ミリ秒）の分子
TIC_DENO	1	基本クロック周期（単位：ミリ秒）の分母

16.4　条件コンパイル用マクロ

RX850V4 のヘッダ・ファイルは以下のマクロにより条件コンパイルされます。
使用する環境にあわせてマクロを定義（コンパイラの -D 起動オプションなど）してください。

表 16 － 12　条件コンパイル用マクロ

分類	マクロ	内容
C コンパイラ・パッケージ	__nec__	CA850/CX を使用
ターゲット・デバイス	__v850__	V850 コア
	__v850e__	V850ES/V850E1/V850E2 コア
	__v850e2m__	V850E2M コア
レジスタ・モード種別	__r22__	22 レジスタ・モード
	__r26__	26 レジスタ・モード
	__r32__	32 レジスタ・モード

第 17 章　サービス・コール

本章では，RX850V4 が提供しているサービス・コールについて解説しています。

17.1　概　　要

RX850V4 が提供しているサービス・コールは，ユーザが記述した処理プログラムから RX850V4 が管理している資源（タスク，セマフォなど）を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，RX850V4 が提供しているサービス・コールを管理モジュール別に示します。

- タスク管理機能

act_tsk	iact_tsk	can_act	ican_act
sta_tsk	ista_tsk	ext_tsk	ter_tsk
chg_pri	ichg_pri	get_pri	iget_pri
ref_tsk	iref_tsk	ref_tst	iref_tst

- タスク付属同期機能

slp_tsk	tslp_tsk	wup_tsk	iwup_tsk
can_wup	ican_wup	rel_wai	irel_wai
sus_tsk	isus_tsk	rsm_tsk	irsm_tsk
frsm_tsk	ifrsm_tsk	dly_tsk	

- タスク例外処理機能

ras_tex	iras_tex	dis_tex	ena_tex
sns_tex	ref_tex	iref_tex	

- 同期通信機能（セマフォ）

wai_sem	pol_sem	ipol_sem	twai_sem
sig_sem	isig_sem	ref_sem	iref_sem

- 同期通信機能（イベントフラグ）

set_flg	iset_flg	clr_flg	iclr_flg
wai_flg	pol_flg	ipol_flg	twai_flg
ref_flg	iref_flg		

- 同期通信機能（データ・キュー）

snd_dtq	psnd_dtq	ipsnd_dtq	tsnd_dtq
fsnd_dtq	ifsnd_dtq	rcv_dtq	prcv_dtq
iprcv_dtq	trcv_dtq	ref_dtq	iref_dtq

- 同期通信機能（メールボックス）

snd_mbx	isnd_mbx	rcv_mbx	prcv_mbx
iprcv_mbx	trcv_mbx	ref_mbx	iref_mbx

- 拡張同期通信機能（ミューテックス）

loc_mtx	ploc_mtx	tloc_mtx	unl_mtx
ref_mtx	iref_mtx		

- メモリ・プール管理機能（固定長メモリ・プール）

get_mpf	pget_mpf	ipget_mpf	tget_mpf
rel_mpf	irel_mpf	ref_mpf	iref_mpf

- メモリ・プール管理機能（可変長メモリ・プール）

get_mpl	pget_mpl	ipget_mpl	tget_mpl
rel_mpl	irel_mpl	ref_mpl	iref_mpl

- 時間管理機能

set_tim	iset_tim	get_tim	iget_tim
sta_cyc	ista_cyc	stp_cyc	istp_cyc
ref_cyc	iref_cyc		

- システム状態管理機能

rot_rdq	irot_rdq	vsta_sch	get_tid
iget_tid	loc_cpu	iloc_cpu	unl_cpu
iunl_cpu	sns_loc	dis_dsp	ena_dsp
sns_dsp	sns_ctx	sns_dpn	

- 割り込み管理機能

dis_int	ena_int	chg_ims	ichg_ims
get_ims	iget_ims		

- サービス・コール管理機能

cal_svc	ical_svc

17.1.1　サービス・コールの呼び出し

サービス・コールを 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
  サービス・コールを 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，サービス・コールのパラメータは RX850V4 に引き数として渡され，該当処理が実行されます。

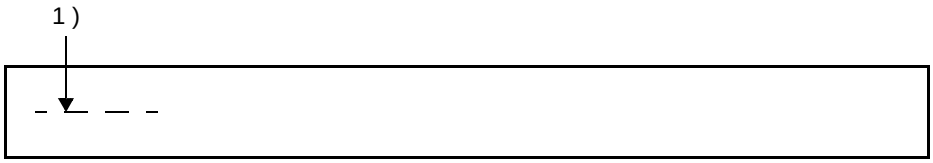
- アセンブリ言語
  サービス・コールをアセンブリ言語で記述された処理プログラムから発行する場合，ユーザが開発環境として使用する C コンパイラ・パッケージの関数呼び出し規約にしたがったパラメータ，および，戻り番地の設定を行ったのち，jarl 命令による呼び出しを行うことにより，サービス・コールのパラメータは RX850V4 に引き数として渡され，該当処理が実行されます。

備考　RX850V4 が提供するサービス・コール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。

kernel.h：　標準ヘッダ・ファイル

17.2　サービス・コール解説

次項から RX850V4 が提供しているサービス・コールについて，以下の記述フォーマットにしたがって解説します。



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**

マクロ	数値	意味

1） 名称
サービス・コールの名称を示しています。

2） 概要
サービス・コールの機能概要を示しています。

3） C 言語形式
サービス・コールを 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。

4） パラメータ
サービス・コール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： RX850V4 への入力パラメータ
O： RX850V4 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
サービス・コールの機能詳細を示しています。

6） 戻り値
サービス・コールからの戻り値を以下の形式で示しています。

マクロ	数値	意味
A	B	C

A） 戻り値のマクロ

B） 戻り値の数値

C） 戻り値の意味

17.2.1　タスク管理機能

以下に，RX850V4 がタスク管理機能として提供しているサービス・コールの一覧を示します。

表 17 － 1　タスク管理機能

サービス・コール名	機能概要	発行有効範囲
act_tsk	タスクの起動（起動要求をキューイングする）	タスク，制約タスク， 非タスク，初期化ルーチン
iact_tsk	タスクの起動（起動要求をキューイングする）	タスク，制約タスク， 非タスク，初期化ルーチン
can_act	起動要求のキューイング解除	タスク，制約タスク， 非タスク，初期化ルーチン
ican_act	起動要求のキューイング解除	タスク，制約タスク， 非タスク，初期化ルーチン
sta_tsk	タスクの起動（起動要求をキューイングしない）	タスク，制約タスク， 非タスク，初期化ルーチン
ista_tsk	タスクの起動（起動要求をキューイングしない）	タスク，制約タスク， 非タスク，初期化ルーチン
ext_tsk	タスクの終了	タスク，制約タスク
ter_tsk	タスクの強制終了	タスク，制約タスク， 初期化ルーチン
chg_pri	タスク優先度の変更	タスク，制約タスク， 非タスク，初期化ルーチン
ichg_pri	タスク優先度の変更	タスク，制約タスク， 非タスク，初期化ルーチン
get_pri	タスク優先度の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iget_pri	タスク優先度の参照	タスク，制約タスク， 非タスク，初期化ルーチン
ref_tsk	タスク詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iref_tsk	タスク詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
ref_tst	タスク基本情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iref_tst	タスク基本情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン

act_tsk
iact_tsk

概要

タスクの起動（起動要求をキューイングする）

C 言語形式

```
ER      act_tsk ( ID tskid );
ER      iact_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID TSK_SELF：　自タスク 数値：　タスクの ID

機能

tskid で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち，初期優先度のレディ・キューの最後尾にキューイングします。これにより，対象タスクは，RX850V4 のスケジューリング対象となります。

ただし，本サービス・コールを発行した際，対象タスクが DORMANT 状態以外の場合には，対象タスクのキューイング処理，および，状態操作処理は行わず，対象タスクに起動要求をキューイング（起動要求カウンタに 0x1 を加算）しています。

備考 1　RX850V4 が管理する起動要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールでは，起動要求数が 127 回を越えるような場合には，起動要求の発行（起動要求カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

備考 2　本サービス・コールの発行により起動されたタスクには，拡張情報として“タスク情報で指定した拡張情報”が渡されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス・コールを発行した際，*tskid* に TSK_SELF を指定した
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象タスクが生成されていない
E_QOVR	-43	起動要求数が 127 回を越えた

can_act
ican_act

概要

起動要求のキューイング解除

C 言語形式

```
ER_UINT can_act ( ID tskid );
ER_UINT ican_act ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID TSK_SELF：　自タスク 数値：　タスクの ID

機能

tskid で指定されたタスクにキューイングされている起動要求をすべて解除（起動要求カウンタに 0x0 を設定）します。
なお，正常終了時は戻り値として本サービス・コールの発行により解除した起動要求数を返します。

備考　本サービス・コールでは，状態操作処理は行わず，起動要求カウンタの設定処理のみを行います。
したがって，READY 状態などから DORMANT 状態に遷移することはありません。

戻り値

マクロ	数値	意味
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス・コールを発行した際，*tskid* に TSK_SELF を指定した
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象タスクが生成されていない
－	0	正常終了 - 起動要求数が 0 である - 対象タスクが DORMANT 状態である
正の値	－	正常終了（解除した起動要求数）

sta_tsk
ista_tsk

概要

タスクの起動（起動要求をキューイングしない）

C 言語形式

```
ER      sta_tsk ( ID tskid, VP_INT stacd );
ER      ista_tsk ( ID tskid, VP_INT stacd );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID
I	VP_INT　*stacd*;	タスクの拡張情報

機能

tskid で指定されたタスクを DORMANT 状態から READY 状態へと遷移させたのち，初期優先度のレディ・キューの最後尾にキューイングします。これにより，対象タスクは，RX850V4 のスケジューリング対象となります。

ただし，本サービス・コールでは，起動要求のキューイングが行われません。このため，対象タスクが DORMANT 状態以外の場合には，対象タスクの状態操作処理は行わず，戻り値として E_OBJ を返します。

なお，*stacd* には，対象タスクに引き渡す拡張情報を指定します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *tskid* ≦ 0x0 - *tskid* ＞生成されているタスクの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	対象タスクが DORMANT 状態でない
E_NOEXS	-42	対象タスクが生成されていない

ext_tsk

概要

タスクの終了

C 言語形式

```
void    ext_tsk ( void );
```

パラメータ

なし

機能

自タスクを RUNNING 状態から DORMANT 状態へと遷移させ，レディ・キューから外します。これにより，自タスクは，RX850V4 のスケジューリング対象から除外されます。

ただし，本サービス・コールを発行した際，自タスクの起動要求がキューイングされていた（起動要求カウンタが 0x0 以外の値であった）場合には，自タスクの状態操作（DORMANT 状態への状態遷移処理）を行ったのち，自タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。

備考 1　本サービス・コールでは，自タスクの状態操作（DORMANT 状態への状態遷移処理）を行う際に，

- 優先度（現在優先度）
- 起床要求数
- サスペンド要求数
- 割り込み状態

といった情報をタスク生成時に設定される値で初期化しています。
また，自タスクがミューテックスをロックしていた場合には，ロック状態の解除（unl_mtx と同等の処理）もあわせて行われます。

備考 2　タスク内で return 命令が発行された場合，本サービス・コールと同等の処理が実行されます。

戻り値

なし

ter_tsk

概要

タスクの強制終了

C 言語形式

```
ER      ter_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID

機能

tskid で指定されたタスクを強制的に DORMANT 状態へと遷移させます。これにより，対象タスクは，RX850V4 のスケジューリング対象から除外されます。

ただし，本サービス・コールを発行した際，対象タスクの起動要求がキューイングされていた（起動要求カウンタが 0x0 以外の値であった）場合には，対象タスクの状態操作（DORMANT 状態への状態遷移処理）を行ったのち，対象タスクの起動（DORMANT 状態から READY 状態への状態遷移処理）もあわせて行われます。

備考　本サービス・コールでは，対象タスクの状態操作（DORMANT 状態への状態遷移処理）を行う際に，

- 優先度（現在優先度）
- 起床要求数
- サスペンド要求数
- 割り込み状態

といった情報をタスク生成時に設定される値で初期化しています。
また，対象タスクがミューテックスをロックしていた場合には，ロック状態の解除（unl_mtx と同等の処理）もあわせて行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	対象タスクが制約タスクである
E_ID	-18	ID の指定が不正である - *tskid* ≦ 0x0 - *tskid* ＞生成されているタスクの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した
E_ILUSE	-28	対象タスクが自タスクである
E_OBJ	-41	対象タスクが DORMANT 状態である

マクロ	数値	意味
E_NOEXS	-42	対象タスクが生成されていない

chg_pri
ichg_pri

概要

タスク優先度の変更

C 言語形式

```
ER      chg_pri ( ID tskid, PRI tskpri );
ER      ichg_pri ( ID tskid, PRI tskpri );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID TSK_SELF：　自タスク 数値：　タスクの ID
I	PRI　*tskpri*;	タスクの優先度 TPRI_INI：　タスクの初期優先度 数値：　タスクの優先度

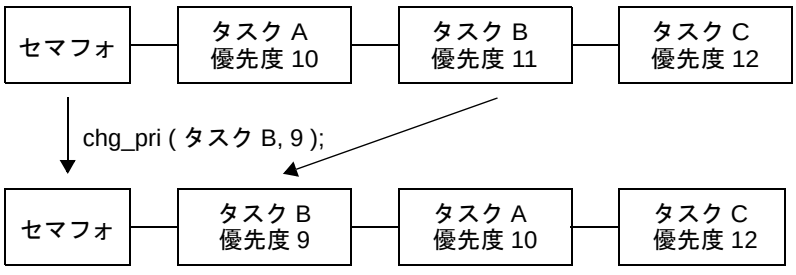
機能

tskid で指定されたタスクの優先度（現在優先度）を *tskpri* で指定された値に変更します。

対象タスクが RUNNING 状態，または READY 状態であった場合には，優先度を変更したのち，対象タスクを *tskpri* で指定された優先度に応じたレディ・キューの最後尾につなぎかえます。

備考　対象タスクが何らかの待ちキューに優先度順でキューイングされていた場合，本サービス・コールの発行により，待ち順序が変わることがあります。

例　セマフォの待ちキューに 3 つのタスク（タスク A：優先度 10，タスク B：優先度 11，タスク C：優先度 12）が優先度順でキューイングされているとき，タスク B の優先度を 11 から 9 に変更した場合，待ちキューの待ち順序は，以下のように変更されます。



戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	対象タスクが制約タスクである

マクロ	数値	意味
E_PAR	-17	優先度の指定が不正である - *tskpri* ＜ 0x0 - *tskpri* ＞最大タスク優先度
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス・コールを発行した際, *tskid* に TSK_SELF を指定した
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	対象タスクが DORMANT 状態である
E_NOEXS	-42	対象タスクが生成されていない

get_pri
iget_pri

概要

タスク優先度の参照

C 言語形式

```
ER      get_pri ( ID tskid, PRI *p_tskpri );
ER      iget_pri ( ID tskid, PRI *p_tskpri );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID TSK_SELF：　自タスク 数値：　タスクの ID
O	PRI　*p_tskpri;	現在優先度を格納する領域へのポインタ

機能

tskid で指定されたタスクの現在優先度を *p_tskpri* で指定された領域に格納します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス・コールを発行した際, *tskid* に TSK_SELF を指定した
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	対象タスクが DORMANT 状態である
E_NOEXS	-42	対象タスクが生成されていない

ref_tsk
iref_tsk

概要

タスク詳細情報の参照

C 言語形式

```
ER      ref_tsk ( ID tskid, T_RTSK *pk_rtsk );
ER      iref_tsk ( ID tskid, T_RTSK *pk_rtsk );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID TSK_SELF：　自タスク 数値：　タスクの ID
O	T_RTSK　**pk_rtsk*;	タスク詳細情報を格納する領域へのポインタ

【 タスク詳細情報 T_RTSK の構造 】

```
typedef struct  t_rtsk {
        STAT    tskstat;        /* 現在状態 */
        PRI     tskpri;         /* 現在優先度 */
        PRI     tskbpri;        /* システム予約領域 */
        STAT    tskwait;        /* 待ち要因 */
        ID      wobjid;         /* 管理オブジェクトの ID*/
        TMO     lefttmo;        /* 残り時間 */
        UINT    actcnt;         /* 起動要求数 */
        UINT    wupcnt;         /* 起床要求数 */
        UINT    suscnt;         /* サスペンド要求数 */
        ATR     tskatr;         /* 属性 */
        PRI     itskpri;        /* 初期優先度 */
        ID      memid;          /* システム予約領域 */
} T_RTSK;
```

機能

tskid で指定されたタスクのタスク詳細情報（現在状態，現在優先度など）を *pk_rtsk* で指定された領域に格納します。

備考　タスク詳細情報 T_RTSK についての詳細は，「16.2.1　タスク詳細情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス・コールを発行した際, *tskid* に TSK_SELF を指定した
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象タスクが生成されていない

ref_tst
iref_tst

概要

タスク基本情報の参照

C 言語形式

```
ER      ref_tst ( ID tskid, T_RTST *pk_rtst );
ER      iref_tst ( ID tskid, T_RTST *pk_rtst );
```

パラメータ

I/O	パラメータ	説明
I	ID　tskid;	タスクの ID TSK_SELF：　自タスク 数値：　タスクの ID
O	T_RTST　*pk_rtst;	タスク基本情報を格納する領域へのポインタ

【 タスク基本情報 T_RTST の構造 】

```
typedef struct  t_rtst {
        STAT    tskstat;        /* 現在状態 */
        STAT    tskwait;        /* 待ち要因 */
} T_RTST;
```

機能

tskid で指定されたタスクのタスク基本情報（現在状態，待ち要因）を *pk_rtst* で指定された領域に格納します。
タスク情報のうち，現在状態，待ち要因のみを参照したい場合に使用します。
取得する情報が少ないので ref_tsk，iref_tsk より高速に応答します。

備考　　タスク基本情報 T_RTST についての詳細は，「16.2.2　タスク基本情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス・コールを発行した際，*tskid* に TSK_SELF を指定した
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象タスクが生成されていない

17.2.2　タスク付属同期機能

以下に，RX850V4 がタスク付属同期機能として提供しているサービス・コールの一覧を示します。

表 17 － 2　タスク付属同期機能

サービス・コール名	機能概要	発行有効範囲
slp_tsk	起床待ち状態への移行（永久待ち）	タスク
tslp_tsk	起床待ち状態への移行（タイムアウト付き）	タスク
wup_tsk	タスクの起床	タスク，制約タスク， 非タスク，初期化ルーチン
iwup_tsk	タスクの起床	タスク，制約タスク， 非タスク，初期化ルーチン
can_wup	起床要求の解除	タスク，制約タスク， 非タスク，初期化ルーチン
ican_wup	起床要求の解除	タスク，制約タスク， 非タスク，初期化ルーチン
rel_wai	WAITING 状態の強制解除	タスク，制約タスク， 非タスク，初期化ルーチン
irel_wai	WAITING 状態の強制解除	タスク，制約タスク， 非タスク，初期化ルーチン
sus_tsk	SUSPENDED 状態への移行	タスク，制約タスク， 非タスク，初期化ルーチン
isus_tsk	SUSPENDED 状態への移行	タスク，制約タスク， 非タスク，初期化ルーチン
rsm_tsk	SUSPENDED 状態の解除	タスク，制約タスク， 非タスク，初期化ルーチン
irsm_tsk	SUSPENDED 状態の解除	タスク，制約タスク， 非タスク，初期化ルーチン
frsm_tsk	SUSPENDED 状態の強制解除	タスク，制約タスク， 非タスク，初期化ルーチン
ifrsm_tsk	SUSPENDED 状態の強制解除	タスク，制約タスク， 非タスク，初期化ルーチン
dly_tsk	時間経過待ち状態への移行	タスク

slp_tsk

概要

起床待ち状態への移行（永久待ち）

C 言語形式

```
ER      slp_tsk ( void );
```

パラメータ

なし

機能

自タスクを RUNNING 状態から WAITING 状態（起床待ち状態）へと遷移させます。

ただし，本サービス・コールを発行した際，自タスクの起床要求がキューイングされていた（起床要求カウンタが 0x0 以外）場合には，状態操作処理は行わず，起床要求カウンタから 0x1 を減算します。

なお，起床待ち状態の解除は，以下の場合に行われ，起床待ち状態から READY 状態へと遷移します。

起床待ち状態の解除操作	エラー・コード
wup_tsk の発行により，起床要求が発行された	E_OK
iwup_tsk の発行により，起床要求が発行された	E_OK
rel_wai の発行により，起床待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，起床待ち状態を強制的に解除された	E_RLWAI

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_RLWAI	-49	rel_wai，または irel_wai の発行により，起床待ち状態を強制的に解除された

tslp_tsk

概要

起床待ち状態への移行（タイムアウト付き）

C 言語形式

```
ER      tslp_tsk ( TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	TMO　*tmout*;	待ち時間（単位ミリ秒） TMO_FEVR：　永久待ち TMO_POL：　ポーリング 数値：　待ち時間

機能

自タスクを RUNNING 状態からタイムアウト付きの WAITING 状態（起床待ち状態）へと遷移させます。
ただし，本サービス・コールを発行した際，自タスクの起床要求がキューイングされていた（起床要求カウンタが 0x0 以外）場合には，状態操作処理は行わず，起床要求カウンタから 0x1 を減算します。
なお，起床待ち状態の解除は，以下の場合に行われ，起床待ち状態から READY 状態へと遷移します。

起床待ち状態の解除操作	エラー・コード
wup_tsk の発行により，起床要求が発行された	E_OK
iwup_tsk の発行により，起床要求が発行された	E_OK
rel_wai の発行により，起床待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，起床待ち状態を強制的に解除された	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

備考　　待ち時間 *tmout* に TMO_FEVR が指定された際には“slp_tsk と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_PAR	-17	待ち時間の指定が不正（*tmout* ＜ TMO_FEVR）である
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した

マクロ	数値	意味
E_RLWAI	-49	rel_wai，または irel_wai の発行により，起床待ち状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した

wup_tsk
iwup_tsk

概要

タスクの起床

C 言語形式

```
ER      wup_tsk ( ID tskid );
ER      iwup_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID TSK_SELF：　自タスク 数値：　タスクの ID

機能

tskid で指定されたタスクを WAITING 状態（起床待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移させます。

ただし，本サービス・コールを発行した際，対象タスクが起床待ち状態以外の場合には，状態操作処理は行わず，起床要求カウンタに 0x1 を加算します。

備考　RX850V4 が管理する起床要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールでは，起床要求数が 127 回を越えるような場合には，起床要求のキューイング（起床要求カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	対象タスクが制約タスクである
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス･コールを発行した際, *tskid* に TSK_SELF を指定した
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	対象タスクが DORMANT 状態である
E_NOEXS	-42	対象タスクが生成されていない
E_QOVR	-43	起床要求数が 127 回を越えた

can_wup
ican_wup

概要

起床要求の解除

C 言語形式

```
ER_UINT can_wup ( ID tskid );
ER_UINT ican_wup ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID　　*tskid*;	タスクの ID TSK_SELF :　自タスク 数値 :　　タスクの ID

機能

tskid で指定されたタスクにキューイングされている起床要求をすべて解除（起床要求カウンタに 0x0 を設定）します。
なお，本サービス・コールは戻り値として解除した起床要求数を返します。

戻り値

マクロ	数値	意味
E_NOSPT	-9	対象タスクが制約タスクである
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス･コールを発行した際, *tskid* に TSK_SELF を指定した
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	対象タスクが DORMANT 状態である
E_NOEXS	-42	対象タスクが生成されていない
その他	―	正常終了（解除した起床要求数）

rel_wai
irel_wai

概要

WAITING 状態の強制解除

C 言語形式

```
ER      rel_wai ( ID tskid );
ER      irel_wai ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID

機能

tskid で指定されたタスクの WAITING 状態を強制的に解除します。これにより，対象タスクは待ちキューから外れ，WAITING 状態から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

なお，本サービス・コールの発行により WAITING 状態を解除されたタスクには，WAITING 状態へと遷移するきっかけとなったサービス・コール（slp_tsk，wai_sem など）の戻り値として E_RLWAI を返します。

備考 1　本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが WAITING 状態，または WAITING-SUSPENDED 状態以外の場合には，戻り値として E_OBJ を返します。

備考 2　本サービス・コールでは，SUSPENDED 状態の解除は行われません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	対象タスクが制約タスクである
E_ID	-18	ID の指定が不正である - *tskid* ≦ 0x0 - *tskid* ＞生成されているタスクの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	対象タスクが WAITING 状態，または WAITING-SUSPENDED 状態でない
E_NOEXS	-42	対象タスクが生成されていない

sus_tsk
isus_tsk

概要

SUSPENDED 状態への移行

C 言語形式

```
ER      sus_tsk ( ID tskid );
ER      isus_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID TSK_SELF：　自タスク 数値：　タスクの ID

機能

tskid で指定されたタスクを RUNNING 状態から SUSPENDED 状態へ，READY 状態から SUSPENDED 状態へ，または WAITING 状態から WAITING-SUSPENDED 状態へと遷移させます。

ただし，本サービス・コールを発行した際，対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態へと遷移していた場合には，状態操作処理は行わず，サスペンド要求のキューイング（カウンタの加算処理）のみを行います。

備考　RX850V4 が管理するサスペンド要求カウンタは，7 ビット幅で構成されています。このため，本サービス・コールでは，サスペンド要求数が 127 回を越えるような場合には，サスペンド要求の発行（サスペンド要求カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	対象タスクが制約タスクである
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス・コールを発行した際，*tskid* に TSK_SELF を指定した
E_CTX	-25	コンテキスト・エラー - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した際，*tskid* に自タスクを指定した
E_OBJ	-41	対象タスクが DORMANT 状態である
E_NOEXS	-42	対象タスクが生成されていない

マクロ	数値	意味
E_QOVR	-43	サスペンド要求数が 127 回を越えた

rsm_tsk
irsm_tsk

概要

SUSPENDED 状態の解除

C 言語形式

```
ER      rsm_tsk ( ID tskid );
ER      irsm_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID

機能

tskid で指定されたタスクを SUSPENDED 状態から READY 状態へ，または WAITING-SUSPENDED 状態から WAITING 状態へと遷移させます。

ただし，本サービス・コールを発行した際，サスペンド要求がキューイングされていた場合には，状態操作処理は行わず，サスペンド要求カウンタの減算処理のみを行います。

備考　本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態以外の場合には，戻り値として E_OBJ を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	対象タスクが制約タスクである
E_ID	-18	ID の指定が不正である - *tskid* ≦ 0x0 - *tskid* ＞生成されているタスクの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態でない
E_NOEXS	-42	対象タスクが生成されていない

frsm_tsk
ifrsm_tsk

概要

SUSPENDED 状態の強制解除

C 言語形式

```
ER      frsm_tsk ( ID tskid );
ER      ifrsm_tsk ( ID tskid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID

機能

tskid で指定されたタスクに発行されているサスペンド要求をすべて解除（サスペンド要求カウンタに 0x0 を設定）します。これにより，対象タスクは SUSPENDED 状態から READY 状態へ，または WAITING-SUSPENDED 状態から WAITING 状態へと遷移します。

備考　本サービス・コールでは，解除要求のキューイングが行われません。このため，対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態以外の場合には，戻り値として E_OBJ を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	対象タスクが制約タスクである
E_ID	-18	ID の指定が不正である - *tskid* ≦ 0x0 - *tskid* ＞生成されているタスクの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	対象タスクが SUSPENDED 状態，または WAITING-SUSPENDED 状態でない
E_NOEXS	-42	対象タスクが生成されていない

dly_tsk

概要

時間経過待ち状態への移行

C 言語形式

```
ER      dly_tsk ( RELTIM dlytim );
```

パラメータ

I/O	パラメータ	説明
I	RELTIM　*dlytim*;	遅延時間（単位：ミリ秒）

機能

自タスクを *dlytim* で指定された遅延時間が経過するまでの間，RUNNING 状態から WAITING 状態（時間経過待ち状態）へと遷移させます。
なお，時間経過待ち状態の解除は，以下の場合に行われ，時間経過待ち状態から READY 状態へと遷移します。

時間経過待ち状態の解除操作	エラー・コード
dlytim で指定された遅延時間が経過した	E_OK
rel_wai の発行により，時間経過待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，時間経過待ち状態を強制的に解除された	E_RLWAI

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_RLWAI	-49	rel_wai，または irel_wai の発行により，起床待ち状態を強制的に解除された

17.2.3　タスク例外処理機能

以下に，RX850V4 がタスク例外処理機能として提供しているサービス・コールの一覧を示します。

表 17 － 3　タスク例外処理機能

サービス・コール名	機能概要	発行有効範囲
ras_tex	タスク例外処理ルーチンの起動	タスク，制約タスク， 非タスク，初期化ルーチン
iras_tex	タスク例外処理ルーチンの起動	タスク，制約タスク， 非タスク，初期化ルーチン
dis_tex	タスク例外処理ルーチンの起動禁止	タスク
ena_tex	タスク例外処理ルーチンの起動許可	タスク
sns_tex	タスク例外処理禁止状態の参照	タスク，制約タスク， 非タスク，初期化ルーチン
ref_tex	タスク例外処理ルーチン詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iref_tex	タスク例外処理ルーチン詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン

ras_tex
iras_tex

概要

タスク例外処理ルーチンの起動

C 言語形式

```
ER      ras_tex ( ID tskid, TEXPTN rasptn );
ER      iras_tex ( ID tskid, TEXPTN rasptn );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID TSK_SELF：　自タスク 数値：　タスクの ID
I	TEXPTN　*rasptn*;	タスク例外処理ルーチンのタスク例外要因

機能

tskid で指定されたタスクにタスク例外処理要求を発行します。これにより，対象タスクが RUNNING 状態へと遷移した際には，対象タスクに登録されているタスク例外処理ルーチンが起動します。

なお，*rasptn* には，対象タスク例外処理ルーチンに引き渡すタスク例外要因を指定します。対象タスク例外処理ルーチンは，タスク例外要因を関数パラメータと同様に取り扱うことで操作可能となります。

備考　本サービス・コールでは，タスク例外処理要求のキューイングが行われません。このため，タスク例外処理ルーチンの起動以前（タスク例外処理要求の発行後，対象タスクが RUNNING 状態へと遷移するまでの間）に複数回のタスク例外処理要求が発行された場合には，2 回目以降の本サービス・コールの発行時には，タスク例外処理要求の発行は行わず，タスク例外要因の保留（タスク例外要因の論理和 OR）のみを行います。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	対象タスクが制約タスクである
E_PAR	-17	タスク例外要因の指定が不正（*rasptn* = 0x0）である
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス・コールを発行した際，*tskid* に TSK_SELF を指定した
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した

マクロ	数値	意味
E_OBJ	-41	管理オブジェクトの状態が不正である - 対象タスクが DORMANT 状態である - 対象タスクにタスク例外処理ルーチンが登録されていない
E_NOEXS	-42	対象タスクが生成されていない

dis_tex

概要

タスク例外処理ルーチンの起動禁止

C 言語形式

```
ER      dis_tex ( void );
```

パラメータ

なし

機能

自タスクに登録されているタスク例外処理ルーチンの禁止状態をタスク例外処理許可状態からタスク例外処理禁止状態へと遷移させます。これにより，本サービス・コールの発行から ena_tex が発行されるまでの間，対象タスク例外処理ルーチンは RX850V4 の起動対象から除外されます。

なお，RX850V4 では，本サービス・コールの発行から ena_tex が発行されるまでの間にタスク例外処理要求（ras_tex，または iras_tex）が発行された場合には，タスク例外処理要求の受け付けなどといった処理を行うだけであり，実際の起動処理は対象タスク例外処理ルーチンがタスク例外処理許可状態へと遷移するまで遅延されます。

備考 1　本サービス・コールでは，禁止要求のキューイングが行われません。このため，対象タスク例外処理ルーチンがタスク例外処理禁止状態に遷移していた場合には，何も処理は行わず，エラーとしても扱いません。

備考 2　RX850V4 では，タスクの起動処理として，本サービス・コールと同等の処理もあわせて実行しています。したがって，タスクが DORMANT 状態から READY 状態へと遷移した際には，タスク例外処理禁止状態となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	自タスクにタスク例外処理ルーチンが登録されていない

ena_tex

概要

タスク例外処理ルーチンの起動許可

C 言語形式

```
ER      ena_tex ( void );
```

パラメータ

なし

機能

自タスクに登録されているタスク例外処理ルーチンの禁止状態をタスク例外処理禁止状態からタスク例外処理許可状態へと遷移させます。これにより，対象タスク例外処理ルーチンは RX850V4 の起動対象となります。

なお，RX850V4 では，dis_tex の発行から本サービス・コールが発行されるまでの間にタスク例外処理要求（ras_tex，または iras_tex）が発行された場合には，タスク例外処理要求の受け付けなどといった処理を行うだけであり，実際の起動処理は対象タスク例外処理ルーチンがタスク例外処理許可状態へと遷移するまで遅延されます。

備考　本サービス・コールでは，起動要求のキューイングが行われません。このため，対象タスク例外処理ルーチンがタスク例外処理許可状態に遷移していた場合には，何も処理は行わず，エラーとしても扱いません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	自タスクにタスク例外処理ルーチンが登録されていない

sns_tex

概要

タスク例外処理禁止状態の参照

C 言語形式

```
BOOL    sns_tex ( void );
```

パラメータ

なし

機能

自タスク，または割り込み／例外が発生したときに RUNNING 状態であったタスクに登録されているタスク例外処理ルーチンの状態（タスク例外処理禁止状態，タスク例外処理許可状態）を獲得します。
戻り値としてタスク例外処理ルーチンの状態を返します。

戻り値

マクロ	数値	意味
TRUE	1	正常終了 - タスク例外処理禁止状態 - RUNNING 状態のタスクが存在しない - RUNNING 状態のタスクにタスク例外処理ルーチンが登録されていない
FALSE	0	正常終了 - タスク例外処理許可状態
E_NOSPT	-9	制約タスクから本サービス・コールを発行した

ref_tex
iref_tex

概要

タスク例外処理ルーチン詳細情報の参照

C 言語形式

```
ER      ref_tex ( ID tskid, T_RTEX *pk_rtex );
ER      iref_tex ( ID tskid, T_RTEX *pk_rtex );
```

パラメータ

I/O	パラメータ	説明
I	ID　*tskid*;	タスクの ID TSK_SELF：　自タスク 数値：　タスクの ID
O	T_RTEX　**pk_rtex*;	タスク例外処理ルーチン詳細情報を格納する領域へのポインタ

【 タスク例外処理ルーチン詳細情報 T_RTEX の構造 】

```
typedef struct  t_rtex {
        STAT    texstat;        /* 現在状態 */
        TEXPTN  pndptn;         /* 保留例外要因 */
        ATR     texatr;         /* 属性 */
} T_RTEX;
```

機能

tskid で指定されたタスクに登録されているタスク例外処理ルーチンのタスク例外処理ルーチン詳細情報（現在状態，保留例外要因など）を *pk_rtex* で指定された領域に格納します。

指定されたタスクにタスク例外処理ルーチンが登録されていなかった場合は，戻り値として E_OBJ を返します。

備考　タスク例外処理ルーチン詳細情報 T_RTEX についての詳細は，「16.2.3　タスク例外処理ルーチン詳細情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	対象タスクが制約タスクである
E_ID	-18	ID の指定が不正である - *tskid* ＜ 0x0 - *tskid* ＞生成されているタスクの最大 ID - 非タスクから本サービス・コールを発行した際，*tskid* に TSK_SELF を指定した

マクロ	数値	意味
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_OBJ	-41	管理オブジェクトの状態が不正である - 対象タスクが DORMANT 状態である - 対象タスクにタスク例外処理ルーチンが登録されていない
E_NOEXS	-42	対象タスクが生成されていない

17.2.4　同期通信機能（セマフォ）

以下に，RX850V4 が同期通信機能（セマフォ）として提供しているサービス・コールの一覧を示します。

表 17 － 4　同期通信機能（セマフォ）

サービス・コール名	機能概要	発行有効範囲
wai_sem	資源の獲得	タスク
pol_sem	資源の獲得（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
ipol_sem	資源の獲得（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
twai_sem	資源の獲得（タイムアウト付き）	タスク
sig_sem	資源の返却	タスク，制約タスク， 非タスク，初期化ルーチン
isig_sem	資源の返却	タスク，制約タスク， 非タスク，初期化ルーチン
ref_sem	セマフォ詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iref_sem	セマフォ詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン

wai_sem

概要

資源の獲得

C 言語形式

```
ER      wai_sem ( ID semid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*semid*;	セマフォの ID

機能

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。

ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，自タスクを対象セマフォの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（資源獲得待ち状態）へと遷移させます。

なお，資源獲得待ち状態の解除は，以下の場合に行われ，資源獲得待ち状態から READY 状態へと遷移します。

資源獲得待ち状態の解除操作	エラー・コード
sig_sem の発行により，対象セマフォに資源が返却された	E_OK
isig_sem の発行により，対象セマフォに資源が返却された	E_OK
rel_wai の発行により，資源獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，資源獲得待ち状態を強制的に解除された	E_RLWAI

備考　自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_ID	-18	ID の指定が不正である - *semid* ≦ 0x0 - *semid* ＞生成されているセマフォの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した

マクロ	数値	意味
E_NOEXS	-42	対象セマフォが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，資源獲得待ち状態を強制的に解除された

pol_sem
ipol_sem

概要

資源の獲得（ポーリング）

C 言語形式

```
ER      pol_sem ( ID semid );
ER      ipol_sem ( ID semid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*semid*;	セマフォの ID

機能

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。

ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，戻り値として E_TMOUT を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *semid* ≦ 0x0 - *semid* ＞生成されているセマフォの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象セマフォが生成されていない
E_TMOUT	-50	対象セマフォの資源数が 0x0 である

twai_sem

概要

資源の獲得（タイムアウト付き）

C 言語形式

```
ER      twai_sem ( ID semid, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID　*semid*;	セマフォの ID
I	TMO　*tmout*;	待ち時間（単位：ミリ秒） TMO_FEVR：永久待ち TMO_POL：ポーリング 数値：待ち時間

機能

semid で指定されたセマフォから資源を獲得（セマフォ・カウンタから 0x1 を減算）します。

ただし，本サービス・コールを発行した際，対象セマフォから資源を獲得することができなかった（空き資源が存在しなかった）場合には，資源の獲得は行わず，自タスクを対象セマフォの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（資源獲得待ち状態）へと遷移させます。

なお，資源獲得待ち状態の解除は，以下の場合に行われ，資源獲得待ち状態から READY 状態へと遷移します。

資源獲得待ち状態の解除操作	エラー・コード
sig_sem の発行により，対象セマフォに資源が返却された	E_OK
isig_sem の発行により，対象セマフォに資源が返却された	E_OK
rel_wai の発行により，資源獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，資源獲得待ち状態を強制的に解除された	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

備考 1　自タスクを対象セマフォの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　待ち時間 *tmout* に TMO_FEVR が指定された際には“wai_sem と同等の処理”を，TMO_POL が指定された際には“pol_sem，ipol_sem と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した

マクロ	数値	意味
E_PAR	-17	待ち時間の指定が不正（*tmout* ＜ TMO_FEVR）である
E_ID	-18	ID の指定が不正である - *semid* ≦ 0x0 - *semid* ＞生成されているセマフォの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象セマフォが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，資源獲得待ち状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した

sig_sem
isig_sem

概要

資源の返却

C 言語形式

```
ER      sig_sem ( ID semid );
ER      isig_sem ( ID semid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*semid*;	セマフォの ID

機能

semid で指定されたセマフォに資源を返却（セマフォ・カウンタに 0x1 を加算）します。

ただし，本サービス・コールを発行した際，対象セマフォの待ちキューにタスクがキューイングされていた場合には，資源の返却（セマフォ・カウンタの加算処理）は行わず，該当タスク（待ちキューの先頭タスク）に資源を渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（資源獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考　RX850V4 では，セマフォの資源数として取り得る最大値（最大資源数）をコンフィギュレーション時に定義させています。このため，本サービス・コールでは，資源数が最大資源数を越えるような場合には，資源の返却（セマフォ・カウンタの加算処理）は行わず，戻り値として E_QOVR を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *semid* ≦ 0x0 - *semid* ＞生成されているセマフォの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象セマフォが生成されていない
E_QOVR	-43	資源数が最大資源数を越えた

ref_sem
iref_sem

概要

セマフォ詳細情報の参照

C 言語形式

```
ER      ref_sem ( ID semid, T_RSEM *pk_rsem );
ER      iref_sem ( ID semid, T_RSEM *pk_rsem );
```

パラメータ

I/O	パラメータ	説明
I	ID　　*semid*;	セマフォの ID
O	T_RSEM　*pk_rsem*;	セマフォ詳細情報を格納する領域へのポインタ

【 セマフォ詳細情報 T_RSEM の構造 】

```
typedef struct  t_rsem {
        ID      wtskid;         /* 待ちタスクの有無 */
        UINT    semcnt;         /* 現在資源数 */
        ATR     sematr;         /* 属性 */
        UINT    maxsem;         /* 最大資源数 */
} T_RSEM;
```

機能

semid で指定されたセマフォのセマフォ詳細情報（待ちタスクの有無，現在資源数など）を *pk_rsem* で指定された領域に格納します。

備考　　セマフォ詳細情報 T_RSEM についての詳細は，「16.2.4　セマフォ詳細情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *semid* ≦ 0x0 - *semid* ＞生成されているセマフォの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象セマフォが生成されていない

17.2.5　同期通信機能（イベントフラグ）

以下に，RX850V4 が同期通信機能（イベントフラグ）として提供しているサービス・コールの一覧を示します。

表 17 － 5　同期通信機能（イベントフラグ）

サービス・コール名	機能概要	発行有効範囲
set_flg	ビット・パターンのセット	タスク，制約タスク， 非タスク，初期化ルーチン
iset_flg	ビット・パターンのセット	タスク，制約タスク， 非タスク，初期化ルーチン
clr_flg	ビット・パターンのクリア	タスク，制約タスク， 非タスク，初期化ルーチン
iclr_flg	ビット・パターンのクリア	タスク，制約タスク， 非タスク，初期化ルーチン
wai_flg	ビット・パターンのチェック	タスク
pol_flg	ビット・パターンのチェック（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
ipol_flg	ビット・パターンのチェック（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
twai_flg	ビット・パターンのチェック（タイムアウト付き）	タスク
ref_flg	イベントフラグ詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iref_flg	イベントフラグ詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン

set_flg
iset_flg

概要

ビット・パターンのセット

C 言語形式

```
ER      set_flg ( ID flgid, FLGPTN setptn );
ER      iset_flg ( ID flgid, FLGPTN setptn );
```

パラメータ

I/O	パラメータ	説明
I	ID　*flgid*;	イベントフラグの ID
I	FLGPTN　*setptn*;	セットするビット・パターン

機能

flgid で指定されたイベントフラグのビット・パターンと *setptn* で指定されたビット・パターンの論理和 OR をとり，その結果を対象イベントフラグにセットします。

なお，本サービス・コールを発行した際，対象イベントフラグの待ちキューにキューイングされているタスクの要求条件を満足した場合には，ビット・パターンのセット（論理和 OR の設定処理）とともに，該当タスクを待ちキューから外します。これにより，該当タスクは，WAITING 状態（イベントフラグ待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考　本サービス・コールを発行した際，対象イベントフラグのビット・パターンが B'1100，*setptn* で指定されたビット・パターンが B'1010 であった場合には，対象イベントフラグのビット・パターンは B'1110 となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *flgid* ≦ 0x0 - *flgid* ＞生成されているイベントフラグの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象イベントフラグが生成されていない

clr_flg
iclr_flg

概要

ビット・パターンのクリア

C 言語形式

```
ER      clr_flg ( ID flgid, FLGPTN clrptn );
ER      iclr_flg ( ID flgid, FLGPTN clrptn );
```

パラメータ

I/O	パラメータ	説明
I	ID　*flgid*;	イベントフラグの ID
I	FLGPTN　*clrptn*;	クリアするビット・パターン

機能

flgid で指定されたイベントフラグのビット・パターンと *clrptn* で指定されたビット・パターンの論理積 AND をとり，その結果を対象イベントフラグに設定します。

備考　本サービス・コールを発行した際，対象イベントフラグのビット・パターンが B'1100，*clrptn* で指定されたビット・パターンが B'1010 であった場合には，対象イベントフラグのビット・パターンは B'1000 となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *flgid* ≦ 0x0 - *flgid* ＞生成されているイベントフラグの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象イベントフラグが生成されていない

wai_flg

概要

ビット・パターンのチェック

C 言語形式

```
ER      wai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
```

パラメータ

I/O	パラメータ	説明
I	ID　*flgid*;	イベントフラグの ID
I	FLGPTN　*waiptn*;	要求するビット・パターン
I	MODE　*wfmode*;	要求条件の指定 TWF_ANDW： AND 待ち TWF_ORW：　OR 待ち
O	FLGPTN　*p_flgptn*;	条件成立時のビット・パターンを格納する領域へのポインタ

機能

waiptn で指定された要求ビット・パターンと *wfmode* で指定された要求条件を満足するビット・パターンが *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。

なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンを *p_flgptn* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，自タスクを対象イベントフラグの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（イベントフラグ待ち状態）へと遷移させます。

なお，イベントフラグ待ち状態の解除は，以下の場合に行われ，イベントフラグ待ち状態から READY 状態へと遷移します。

イベントフラグ待ち状態の解除操作	エラー・コード
set_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された	E_OK
iset_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された	E_OK
rel_wai の発行により，イベントフラグ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，イベントフラグ待ち状態を強制的に解除された	E_RLWAI

以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* = TWF_ANDW
  waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。

- *wfmode* = TWF_ORW
  waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

備考 1　RX850V4 では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した場合，RX850V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　自タスクを対象イベントフラグ（TA_WMUL 属性）の待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 3　対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際，RX850V4 はビット・パターンのクリア（0x0 の設定）を行います。

備考 4　rel_wai，または irel_wai の発行によりイベントフラグ待ち状態を解除された場合，*p_flgptn* で指定された領域の内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_PAR	-17	パラメータの指定が不正である - 要求ビット・パターンの指定が不正（*waiptn* = 0x0）である - 要求条件 *wfmode* の指定が不正である
E_ID	-18	ID の指定が不正である - *flgid* ≦ 0x0 - *flgid* ＞生成されているイベントフラグの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_ILUSE	-28	すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した
E_NOEXS	-42	対象イベントフラグが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，イベントフラグ待ち状態を強制的に解除された

pol_flg
ipol_flg

概要

ビット・パターンのチェック（ポーリング）

C 言語形式

```
ER      pol_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
ER      ipol_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn );
```

パラメータ

I/O	パラメータ	説明
I	ID　*flgid*;	イベントフラグの ID
I	FLGPTN　*waiptn*;	要求するビット・パターン
I	MODE　*wfmode*;	要求条件の指定 TWF_ANDW：AND 待ち TWF_ORW：　OR 待ち
O	FLGPTN　**p_flgptn*;	条件成立時のビット・パターンを格納する領域へのポインタ

機能

waiptn で指定された要求ビット・パターンと *wfmode* で指定された要求条件を満足するビット・パターンが *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。

なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンを *p_flgptn* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，戻り値として E_TMOUT を返します。

以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* = TWF_ANDW
  waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。

- *wfmode* = TWF_ORW
  waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

備考 1　RX850V4　では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した場合，RX850V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際，RX850V4 はビット・パターンのクリア（0x0 の設定）を行います。

備考 3　本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合，*p_flgptn* で指定された領域の内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	パラメータの指定が不正である - 要求ビット・パターンの指定が不正（*waiptn* = 0x0）である - 要求条件 *wfmode* の指定が不正である
E_ID	-18	ID の指定が不正である - *flgid* ≦ 0x0 - *flgid* ＞生成されているイベントフラグの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_ILUSE	-28	すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した
E_NOEXS	-42	対象イベントフラグが生成されていない
E_TMOUT	-50	対象イベントフラグのビット・パターンが要求条件を満足していない

twai_flg

概要

ビット・パターンのチェック（タイムアウト付き）

C 言語形式

```
ER      twai_flg ( ID flgid, FLGPTN waiptn, MODE wfmode, FLGPTN *p_flgptn, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID　*flgid*;	イベントフラグの ID
I	FLGPTN　*waiptn*;	要求するビット・パターン
I	MODE　*wfmode*;	要求条件の指定 TWF_ANDW：AND 待ち TWF_ORW：OR 待ち
O	FLGPTN　**p_flgptn*;	条件成立時のビット・パターンを格納する領域へのポインタ
I	TMO　*tmout*;	待ち時間（単位：ミリ秒） TMO_FEVR：永久待ち TMO_POL：ポーリング 数値：待ち時間

機能

waiptn で指定された要求ビット・パターンと *wfmode* で指定された要求条件を満足するビット・パターンが *flgid* で指定されたイベントフラグに設定されているか否かをチェックします。

なお，要求条件を満足するビット・パターンが対象イベントフラグに設定されていた場合には，対象イベントフラグのビット・パターンを *p_flgptn* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象イベントフラグのビット・パターンが要求条件を満足していなかった場合には，自タスクを対象イベントフラグの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（イベントフラグ待ち状態）へと遷移させます。

なお，イベントフラグ待ち状態の解除は，以下の場合に行われ，イベントフラグ待ち状態から READY 状態へと遷移します。

イベントフラグ待ち状態の解除操作	エラー・コード
set_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された	E_OK
iset_flg の発行により，対象イベントフラグに要求条件を満足するビット・パターンが設定された	E_OK
rel_wai の発行により，イベントフラグ待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，イベントフラグ待ち状態を強制的に解除された	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

以下に，要求条件 *wfmode* の指定形式を示します。

- *wfmode* = TWF_ANDW
  waiptn で 1 を設定している全ビットが対象イベントフラグに設定されているか否かをチェックします。

- *wfmode* = TWF_ORW
  waiptn で 1 を設定しているビットのうち，いずれかのビットが対象イベントフラグに設定されているか否かをチェックします。

備考 1　RX850V4　では，イベントフラグの待ちキューに複数のタスクをキューイング可能とするか否かをコンフィギュレーション時に定義させています。このため，すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した場合，RX850V4 は要求条件の即時成立／不成立を問わず，戻り値として E_ILUSE を返します。

TA_WSGL 属性：　キューイング可能なタスクは，1 個
TA_WMUL 属性：　キューイング可能なタスクは，複数

備考 2　自タスクを対象イベントフラグ（TA_WMUL　属性）の待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 3　対象イベントフラグ（TA_CLR 属性）の要求条件が満足した際，RX850V4 はビット・パターンのクリア（0x0 の設定）を行います。

備考 4　rel_wai，または　irel_wai　の発行，または待ち時間の経過によりイベントフラグ待ち状態を解除された場合，*p_flgptn* で指定された領域の内容は不定となります。

備考 5　待ち時間 *tmout* に TMO_FEVR が指定された際には“wai_flg と同等の処理”を，TMO_POL が指定された際には“pol_flg，ipol_flg と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_PAR	-17	パラメータの指定が不正である - 要求ビット・パターンの指定が不正（*waiptn* = 0x0）である - 要求条件 *wfmode* の指定が不正である - *tmout* ＜ TMO_FEVR
E_ID	-18	ID の指定が不正である - *flgid* ≦ 0x0 - *flgid* ＞生成されているイベントフラグの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_ILUSE	-28	すでに待ちタスクがキューイングされているイベントフラグ（TW_WSGL 属性）に対して本サービス・コールを発行した
E_NOEXS	-42	対象イベントフラグが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，イベントフラグ待ち状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した

ref_flg
iref_flg

概要

イベントフラグ詳細情報の参照

C 言語形式

```
ER      ref_flg ( ID flgid, T_RFLG *pk_rflg );
ER      iref_flg ( ID flgid, T_RFLG *pk_rflg );
```

パラメータ

I/O	パラメータ	説明
I	ID　　*flgid*;	イベントフラグの ID
O	T_RFLG　*pk_rflg*;	イベントフラグ詳細情報を格納する領域へのポインタ

【 イベントフラグ詳細情報 T_RFLG の構造 】

```
typedef struct  t_rflg {
        ID      wtskid;         /* 待ちタスクの有無 */
        FLGPTN  flgptn;         /* 現在ビット・パターン */
        ATR     flgatr;         /* 属性 */
} T_RFLG;
```

機能

flgid で指定されたイベントフラグのイベントフラグ詳細情報（待ちタスクの有無，現在ビット・パターンなど）を *pk_rflg* で指定された領域に格納します。

備考　イベントフラグ詳細情報 T_RFLG についての詳細は，「16.2.5　イベントフラグ詳細情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *flgid* ≦ 0x0 - *flgid* ＞生成されているイベントフラグの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象イベントフラグが生成されていない

17.2.6　同期通信機能（データ・キュー）

以下に，RX850V4 が同期通信機能（データ・キュー）として提供しているサービス・コールの一覧を示します。

表 17 － 6　同期通信機能（データ・キュー）

サービス・コール名	機能概要	発行有効範囲
snd_dtq	データの送信	タスク
psnd_dtq	データの送信（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
ipsnd_dtq	データの送信（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
tsnd_dtq	データの送信（タイムアウト付き）	タスク
fsnd_dtq	データの強制送信	タスク，制約タスク， 非タスク，初期化ルーチン
ifsnd_dtq	データの強制送信	タスク，制約タスク， 非タスク，初期化ルーチン
rcv_dtq	データの受信	タスク
prcv_dtq	データの受信（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
iprcv_dtq	データの受信（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
trcv_dtq	データの受信（タイムアウト付き）	タスク
ref_dtq	データ・キュー詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iref_dtq	データ・キュー詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン

snd_dtq

概要

データの送信

C 言語形式

```
ER      snd_dtq ( ID dtqid, VP_INT data );
```

パラメータ

I/O	パラメータ	説明
I	ID　*dtqid*;	データ・キューの ID
I	VP_INT　*data*;	送信するデータ

機能

dtqid で指定されたデータ・キューのデータ・キュー領域に *data* で指定されたデータを書き込みます。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，自タスクを対象データ・キューの送信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（データ送信待ち状態）へと遷移させます。

なお，データ送信待ち状態の解除は，以下の場合に行われ，データ送信待ち状態から READY 状態へと遷移します。

データ送信待ち状態の解除操作	エラー・コード
rcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
prcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
iprcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
trcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
rel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI

また，本サービス・コールを発行した際，対象データ・キューの受信待ちキューにタスクがキューイングされていた場合には，データの書き込みは行わず，該当タスクにデータを渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考 1　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

備考 2　自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_ID	-18	ID の指定が不正である - *dtqid* ≦ 0x0 - *dtqid* ＞生成されているデータ・キューの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象データ・キューが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，データ送信待ち状態を強制的に解除された

psnd_dtq
ipsnd_dtq

概要

データの送信（ポーリング）

C 言語形式

```
ER      psnd_dtq ( ID dtqid, VP_INT data );
ER      ipsnd_dtq ( ID dtqid, VP_INT data );
```

パラメータ

I/O	パラメータ	説明
I	ID　*dtqid*;	データ・キューの ID
I	VP_INT　*data*;	送信するデータ

機能

dtqid で指定されたデータ・キューのデータ・キュー領域に *data* で指定されたデータを書き込みます。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，戻り値として E_TMOUT を返します。

また，本サービス・コールを発行した際，対象データ・キューの受信待ちキューにタスクがキューイングされていた場合には，データの書き込みは行わず，該当タスクにデータを渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *dtqid* ≦ 0x0 - *dtqid* ＞生成されているデータ・キューの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象データ・キューが生成されていない
E_TMOUT	-50	対象データ・キューのデータ・キュー領域に空き領域が存在しない

tsnd_dtq

概要

データの送信（タイムアウト付き）

C 言語形式

```
ER      tsnd_dtq ( ID dtqid, VP_INT data, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID　*dtqid*;	データ・キューの ID
I	VP_INT　*data*;	送信するデータ
I	TMO　*tmout*;	待ち時間（単位：ミリ秒） TMO_FEVR：　永久待ち TMO_POL：　ポーリング 数値：　待ち時間

機能

dtqid で指定されたデータ・キューのデータ・キュー領域に *data* で指定されたデータを書き込みます。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，データの書き込みは行わず，自タスクを対象データ・キューの送信待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（データ送信待ち状態）へと遷移させます。

なお，データ送信待ち状態の解除は，以下の場合に行われ，データ送信待ち状態から READY 状態へと遷移します。

データ送信待ち状態の解除操作	エラー・コード
rcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
prcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
iprcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
trcv_dtq の発行により，対象データ・キューのデータ・キュー領域に空き領域が確保された	E_OK
rel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ送信待ち状態を強制的に解除された	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

また，本サービス・コールを発行した際，対象データ・キューの受信待ちキューにタスクがキューイングされていた場合には，データの書き込みは行わず，該当タスクにデータを渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考 1　データを対象データ・キューのデータ・キュー領域に書き込む際の書き込み方法は，データの送信要求を行った順に行われます。

備考 2　自タスクを対象データ・キューの送信待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には“snd_dtq と同等の処理”を，TMO_POL が指定された際には“psnd_dtq，ipsnd_dtq と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_PAR	-17	待ち時間の指定が不正（*tmout* ＜ TMO_FEVR）である
E_ID	-18	ID の指定が不正である - *dtqid* ≦ 0x0 - *dtqid* ＞生成されているデータ・キューの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象データ・キューが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，データ送信待ち状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した

fsnd_dtq
ifsnd_dtq

概要

データの強制送信

C 言語形式

```
ER      fsnd_dtq ( ID dtqid, VP_INT data );
ER      ifsnd_dtq ( ID dtqid, VP_INT data );
```

パラメータ

I/O	パラメータ	説明
I	ID　*dtqid*;	データ・キューの ID
I	VP_INT　*data*;	送信するデータ

機能

dtqid で指定されたデータ・キューのデータ・キュー領域に *data* で指定されたデータを書き込みます。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域にデータを書き込むための空き領域が存在しなかった場合には，書き込まれてから最も時間が経過しているデータの領域に該当データを上書きします。

また，本サービス・コールを発行した際，対象データ・キューの受信待ちキューにタスクがキューイングされていた場合には，データの書き込みは行わず，該当タスクにデータを渡します。これにより，該当タスクは，受信待ちキューから外れ，WAITING 状態（データ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *dtqid* ≦ 0x0 - *dtqid* ＞生成されているデータ・キューの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_ILUSE	-28	データ・キュー領域のデータ数が 0x0 のデータ・キューに対して本サービス・コールを発行した
E_NOEXS	-42	対象データ・キューが生成されていない

rcv_dtq

概要

データの受信

C 言語形式

```
ER      rcv_dtq ( ID dtqid, VP_INT *p_data );
```

パラメータ

I/O	パラメータ	説明
I	ID　*dtqid*;	データ・キューの ID
O	VP_INT　**p_data*;	データを格納する領域へのポインタ

機能

dtqid で指定されたデータ・キューのデータ・キュー領域からデータを読み込み，*p_data* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，自タスクを対象データ・キューの受信待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（データ受信待ち状態）へと遷移させます。

なお，データ受信待ち状態の解除は，以下の場合に行われ，データ受信待ち状態から READY 状態へと遷移します。

データ受信待ち状態の解除操作	エラー・コード
snd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
psnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ipsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
tsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
fsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ifsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
rel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI

備考 1　自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は，データの受信要求を行った順に行われます。

備考 2　rel_wai，または irel_wai の発行によりデータ受信待ち状態を解除された場合，*p_data* で指定された領域の内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

マクロ	数値	意味
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_ID	-18	ID の指定が不正である - *dtqid* ≦ 0x0 - *dtqid* ＞生成されているデータ・キューの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象データ・キューが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，データ受信待ち状態を強制的に解除された

prcv_dtq
iprcv_dtq

概要

データの受信（ポーリング）

C 言語形式

```
ER      prcv_dtq ( ID dtqid, VP_INT *p_data );
ER      iprcv_dtq ( ID dtqid, VP_INT *p_data );
```

パラメータ

I/O	パラメータ	説明
I	ID　*dtqid*;	データ・キューの ID
O	VP_INT　**p_data*;	データを格納する領域へのポインタ

機能

dtqid で指定されたデータ･キューのデータ･キュー領域からデータを読み込み，*p_data* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，戻り値として E_TMOUT を返します。

備考　本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合，*p_data* で指定された領域の内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *dtqid* ≦ 0x0 - *dtqid* ＞生成されているデータ・キューの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象データ・キューが生成されていない
E_TMOUT	-50	対象データ・キューのデータ・キュー領域にデータが書き込まれていない

trcv_dtq

概要

データの受信（タイムアウト付き）

C 言語形式

```
ER      trcv_dtq ( ID dtqid, VP_INT *p_data, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID　*dtqid*;	データ・キューの ID
O	VP_INT　**p_data*;	データを格納する領域へのポインタ
I	TMO　*tmout*;	待ち時間（単位：ミリ秒） TMO_FEVR：　永久待ち TMO_POL：　ポーリング 数値：　待ち時間

機能

dtqid で指定されたデータ・キューのデータ・キュー領域からデータを読み込み，*p_data* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象データ・キューのデータ・キュー領域からデータを読み込むことができなかった（データ・キュー領域にデータが書き込まれていなかった）場合には，データの読み込みは行わず，自タスクを対象データ・キューの受信待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（データ受信待ち状態）へと遷移させます。

なお，データ受信待ち状態の解除は，以下の場合に行われ，データ受信待ち状態から READY 状態へと遷移します。

データ受信待ち状態の解除操作	エラー・コード
snd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
psnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ipsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
tsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
fsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
ifsnd_dtq の発行により，対象データ・キューのデータ・キュー領域にデータが書き込まれた	E_OK
rel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，データ受信待ち状態を強制的に解除された	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

備考 1　自タスクを対象データ・キューの受信待ちキューにキューイングする際のキューイング方式は，データの受信要求を行った順に行われます。

備考 2　rel_wai，または irel_wai の発行，または待ち時間の経過によりデータ受信待ち状態を解除された場合，*p_data* で指定された領域の内容は不定となります。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には“rcv_dtq と同等の処理”を，TMO_POL が指定された際には“prcv_dtq，iprcv_dtq と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_PAR	-17	待ち時間の指定が不正（*tmout* ＜ TMO_FEVR）である
E_ID	-18	ID の指定が不正である - *dtqid* ≦ 0x0 - *dtqid* ＞生成されているデータ・キューの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象データ・キューが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，データ受信待ち状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した

ref_dtq
iref_dtq

概要

データ・キュー詳細情報の参照

C 言語形式

```
ER      ref_dtq ( ID dtqid, T_RDTQ *pk_rdtq );
ER      iref_dtq ( ID dtqid, T_RDTQ *pk_rdtq );
```

パラメータ

I/O	パラメータ	説明
I	`ID dtqid;`	データ・キューの ID
O	`T_RDTQ *pk_rdtq;`	データ・キュー詳細情報を格納する領域へのポインタ

【 データ・キュー詳細情報 T_RDTQ の構造 】

```
typedef struct  t_rdtq {
        ID      stskid;         /* データ送信待ちタスクの有無 */
        ID      rtskid;         /* データ受信待ちタスクの有無 */
        UINT    sdtqcnt;        /* 未受信データの総数 */
        ATR     dtqatr;         /* 属性 */
        UINT    dtqcnt;         /* データ数 */
        ID      memid;          /* システム予約領域 */
} T_RDTQ;
```

機能

dtqid で指定されたデータ・キューのデータ・キュー詳細情報（待ちタスクの有無，未受信データの総数など）を *pk_rdtq* で指定された領域に格納します。

備考　データ・キュー詳細情報 T_RDTQ についての詳細は，「16.2.6　データ・キュー詳細情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *dtqid* ≦ 0x0 - *dtqid* ＞生成されているデータ・キューの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象データ・キューが生成されていない

17.2.7　同期通信機能（メールボックス）

以下に，RX850V4 が同期通信機能（メールボックス）として提供しているサービス・コールの一覧を示します。

表 17 － 7　同期通信機能（メールボックス）

サービス・コール名	機能概要	発行有効範囲
snd_mbx	メッセージの送信	タスク，制約タスク， 非タスク，初期化ルーチン
isnd_mbx	メッセージの送信	タスク，制約タスク， 非タスク，初期化ルーチン
rcv_mbx	メッセージの受信	タスク
prcv_mbx	メッセージの受信（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
iprcv_mbx	メッセージの受信（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
trcv_mbx	メッセージの受信（タイムアウト付き）	タスク
ref_mbx	メールボックス詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iref_mbx	メールボックス詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン

snd_mbx
isnd_mbx

概要

メッセージの送信

C 言語形式

```
ER      snd_mbx ( ID mbxid, T_MSG *pk_msg );
ER      isnd_mbx ( ID mbxid, T_MSG *pk_msg );
```

パラメータ

I/O	パラメータ	説明
I	ID　　*mbxid*;	メールボックスの ID
I	T_MSG　**pk_msg*;	メッセージを格納した領域へのポインタ

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  t_msg {
        struct  t_msg   *msgnext;       /* システム予約領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  t_msg_pri {
        struct  t_msg   msgque;         /* システム予約領域 */
        PRI     msgpri;                 /* 優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスに *pk_msg* で指定されたメッセージを送信します。

ただし，本サービス・コールを発行した際，対象メールボックスの待ちキューにタスクがキューイングされていた場合には，メッセージの送信（メッセージのキューイング処理）は行わず，該当タスクにメッセージを渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（メッセージ受信待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考 1　メッセージを対象メールボックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　RX850V4 のメールボックスは，メッセージの先頭アドレスを受信側処理プログラムに渡すだけであり，メッセージの内容が他領域にコピーされるわけではありません。したがって，本サービス・コールの発行後であってもメッセージの内容を書き換えることができます。

備考 3　メッセージ T_MSG，T_MSG_PRI についての詳細は，「16.2.7　メッセージ」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	優先度の指定が不正である - msgpri ≦ 0x0 - msgpri ＞最大メッセージ優先度
E_ID	-18	ID の指定が不正である - *mbxid* ≦ 0x0 - *mbxid* ＞生成されているメールボックスの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象メールボックスが生成されていない

rcv_mbx

概要

メッセージの受信

C 言語形式

```
ER      rcv_mbx ( ID mbxid, T_MSG **ppk_msg );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mbxid*;	メールボックスの ID
O	T_MSG　***ppk_msg*;	メッセージの先頭アドレスを格納する領域へのポインタ

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  t_msg {
        struct  t_msg   *msgnext;       /* システム予約領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  t_msg_pri {
        struct  t_msg   msgque;         /* システム予約領域 */
        PRI     msgpri;                 /* 優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスからメッセージを受信し，その先頭アドレスを *ppk_msg* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，自タスクを対象メールボックスの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（メッセージ受信待ち状態）へと遷移させます。

なお，メッセージ受信待ち状態の解除は，以下の場合に行われ，メッセージ受信待ち状態から READY 状態へと遷移します。

メッセージ受信待ち状態の解除操作	エラー・コード
snd_mbx の発行により，対象メールボックスにメッセージが送信された	E_OK
isnd_mbx の発行により，対象メールボックスにメッセージが送信された	E_OK
rel_wai の発行により，メッセージ受信待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，メッセージ受信待ち状態を強制的に解除された	E_RLWAI

備考 1　自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　rel_wai，または irel_wai の発行によりメッセージ受信待ち状態を解除された場合，*ppk_msg* で指定された領域の内容は不定となります。

備考 3　メッセージ T_MSG，T_MSG_PRI についての詳細は，「16.2.7　メッセージ」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_ID	-18	ID の指定が不正である - *mbxid* ≦ 0x0 - *mbxid* ＞生成されているメールボックスの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象メールボックスが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，メッセージ受信待ち状態を強制的に解除された

prcv_mbx
iprcv_mbx

概要

メッセージの受信（ポーリング）

C 言語形式

```
ER      prcv_mbx ( ID mbxid, T_MSG **ppk_msg );
ER      iprcv_mbx ( ID mbxid, T_MSG **ppk_msg );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mbxid*;	メールボックスの ID
O	T_MSG　***ppk_msg*;	メッセージの先頭アドレスを格納する領域へのポインタ

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  t_msg {
        struct  t_msg   *msgnext;       /* システム予約領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  t_msg_pri {
        struct  t_msg   msgque;         /* システム予約領域 */
        PRI     msgpri;                 /* 優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスからメッセージを受信し，その先頭アドレスを *ppk_msg* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，戻り値として E_TMOUT を返します。

備考 1　本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合，*ppk_msg* で指定された領域の内容は不定となります。

備考 2　メッセージ T_MSG，T_MSG_PRI についての詳細は，「16.2.7　メッセージ」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *mbxid* ≦ 0x0 - *mbxid* ＞生成されているメールボックスの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象メールボックスが生成されていない
E_TMOUT	-50	対象メールボックスの待ちキューにメッセージがキューイングされていない

trcv_mbx

概要

メッセージの受信（タイムアウト付き）

C 言語形式

```
ER      trcv_mbx ( ID mbxid, T_MSG **ppk_msg, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mbxid*;	メールボックスの ID
O	T_MSG　***ppk_msg*;	メッセージの先頭アドレスを格納する領域へのポインタ
I	TMO　*tmout*;	待ち時間（単位：ミリ秒） TMO_FEVR：　永久待ち TMO_POL：　ポーリング 数値：　待ち時間

【 TA_MFIFO 属性用メッセージ T_MSG の構造 】

```
typedef struct  t_msg {
        struct  t_msg   *msgnext;       /* システム予約領域 */
} T_MSG;
```

【 TA_MPRI 属性用メッセージ T_MSG_PRI の構造 】

```
typedef struct  t_msg_pri {
        struct  t_msg   msgque;         /* システム予約領域 */
        PRI     msgpri;                 /* 優先度 */
} T_MSG_PRI;
```

機能

mbxid で指定されたメールボックスからメッセージを受信し，その先頭アドレスを *ppk_msg* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象メールボックスからメッセージを受信することができなかった（待ちキューにメッセージがキューイングされていなかった）場合には，メッセージの受信は行わず，自タスクを対象メールボックスの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（メッセージ受信待ち状態）へと遷移させます。

なお，メッセージ受信待ち状態の解除は，以下の場合に行われ，メッセージ受信待ち状態から READY 状態へと遷移します。

メッセージ受信待ち状態の解除操作	エラー・コード
snd_mbx の発行により，対象メールボックスにメッセージが送信された	E_OK
isnd_mbx の発行により，対象メールボックスにメッセージが送信された	E_OK
rel_wai の発行により，メッセージ受信待ち状態を強制的に解除された	E_RLWAI

メッセージ受信待ち状態の解除操作	エラー・コード
irel_wai の発行により，メッセージ受信待ち状態を強制的に解除された	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

備考 1　自タスクを対象メールボックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　rel_wai，または irel_wai の発行，または待ち時間の経過によりメッセージ受信待ち状態を解除された場合，*ppk_msg* で指定された領域の内容は不定となります。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には“rcv_mbx と同等の処理”を，TMO_POL が指定された際には“prcv_mbx，iprcv_mbx と同等の処理”を実行します。

備考 4　メッセージ T_MSG，T_MSG_PRI についての詳細は，「16.2.7　メッセージ」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_PAR	-17	待ち時間の指定が不正（*tmout* ＜ TMO_FEVR）である
E_ID	-18	ID の指定が不正である - *mbxid* ≦ 0x0 - *mbxid* ＞生成されているメールボックスの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象メールボックスが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，メッセージ受信待ち状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した

ref_mbx
iref_mbx

概要

メールボックス詳細情報の参照

C 言語形式

```
ER      ref_mbx ( ID mbxid, T_RMBX *pk_rmbx );
ER      iref_mbx ( ID mbxid, T_RMBX *pk_rmbx );
```

パラメータ

I/O	パラメータ	説明
I	`ID  mbxid;`	メールボックスの ID
O	`T_RMBX  *pk_rmbx;`	メールボックス詳細情報を格納する領域へのポインタ

【 メールボックス詳細情報 T_RMBX の構造 】

```
typedef struct  t_rmbx {
        ID      wtskid;         /* 待ちタスクの有無 */
        T_MSG   *pk_msg;        /* 待ちメッセージの有無 */
        ATR     mbxatr;         /* 属性 */
} T_RMBX;
```

機能

mbxid で指定されたメールボックスのメールボックス詳細情報（待ちタスクの有無，待ちメッセージの有無など）を *pk_rmbx* で指定された領域に格納します。

備考　メールボックス詳細情報 T_RMBX についての詳細は，「16.2.8　メールボックス詳細情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *mbxid* ≦ 0x0 - *mbxid* ＞生成されているメールボックスの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象メールボックスが生成されていない

17.2.8　拡張同期通信機能（ミューテックス）

以下に，RX850V4 が拡張同期通信機能（ミューテックス）として提供しているサービス・コールの一覧を示します。

表 17 － 8　拡張同期通信機能（ミューテックス）

サービス・コール名	機能概要	発行有効範囲
loc_mtx	ミューテックスのロック	タスク
ploc_mtx	ミューテックスのロック（ポーリング）	タスク，制約タスク
tloc_mtx	ミューテックスのロック（タイムアウト付き）	タスク
unl_mtx	ミューテックスのロック解除	タスク，制約タスク
ref_mtx	ミューテックス詳細情報の参照	タスク，制約タスク 非タスク，初期化ルーチン
iref_mtx	ミューテックス詳細情報の参照	タスク，制約タスク 非タスク，初期化ルーチン

loc_mtx

概要

ミューテックスのロック

C 言語形式

```
ER      loc_mtx ( ID mtxid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mtxid*;	ミューテックスの ID

機能

mtxid で指定されたミューテックスをロックします。

ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，自タスクを対象ミューテックスの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（ミューテックス待ち状態）へと遷移させます。

なお，ミューテックス待ち状態の解除は，以下の場合に行われ，ミューテックス待ち状態から READY 状態へと遷移します。

ミューテックス待ち状態の解除操作	エラー・コード
unl_mtx の発行により，対象ミューテックスのロック状態が解除された	E_OK
ext_tsk の発行により，対象ミューテックスのロック状態が解除された	E_OK
ter_tsk の発行により，対象ミューテックスのロック状態が解除された	E_OK
rel_wai の発行により，ミューテックス待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，ミューテックス待ち状態を強制的に解除された	E_RLWAI

備考 1　自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　RX850V4 では，自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_ID	-18	ID の指定が不正である - *mtxid* ≦ 0x0 - *mtxid* ＞生成されているミューテックスの最大 ID

マクロ	数値	意味
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_ILUSE	-28	すでにロック状態へと遷移させているミューテックスに対して本サービス・コールを発行した
E_NOEXS	-42	対象ミューテックスが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，ミューテックス待ち状態を強制的に解除された

ploc_mtx

概要

ミューテックスのロック（ポーリング）

C 言語形式

```
ER      ploc_mtx ( ID mtxid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mtxid*;	ミューテックスの ID

機能

mtxid で指定されたミューテックスをロックします。
ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，戻り値として E_TMOUT を返します。

備考　RX850V4 では，対象ミューテックスをロック状態へと遷移させたタスクがロック状態を解除することなく，本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_ID	-18	ID の指定が不正である - *mtxid* ≦ 0x0 - *mtxid* ＞生成されているミューテックスの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した
E_ILUSE	-28	すでにロック状態へと遷移させているミューテックスに対して本サービス・コールを発行した
E_NOEXS	-42	対象ミューテックスが生成されていない
E_TMOUT	-50	すでに他タスクが対象ミューテックスをロックしている

tloc_mtx

概要

ミューテックスのロック（タイムアウト付き）

C 言語形式

```
ER      tloc_mtx ( ID mtxid, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mtxid*;	ミューテックスの ID
I	TMO　*tmout*;	待ち時間（単位：ミリ秒） TMO_FEVR：永久待ち TMO_POL：ポーリング 数値：待ち時間

機能

mtxid で指定されたミューテックスをロックします。

ただし，本サービス・コールを発行した際，対象ミューテックスをロックすることができなかった（すでに他タスクがロックしていた）場合には，自タスクを対象ミューテックスの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（ミューテックス待ち状態）へと遷移させます。

なお，ミューテックス待ち状態の解除は，以下の場合に行われ，ミューテックス待ち状態から READY 状態へと遷移します。

ミューテックス待ち状態の解除操作	エラー・コード
unl_mtx の発行により，対象ミューテックスのロック状態が解除された	E_OK
ext_tsk の発行により，対象ミューテックスのロック状態が解除された	E_OK
ter_tsk の発行により，対象ミューテックスのロック状態が解除された	E_OK
rel_wai の発行により，ミューテックス待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，ミューテックス待ち状態を強制的に解除された	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

備考 1　自タスクを対象ミューテックスの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　RX850V4 では，自タスクがロックしているミューテックスに対して本サービス・コールを再発行（ミューテックスの多重ロック）した際には，戻り値として E_ILUSE を返します。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には“loc_mtx と同等の処理”を，TMO_POL が指定された際には“ploc_mtx と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_PAR	-17	待ち時間の指定が不正（*tmout* ＜ TMO_FEVR）である
E_ID	-18	ID の指定が不正である - *mtxid* ≦ 0x0 - *mtxid* ＞生成されているミューテックスの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_ILUSE	-28	すでにロック状態へと遷移させているミューテックスに対して本サービス･コールを発行した
E_NOEXS	-42	対象ミューテックスが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，ミューテックス待ち状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した

unl_mtx

概要

ミューテックスのロック解除

C 言語形式

```
ER      unl_mtx ( ID mtxid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mtxid*;	ミューテックスの ID

機能

mtxid で指定されたミューテックスのロック状態を解除します。

ただし，本サービス・コールを発行した際，対象ミューテックスの待ちキューにタスクがキューイングされていた場合には，ミューテックスのロック解除処理後，ただちに該当タスク（待ちキューの先頭タスク）によるミューテックスのロック処理が行われます。

このとき，該当タスクは，待ちキューから外れ，WAITING 状態（ミューテックス待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考　ミューテックスのロック解除が可能なタスクは“対象ミューテックスをロックしたタスク”に限られます。このため，自タスクがロックしていないミューテックスに対して本サービス・コールを発行した場合には，何も処理は行わず，戻り値として E_ILUSE を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *mtxid* ≦ 0x0 - *mtxid* ＞生成されているミューテックスの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した
E_ILUSE	-28	サービス・コールの使用方法が不正である - すでにロック状態が解除されている - ロック状態へと遷移させたのは他タスクである
E_NOEXS	-42	対象ミューテックスが生成されていない

ref_mtx
iref_mtx

概要

ミューテックス詳細情報の参照

C 言語形式

```
ER      ref_mtx ( ID mtxid, T_RMTX *pk_rmtx );
ER      iref_mtx ( ID mtxid, T_RMTX *pk_rmtx );
```

パラメータ

I/O	パラメータ	説明
I	ID　　*mtxid*;	ミューテックスの ID
O	T_RMTX　*pk_rmtx*;	ミューテックス詳細情報を格納する領域へのポインタ

【 ミューテックス詳細情報 T_RMTX の構造 】

```
typedef struct  t_rmtx {
        ID      htskid;         /* ロックの有無 */
        ID      wtskid;         /* 待ちタスクの有無 */
        ATR     mtxatr;         /* 属性 */
        PRI     ceilpri;        /* システム予約領域 */
} T_RMTX;
```

機能

mtxid で指定されたミューテックスのミューテックス詳細情報（ロックの有無，待ちタスクの有無など）を *pk_rmtx* で指定された領域に格納します。

備考　ミューテックス詳細情報 T_RMTX についての詳細は，「16.2.9　ミューテックス詳細情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *mtxid* ≦ 0x0 - *mtxid* ＞生成されているミューテックスの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象ミューテックスが生成されていない

17.2.9　メモリ・プール管理機能（固定長メモリ・プール）

以下に，RX850V4 がメモリ・プール管理機能（固定長メモリ・プール）として提供しているサービス・コールの一覧を示します。

表 17 － 9　メモリ・プール管理機能（固定長メモリ・プール）

サービス・コール名	機能概要	発行有効範囲
get_mpf	固定長メモリ・ブロックの獲得	タスク
pget_mpf	固定長メモリ・ブロックの獲得（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
ipget_mpf	固定長メモリ・ブロックの獲得（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
tget_mpf	固定長メモリ・ブロックの獲得（タイムアウト付き）	タスク
rel_mpf	固定長メモリ・ブロックの返却	タスク，制約タスク， 非タスク，初期化ルーチン
irel_mpf	固定長メモリ・ブロックの返却	タスク，制約タスク， 非タスク，初期化ルーチン
ref_mpf	固定長メモリ・プール詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iref_mpf	固定長メモリ・プール詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン

get_mpf

概要

固定長メモリ・ブロックの獲得

C 言語形式

```
ER      get_mpf ( ID mpfid, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mpfid*;	固定長メモリ・プールの ID
O	VP　**p_blk*;	固定長メモリ・プールの先頭アドレスを格納する領域へのポインタ

機能

mpfid で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお，固定長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われ，固定長メモリ・ブロック獲得待ち状態から READY 状態へと遷移します。

固定長メモリ・ブロック獲得待ち状態の解除操作	エラー・コード
rel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された	E_OK
irel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された	E_OK
rel_wai の発行により，固定長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，固定長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI

備考 1　自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　rel_wai，または irel_wai の発行により固定長メモリ・ブロック獲得待ち状態を解除された場合，*p_blk* で指定された領域の内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した

マクロ	数値	意味
E_ID	-18	ID の指定が不正である - *mpfid* ≦ 0x0 - *mpfid* ＞生成されている固定長メモリ・プールの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象固定長メモリ・プールが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，固定長メモリ・ブロック獲得待ち状態を強制的に解除された

pget_mpf
ipget_mpf

概要

固定長メモリ・ブロックの獲得（ポーリング）

C 言語形式

```
ER      pget_mpf ( ID mpfid, VP *p_blk );
ER      ipget_mpf ( ID mpfid, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mpfid*;	固定長メモリ・プールの ID
O	VP　**p_blk*;	固定長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mpfid で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，戻り値として E_TMOUT を返します。

備考　本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合，*p_blk* で指定された領域の内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *mpfid* ≦ 0x0 - *mpfid* ＞生成されている固定長メモリ・プールの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象固定長メモリ・プールが生成されていない
E_TMOUT	-50	対象固定長メモリ・プールに空き固定長メモリ・ブロックが存在しない

tget_mpf

概要

固定長メモリ・ブロックの獲得（タイムアウト付き）

C 言語形式

```
ER      tget_mpf ( ID mpfid, VP *p_blk, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mpfid*;	固定長メモリ・プールの ID
O	VP　**p_blk*;	固定長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ
I	TMO　*tmout*;	待ち時間（単位：ミリ秒） TMO_FEVR：　永久待ち TMO_POL：　ポーリング 数値：　待ち時間

機能

mpfid で指定された固定長メモリ・プールから固定長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象固定長メモリ・プールから固定長メモリ・ブロックを獲得することができなかった（空き固定長メモリ・ブロックが存在しなかった）場合には，固定長メモリ・ブロックの獲得は行わず，自タスクを対象固定長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（固定長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお，固定長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われ，固定長メモリ・ブロック獲得待ち状態から READY 状態へと遷移します。

固定長メモリ・ブロック獲得待ち状態の解除操作	エラー・コード
rel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された	E_OK
irel_mpf の発行により，対象固定長メモリ・プールに固定長メモリ・ブロックが返却された	E_OK
rel_wai の発行により，固定長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，固定長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

備考 1　自タスクを対象固定長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 2　rel_wai，または irel_wai の発行，または待ち時間の経過により固定長メモリ・ブロック獲得待ち状態を解除された場合，*p_blk* で指定された領域の内容は不定となります。

備考 3　待ち時間 *tmout* に TMO_FEVR が指定された際には“get_mpf と同等の処理”を，TMO_POL が指定された際には“pget_mpf，ipget_mpf と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_PAR	-17	待ち時間の指定が不正（*tmout* ＜ TMO_FEVR）である
E_ID	-18	ID の指定が不正である - *mpfid* ≦ 0x0 - *mpfid* ＞生成されている固定長メモリ・プールの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象固定長メモリ・プールが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，固定長メモリ・ブロック獲得待ち状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した

rel_mpf
irel_mpf

概要

固定長メモリ・ブロックの返却

C 言語形式

```
ER      rel_mpf ( ID mpfid, VP blk );
ER      irel_mpf ( ID mpfid, VP blk );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mpfid*;	固定長メモリ・プールの ID
I	VP　*blk*;	固定長メモリ・ブロックの先頭アドレス

機能

mpfid で指定された固定長メモリ・プールに *blk* で指定された固定長メモリ・ブロックを返却します。

ただし，本サービス・コールを発行した際，対象固定長メモリ・プールの待ちキューにタスクがキューイングされていた場合には，固定長メモリ・ブロックの返却は行わず，該当タスク（待ちキューの先頭タスク）に固定長メモリ・ブロックを渡します。これにより，該当タスクは，待ちキューから外れ，WAITING 状態（固定長メモリ・ブロック獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考 1　RX850V4 では，固定長メモリ・ブロックを返却する際，メモリ・クリア処理を行っていません。したがって，返却された固定長メモリ・ブロックの内容は不定となります。

備考 2　固定長メモリ・ブロックを返却する際は，必ず獲得した固定長メモリ・プールに対して本サービス・コールを発行してください。異なる固定長メモリ・プールに対して本サービス・コールを発行してもエラーにはなりませんが，以後の動作は保証されません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *mpfid* ≦ 0x0 - *mpfid* ＞生成されている固定長メモリ・プールの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象固定長メモリ・プールが生成されていない

ref_mpf
iref_mpf

概要

固定長メモリ・プール詳細情報の参照

C 言語形式

```
ER      ref_mpf ( ID mpfid, T_RMPF *pk_rmpf );
ER      iref_mpf ( ID mpfid, T_RMPF *pk_rmpf );
```

パラメータ

I/O	パラメータ	説明
I	ID　　*mpfid*;	固定長メモリ・プールの ID
O	T_RMPF　*pk_rmpf*;	固定長メモリ・プール詳細情報を格納する領域へのポインタ

【 固定長メモリ・プール詳細情報 T_RMPF の構造 】

```
typedef struct  t_rmpf {
        ID      wtskid;         /* 待ちタスクの有無 */
        UINT    fblkcnt;        /* 空き固定長メモリ・ブロックの総数 */
        ATR     mpfatr;         /* 属性 */
        ID      memid;          /* システム予約領域 */
} T_RMPF;
```

機能

mpfid で指定された固定長メモリ・プールの固定長メモリ・プール詳細情報（待ちタスクの有無，空き固定長メモリ・ブロックの総数など）を *pk_rmpf* で指定された領域に格納します。

備考　固定長メモリ・プール詳細情報 T_RMPF についての詳細は，「16.2.10　固定長メモリ・プール詳細情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *mpfid* ≦ 0x0 - *mpfid* ＞生成されている固定長メモリ・プールの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象固定長メモリ・プールが生成されていない

17.2.10 メモリ・プール管理機能（可変長メモリ・プール）

以下に，RX850V4 がメモリ・プール管理機能（可変長メモリ・プール）として提供しているサービス・コールの一覧を示します。

表 17 － 10　メモリ・プール管理機能（可変長メモリ・プール）

サービス・コール名	機能概要	発行有効範囲
get_mpl	可変長メモリ・ブロックの獲得	タスク
pget_mpl	可変長メモリ・ブロックの獲得（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
ipget_mpl	可変長メモリ・ブロックの獲得（ポーリング）	タスク，制約タスク， 非タスク，初期化ルーチン
tget_mpl	可変長メモリ・ブロックの獲得（タイムアウト付き）	タスク
rel_mpl	可変長メモリ・ブロックの返却	タスク，制約タスク， 非タスク，初期化ルーチン
irel_mpl	可変長メモリ・ブロックの返却	タスク，制約タスク， 非タスク，初期化ルーチン
ref_mpl	可変長メモリ・プール情詳細報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iref_mpl	可変長メモリ・プール詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン

get_mpl

概要

可変長メモリ・ブロックの獲得

C 言語形式

```
ER      get_mpl ( ID mplid, UINT blksz, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mplid*;	可変長メモリ・プールの ID
I	UINT　*blksz*;	可変長メモリ・ブロックの要求サイズ（単位：バイト）
O	VP　**p_blk*;	可変長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mplid で指定された可変長メモリ・プールから *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態から WAITING 状態（可変長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお，可変長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われ，可変長メモリ・ブロック獲得待ち状態から READY 状態へと遷移します。

可変長メモリ・ブロック獲得待ち状態の解除操作	エラー・コード
rel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された	E_OK
irel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された	E_OK
rel_wai の発行により，可変長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，可変長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI

備考 1　RX850V4 では，可変長メモリ・ブロックの獲得処理を“4 の整数倍値”を単位として行います。したがって，*blksz* に 4 の整数倍値以外の値が指定された場合には，4 の整数倍値に繰り上げられます。

備考 2　自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 3　rel_wai，または irel_wai の発行により可変長メモリ・ブロック獲得待ち状態を解除された場合，*p_blk* で指定された領域の内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_PAR	-17	要求サイズの指定が不正である - *blksz* = 0x0 - *blksz* ＞ 0x7fffffff
E_ID	-18	ID の指定が不正である - *mplid* ≦ 0x0 - *mplid* ＞生成されている可変長メモリ・プールの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象可変長メモリ・プールが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，可変長メモリ・ブロック獲得待ち状態を強制的に解除された

pget_mpl
ipget_mpl

概要

可変長メモリ・ブロックの獲得（ポーリング）

C 言語形式

```
ER      pget_mpl ( ID mplid, UINT blksz, VP *p_blk );
ER      ipget_mpl ( ID mplid, UINT blksz, VP *p_blk );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mplid*;	可変長メモリ・プールの ID
I	UINT　*blksz*;	可変長メモリ・ブロックの要求サイズ（単位：バイト）
O	VP　**p_blk*;	可変長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ

機能

mplid で指定された可変長メモリ・プールから *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，戻り値として E_TMOUT を返します。

備考 1　RX850V4 では，可変長メモリ・ブロックの獲得処理を“4 の整数倍値”を単位として行います。したがって，*blksz* に 4 の整数倍値以外の値が指定された場合には，4 の整数倍値に繰り上げられます。

備考 2　本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合，*p_blk* で指定された領域の内容は不定となります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	要求サイズの指定が不正である - *blksz* = 0x0 - *blksz* ＞ 0x7fffffff
E_ID	-18	ID の指定が不正である - *mplid* ≦ 0x0 - *mplid* ＞生成されている可変長メモリ・プールの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象可変長メモリ・プールが生成されていない

マクロ	数値	意味
E_TMOUT	-50	対象可変長メモリ・プールに要求サイズ分の連続する空き領域が存在しない

tget_mpl

概要

可変長メモリ・ブロックの獲得（タイムアウト付き）

C 言語形式

```
ER      tget_mpl ( ID mplid, UINT blksz, VP *p_blk, TMO tmout );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mplid*;	可変長メモリ・プールの ID
I	UINT　*blksz*;	可変長メモリ・ブロックの要求サイズ（単位：バイト）
O	VP　**p_blk*;	可変長メモリ・ブロックの先頭アドレスを格納する領域へのポインタ
I	TMO　*tmout*;	待ち時間（単位：ミリ秒） TMO_FEVR：　永久待ち TMO_POL：　ポーリング 数値：　待ち時間

機能

mplid で指定された可変長メモリ・プールから *blksz* で指定されたサイズの可変長メモリ・ブロックを獲得し，その先頭アドレスを *p_blk* で指定された領域に格納します。

ただし，本サービス・コールを発行した際，対象可変長メモリ・プールから可変長メモリ・ブロックを獲得することができなかった（要求サイズ分の連続する空き領域が存在しなかった）場合には，可変長メモリ・ブロックの獲得は行わず，自タスクを対象可変長メモリ・プールの待ちキューにキューイングしたのち，RUNNING 状態からタイムアウト付きの WAITING 状態（可変長メモリ・ブロック獲得待ち状態）へと遷移させます。

なお，可変長メモリ・ブロック獲得待ち状態の解除は，以下の場合に行われ，可変長メモリ・ブロック獲得待ち状態から READY 状態へと遷移します。

可変長メモリ・ブロック獲得待ち状態の解除操作	エラー・コード
rel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された	E_OK
irel_mpl の発行により，対象可変長メモリ・プールに要求サイズを満足する可変長メモリ・ブロックが返却された	E_OK
rel_wai の発行により，可変長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
irel_wai の発行により，可変長メモリ・ブロック獲得待ち状態を強制的に解除された	E_RLWAI
tmout で指定された待ち時間が経過した	E_TMOUT

備考 1　RX850V4 では，可変長メモリ・ブロックの獲得処理を“4 の整数倍値”を単位として行います。したがって，*blksz* に 4 の整数倍値以外の値が指定された場合には，4 の整数倍値に繰り上げられます。

備考 2　自タスクを対象可変長メモリ・プールの待ちキューにキューイングする際のキューイング方式は，コンフィギュレーション時に定義された順（FIFO 順，優先度順）に行われます。

備考 3　rel_wai，または irel_wai の発行，または待ち時間の経過により可変長メモリ・ブロック獲得待ち状態を解除された場合，*p_blk* で指定された領域の内容は不定となります。

備考 4　待ち時間 *tmout* に TMO_FEVR が指定された際には“get_mpl と同等の処理”を，TMO_POL が指定された際には“pget_mpl，ipget_mpl と同等の処理”を実行します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_NOSPT	-9	制約タスクから本サービス・コールを発行した
E_PAR	-17	パラメータの指定が不正である - *blksz* = 0x0 - *blksz* ＞ 0x7fffffff - *tmout* ＜ TMO_FEVR
E_ID	-18	ID の指定が不正である - *mplid* ≦ 0x0 - *mplid* ＞生成されている可変長メモリ・プールの最大 ID
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した
E_NOEXS	-42	対象可変長メモリ・プールが生成されていない
E_RLWAI	-49	rel_wai，または irel_wai の発行により，可変長メモリ・ブロック獲得待ち状態を強制的に解除された
E_TMOUT	-50	待ち時間が経過した

rel_mpl
irel_mpl

概要

可変長メモリ・ブロックの返却

C 言語形式

```
ER      rel_mpl ( ID mplid, VP blk );
ER      irel_mpl ( ID mplid, VP blk );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mplid*;	可変長メモリ・プールの ID
I	VP　*blk*;	可変長メモリ・ブロックの先頭アドレス

機能

mplid で指定された可変長メモリ・プールに *blk* で指定された可変長メモリ・ブロックを返却します。

可変長メモリ・ブロックを返却したあと，対象可変長メモリ・プールの待ちキューにキューイングされているタスクをキューの先頭から調べていき，待ちタスクが要求するサイズのメモリを割り当てられる場合はメモリを割り当てます。この動作を待ちキューにタスクがなくなるか，メモリが割り当てられなくなるまで繰り返します。これにより，メモリを獲得できたタスクは，待ちキューから外れ，WAITING 状態（可変長メモリ・ブロック獲得待ち状態）から READY 状態へ，または WAITING-SUSPENDED 状態から SUSPENDED 状態へと遷移します。

備考 1　RX850V4 では，可変長メモリ・ブロックを返却する際，メモリ・クリア処理を行っていません。したがって，返却された可変長メモリ・ブロックの内容は不定となります。

備考 2　可変長メモリ・ブロックを返却する際は，必ず獲得した可変長メモリ・プールに対して本サービス・コールを発行してください。異なる可変長メモリ・プールに対して本サービス・コールを発行してもエラーにはなりませんが，以後の動作は保証されません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *mplid* ≦ 0x0 - *mplid* ＞生成されている可変長メモリ・プールの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象可変長メモリ・プールが生成されていない

ref_mpl
iref_mpl

概要

可変長メモリ・プール詳細情報の参照

C 言語形式

```
ER      ref_mpl ( ID mplid, T_RMPL *pk_rmpl );
ER      iref_mpl ( ID mplid, T_RMPL *pk_rmpl );
```

パラメータ

I/O	パラメータ	説明
I	ID　*mplid*;	可変長メモリ・プールの ID
O	T_RMPL　**pk_rmpl*;	可変長メモリ・プール詳細情報を格納する領域へのポインタ

【 可変長メモリ・プール詳細情報 T_RMPL の構造 】

```
typedef struct  t_rmpl {
        ID      wtskid;         /* 待ちタスクの有無 */
        SIZE    fmplsz;         /* 空き可変長メモリ・ブロックの合計サイズ */
        UINT    fblksz;         /* 空き可変長メモリ・ブロックの最大サイズ */
        ATR     mplatr;         /* 属性 */
        ID      memid;          /* システム予約領域 */
} T_RMPL;
```

機能

mplid で指定された可変長メモリ・プールの可変長メモリ・プール詳細情報（待ちタスクの有無，空き可変長メモリ・ブロックの合計サイズなど）を *pk_rmpl* で指定された領域に格納します。

備考　可変長メモリ・プール詳細情報 T_RMPL についての詳細は，「16.2.11　可変長メモリ・プール詳細情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *mplid* ≦ 0x0 - *mplid* ＞生成されている可変長メモリ・プールの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象可変長メモリ・プールが生成されていない

17.2.11 時間管理機能

以下に，RX850V4 が時間管理機能として提供しているサービス・コールの一覧を示します。

表 17 － 11　時間管理機能

サービス・コール名	機能概要	発行有効範囲
set_tim	システム時刻の設定	タスク，制約タスク， 非タスク，初期化ルーチン
iset_tim	システム時刻の設定	タスク，制約タスク， 非タスク，初期化ルーチン
get_tim	システム時刻の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iget_tim	システム時刻の参照	タスク，制約タスク， 非タスク，初期化ルーチン
sta_cyc	周期ハンドラの動作開始	タスク，制約タスク， 非タスク，初期化ルーチン
ista_cyc	周期ハンドラの動作開始	タスク，制約タスク， 非タスク，初期化ルーチン
stp_cyc	周期ハンドラの動作停止	タスク，制約タスク， 非タスク，初期化ルーチン
istp_cyc	周期ハンドラの動作停止	タスク，制約タスク， 非タスク，初期化ルーチン
ref_cyc	周期ハンドラ詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン
iref_cyc	周期ハンドラ詳細情報の参照	タスク，制約タスク， 非タスク，初期化ルーチン

set_tim
iset_tim

概要

システム時刻の設定

C 言語形式

```
ER      set_tim ( SYSTIM *p_systim );
ER      iset_tim ( SYSTIM *p_systim );
```

パラメータ

I/O	パラメータ	説明
I	SYSTIM　*p_systim;	システム時刻情報を格納した領域へのポインタ

【 システム時刻情報 SYSTIM の構造 】

```
typedef struct  t_systim {
        UW      ltime;          /* システム時刻（下位 32 ビット）*/
        UH      utime;          /* システム時刻（上位 16 ビット）*/
} SYSTIM;
```

機能

RX850V4 のシステム時刻（単位：ミリ秒）を *p_systim* で指定された時間に変更します。

備考　　システム時刻情報 SYSTIM についての詳細は，「16.2.12　システム時刻情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した

get_tim
iget_tim

概要

システム時刻の参照

C 言語形式

```
ER      get_tim ( SYSTIM *p_systim );
ER      iget_tim ( SYSTIM *p_systim );
```

パラメータ

I/O	パラメータ	説明
O	SYSTIM　*p_systim;	システム時刻情報を格納する領域へのポインタ

【 システム時刻情報 SYSTIM の構造 】

```
typedef struct  t_systim {
        UW      ltime;          /* システム時刻（下位 32 ビット）*/
        UH      utime;          /* システム時刻（上位 16 ビット）*/
} SYSTIM;
```

機能

RX850V4 のシステム時刻（単位：ミリ秒）を *p_systim* で指定された領域に格納します。

備考 1　RX850V4 では，システム時刻として表現できない数値（48 ビット幅からオーバフローした数値）については無視しています。

備考 2　システム時刻情報 SYSTIM についての詳細は，「16.2.12　システム時刻情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した

sta_cyc
ista_cyc

概要

周期ハンドラの動作開始

C 言語形式

```
ER      sta_cyc ( ID cycid );
ER      ista_cyc ( ID cycid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*cycid*;	周期ハンドラの ID

機能

cycid で指定された周期ハンドラの動作状態を停止状態（STP 状態）から動作状態（STA 状態）へと遷移させます。これにより，対象周期ハンドラは，RX850V4 の起動対象となります。

なお，本サービス・コールの発行から 1 回目の起動要求が発行されるまでの相対時間間隔は，コンフィギュレーション時に対象周期ハンドラに対して TA_PHS 属性を指定しているか否かにより異なります。

- “指定あり”の場合
  コンフィギュレーション時に定義した起動位相（初期起動位相 *cycphs*，起動周期 *cyctim*）で対象周期ハンドラに対する起動タイミング設定処理が行われます。
  ただし，対象周期ハンドラの動作状態が開始状態の場合には，本サービス・コールを発行しても何も処理は行わず，エラーとしても扱いません。

- “指定なし”の場合
  本サービス・コールの発行を基準点とした起動位相（起動周期 *cyctim*）で対象周期ハンドラに対する起動タイミング設定処理が行われます。
  なお，起動タイミング設定処理については，対象周期ハンドラの動作状態に関係なく実行されます。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *cycid* ≦ 0x0 - *cycid* ＞生成されている周期ハンドラの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象周期ハンドラが生成されていない

stp_cyc
istp_cyc

概要

周期ハンドラの動作停止

C 言語形式

```
ER      stp_cyc ( ID cycid );
ER      istp_cyc ( ID cycid );
```

パラメータ

I/O	パラメータ	説明
I	ID　*cycid*;	周期ハンドラの ID

機能

cycid で指定された周期ハンドラの動作状態を動作状態（STA 状態）から停止状態（STP 状態）へと遷移させます。これにより，本サービス・コールの発行から sta_cyc，または ista_cyc が発行されるまでの間，対象周期ハンドラは，RX850V4 の起動対象から除外されます。

備考　本サービス・コールでは，停止要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，対象周期ハンドラの動作状態が停止状態（STP 状態）へと遷移していた場合には，何も処理は行わず，エラーとしても扱いません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *cycid* ≦ 0x0 - *cycid* ＞生成されている周期ハンドラの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象周期ハンドラが生成されていない

ref_cyc
iref_cyc

概要

周期ハンドラ詳細情報の参照

C 言語形式

```
ER      ref_cyc ( ID cycid, T_RCYC *pk_rcyc );
ER      iref_cyc ( ID cycid, T_RCYC *pk_rcyc );
```

パラメータ

I/O	パラメータ	説明
I	ID　　*cycid*;	周期ハンドラの ID
O	T_RCYC　*pk_rcyc;	周期ハンドラ詳細情報を格納する領域へのポインタ

【 周期ハンドラ詳細情報 T_RCYC の構造 】

```
typedef struct  t_rcyc {
        STAT    cycstat;        /* 現在状態 */
        RELTIM  lefttim;        /* 残り時間 */
        ATR     cycatr;         /* 属性 */
        RELTIM  cyctim;         /* 起動周期 */
        RELTIM  cycphs;         /* 初期起動位相 */
} T_RCYC;
```

機能

cycid で指定された周期ハンドラの周期ハンドラ詳細情報（現在状態，残り時間など）を *pk_rcyc* で指定された領域に格納します。

備考　　周期ハンドラ詳細情報 T_RCYC についての詳細は，「16.2.13　周期ハンドラ詳細情報」を参照してください。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_ID	-18	ID の指定が不正である - *cycid* ≦ 0x0 - *cycid* ＞生成されている周期ハンドラの最大 ID
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した
E_NOEXS	-42	対象周期ハンドラが生成されていない

17.2.12 システム状態管理機能

以下に，RX850V4 がシステム状態管理機能として提供しているサービス・コールの一覧を示します。

表 17 － 12　システム状態管理機能

サービス・コール名	機能概要	発行有効範囲
rot_rdq	レディ・キューの回転	タスク，制約タスク， 非タスク，初期化ルーチン
irot_rdq	レディ・キューの回転	タスク，制約タスク， 非タスク，初期化ルーチン
vsta_sch	スケジューラの強制起動	タスク，制約タスク
get_tid	RUNNING 状態のタスクの参照	タスク，制約タスク， 非タスク，初期化ルーチン
iget_tid	RUNNING 状態のタスクの参照	タスク，制約タスク， 非タスク，初期化ルーチン
loc_cpu	CPU ロック状態への移行	タスク，制約タスク， 非タスク
iloc_cpu	CPU ロック状態への移行	タスク，制約タスク， 非タスク
unl_cpu	CPU ロック状態の解除	タスク，制約タスク， 非タスク
iunl_cpu	CPU ロック状態の解除	タスク，制約タスク， 非タスク
sns_loc	CPU ロック状態の参照	タスク，制約タスク， 非タスク，初期化ルーチン
dis_dsp	ディスパッチ禁止状態への移行	タスク，制約タスク
ena_dsp	ディスパッチ禁止状態の解除	タスク，制約タスク
sns_dsp	ディスパッチ禁止状態の参照	タスク，制約タスク， 非タスク，初期化ルーチン
sns_ctx	コンテキスト種別の参照	タスク，制約タスク， 非タスク，初期化ルーチン
sns_dpn	ディスパッチ保留状態の参照	タスク，制約タスク， 非タスク，初期化ルーチン

rot_rdq
irot_rdq

概要

レディ・キューの回転

C 言語形式

```
ER      rot_rdq ( PRI tskpri );
ER      irot_rdq ( PRI tskpri );
```

パラメータ

I/O	パラメータ	説明
I	PRI　*tskpri*;	タスクの優先度 TPRI_SELF：　自タスクの現在優先度 数値：　　タスクの優先度

機能

tskpri で指定された優先度に対応したレディ・キューの先頭タスクを最後尾につなぎかえ，タスクの実行順序を明示的に変更します。

備考 1　本サービス・コールでは，レディ・キューの対象優先度にタスクが 1 つもキューイングされていなかった場合，および，対象優先度の先頭タスクが制約タスクであった場合には，何も処理は行わず，エラーとしても扱いません。

備考 2　本サービス・コールを周期ハンドラなどから一定周期で発行することにより，ラウンドロビン・スケジューリングを実現することができます。

備考 3　RX850V4　　におけるレディ・キューは，優先度をキーとしたハッシュ・テーブルであり，実行可能な状態（RUNNING 状態，または READY 状態）へと遷移したタスクの管理ブロック（タスク管理ブロック）が FIFO 順でキューイングされます。このため，スケジューラは，起動された際にレディ・キューの優先度高位から検出処理を実行し，キューイングされているタスクを検出した場合には，該当優先度の先頭タスクに CPU の利用権を与えることにより，RX850V4 のスケジューリング方式（優先度方式，FCFS 方式）を実現しています。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	優先度の指定が不正である - *tskpri* ＜ 0x0 - *tskpri* ＞優先度範囲 - 非タスクから本サービス・コールを発行した際，*tskpri* に TPRI_SELF を指定した
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した

vsta_sch

概要

スケジューラの強制起動

C 言語形式

```
ER      vsta_sch ( void );
```

パラメータ

なし

機能

RX850V4 のスケジューラを明示的に強制起動します。このため，スケジューリング要求が保留されていた際には，“タスクの切り替え”が発生する場合があります。

備考　RX850V4 では，本サービス・コールを“コンフィギュレーション時にプリエンプトの受け付け状態を禁止状態と定義したタスクから RX850V4 のスケジューラを起動するための機能”として提供しています。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した - ディスパッチ禁止状態から本サービス・コールを発行した

get_tid
iget_tid

概要

RUNNING 状態のタスクの参照

C 言語形式

```
ER      get_tid ( ID *p_tskid );
ER      iget_tid ( ID *p_tskid );
```

パラメータ

I/O	パラメータ	説明
O	ID　*p_tskid;	ID を格納する領域へのポインタ

機能

RUNNING 状態のタスクの ID を *p_tskid* で指定された領域に格納します。

備考　本サービス・コールでは，RUNNING 状態へと遷移しているタスクが存在しなかった場合（IDLE 状態）には，*p_tskid* で指定された領域に TSK_NONE を格納しています。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した

loc_cpu
iloc_cpu

概要

CPU ロック状態への移行

C 言語形式

```
ER      loc_cpu ( void );
ER      iloc_cpu ( void );
```

パラメータ

なし

機能

システム状態種別を非 CPU ロック状態から CPU ロック状態へと変更します。これにより，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間，“マスカブル割り込みの受け付け処理”，および，一部のサービス・コールを除き，サービス・コールの発行が禁止されます。

発行可能なサービス・コール	機能概要
sns_tex	タスク例外処理禁止状態の参照
loc_cpu，iloc_cpu	CPU ロック状態への移行
unl_cpu，iunl_cpu	CPU ロック状態の解除
sns_loc	CPU ロック状態の参照
sns_dsp	ディスパッチ禁止状態の参照
sns_ctx	コンテキスト種別の参照
sns_dpn	ディスパッチ保留状態の参照

なお，RX850V4 では，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間にマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を unl_cpu，または iunl_cpu が発行されるまで遅延しています。

備考 1　本サービス・コールの内部処理（割り込みマスク設定処理，割り込みマスク獲得処理）は，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。なお，サンプル・ソース・ファイルでは，割り込みマスク設定処理，割り込みマスク獲得処理として“割り込み制御レジスタ xxICn，または，割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に対する操作”を行っています。

<rx_sample>¥src¥usr_getmsk.c，usr_intmsk.c

備考 2　本サービス・コールの発行により変更した CPU ロック状態の解除は，本サービス・コールを発行した処理プログラムが終了する以前に行う必要があります。

備考 3　本サービス・コールでは，ロック要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別が CPU ロック状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 4　RX850V4 では，一定周期で発生する基本クロック用タイマ割り込みを利用して時間管理機能を実現しています。このため，本サービス・コールの発行により，該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には，時間管理機能が正常に動作しなくなる場合があります。

備考 5　RX850V4 では，本サービス・コールの発行から unl_cpu，または iunl_cpu が発行されるまでの間に“本サービス・コール，または sns_*xxx* 以外のサービス・コール”を発行した場合には，戻り値として E_CTX を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

unl_cpu
iunl_cpu

概要

CPU ロック状態の解除

C 言語形式

```
ER      unl_cpu ( void );
ER      iunl_cpu ( void );
```

パラメータ

なし

機能

システム状態種別を CPU ロック状態から非 CPU ロック状態へと変更します。これにより，loc_cpu，または iloc_cpu の発行により抑制（禁止）されていた“マスカブル割り込みの受け付け処理”，および，サービス・コールの発行が許可されます。

なお，RX850V4 では，loc_cpu，または iloc_cpu の発行から本サービス・コールが発行されるまでの間にマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を本サービス・コールが発行されるまで遅延しています。

備考 1　本サービス・コールの内部処理（割り込みマスク設定処理）は，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。
なお，サンプル・ソース・ファイルでは，割り込みマスク設定処理として“割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に対する操作”を行っています。

<rx_sample>¥src¥usr_setmsk.c

備考 2　本サービス・コールでは，解除要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別が非 CPU ロック状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 3　本サービス・コールでは，dis_dsp の発行により変更されたディスパッチ禁止状態の解除処理は行われません。したがって，CPU ロック状態以前のシステム状態種別がディスパッチ禁止状態であった場合には，本サービス・コール発行後のシステム状態種別は，ディスパッチ禁止状態となります。

備考 4　本サービス・コールでは，dis_int の発行により禁止されたマスカブル割り込みの受け付けの許可処理は行われません。したがって，CPU ロック状態以前のマスカブル割り込みの受け付けが禁止状態であった場合には，本サービス・コール発行後のマスカブル割り込みの受け付けは，禁止状態となります。

備考 5　RX850V4 では，loc_cpu，または iloc_cpu の発行から本サービス・コールが発行されるまでの間に“loc_cpu，iloc_cpu，または sns_*xxx* 以外のサービス・コール”を発行した場合には，戻り値として E_CTX を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了

sns_loc

概要

CPU ロック状態の参照

C 言語形式

```
BOOL    sns_loc ( void );
```

パラメータ

なし

機能

本サービス・コールを発行した際のシステム状態種別（CPU ロック状態，非 CPU ロック状態）を獲得します。
なお，本サービス・コールの発行により獲得したシステム状態種別については，戻り値として返します。

戻り値

マクロ	数値	意味
TRUE	1	正常終了（CPU ロック状態）
FALSE	0	正常終了（非 CPU ロック状態）

dis_dsp

概要

ディスパッチ禁止状態への移行

C 言語形式

```
ER      dis_dsp ( void );
```

パラメータ

なし

機能

システム状態種別をディスパッチ許可状態からディスパッチ禁止状態へと変更します。これにより，本サービス・コールの発行から ena_dsp が発行されるまでの間，“タスクのディスパッチ処理”が抑制（禁止）されます。

なお，RX850V4 では，本サービス・コールの発行から ena_dsp が発行されるまでの間に“タスクのディスパッチ処理”を伴うサービス・コール（chg_pri，sig_sem など）が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際の“タスクのディスパッチ処理”は，ena_dsp が発行されるまで遅延され，一括して行うようにしています。

備考 1　本サービス・コールの発行により変更したディスパッチ禁止状態の解除は，本サービス・コールを発行したタスクが DORMANT 状態へと遷移する以前に行う必要があります。

備考 2　本サービス・コールでは，禁止要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別がディスパッチ禁止状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 3　RX850V4 では，本サービス・コールの発行から ena_dsp が発行されるまでの間に“自タスクを状態遷移させる可能性のあるサービス・コール（wai_sem，wai_flg など）”を発行した場合には，要求条件の即時成立／不成立を問わず，戻り値として E_CTX を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した

ena_dsp

概要

ディスパッチ禁止状態の解除

C 言語形式

```
ER      ena_dsp ( void );
```

パラメータ

なし

機能

システム状態種別をディスパッチ禁止状態からディスパッチ許可状態へと変更します。これにより，dis_dsp の発行により抑制（禁止）されていた“タスクのディスパッチ処理”が許可されます。

なお，RX850V4 では，dis_dsp の発行から本サービス・コールが発行されるまでの間に“タスクのディスパッチ処理”を伴うサービス・コール（chg_pri，sig_sem など）が発行された場合には，キュー操作，カウンタ操作などといった処理を行うだけであり，実際の“タスクのディスパッチ処理”は，本サービス・コールが発行されるまで遅延され，一括して行うようにしています。

備考 1　本サービス・コールでは，許可要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，システム状態種別がディスパッチ許可状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 2　RX850V4 では，dis_dsp の発行から本サービス・コールが発行されるまでの間に“自タスクを状態遷移させる可能性のあるサービス・コール（wai_sem，wai_flg など）”を発行した場合には，要求条件の即時成立／不成立を問わず，戻り値として E_CTX を返します。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	コンテキスト・エラー - 非タスクから本サービス・コールを発行した - CPU ロック状態から本サービス・コールを発行した

sns_dsp

概要

ディスパッチ禁止状態の参照

C 言語形式

```
BOOL      sns_dsp ( void );
```

パラメータ

なし

機能

本サービス・コールを発行した際のシステム状態種別（ディスパッチ禁止状態，ディスパッチ許可状態）を獲得します。
なお，本サービス・コールの発行により獲得したシステム状態種別については，戻り値として返します。

戻り値

マクロ	数値	意味
TRUE	1	正常終了（ディスパッチ禁止状態）
FALSE	0	正常終了（ディスパッチ許可状態）

sns_ctx

概要

コンテキスト種別の参照

C 言語形式

```
BOOL    sns_ctx ( void );
```

パラメータ

なし

機能

本サービス・コールを発行した処理プログラムのコンテキスト種別（非タスク・コンテキスト，タスク・コンテキスト）を獲得します。
なお，本サービス・コールの発行により獲得したコンテキスト種別については，戻り値として返します。

戻り値

マクロ	数値	意味
TRUE	1	正常終了（非タスク・コンテキスト）
FALSE	0	正常終了（タスク・コンテキスト）

sns_dpn

概要

ディスパッチ保留状態の参照

C 言語形式

```
BOOL    sns_dpn ( void );
```

パラメータ

なし

機能

本サービス・コールを発行した際のシステム状態種別（ディスパッチ保留状態であるか否か）を獲得します。
なお，本サービス・コールの発行により獲得したシステム状態種別については，戻り値として返します。

戻り値

マクロ	数値	意味
TRUE	1	正常終了（ディスパッチ保留状態）
FALSE	0	正常終了（非ディスパッチ保留状態）

17.2.13 割り込み管理機能

以下に，RX850V4 が割り込み管理機能として提供しているサービス・コールの一覧を示します。

表 17 － 13　割り込み管理機能

サービス・コール名	機能概要	発行有効範囲
dis_int	マスカブル割り込みの受け付け禁止	タスク，制約タスク， 非タスク，初期化ルーチン
ena_int	マスカブル割り込みの受け付け許可	タスク，制約タスク， 非タスク，初期化ルーチン
chg_ims	割り込みマスク・パターンの変更	タスク，制約タスク， 非タスク，初期化ルーチン
ichg_ims	割り込みマスク・パターンの変更	タスク，制約タスク， 非タスク，初期化ルーチン
get_ims	割り込みマスク・パターンの参照	タスク，制約タスク， 非タスク，初期化ルーチン
iget_ims	割り込みマスク・パターンの参照	タスク，制約タスク， 非タスク，初期化ルーチン

dis_int

概要

マスカブル割り込みの受け付け禁止

C 言語形式

```
ER      dis_int ( INTNO intno );
```

パラメータ

I/O	パラメータ	説明
I	INTNO　*intno*;	例外コード番号

機能

intno で指定された例外コード番号に対応したマスカブル割り込みの受け付けを許可状態から禁止状態へと変更します。

なお，RX850V4 では，本サービス・コールの発行から ena_int が発行されるまでの間に *intno* で指定された例外コード番号に対応したマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を ena_int が発行されるまで遅延しています。

備考 1　本サービス・コールは，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。
なお，サンプル・ソース・ファイルでは，マスカブル割り込みの受け付け禁止処理として“割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に対する操作”を行っています。

<rx_sample>¥src¥usr_disint.c

備考 2　本サービス・コールでは，禁止要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，対応するマスカブル割り込みの受け付けが禁止状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

備考 3　RX850V4 では，一定周期で発生する基本クロック用タイマ割り込みを利用して時間管理機能を実現しています。このため，本サービス・コールの発行により，該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には，時間管理機能が正常に動作しなくなる場合があります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	例外コード番号の指定が不正である
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した

ena_int

概要

マスカブル割り込みの受け付け許可

C 言語形式

```
ER      ena_int ( INTNO intno );
```

パラメータ

I/O	パラメータ	説明
I	INTNO　*intno*;	例外コード番号

機能

intno で指定された例外コード番号に対応したマスカブル割り込みの受け付けを禁止状態から許可状態へと変更します。

なお，RX850V4 では，dis_int の発行から本サービス・コールが発行されるまでの間に *intno* で指定された例外コード番号に対応したマスカブル割り込みが発生した場合には，該当割り込み処理（割り込みハンドラ）への移行を本サービス・コールが発行されるまで遅延しています。

備考 1　本サービス・コールは，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。
なお，サンプル・ソース・ファイルでは，マスカブル割り込みの受け付け許可処理として“割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn に対する操作”を行っています。

<rx_sample>¥src¥usr_enaint.c

備考 2　本サービス・コールでは，許可要求のキューイングが行われません。このため，すでに本サービス・コールが発行され，対応するマスカブル割り込みの受け付けが許可状態へと変更されていた場合には，何も処理は行わず，エラーとしても扱いません。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_PAR	-17	例外コード番号の指定が不正である
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した

chg_ims
ichg_ims

概要

割り込みマスク・パターンの変更

C 言語形式

```
ER      chg_ims ( UH *p_intms );
ER      ichg_ims ( UH *p_intms );
```

パラメータ

I/O	パラメータ	説明
I	UH　*p_intms;	割り込みマスク・パターンを格納した領域へのポインタ

機能

CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または，割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）を *p_intms* で指定された状態へと変更します。

以下に，*p_intms* で指定された領域に設定する値（割り込みマスク・フラグ）の意味を示します。

0：　マスカブル割り込みの受け付けは許可状態
1：　マスカブル割り込みの受け付けは禁止状態

備考 1　本サービス・コールの内部処理（割り込みマスク設定処理）は，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。

<rx_sample>¥src¥usr_setmsk.c

備考 2　RX850V4 では，一定周期で発生する基本クロック用タイマ割り込みを利用して時間管理機能を実現しています。このため，本サービス・コールの発行により，該当基本クロック用タイマ割り込みの受け付けを禁止状態へと変更した際には，時間管理機能が正常に動作しなくなる場合があります。

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した

get_ims
iget_ims

概要

割り込みマスク・パターンの参照

C 言語形式

```
ER      get_ims ( UH *p_intms );
ER      iget_ims ( UH *p_intms );
```

パラメータ

I/O	パラメータ	説明
O	UH　*p_intms;	割り込みマスク・パターンを格納する領域へのポインタ

機能

CPU の割り込みマスク・パターン（割り込み制御レジスタ xxICn，または , 割り込みマスク・レジスタ IMRm の割り込みマスク・フラグ xxMKn の値）を *p_intms* で指定された領域に格納します。

以下に，*p_intms* で指定された領域に格納される値（割り込みマスク・フラグ）の意味を示します。

0：　マスカブル割り込みの受け付けは許可状態
1：　マスカブル割り込みの受け付けは禁止状態

備考　本サービス・コールの内部処理（割り込みマスク獲得処理）は，ユーザの実行環境に依存した処理であるため，ターゲット依存部として切り出し，サンプル・ソース・ファイルを提供しています。

<rx_sample>¥src¥usr_getmsk.c

戻り値

マクロ	数値	意味
E_OK	0	正常終了
E_CTX	-25	CPU ロック状態から本サービス・コールを発行した

17.2.14 サービス・コール管理機能

以下に，RX850V4 がサービス・コール管理機能として提供しているサービス・コールの一覧を示します。

表 17 － 14　サービス・コール管理機能

サービス・コール名	機能概要	発行有効範囲
cal_svc	拡張サービス・コール・ルーチンの呼び出し	タスク，制約タスク， 非タスク，初期化ルーチン
ical_svc	拡張サービス・コール・ルーチンの呼び出し	タスク，制約タスク， 非タスク，初期化ルーチン

cal_svc
ical_svc

概要

拡張サービス・コール・ルーチンの呼び出し

C 言語形式

```
ER_UINT cal_svc ( FN fncd, VP_INT par1, VP_INT par2, VP_INT par3 );
ER_UINT ical_svc ( FN fncd, VP_INT par1, VP_INT par2, VP_INT par3 );
```

パラメータ

I/O	パラメータ	説明
I	FN　*fncd*;	拡張サービス・コール・ルーチンの機能コード番号
I	VP_INT　*par1*;	拡張サービス・コール・ルーチンへの引き継ぎデータ 1
I	VP_INT　*par2*;	拡張サービス・コール・ルーチンへの引き継ぎデータ 2
I	VP_INT　*par3*;	拡張サービス・コール・ルーチンへの引き継ぎデータ 3

機能

fncd で指定された拡張サービス・コール・ルーチンを呼び出します。

備考　本サービス・コールを使用して呼び出すことができる拡張サービス・コール・ルーチンは，総引き継ぎデータ数が 4 つ未満のものに限られます。

戻り値

マクロ	数値	意味
E_RSFN	-10	機能コード番号の指定が不正である - *fncd* ≦ 0x0 - *fncd* ＞ 0xff - 対象拡張サービス・コール・ルーチンが生成されていない
その他	－	正常終了（拡張サービス・コール・ルーチンの戻り値）

第 18 章　システム・コンフィギュレーション・ファイル

本章では，RX850V4 に提供するデータを保持した情報ファイル（システム情報テーブル・ファイル，システム情報ヘッダ・ファイル）を生成する際に必要となるシステム・コンフィギュレーション・ファイルの記述方法について解説しています。

18.1　概　　要

以下に，システム・コンフィギュレーション・ファイルの表記方法を示します。

- 文字コード
  システム・コンフィギュレーション・ファイルは，ASCII コードで記述します。
  なお，CF850V4 では，英小文字（a ～ z）と英大文字（A ～ Z）の区別が行われます。

  備考　　日本語については，SJIS コードのものをコメント内でのみ記述することができます。

- コメント
  システム・コンフィギュレーション・ファイルでは，/* から */ で囲まれた部分，および，//（連続する 2 個のスラッシュ）から行末までの部分がコメントとして扱われます。

- 数値
  システム・コンフィギュレーション・ファイルでは，数字（0 ～ 9）で始まる単語が数値として扱われます。
  なお，CF850V4 では，数値を以下のように区別しています。

  8 進数：　　0 で始まる単語
  10 進数：　0 以外の数字で始まる単語
  16 進数：　0x，または 0X で始まる単語

  備考　　特に指定のないかぎり，数値として指定可能な範囲は，0x0 ～ 0xffffffff に限られます。

- シンボル名
  システム・コンフィギュレーション・ファイルでは，英字（a ～ z，A ～ Z），または _（アンダースコア）で始まる単語がシンボル名として扱われます。
  また，システム・コンフィギュレーション・ファイルでは，“シンボル名 + オフセット”といった形式の記述も可能ではありますが，オフセットは定数式に限定されます。
  以下に，シンボル名の記述例（良い例，悪い例）を示します。
  なお, CF850V4 では, シンボル名と名前をシステム･コンフィギュレーション･ファイルの文脈から区別しています。

  【良い例】
  ```
  func + 0x80000          // func は関数名
  symbol + 0x90 * 80      // symbol は変数名
  symbol + BASE           // BASE はデータ・マクロ
  ```

  【悪い例】
  ```
  ( func + 0x8000 )       // 先頭文字が不正である
  0x8000 + func           // 先頭文字が不正である
  BASE + func             // データ・マクロ BASE がシンボル名として扱われる
  func * 0x8000           // オフセットの形式になっていない
  ```

  備考　　シンボル名として指定可能な文字数は，オフセット，空白を含めて 4095 文字以内に限られます。

- 名前
  システム・コンフィギュレーション・ファイルでは，英字（a ～ z，A ～ Z），または _（アンダースコア）で始まる単語が名前として扱われます。
  なお, CF850V4 では, 名前とシンボル名をシステム･コンフィギュレーション･ファイルの文脈から区別しています。

  備考　　名前として指定可能な文字数は，255 文字以内に限られます。

- 前処理指令
  システム・コンフィギュレーション・ファイルでは，以下に示した前処理指令の記述が可能です。

#define	#elif	#else	#endif
#if	#ifdef	#ifndef	#include
#undef			

- キーワード
  以下に示した単語は，CF850V4 によってキーワードとして予約されています。このため，これらの単語を指定された用途以外に使用することは禁止されています。

ATT_INI	CLK_INTNO	CPU_TYPE	CRE_CYC
CRE_DTQ	CRE_FLG	CRE_MBX	CRE_MPF
CRE_MPL	CRE_MTX	CRE_SEM	CRE_TSK
DEF_EXC	DEF_INH	DEF_SVC	DEF_TEX
DEF_TIM	INCLUDE	INT_STK	MAX_CYC
MAX_DTQ	MAX_FLG	MAX_INT	MAX_MBX
MAX_MPF	MAX_MPL	MAX_MTX	MAX_PRI
MAX_SEM	MAX_SVC	MAX_TSK	MEM_AREA
NULL	r22	r26	r32
REG_MODE	RX_SERIES	SERVICECALL	SIZE_AUTO
STK_CHK	SYS_STK	TA_ACT	TA_ASM
TA_CLR	TA_DISINT	TA_DISPREEMPT	TA_ENAINT
TA_HLNG	TA_MFIFO	TA_MPRI	TA_OFF
TA_ON	TA_PHS	TA_RSTR	TA_STA
TA_TFIFO	TA_TPRI	TA_WMUL	TA_WSGL
TBIT_FLGPTN	TBIT_TEXPTN	TIC_DENO	TIC_NUME
TKERNEL_MAKER	TKERNEL_PRID	TKERNEL_PRVER	TKERNEL_SPVER
TMAX_ACTCNT	TMAX_MPRI	TMAX_SEMCNT	TMAX_SUSCNT
TMAX_TPRI	TMAX_WUPCNT	TMIN_MPRI	TMIN_TPRI
TSZ_DTQ	TSZ_MBF	TSZ_MPF	TSZ_MPL
TSZ_MPROHD	V850	V850ES	V850E1
V850E2	VATT_IDL	VDEF_RTN	

備考　CF850V4 では，上記に示した単語のほかに，サービス・コール名（act_tsk，slp_tsk，ras_tex など），および，_kernel_ で始まる単語をキーワードとして予約しています。

18.2　コンフィギュレーション情報

システム・コンフィギュレーション・ファイルに記述するデータ（コンフィギュレーション情報）は，以下に示した 3 種類に大別されます。

- 宣言情報
  システム・コンフィギュレーション・ファイル内で使用するデータ・マクロの実体が定義されているヘッダ・ファイル（ヘッダ・ファイル名）に関するデータ
  - ヘッダ・ファイル情報
- システム情報
  RX850V4 が動作するうえで必要となる OS 資源（リアルタイム OS 名，CPU 種別など）に関するデータ
  - RX シリーズ情報
  - 基本情報
  - 初期 FPSR レジスタ情報
  - メモリ領域情報
- 静的 API 情報
  システムで使用する管理オブジェクト（タスク，タスク例外処理ルーチンなど）に関するデータ
  - タスク情報
  - タスク例外処理ルーチン情報
  - セマフォ情報
  - イベントフラグ情報
  - データ・キュー情報
  - メールボックス情報
  - ミューテックス情報
  - 固定長メモリ・プール情報
  - 可変長メモリ・プール情報
  - 周期ハンドラ情報
  - 割り込みハンドラ情報
  - CPU 例外ハンドラ情報
  - 拡張サービス・コール・ルーチン情報
  - 初期化ルーチン情報
  - アイドル・ルーチン情報

18.2.1　記述上の注意点

システム・コンフィギュレーション・ファイルにデータ（コンフィギュレーション情報）を記述する場合，以下の順序で行います。

1）宣言情報の記述
2）システム情報の記述
3）静的 API 情報の記述

なお，システム情報，静的 API 情報については，順不動で記述することができます。
以下に，システム・コンフィギュレーション・ファイルの記述イメージを示します。

図 18 － 1　システム・コンフィギュレーション・ファイルの記述イメージ

```
-- 宣言情報（ヘッダ・ファイル情報）の記述
............
............

-- システム情報（RX シリーズ情報，メモリ領域情報など）の記述
............
............

-- 静的 API 情報（タスク情報，タスク例外処理ルーチン情報など）の記述
............
............
```

18.3　宣言情報

システム・コンフィギュレーション・ファイルに記述する宣言情報の記述形式を以下に示します。
ただし，表記中のゴシック書体は予約語であることを，イタリック書体はユーザが該当する数値，シンボル名，キーワードを記述する部分であることを表しています。

18.3.1　ヘッダ・ファイル情報

ヘッダ・ファイル情報では，

1）*ヘッダ・ファイル名 h_file*

といった項目を個々のヘッダ・ファイルに対して定義します。
なお，ヘッダ・ファイル情報として定義可能な数に制限はありません。
以下に，ヘッダ・ファイル情報の記述形式を示します。

```
INCLUDE ( "h_file" );
```

以下に，ヘッダ・ファイル情報で記述する項目について示します。

1）*ヘッダ・ファイル名 h_file*

h_file に指定したヘッダ・ファイルの宣言を，CF850V4 が出力するシステム情報ヘッダ・ファイルに反映します。この宣言により，CF850V4 が出力したシステム情報ヘッダ・ファイルをインクルードしたファイルから *h_file* 内のマクロ定義を参照することが可能となります。

備考　*h_file* に <sample.h> と指定した際には，ヘッダ・ファイルの定義（インクルード処理）が

```
#include        <sample.h>
```

といった形式で，¥"sample.h¥" と指定した際には

```
#include        "sample.h"
```

といった形式でシステム情報ヘッダ・ファイルに出力されます。

18.4　システム情報

システム・コンフィギュレーション・ファイルに記述するシステム情報の記述形式を以下に示します。
ただし，表記中のゴシック書体は予約語であることを，イタリック書体はユーザが該当する数値，シンボル名，キーワードを記述する部分であることを表しています。
また，“[　]”で囲まれた項目は，省略可能な項目であることを表しています。

18.4.1　RX シリーズ情報

RX シリーズ情報では，

1）*リアルタイム OS 名 rtos_nam*
2）*バージョン番号 rtos_ver*

といった項目を定義します。
なお，RX シリーズ情報として定義可能な数は，1 個に限られます。
以下に，RX シリーズ情報の記述形式を示します。

```
RX_SERIES ( rtos_nam, rtos_ver );
```

以下に，RX シリーズ情報で記述する項目について示します。

1）*リアルタイム OS 名 rtos_nam*

リアルタイム OS の名前（リアルタイム OS 名）を指定します。
なお，*rtos_nam* として指定可能な値は“RX850V4”に限られます。

2）*バージョン番号 rtos_ver*

リアルタイム OS のバージョン番号を指定します。
なお，*rtos_ver* として指定可能な値は“V4*xx*”に限られます。

備考　　*xx* として指定可能な値は 00 ～ 99 に限られます。

18.4.2　基本情報

基本情報では，

1）*CPU 種別 chip_type*
2）*レジスタ・モード種別 reg_mode*
3）*基本クロック周期 tim_base*
4）*基本クロック用タイマ割り込みの例外コード番号 tim_intno*
5）*システム・スタックのサイズ sys_stksz*
6）*スタック・チェックの有無 stkchk*
7）*最大タスク優先度 maxtpri*
8）*最大割り込みハンドラ数 maxint，最大例外コード番号 maxintno*

といった項目を定義します。
なお，基本情報として定義可能な数は，1 個に限られます。
以下に，基本情報の記述形式を示します。

```
[ CPU_TYPE ( chip_type ); ]
[ REG_MODE ( reg_mode  ); ]
[ DEF_TIM ( tim_base ); ]
CLK_INTNO ( tim_intno );
SYS_STK ( sys_stksz );
[ STK_CHK ( stkchk ); ]
[ MAX_PRI ( maxtpri ); ]
MAX_INT ( maxint, maxintno );
```

以下に，基本情報で記述する項目について示します。

1）*CPU 種別 chip_type*

ターゲット・デバイスの CPU 種別を指定します。
なお，*chip_type* として指定可能な値は“V850ES，V850E1，V850E2，V850E2M のいずれか”に限られます。

V850ES：　V850ES コア
V850E1：　V850E1 コア
V850E2：　V850E2 コア
V850E2M：　V850E2M コア

省略時　ターゲット・デバイスの CPU 種別は“V850E1”となります。

2）*レジスタ・モード種別 reg_mode*

システム構築時にリンクするカーネル・ライブラリ librxc.a のレジスタ・モード種別を指定します。
なお，*reg_mode* として指定可能な値は“r22，r26，r32 のいずれか”に限られます。

r22：　22 レジスタ・モード
r26：　26 レジスタ・モード
r32：　32 レジスタ・モード

省略時　システム構築時にリンクするカーネル・ライブラリ librxc.a のレジスタ・モード種別は“r32”となります。

備考　CF850V4 の起動オプションとして -reg*xx* が指定された場合，*reg_mode* の定義は無視され，CF850V4 の起動オプションが有効情報として扱われます。

3）*基本クロック周期 tim_base*

RX850V4 の基本クロック周期（単位：ミリ秒）を指定します。
なお，*tim_base* として指定可能な値は“0x1 ～ 0xffff”に限られます。

省略時　RX850V4 の基本クロック周期は“0x1 ミリ秒”となります。

備考　基本クロック周期とは，RX850V4 が提供する時間管理機能を実現するうえで必要となる基本クロック用タイマ割り込み *tim_intno* の発生周期を意味しています。したがって，RX850V4 が時間管理用に利用するハードウエア（タイマ・コントローラなど）を初期化する際には，*tim_base* で定義された周期で基本クロック用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。

4) *基本クロック用タイマ割り込みの例外コード番号 tim_intno*

RX850V4 が提供する時間管理機能を実現するうえで必要となる基本クロック用タイマ割り込みの例外コード番号を指定します。
なお，*tim_into* として指定可能な値は，デバイス・ファイルで規定されている割り込み要因名，または 16 バイト境界値に限られます。
なお，*tim_intno* に“割り込み要因名”を指定した場合，CF850V4 の起動オプションとして -cpu △ *name* の指定が必須となります。

5) *システム・スタックのサイズ sys_stksz*

システム・スタックのサイズ（単位：バイト）を指定します。
なお，*sys_stksz* として指定可能な値は“0x0 ～ 0x7ffffffc の 4 バイト境界値”に限られます。

備考 1　システム・スタックのサイズを算出する際の計算式については，「18.6　メモリ容量計算式」を参照してください。

備考 2　システム・スタック用メモリ領域は“.rx_memory セクション”から確保されます。

備考 3　実際に確保されるスタック・サイズは，指定したスタック・サイズに“20 ＋ *frmsz*（割り込みハンドラのコンテキスト領域のサイズ）”が加算されたサイズです。*frmsz* については，表 18 － 3 を参照してください。

6) *スタック・チェックの有無 stkchk*

RX850V4 が処理を実行する際にスタックのオーバフロー・チェックを行うか否かを指定します。
なお，*stkchk* として指定可能な値は“TA_ON，TA_OFF のいずれか”に限られます。

TA_ON :　オーバフロー・チェックを行う
TA_OFF :　オーバフロー・チェックを行わない

備考　RX850V4　が処理を実行する際にスタックのオーバフロー・チェックを行うか否かは“オーバフロー・チェックを行わない”となります。

7) *最大タスク優先度 maxtpri*

タスクの優先度範囲（最大タスク優先度 : タスク情報で定義する優先度の最大値，または処理プログラムで chg_pri などのサービス・コールを発行する際に指定する優先度の最大値）を指定します。
なお，*maxtpri* として指定可能な値は“0x1 ～ 0x20”に限られます。

省略時　最大タスク優先度は“0x20”となります。

8) *最大割り込みハンドラ数 maxint，最大例外コード番号 maxintno*

割り込みハンドラの最大登録数，および，対象 CPU が有する例外コード番号の最大値を指定します。
なお, *maxint*として指定可能な値は“0x0～0xff”に, *maxintno*として指定可能な値は“0x80～0x1060”に限られます。

備考　*maxint* として指定する値は，“割り込みハンドラ情報として定義した割り込みハンドラの総数”となります。

18.4.3　初期 FPSR レジスタ情報

初期 FPSR レジスタ情報では，

1）*初期 FPSR レジスタ値 fpsr*

といった項目を個々の処理プログラム（タスク，周期ハンドラ，割り込みハンドラなど）を起動する際に設定される“浮動小数点演算の設定／ステータス・レジスタ FPSR”に対して定義します。

以下に，初期 FPSR レジスタ情報の記述形式を示します。

```
[ DEF_FPSR ( fpsr ); ]
```

以下に，初期 FPSR レジスタ情報で記述する項目について示します。

1）*初期 FPSR レジスタ値 fpsr*

処理プログラムを起動する際に使用する FPSR の値を指定します。
なお，*fpsr* として指定可能な値は“0x0 ～ 0xffffffff”に限られます。
ただし，ハードウエアで規定された値以外を指定した場合，動作は保証されません。具体的な値については，ハードウエアのドキュメントを参照してください。

省略時　初期 FPSR レジスタ値は“0x00020000”となります。

注意　本項目は，V850E2M のデバイスを指定した場合のみ有効になります。それ以外のデバイスを指定した場合，本項目の指定は無視されます。

18.4.4　メモリ領域情報

メモリ領域情報では，

1）*メモリ領域名 sec_nam*

2）*メモリ領域サイズ secsz*

といった項目を個々の管理オブジェクト（タスク・スタック，データ・キュー領域，固定長メモリ・プール，可変長メモリ・プールなど）用メモリ領域に対して定義します。
なお，メモリ領域情報として定義可能な数は，0x0 ～ 0xff 個（1 つのセクションに対して 1 個）に限られます。
以下に，メモリ領域情報の記述形式を示します。

```
MEM_AREA ( sec_nam, secsz );
```

以下に，メモリ領域情報で記述する項目について示します。

1）*メモリ領域名 sec_nam*

管理オブジェクト用に使用するメモリ領域の名前を指定します。
なお，*sec_nam* として指定可能な値は“リンク・ディレクティブ・ファイルで定義されているセクション名 *.sec_nam* から . を除いた名前”に限られます。

備考　リンク・ディレクティブ・ファイルについての詳細は，「CubeSuite　V850　コーディング編」，または「CubeSuite コーディング編（CX コンパイラ）」のユーザーズ・マニュアルを参照してください。

2）*メモリ領域サイズ secsz*

管理オブジェクト用に使用するメモリ領域のサイズ（単位：バイト）を指定します。
なお，*secsz* として指定可能な値は“0x0 ～ 0x7ffffffc の 4 バイト境界値，または SIZE_AUTO”に限られます。

SIZE_AUTO：　基本情報，タスク情報などで定義した管理オブジェクトの合計サイズ

備考　メモリ領域サイズを算出する際の計算式については，「18.6　メモリ容量計算式」を参照してください。

18.5　静的 API 情報

システム・コンフィギュレーション・ファイルに記述する静的 API 情報の記述形式を以下に示します。
ただし，表記中のゴシック書体は予約語であることを，イタリック書体はユーザが該当する数値，シンボル名，キーワードを記述する部分であることを表しています。
また，“[　]”で囲まれた項目は，省略可能な項目であることを表しています。

18.5.1　タスク情報

タスク情報では，

1）*ID tskid*
2）*属性（記述言語，初期起動状態など）tskatr*
3）*拡張情報 exinf*
4）*起動アドレス task*
5）*初期優先度 itskpri*
6）*スタック・サイズ stksz，メモリ領域名 sec_nam*
7）*システム予約領域 stk*

といった項目を個々のタスクに対して定義します。
なお，タスク情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，タスク情報の記述形式を示します。

```
CRE_TSK ( tskid, { tskatr, exinf, task, itskpri, stksz[:sec_nam ], stk } );
```

以下に，タスク情報で記述する項目について示します。

1）*ID tskid*

タスクの ID を指定します。
なお，*tskid* として指定可能な値は“0x1 ～ 0xff，または名前”に限られます。

備考　*tskid* に“名前”を指定した場合，CF850V4 は ID の自動割り付け処理を行います。なお，名前と ID の対応は，下記形式でシステム情報ヘッダ・ファイルに出力されます。

```
#define tskid    数値
```

2）*属性（記述言語，初期起動状態など）tskatr*

タスクの属性（記述言語，初期起動状態など）を指定します。
なお，*tskatr* として指定可能な値は“TA_HLNG，TA_ASM のいずれか，および，TA_ACT，TA_RSTR，TA_DISPREEMPT，および，TA_ENAINT，TA_DISINT のいずれか”に限られます。

【 タスクの記述言語 】
TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

【 タスクの初期起動状態 】
TA_ACT：　READY 状態

【 タスクの種別 】
TA_RSTR：　制約タスク

【 プリエンプトの受け付け状態 】
TA_DISPREEMPT：　DORMANT 状態から READY 状態へと遷移した際には，受け付け禁止状態

【 初期割り込み状態 】
TA_ENAINT：　DORMANT 状態から READY 状態へと遷移した際には，受け付け許可状態
TA_DISINT：　DORMANT 状態から READY 状態へと遷移した際には，受け付け禁止状態

備考 1　TA_ACT の指定を省略した場合，タスクの初期起動状態は“DORMANT 状態”となります。
備考 2　TA_RSTR の指定を省略した場合，タスクの種別は“通常タスク”となります。

備考 3　TA_DISPREEMPT の指定を省略した場合，プリエンプトの受け付け状態は“DORMANT 状態から READY 状態へと遷移した際には，受け付け許可状態”となります。

備考 4　TA_ENAINT，および，TA_DISINT の指定を省略した場合，初期割り込み状態は“DORMANT 状態から READY 状態へと遷移した際には，受け付け許可状態”となります。

3）*拡張情報 exinf*

タスクに引き渡す拡張情報を指定します。
なお，*exinf* として指定可能な値は“0x0 ～ 0xffffffff，またはシンボル名”に限られます。

備考　対象タスクは，*exinf* を関数パラメータと同様に取り扱うことで操作可能となります。

4）*起動アドレス task*

タスクの起動アドレスを指定します。
なお，*task* として指定可能な値は“0x0 ～ 0xfffffffe の 2 バイト境界値，またはシンボル名”に限られます。

5）*初期優先度 itskpri*

タスクの初期優先度を指定します。
なお，*itskpri* として指定可能な値は“0x1 ～基本情報で定義した最大タスク優先度 *maxtpri*”に限られます。

6）*スタック・サイズ stksz，メモリ領域名 sec_nam*

タスク・スタックのサイズ（単位：バイト），および，タスク・スタック用に確保するメモリ領域の名前を指定します。
なお，*stksz* として指定可能な値は“0x0 ～ 0x7ffffffc の 4 バイト境界値”に，*sec_nam* として指定可能な値は“メモリ領域情報で定義したメモリ領域名 *sec_nam*”に限られます。

備考 1　スタック・サイズを算出する際の計算式については，「18.6　メモリ容量計算式」を参照してください。

備考 2　*sec_nam* の指定を省略した場合，タスク・スタック用に確保するメモリ領域は“.rx_memory セクション”となります。

備考 3　実際に確保されるスタック・サイズは，指定したスタック・サイズに *ctxsz*（タスクのコンテキスト領域のサイズ）が加算されたサイズです。*ctxsz* については，表 18 － 4 と表 18 － 5 を参照してください。

7）*システム予約領域 stk*

システム予約領域です。
なお，*stk* として指定可能な値は“NULL”に限られます。

18.5.2　タスク例外処理ルーチン情報

タスク例外処理ルーチン情報では，

1) *ID tskid*

2) *属性（記述言語）texatr*

3) *起動アドレス texrtn*

といった項目を個々のタスク例外処理ルーチンに対して定義します。
なお，タスク例外処理ルーチン情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，タスク例外処理ルーチン情報の記述形式を示します。

```
DEF_TEX ( tskid, { texatr, texrtn } );
```

以下に，タスク例外処理ルーチン情報で記述する項目について示します。

1) *ID tskid*

タスク例外処理ルーチンを登録するタスクの ID を指定します。
なお，*tskid* として指定可能な値は“タスク情報で定義した ID *tskid*”に限られます。

2) *属性（記述言語）texatr*

タスク例外処理ルーチンの属性（記述言語）を指定します。
なお，*texatr* として指定可能な値は“TA_HLNG，TA_ASM のいずれか”に限られます。

TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

3) *起動アドレス texrtn*

タスク例外処理ルーチンの起動アドレスを指定します。
なお，*texrtn* として指定可能な値は“0x0 ～ 0xfffffffe の 2 バイト境界値，またはシンボル名”に限られます。

18.5.3　セマフォ情報

セマフォ情報では，

1）*ID semid*

2）*属性（キューイング方式）sematr*

3）*初期資源数 isemcnt*

4）*最大資源数 maxsem*

といった項目を個々のセマフォに対して定義します。
なお，セマフォ情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，セマフォ情報の記述形式を示します。

```
CRE_SEM ( semid, { sematr, isemcnt, maxsem } );
```

以下に，セマフォ情報で記述する項目について示します。

1）*ID semid*

セマフォの ID を指定します。
なお，*semid* として指定可能な値は“0x1 ～ 0xff，または名前”に限られます。

備考　*semid* に“名前”を指定した場合，CF850V4 は ID の自動割り付け処理を行います。なお，名前と ID の対応は，下記形式でシステム情報ヘッダ・ファイルに出力されます。

```
#define semid    数値
```

2）*属性（キューイング方式）sematr*

セマフォの属性（キューイング方式）を指定します。
なお，*sematr* として指定可能な値は“TA_TFIFO，TA_TPRI のいずれか”に限られます。

TA_TFIFO：　資源の獲得要求を行った順
TA_TPRI：　タスクの優先度順

3）*初期資源数 isemcnt*

セマフォの初期資源数を指定します。
なお，*isemcnt* として指定可能な値は“0x0 ～最大資源数 *maxsem*”に限られます。

4）*最大資源数 maxsem*

セマフォの最大資源数を指定します。
なお，*maxsem* として指定可能な値は“0x1 ～ 0xffff”に限られます。

18.5.4　イベントフラグ情報

イベントフラグ情報では，

1）*ID flgid*

2）*属性（キューイング方式，キューイング可能なタスクの最大数など）flgatr*

3）*初期ビット・パターン iflgptn*

といった項目を個々のイベントフラグに対して定義します。
なお，イベントフラグ情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，イベントフラグ情報の記述形式を示します。

```
CRE_FLG ( flgid, { flgatr, iflgptn } );
```

以下に，イベントフラグ情報で記述する項目について示します。

1）*ID flgid*

イベントフラグの ID を指定します。
なお，*flgid* として指定可能な値は“0x1 ～ 0xff，または名前”に限られます。

備考　*flgid* に“名前”を指定した場合，CF850V4 は ID の自動割り付け処理を行います。なお，名前と ID の対応は，下記形式でシステム情報ヘッダ・ファイルに出力されます。

```
#define flgid    数値
```

2）*属性（キューイング方式，キューイング可能なタスクの最大数など）flgatr*

イベントフラグの属性（キューイング方式，キューイング可能なタスクの最大数など）を指定します。
なお，*flgatr* として指定可能な値は“TA_TFIFO，TA_TPRI のいずれか，TA_WSGL，TA_WMUL のいずれか，TA_CLR”に限られます。

【 タスク・キューイング方式 】
TA_TFIFO：　ビット・パターンのチェックを行った順
TA_TPRI：　タスクの優先度順

【 キューイング可能なタスクの最大数 】
TA_WSGL：　1 個
TA_WMUL：　複数

【 ビット・パターンのクリア 】
TA_CLR：　要求条件が満足した際，ビット・パターンをクリア（0x0 の設定）

備考 1　TA_TFIFO，および，TA_TPRI の指定を省略した場合，タスク・キューイング方式は“ビット・パターンのチェックを行った順”となります。

備考 2　TA_CLR の指定を省略した場合，ビット・パターンのクリアは“要求条件が満足した際，ビット・パターンをクリアしない”となります。

3）*初期ビット・パターン iflgptn*

イベントフラグの初期ビット・パターンを指定します。
なお，*iflgptn* として指定可能な値は“0x0 ～ 0xffffffff”に限られます。

18.5.5　データ・キュー情報

データ・キュー情報では，

1）*ID dtqid*

2）*属性（キューイング方式）dtqatr*

3）*データ数 dtqcnt，メモリ領域名 sec_nam*

4）*システム予約領域 dtq*

といった項目を個々のデータ・キューに対して定義します。
なお，データ・キュー情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，データ・キュー情報の記述形式を示します。

```
CRE_DTQ ( dtqid, { dtqatr, dtqcnt[:sec_nam ], dtq } );
```

以下に，データ・キュー情報で記述する項目について示します。

1）*ID dtqid*

データ・キューの ID を指定します。
なお，*dtqid* として指定可能な値は“0x1 ～ 0xff，または名前”に限られます。

備考　*dtqid* に“名前”を指定した場合，CF850V4 は ID の自動割り付け処理を行います。なお，名前と ID の対応は，下記形式でシステム情報ヘッダ・ファイルに出力されます。

```
#define dtqid    数値
```

2）*属性（キューイング方式）dtqatr*

データ・キューの属性（キューイング方式）を指定します。
なお，*dtqatr* として指定可能な値は“TA_TFIFO，TA_TPRI のいずれか”に限られます。

TA_TFIFO：　データの送信要求を行った順
TA_TPRI：　タスクの優先度順

3）*データ数 dtqcnt，メモリ領域名 sec_nam*

データ・キューのデータ・キュー領域にキューイング可能なデータの最大数，および，データ・キュー領域用に確保するメモリ領域の名前を指定します。
なお，*dtqcnt* として指定可能な値は“0x0 ～ 0xff”に，*sec_nam* として指定可能な値は“メモリ領域情報で定義したメモリ領域名 *sec_nam*”に限られます。

備考　*sec_nam* の指定を省略した場合，データ・キュー領域用に確保するメモリ領域は“.rx_memory セクション”となります。

4）*システム予約領域 dtq*

システム予約領域です。
なお，*dtq* として指定可能な値は“NULL”に限られます。

18.5.6　メールボックス情報

メールボックス情報では，

1）*ID mbxid*

2）*属性（キューイング方式）mbxatr*

3）*最大メッセージ優先度 maxmpri*

4）*システム予約領域 mprihd*

といった項目を個々のメールボックスに対して定義します。
なお，メールボックス情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，メールボックス情報の記述形式を示します。

```
CRE_MBX ( mbxid, { mbxatr, maxmpri, mprihd } );
```

以下に，メールボックス情報で記述する項目について示します。

1）*ID mbxid*

メールボックスの ID を指定します。
なお，*mbxid* として指定可能な値は“0x1 ～ 0xff，または名前”に限られます。

備考　*mbxid* に“名前”を指定した場合，CF850V4 は ID の自動割り付け処理を行います。なお，名前と ID の対応は，下記形式でシステム情報ヘッダ・ファイルに出力されます。

```
#define mbxid    数値
```

2）*属性（キューイング方式）mbxatr*

メールボックスの属性（キューイング方式）を指定します。
なお，*mbxatr* として指定可能な値は“TA_TFIFO，TA_TPRI のいずれか，TA_MFIFO，TA_MPRI のいずれか”に限られます。

【 タスク・キューイング方式 】
TA_TFIFO：　メッセージの受信要求を行った順
TA_TPRI：　タスクの優先度順

【 メッセージ・キューイング方式 】
TA_MFIFO：　メッセージの送信要求を行った順
TA_MPRI：　メッセージの優先度順

3）*最大メッセージ優先度 maxmpri*

本情報で定義されたメールボックスに対して送信されるメッセージの優先度範囲（最大メッセージ優先度）を指定します。
なお，*maxmpri* として指定可能な値は“0x1 ～ 0x7fff”に限られます。

備考　属性 *mbxatr* に TA_MFIFO が指定された場合，本項目の指定は無効となります。

4）*システム予約領域 mprihd*

システム予約領域です。
なお，*mprihd* として指定可能な値は“NULL”に限られます。

18.5.7　ミューテックス情報

ミューテックス情報では，

1）*ID mtxid*

2）*属性（キューイング方式）mtxatr*

3）*システム予約領域 ceilpri*

といった項目を個々のキューテックスに対して定義します。
なお，ミューテックス情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，ミューテックス情報の記述形式を示します。

```
CRE_MTX ( mtxid, { mtxatr, ceilpri } );
```

以下に，ミューテックス情報で記述する項目について示します。

1）*ID mtxid*

ミューテックスの ID を指定します。
なお，*mtxid* として指定可能な値は“0x1 ～ 0xff，または名前”に限られます。

備考　*mtxid* に“名前”を指定した場合，CF850V4 は ID の自動割り付け処理を行います。なお，名前と ID の対応は，下記形式でシステム情報ヘッダ・ファイルに出力されます。

```
#define mtxid    数値
```

2）*属性（キューイング方式）mtxatr*

ミューテックスの属性（キューイング方式）を指定します。
なお，*mtxatr* として指定可能な値は“TA_TFIFO，TA_TPRI のいずれか”に限られます。

TA_TFIFO：　ミューテックスのロック要求を行った順
TA_TPRI：　タスクの優先度順

3）*システム予約領域 ceilpri*

システム予約領域です。
なお，*ceilpri* として指定可能な値は“0x1 ～基本情報で定義した最大タスク優先度 *maxtpri*”に限られます。

18.5.8　固定長メモリ・プール情報

固定長メモリ・プール情報では，

1）*ID mpfid*

2）*属性（キューイング方式）mpfatr*

3）*全メモリ・ブロック数 blkcnt*

4）*ブロック単位サイズ blksz，メモリ領域名 sec_nam*

5）*システム予約領域 mpf*

といった項目を個々の固定長メモリ・プールに対して定義します。
なお，固定長メモリ・プール情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，固定長メモリ・プール情報の記述形式を示します。

```
CRE_MPF ( mpfid, { mpfatr, blkcnt, blksz[:sec_nam ], mpf } );
```

以下に，固定長メモリ・プール情報で記述する項目について示します。

1）*ID mpfid*

固定長メモリ・プールの ID を指定します。
なお，*mpfid* として指定可能な値は“0x1 ～ 0xff，または名前”に限られます。

備考　*mpfid* に“名前”を指定した場合，CF850V4 は ID の自動割り付け処理を行います。なお，名前と ID の対応は，下記形式でシステム情報ヘッダ・ファイルに出力されます。

```
#define mpfid    数値
```

2）*属性（キューイング方式）mpfatr*

固定長メモリ・プールの属性（キューイング方式）を指定します。
なお，*mpfatr* として指定可能な値は“TA_TFIFO，TA_TPRI のいずれか”に限られます。

TA_TFIFO：　固定長メモリ・ブロックの獲得要求を行った順
TA_TPRI：　タスクの優先度順

3）*全メモリ・ブロック数 blkcnt*

固定長メモリ・プールの全メモリ・ブロック数を指定します。
なお，*blkcnt* として指定可能な値は“0x1 ～ 0x7fff”に限られます。

4）*ブロック単位サイズ blksz，メモリ領域名 sec_nam*

1 ブロック当たりのサイズ（単位：バイト），および，固定長メモリ・プール用に確保するメモリ領域の名前を指定します。
なお，*blksz* として指定可能な値は“0x1 ～ 0x7ffffffc の 4 バイト境界値”に，*sec_nam* として指定可能な値は“メモリ領域情報で定義したメモリ領域名 *sec_nam*”に限られます。

備考　*sec_nam* の指定を省略した場合，固定長メモリ・プール用に確保するメモリ領域は“.rx_memory セクション”となります。

5）*システム予約領域 mpf*

システム予約領域です。
なお，*mpf* として指定可能な値は“NULL”に限られます。

18.5.9　可変長メモリ・プール情報

可変長メモリ・プール情報では，

1）*ID mplid*

2）*属性（キューイング方式）mplatr*

3）*プール・サイズ mplsz，メモリ領域名 sec_nam*

4）*システム予約領域 mpl*

といった項目を個々の可変長メモリ・プールに対して定義します。
なお，可変長メモリ・プール情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，可変長メモリ・プール情報の記述形式を示します。

```
CRE_MPL ( mplid, { mplatr, mplsz[:sec_nam ], mpl } );
```

以下に，可変長メモリ・プール情報で記述する項目について示します。

1）*ID mplid*

可変長メモリ・プールの ID を指定します。
なお，*mplid* として指定可能な値は“0x1 ～ 0xff，または名前”に限られます。

備考　*mplid* に“名前”を指定した場合，CF850V4 は ID の自動割り付け処理を行います。なお，名前と ID の対応は，下記形式でシステム情報ヘッダ・ファイルに出力されます。

```
#define mplid    数値
```

2）*属性（キューイング方式）mplatr*

可変長メモリ・プールの属性（キューイング方式）を指定します。
なお，*mplatr* として指定可能な値は“TA_TFIFO，TA_TPRI のいずれか”に限られます。

TA_TFIFO：　可変長メモリ・ブロックの獲得要求を行った順
TA_TPRI：　タスクの優先度順

3）*プール・サイズ mplsz，メモリ領域名 sec_nam*

可変長メモリ・プールのサイズ（単位：バイト），および，可変長メモリ・プール用に確保するメモリ領域の名前を指定します。
なお，*mplsz* として指定可能な値は“0x1 ～ 0x7ffffffc の 4 バイト境界値”に，*sec_nam* として指定可能な値は“メモリ領域情報で定義したメモリ領域名 *sec_nam*”に限られます。

備考　*sec_nam* の指定を省略した場合，可変長メモリ・プール用に確保するメモリ領域は“.rx_memory セクション”となります。

4）*システム予約領域 mpl*

システム予約領域です。
なお，*mpl* として指定可能な値は“NULL”に限られます。

18.5.10 周期ハンドラ情報

周期ハンドラ情報では，

1） *ID cycid*

2） *属性（記述言語，初期起動状態など）cycatr*

3） *拡張情報 exinf*

4） *起動アドレス cychdr*

5） *起動周期 cyctim*

6） *初期起動位相 cycphs*

といった項目を個々の周期ハンドラに対して定義します。
なお，周期ハンドラ情報として定義可能な数は，1 つの ID に対して 1 個に限られます。
以下に，周期ハンドラ情報の記述形式を示します。

```
CRE_CYC ( cycid, { cycatr, exinf, cychdr, cyctim, cycphs } );
```

以下に，周期ハンドラ情報で記述する項目について示します。

1） *ID cycid*

周期ハンドラの ID を指定します。
なお，*cycid* として指定可能な値は“0x1 ～ 0xff，または名前”に限られます。

備考　*cycid* に“名前”を指定した場合，CF850V4 は ID の自動割り付け処理を行います。なお，名前と ID の対応は，下記形式でシステム情報ヘッダ・ファイルに出力されます。

```
#define cycid    数値
```

2） *属性（記述言語，初期起動状態など）cycatr*

周期ハンドラの属性（記述言語，初期起動状態など）を指定します。
なお，*cycatr* として指定可能な値は“TA_HLNG，TA_ASM のいずれか，および，TA_STA，TA_PHS”に限られます。

【 周期ハンドラの記述言語 】
TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

【 周期ハンドラの初期起動状態 】
TA_STA：　動作状態（STA 状態）

【 起動位相保存の有無 】
TA_PHS：　保存

備考 1　TA_STA の指定を省略した場合，周期ハンドラの初期起動状態は“停止状態（STP 状態）”となります。

備考 2　TA_PHS の指定を省略した場合，起動位相保存の有無は“未保存”となります。

3） *拡張情報 exinf*

周期ハンドラに引き渡す拡張情報を指定します。
なお，*exinf* として指定可能な値は“0x0 ～ 0xffffffff，またはシンボル名”に限られます。

備考　対象周期ハンドラは，*exinf* を関数パラメータと同様に取り扱うことで操作可能となります。

4） *起動アドレス cychdr*

周期ハンドラの起動アドレスを指定します。
なお，*cychdr* として指定可能な値は“0x0 ～ 0xfffffffe の 2 バイト境界値，またはシンボル名”に限られます。

5） *起動周期 cyctim*

周期ハンドラの起動周期（単位：ミリ秒）を指定します。
なお，*cyctim* として指定可能な値は“0x1 ～ 0x7fffffff”に限られます。

備考　*cyctim* に基本情報で定義した基本クロック周期の整数倍値以外の値を指定した場合，CF850V4 は整数倍値が指定されていたものとして処理を行います。

6) *初期起動位相 cycphs*

周期ハンドラの初期起動位相（単位：ミリ秒）を指定します。
なお，*cycphs* として指定可能な値は“0x1 ～ 0x7fffffff”に限られます。

備考 1　RX850V4 における初期起動位相は，周期ハンドラの生成処理が完了してから 1 回目の起動要求が発行されるまでの相対時間間隔を意味しています。

備考 2　*cycphs* に基本情報で定義した基本クロック周期の整数倍値以外の値を指定した場合，CF850V4 は整数倍値が指定されていたものとして処理を行います。

18.5.11 割り込みハンドラ情報

割り込みハンドラ情報では，

1）*例外コード番号 inhno*

2）*属性（記述言語）inhatr*

3）*起動アドレス inthdr*

といった項目を個々の割り込みハンドラに対して定義します。
なお，割り込みハンドラ情報として定義可能な数は，1 つの例外コードに対して 1 個に限られます。
以下に，割り込みハンドラ情報の記述形式を示します。

```
DEF_INH ( inhno, { inhatr, inthdr } );
```

以下に，割り込みハンドラ情報で記述する項目について示します。

1）*例外コード番号 inhno*

割り込みハンドラを登録するマスカブル割り込みに対応した例外コード番号を指定します。
なお，*inhno* として指定可能な値は，デバイス・ファイルで規定されている割り込み要因名，または 16 バイト境界値に限られます。
なお，*inhno* に“割り込み要因名”を指定した場合，CF850V4 の起動オプションとして -cpu △ *name* の指定が必須となります。

2）*属性（記述言語）inhatr*

割り込みハンドラの属性（記述言語）を指定します。
なお，*inhatr* として指定可能な値は“TA_HLNG，TA_ASM のいずれか”に限られます。

TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

3）*起動アドレス inthdr*

割り込みハンドラの起動アドレスを指定します。
なお，*inthdr* として指定可能な値は“0x0 ～ 0xfffffffe の 2 バイト境界値，またはシンボル名”に限られます。

18.5.12 CPU 例外ハンドラ情報

CPU 例外ハンドラ情報では，

1）*例外コード番号 excno*

2）*属性（記述言語）excatr*

3）*起動アドレス exchdr*

といった項目を個々の CPU 例外ハンドラに対して定義します。
なお，CPU 例外ハンドラ情報として定義可能な数は，1 つの例外コード番号に対して 1 個に限られます。
以下に，CPU 例外ハンドラ情報の記述形式を示します。

```
DEF_EXC ( excno, { excatr, exchdr } );
```

以下に，CPU 例外ハンドラ情報で記述する項目について示します。

1）*例外コード番号 excno*

CPU 例外ハンドラを登録する CPU 例外に対応した例外コード番号を指定します。
なお，*excno* として指定可能な値は，デバイス・ファイルで規定されている割り込み要因名，または 16 バイト境界値に限られます。
なお，*excno* に“割り込み要因名”を指定した場合，CF850V4 の起動オプションとして -cpu △ *name* の指定が必須となります。

備考　ソフトウエア例外（TRAP0*n*：0x4*n*，TRAP1*n*：0x5*n*）のような 16 バイト境界値でない例外コード番号に対して CPU 例外ハンドラを登録する際にも

TRAP0*n* → 0x40
TRAP1*n* → 0x50

といったように必ず 16 バイト境界値を指定する必要があります。

2）*属性（記述言語）excatr*

CPU 例外ハンドラの属性（記述言語）を指定します。
なお，*excatr* として指定可能な値は“TA_HLNG，TA_ASM のいずれか”に限られます。

TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

3）*起動アドレス exchdr*

CPU 例外ハンドラの起動アドレスを指定します。
なお，*exchdr* として指定可能な値は“0x0 ～ 0xfffffffe の 2 バイト境界値，またはシンボル名”に限られます。

18.5.13 拡張サービス・コール・ルーチン情報

拡張サービス・コール・ルーチン情報では，

1）*機能コード番号 fncd*

2）*属性（記述言語）svcatr*

3）*起動アドレス svcrtn*

といった項目を個々の拡張サービス・コール・ルーチンに対して定義します。
なお，拡張サービス・コール・ルーチン情報として定義可能な数は，1 つの機能コード番号に対して 1 個に限られます。
以下に，拡張サービス・コール・ルーチン情報の記述形式を示します。

```
DEF_SVC ( fncd, { svcatr, svcrtn } );
```

以下に，拡張サービス・コール・ルーチン情報で記述する項目について示します。

1）*機能コード番号 fncd*

拡張サービス・コール・ルーチンの機能コード番号を指定します。
なお，*fncd* として指定可能な値は“0x1 ～ 0xff”に限られます。

2）*属性（記述言語）svcatr*

拡張サービス・コール・ルーチンの属性（記述言語）を指定します。
なお，*svcatr* として指定可能な値は“TA_HLNG，TA_ASM のいずれか”に限られます。

TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

3）*起動アドレス svcrtn*

拡張サービス・コール・ルーチンの起動アドレスを指定します。
なお，*svcrtn* として指定可能な値は“0x0 ～ 0xfffffffe の 2 バイト境界値，またはシンボル名”に限られます。

18.5.14 初期化ルーチン情報

初期化ルーチン情報では，

1) *属性（記述言語）iniatr*

2) *拡張情報 exinf*

3) *起動アドレス inirtn*

といった項目を初期化ルーチンに対して定義します。
なお，初期化ルーチン情報として定義可能な数は，0 ～ 254 個に限られます。
以下に，初期化ルーチン情報の記述形式を示します。

```
ATT_INI ( { initatr, exinf, inirtn } );
```

以下に，初期化ルーチン情報で記述する項目について示します。

1) *属性（記述言語）iniatr*

初期化ルーチンの属性（記述言語）を指定します。
なお，*iniatr* として指定可能な値は“TA_HLNG，TA_ASM のいずれか”に限られます。

TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

2) *拡張情報 exinf*

初期化ルーチンに引き渡す拡張情報を指定します。
なお，*exinf* として指定可能な値は“0x0 ～ 0xffffffff，またはシンボル名”に限られます。

備考　対象初期化ルーチンは，*exinf* を関数パラメータと同様に取り扱うことで操作可能となります。

3) *起動アドレス inirtn*

初期化ルーチンの起動アドレスを指定します。
なお，*inirtn* として指定可能な値は“0x0 ～ 0xfffffffe の 2 バイト境界値，またはシンボル名”に限られます。

18.5.15 アイドル・ルーチン情報

アイドル・ルーチン情報では，

1）*属性（記述言語）idlatr*

2）*起動アドレス idlrtn*

といった項目をアイドル・ルーチンに対して定義します。
なお，アイドル・ルーチン情報として定義可能な数は，0 ～ 1 個に限られます。
以下に，アイドル・ルーチン情報の記述形式を示します。

```
VATT_IDL ( { idlatr, idlrtn } );
```

以下に，アイドル・ルーチン情報で記述する項目について示します。

1）*属性（記述言語）idlatr*

アイドル・ルーチンの属性（記述言語）を指定します。
なお，*idlatr* として指定可能な値は“TA_HLNG，TA_ASM のいずれか”に限られます。

TA_HLNG：　C 言語
TA_ASM：　アセンブリ言語

2）*起動アドレス idlrtn*

アイドル・ルーチンの起動アドレスを指定します。
なお，*idlrtn* として指定可能な値は“0x0 ～ 0xfffffffe の 2 バイト境界値，またはシンボル名”に限られます。

18.6　メモリ容量計算式

RX850V4 が使用／管理するメモリ領域は，その用途により，5 種類のセクションに大別されます。

18.6.1　.rx_control セクション

RX850V4 が動作するうえで，および，RX850V4 が提供している機能を実現するうえで必要となる管理オブジェクト（システム管理テーブル，基本タスク管理ブロックなど）が割り付けられる領域です。

以下に，.rx_control セクションに割り付けられるデータのメモリ容量計算式（単位：バイト）を示します。

表 18 － 1　.rx_control セクションのメモリ容量計算式

データ名	メモリ容量計算式（単位：バイト）
システム管理テーブル	72（V850E2M でも 72）
レディ・キュー	align4 (*maxtpri*)
割り込みマスク管理配列	align4 (align16 ((*maxintno* / 16) - 7) / 8)
基本タスク管理ブロック配列	8 × *maxbtsk*
拡張タスク管理ブロック配列	24 × *maxetsk*
タスク例外管理ブロック配列	8 × *maxtex*
セマフォ管理ブロック配列	8 × *maxsem*
イベントフラグ管理ブロック配列	8 × *maxflg*
データ・キュー管理ブロック配列	8 × *maxdtq*
メールボックス管理ブロック配列	12 × *maxmbx*
ミューテックス管理ブロック配列	8 × *maxmtx*
固定長メモリ・プール管理ブロック配列	8 × *maxmpf*
可変長メモリ・プール管理ブロック配列	8 × *maxmpl*
周期ハンドラ管理ブロック配列	8 × *maxcyc*

備考　メモリ容量計算式のキーワードは，以下に示した意味を持ちます。

maxtpri：　基本情報で定義された最大タスク優先度
maxintno：　基本情報で定義された最大例外コード番号
ただし，PM+ でデバイスの品種指定を行う場合，または CF850V4 をコマンド・ラインから起動し，-cpu オプションでデバイスの品種指定を行う場合には，“該当デバイスが有する最大例外コード番号”
maxbtsk：　タスク情報の定義総数
maxttsk：　タスク情報（タスクの種別：非 TA_RSTR）の定義総数
maxtex：　タスク例外処理ルーチン情報の定義総数
maxsem：　セマフォ情報の定義総数
maxflg：　イベントフラグ情報の定義総数
maxdtq：　データ・キュー情報の定義総数
maxmbx：　メールボックス情報の定義総数
maxmtx：　ミューテックス情報の定義総数
maxmpf：　固定長メモリ・プール情報の定義総数
maxmpl：　可変長メモリ・プール情報の定義総数
maxcyc：　周期ハンドラ情報の定義総数

18.6.2　.rx_info セクション

RX850V4 が動作するうえで，および，RX850V4 が提供する機能を実現するうえで必要となる OS 資源（基本クロック周期，最大タスク優先度など）に関するデータが割り付けられる領域です。
以下に，.rx_info セクションに割り付けられる管理オブジェクトのメモリ容量計算式（単位：バイト）を示します。

表 18 － 2　.rx_info セクションのメモリ容量計算式

管理オブジェクト名	メモリ容量計算式（単位：バイト）
システム情報テーブル	208（212【V850E2M】）
起動タスク ID 配列	align4 (*maxact*)
起動周期ハンドラ ID 配列	align4 (*maxsta*)
割り込みマスク情報配列	align4 (align16 ((*maxintno* / 16) - 7) / 8)
タスク情報ブロック配列	24 × *maxtsk*
セマフォ情報ブロック配列	8 × *maxsem*
イベントフラグ情報ブロック配列	8 × *maxflg*
データ・キュー情報ブロック配列	8 × *maxdtq*
メールボックス情報ブロック配列	4 × *maxmbx*
ミューテックス情報ブロック配列	align4 (2 × *maxmtx*)
固定長メモリ・プール情報ブロック配列	12 × *maxmpf*
可変長メモリ・プール情報ブロック配列	12 × *maxmpl*
周期ハンドラ情報ブロック配列	20 × *maxcyc*
拡張サービス･コール･ルーチン情報ブロック配列	8 × *maxsvc*
割り込みハンドラ情報ブロック配列	8 × *maxint*
割り込みハンドラ ID 配列	align4 ((*maxintno* / 16) + 1)
初期化ルーチン情報ブロック配列	12 × *maxini*
アイドル・ルーチン管理ブロック	8
メモリ領域情報ブロック配列	8 × *maxmem*

備考　メモリ容量計算式のキーワードは，以下に示した意味を持ちます。

maxact：　タスク情報（タスクの初期起動状態：TA_ACT）の定義総数
maxsta：　周期ハンドラ情報（ 周期ハンドラの初期起動状態：TA_STA）の定義総数
maxintno：　基本情報で定義された最大例外コード番号
ただし，PM+ でデバイスの品種指定を行う場合，または CF850V4 をコマンド・ラインから起動し，-cpu オプションでデバイスの品種指定を行う場合には，“該当デバイスが有する最大例外コード番号”
maxtsk：　タスク情報の定義総数
maxsem：　セマフォ情報の定義総数
maxflg：　イベントフラグ情報の定義総数
maxdtq：　データ・キュー情報の定義総数
maxmbx：　メールボックス情報の定義総数
maxmtx：　ミューテックス情報の定義総数
maxmpf：　固定長メモリ・プール情報の定義総数
maxmpl：　可変長メモリ・プール情報の定義総数
maxcyc：　周期ハンドラ情報の定義総数
maxsvc：　拡張サービス・コール・ルーチン情報の定義総数
maxint：　割り込みハンドラ情報の定義総数と CPU 例外ハンドラ情報の定義総数の合計値
maxini：　初期化ルーチン情報の定義総数

maxmem：　メモリ領域情報の定義総数

18.6.3　.rx_memory セクション／ユーザ定義セクション

.rx_memory セクション／ユーザ定義セクションは，処理プログラムから利用可能な RX850V4 管理下のメモリ領域が割り付けられる領域です。基本的には .rx_memory セクションにすべてのメモリ領域を割り付けますが，領域を分割したい場合にユーザ定義セクションを利用します。なお，ユーザ定義セクションの定義は，コンフィギュレーション時に“メモリ領域情報”で行います。

両セクションでは，以下のように割り付けることのできるメモリ領域が異なります。

.rx_memory セクション	ユーザ定義セクション
システム・スタック タスク・スタック データ・キュー領域 固定長メモリ・プール 可変長メモリ・プール	タスク・スタック データ・キュー領域 固定長メモリ・プール 可変長メモリ・プール

.rx_memory セクション／ユーザ定義セクションは，ユーザ使用領域とそれを管理するための RX850V4 管理領域に分かれ，.rx_memory セクション／ユーザ定義セクションのサイズは，それぞれ以下の計算式で求められます。

.rx_memory セクションのサイズ＝ RX_SZ（.rx_memory セクション）＋ USR_SZ（.rx_memory セクション）

ユーザ定義セクションのサイズ＝ RX_SZ（ユーザ定義セクション）＋ USR_SZ（ユーザ定義セクション）

- RX_SZ（.rx_memory セクション／ユーザ定義セクション）
  .rx_memory セクション／ユーザ定義セクション内の RX850V4 管理領域のサイズで，以下の計算式で求められます。

$$RX\_SZ = (20 + frmsz) + \sum_{k=1}^{tsknum} ctxsz_k + (4 \times mplnum)$$

備考　上記計算式中の“(20 ＋ *frmsz*)”は，.rx_memory セクションの場合は必要，ユーザ定義セクションの場合は不要です。

frmsz：　割り込みハンドラの実行情報を保存するためのコンテキスト領域
属性，CPU 種別，レジスタ・モード種別により値が変化します。
表 18 － 3 を参照してください。

ctxtsz：　タスクの実行情報を保存するためのコンテキスト領域
ただし，制約タスクは数に含みません。
属性，CPU 種別，レジスタ・モード種別により値が変化します。
表 18 － 4，表 18 － 5 を参照してください。

tsknum：　タスク情報で定義したタスクの総数
ただし，制約タスクは数に含みません。

mplnum：　可変長メモリ・プール情報で定義した可変長メモリ・プール数

表 18 － 3　割り込みハンドラのコンテキスト領域（*frmsz*）

レジスタ・モード	コンテキスト領域
22 レジスタ・モード	60（68【V850E2M】）
26 レジスタ・モード	68（76【V850E2M】）
32 レジスタ・モード	80（88【V850E2M】）

表 18 － 4　タスク（ プリエンプトの受け付け状態：非 TA_DISPREEMPT）のコンテキスト領域（*ctxsz*）

レジスタ・モード	コンテキスト領域
22 レジスタ・モード	88（100【V850E2M】）
26 レジスタ・モード	104（116【V850E2M】）
32 レジスタ・モード	128（140【V850E2M】）

表 18 － 5　タスク（ プリエンプトの受け付け状態：TA_DISPREEMPT）のコンテキスト領域（*ctxsz*）

レジスタ・モード	コンテキスト領域
22 レジスタ・モード	60（68【V850E2M】）
26 レジスタ・モード	68（76【V850E2M】）
32 レジスタ・モード	80（88【V850E2M】）

- USR_SZ（.rx_memory セクション／ユーザ定義セクション）
  .rx_memory セクション／ユーザ定義セクション内のユーザ使用領域のサイズで，以下の計算式で求められます。

$$
\begin{aligned}
USR\_SZ = \ & align4(sys\_stksz) \\
& + \sum_{k=1}^{tsknum} align4(stksz)_k \\
& + \sum_{k=1}^{dtqnum} (dtqcnt \times 4)_k \\
& + \sum_{k=1}^{mpfnum} (align4(blksz)_k \times blkcnt)_k \\
& + \sum_{k=1}^{mplnum} align4(mplsz)_k
\end{aligned}
$$

備考　上記計算式中の“align4(*sys_stksz*)”は，.rx_memory セクションの場合は必要，ユーザ定義セクションの場合は不要です。

sys_stksz：基本情報で定義したシステム・スタックのサイズ

tsknum：タスク情報で定義したタスクの総数
ただし，制約タスクは数に含みません。

stksz：タスク情報で定義したタスクのスタック・サイズ

dtqnum：データ・キュー情報で定義したデータ・キューの総数

dtqcnt：データ・キュー情報で定義したデータ数

mpfnum：固定長メモリ・プール情報で定義した固定長メモリ・プール数

blksz：固定長メモリ・プール情報で定義したブロック単位サイズ

blkcnt：固定長メモリ・プール情報で定義した全メモリ・ブロック数

mplnum：可変長メモリ・プール情報で定義した可変長メモリ・プール数

mplsz：可変長メモリ・プール情報で定義したプール・サイズ

この式へ設定する値は，アプリケーションに応じてユーザが見積もります。
sys_stksz の見積もりに関してのみ，特別に，アプリケーションの情報を元に，以下の計算式で求めます。
なお，この計算式で求めたサイズよりも余裕を見たサイズを設定することを推奨します。

sys_stksz は，以下の *sys_stksz1*，*sys_stksz2*，*sys_stksz3* の中で最大のものとなります。

$$sys\_stksz1 = \sum_{k=1}^{tskprinum} (\mathrm{align4}(rstr\_stksz\_hi) + ctxsz)_k + \sum_{k=1}^{intprinum} (\mathrm{align4}(intsz\_hi) + frmsz)_k$$

$$sys\_stksz2 = idlsz$$

$$sys\_stksz3 = inisz\_hi$$

tskprinum：　基本情報で定義したタスク優先度の総数

rstr_stksz_hi：　タスク優先度 *k* の制約タスクの中で最大のタスク・スタック・サイズ
制約タスクとは，タスク情報（タスクの種別：TA_RSTR）で定義されたタスクです。
タスク・スタック・サイズとは，タスク上で使用されるスタック・サイズです。
タスク優先度 *k* に制約タスクがない場合，タスク優先度 *k* に関して計算は不要です。

ctxtsz：　タスクの実行情報を保存するためのコンテキスト領域
属性，CPU 種別，レジスタ・モード種別により値が変化します。
表 18 － 4，表 18 － 5 を参照してください。

intprinum：　デバイスが持つ割り込み優先度の総数

intsz_hi：　割り込み優先度 *k* の割り込みハンドラ／周期ハンドラの中で最大のスタック・サイズ
割り込み優先度 *k* の周期ハンドラとは，基本情報で定義した基本クロック用タイマ割り込みの割り込み優先度です。
割り込み優先度 *k* に割り込みハンドラ／周期ハンドラがない場合，割り込み優先度 *k* に関して計算は不要です。

frmsz：　割り込みハンドラの実行情報を保存するためのコンテキスト領域
属性，CPU 種別，レジスタ・モード種別により値が変化します。
表 18 － 3 を参照してください。

idlsz：　アイドル・ルーチンで使用するスタック・サイズ

inisz_hi：　初期化ルーチンのスタック・サイズの中で最大のもの

18.6.4　.rx_text セクション

RX850V4 の本体処理（カーネル共通部モジュール，カーネル・モジュール）が割り付けられる領域です。
以下に，.rx_text セクションに割り付けられるメモリ領域の一覧を示します。

- カーネル共通部モジュール
  RX850V4 の核となる処理部分であり，以下に示す機能を提供しています。
  - スケジューリング機能
  - システム初期化処理（カーネル初期化部）

  なお，カーネル共通部モジュールでは，約 4K バイトのメモリ領域を占有します。

- カーネル・モジュール
  RX850V4 が提供するサービス・コールの本体処理部分であり，以下に示す機能を提供しています。
  - タスク管理機能
  - タスク付属同期機能
  - タスク例外処理機能
  - 同期通信機能（セマフォ，イベントフラグ，データ・キュー，メールボックス）
  - 拡張同期通信機能（ミューテックス）
  - メモリ・プール管理機能（固定長メモリ・プール，可変長メモリ・プール）
  - 時間管理機能
  - システム状態管理機能
  - 割り込み管理機能
  - サービス・コール管理機能
  - システム構成管理機能

  なお，カーネル・モジュールでは，約 1K ～ 21K バイトのメモリ領域を占有しますが，システムで使用するサービス・コールの種類に制限を設けることにより必要となるメモリ容量を抑えることが可能です。

18.7　記 述 例

以下に，システム・コンフィギュレーション・ファイルの記述例を示します。

図 18 － 2　システム・コンフィギュレーション・ファイルの記述例

```
-- 宣言情報の記述
INCLUDE ( " ¥"kernel.h¥" " );

-- システム情報の記述
RX_SERIES ( RX850V4, V430 );

CPU_TYPE ( V850E2M );
REG_MODE ( r32 );
DEF_TIM ( 0x1 );
CLK_INTNO ( 0x80 );
SYS_STK ( 0x1000 );
STK_CHK ( TA_OFF );
MAX_PRI ( 0x20 );
MAX_INT ( 0x2, 0x1e );
DEF_FPSR ( 0x00020000 );

MEM_AREA ( usrmem, SIZE_AUTO );

-- 静的 API 情報の記述
CRE_TSK ( taskA, { TA_HLNG|TA_ACT|TA_DISINT, 0x1, taskA, 0x1, 0x800:usrmem, NULL }
);
CRE_TSK ( taskB, { TA_HLNG|TA_ACT, 0x2, taskB, 0x1, 0x800:usrmem, NULL } );

DEF_TEX ( taskA, { TA_HLNG, texrtnA } );
DEF_TEX ( taskB, { TA_HLNG, texrtnB } );

CRE_SEM ( sem, { TA_TFIFO, 0x0, 0x1 } );

CRE_FLG ( flg, { TA_TFIFO|TA_WSGL|TA_CLR, 0x0 } );

CRE_DTQ ( dtq, { TA_TFIFO, 0xff:usrmem, NULL } );

CRE_MBX ( mbx, { TA_TFIFO|TA_MPRI, 0x7fff, NULL } );

CRE_MPF ( mpf, { TA_TFIFO, 0x7fff, 0x1:usrmem, NULL } );

CRE_MPL ( mpl, { TA_TFIFO, 0x8000:usrmem, NULL } );

CRE_CYC ( cyc, { TA_HLNG|TA_STA|TA_PHS, 0x1, cychdr, 0x100, 0x1000 } );

DEF_INH ( 0x1c0, { TA_ASM, inthdr } );

DEF_EXC ( 0x60, { TA_HLNG, exchdr } );

ATT_INI ( { TA_ASM, 0x1, inirtn } );

VATT_IDL ( { TA_HLNG, idlrtn } );
```

備考　RX850V4 では，システム･コンフィギュレーション･ファイルのサンプル･ソース･ファイルを提供しています。

<rx_sample>¥src¥sys.cfg

第 19 章　コンフィギュレータ CF850V4

本章では，RX850V4 がシステム構築時に有益なユーティリティ・ツールとして提供しているコンフィギュレータ CF850V4 について解説しています。

19.1　概　　要

RX850V4 が提供している機能を利用したシステム（ロード・モジュール）を構築する場合，RX850V4 に提供するデータを保持した情報ファイルが必要となります。

基本的に情報ファイルは，規定された形式のデータ羅列であるため，各種エディタを用いて記述することは可能です。しかし，情報ファイルは，記述性／可読性の面で劣ったものとなっているため，記述に際してはかなりの時間と労力を必要とします。

そこで，RX850V4 では，記述性／可読性の面で優れたシステム・コンフィギュレーション・ファイルから情報ファイルへと変換するユーティリティ・ツール（コンフィギュレータ CF850V4）を提供しています。

CF850V4 は，システム・コンフィギュレーション・ファイルを入力ファイルとして読み込んだあと，情報ファイルを出力します。

以下に，CF850V4 が出力する情報ファイルについて示します。

- システム情報テーブル・ファイル
  RX850V4 が動作するうえで必要となる OS 資源（基本クロック周期，最大タスク優先度，管理オブジェクトなど）に関するデータを保持した情報ファイルです。

- システム情報ヘッダ・ファイル
  システム・コンフィギュレーション・ファイルに記述されたオブジェクト名（タスク名，セマフォ名など）と ID の対応付けを保持した情報ファイルです。

- エントリ・ファイル
  割り込み /CPU 例外が発生した際に CPU 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（割り込み前処理，CPU 例外前処理など）への分岐処理を保持したエントリ処理専用ルーチン（割り込みエントリ処理，CPU 例外エントリ処理）です。

19.2　起動方法

19.2.1　コマンド・ラインからの起動

以下に，CF850V4 をコマンド・ラインから起動する際の起動方法を示します。
ただし，入力例中の“C>”はコマンド・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。
また，“[　]”で囲まれた起動オプションは，省略可能な起動オプションであることを表しています。

C>　cf850v4.exe △ [*@cmd_file*] △ [-cpu △ *name*] △ [-devpath=*path*] △ [-reg*xx*] △ [-i △ *sitfile*] △ [-d △ *includefile*] △ [-e *entry*] △ [-ni] △ [-nd] △ [-ne] △ [-t △ *tool*] △ [-T △ *compiler_path*] △ [-I △ *include_path*] △ [-np] △ [-V] △ [-help] △ *file* ［Enter］

以下に，各起動オプションの詳細を示します。

- *@cmd_file*
  コマンド・ファイル名を指定します。

  省略時　コマンド・ライン上で指定された起動オプションが有効となります。

  備考　　コマンド・ファイルについての詳細は，「19.2.3　コマンド・ファイル」を参照してください。

- -cpu △ *name*
  ターゲット・デバイスの品種指定名を指定します。

  省略時　基本情報において指定された品種指定名となります。
  なお，本起動オプションの指定が行われなかった場合，CF850V4 はデバイス・ファイルの読み込みを行いません。このため，システム・コンフィギュレーション・ファイル内で“デバイス・ファイルで規定されている割り込み要因名”を用いた定義が行えなくなります。

- -devpath=*path*
  -cpu △ *name* で指定されたターゲット・デバイスに対応したデバイス・ファイルを *path* フォルダから検索します。

  省略時　カレント・フォルダに対して検索処理を行います。

- -reg*xx*
  CF850V4 からの出力ファイルのファイル形式（レジスタ・モード種別）を指定します。
  なお，*xx* として指定可能なキーワードは“22，26，32 のいずれか”に限られます。

  22：　　22 レジスタ・モード
  26：　　26 レジスタ・モード
  32：　　32 レジスタ・モード

  省略時　RX シリーズ情報において指定されたレジスタ・モード種別となります。
  なお，本起動オプション，および，RX シリーズ情報の両方においてレジスタ・モード種別の指定が行われなかった場合，CF850V4 は -reg32 が指定されていたものとして処理を行います。

- -i △ *sitfile*
  CF850V4 からの出力ファイル名（システム情報テーブル・ファイル名）を指定します。

  省略時　以下に示した起動オプションが指定されていたものとして処理を行います。

  -i △ sit.s

  備考 1　出力ファイル名 *sitfile* として指定可能な文字数は，パス名を含めて 255 文字以内に限られます。

  備考 2　本起動オプションと -ni が同時に指定された場合，-ni を有効起動オプションとして扱います。

- -d △ *includefile*
  CF850V4 からの出力ファイル名（システム情報ヘッダ・ファイル名）を指定します。

  省略時　-d △ kernel_id.h が指定されていたものとして処理を行います。

  備考 1　出力ファイル名 *includefile* として指定可能な文字数は，パス名を含めて 255 文字以内に限られます。

  備考 2　本起動オプションと -nd が同時に指定された場合，-nd を有効起動オプションとして扱います。

- -e △ *entry*
  CF850V4 からの出力ファイル名（エントリ・ファイル名）を指定します。

  省略時　以下に示した起動オプションが指定されていたものとして処理を行います。

  -e △ entry.s

  備考 1　出力ファイル名 *entry* として指定可能な文字数は，パス名を含めて 255 文字以内に限られます。

  備考 2　本起動オプションと -ne が同時に指定された場合，-ne を有効起動オプションとして扱います。

- -ni
  システム情報テーブル・ファイルの出力を抑制します。

  省略時　以下に示した起動オプションが指定されていたものとして処理を行います。

  -i △ sit.s

  備考　本起動オプションと -i △ *sitfile* が同時に指定された場合，本起動オプションを有効起動オプションとして扱います。

- -nd
  システム情報ヘッダ・ファイルの出力を抑制します。

  省略時　-d △ kernel_id が指定されていたものとして処理を行います。

  備考　本起動オプションと -d △ *includefile* が同時に指定された場合，本起動オプションを有効起動オプションとして扱います。

- -ne
  エントリ・ファイルの出力を抑制します。

  省略時　以下に示した起動オプションが指定されていたものとして処理を行います。

  -e △ entry.s

  備考　本起動オプションと -e △ *entry* が同時に指定された場合，本起動オプションを有効起動オプションとして扱います。

- -t △ *tool*
  ユーザが使用する C コンパイラ・パッケージの種類を指定します。
  なお，*tool* として指定可能なキーワードは，NECEL に限られます。

  省略時　-t △ NECEL が指定されていたものとして処理を行います。

- -T △ *compiler_path*
  -t △ *tool* で指定された C コンパイラ・パッケージの C プリプロセッサに対するコマンド検索パスを指定します。

  省略時　環境変数（PATH など）で定義されているフォルダを検索対象フォルダとして処理を行います。

  備考　コマンド検索パス *compiler_path* として指定可能な文字数は，255 文字以内に限られます。

- -I △ *include_path*
  入力ファイル *file* に記述されているヘッダ・ファイル情報の検索対象フォルダ名を指定します。

  省略時　*file* で指定された入力ファイルが格納されているフォルダ，カレント・フォルダ，-t △ *tool* で指定された C コンパイラ・パッケージのデフォルト検索対象フォルダの順序で検索処理を行います。

  備考　検索対象フォルダ名 *include_path* として指定可能な文字数は，255 文字以内に限られます。

- -np
  CF850V4 によるシステム・コンフィギュレーション・ファイルの構文解析処理が完了した際，C プリプロセッサの起動を抑制します。

  省略時　-t △ *tool* で指定された C コンパイラ・パッケージの C プリプロセッサを起動します。

- -V
  CF850V4 のバージョン情報を標準出力に出力します。

  備考　本起動オプションを指定した場合，ほかの起動オプションはすべて無効となり，情報ファイルの出力が抑制されます。

- -help
  CF850V4 の起動オプションに関する情報（種類，用途など）を標準出力に出力します。

  備考　本起動オプションを指定した場合，ほかの起動オプションはすべて無効となり，情報ファイルの出力が抑制されます。

- *file*
  CF850V4 への入力ファイル名（システム・コンフィギュレーション・ファイル名）を指定します。

  備考 1　入力ファイル名 *file* として指定可能な文字数は，パス名を含めて 255 文字以内に限られます。

  備考 2　本入力ファイル名の指定は，-V，または -help の指定時以外に省略することはできません。

19.2.2　CubeSuite からの起動

プロパティ パネルの[システム・コンフィギュレーション・ファイル関連情報]タブで設定した内容に基づき，CubeSuite のビルド時に起動されます。

19.2.3　コマンド・ファイル

CF850V4 では，コマンド・ライン上で指定可能な起動オプションの文字数制限を解消する目的からコマンド・ファイル対応を行っています。
以下に，コマンド・ファイルの記述形式を示します。

1）コメント行
行頭に # が記述された行については，コメント行として扱われます。

2）起動オプション
-cpu，-i，-d，-t，-T，-I については，*-xxx* 部で 1 行，パラメータ部で 1 行の計 2 行書きとなります。
また, -devpath, および, パラメータ部のない起動オプション -reg, -ni, -nd, -np, *file* については, 1 行書きとなります。

3）文字数制限
コマンド・ファイル内の 1 行に対する文字数制限は 4096 文字となっています。

以下に，コマンド・ファイルの記述例を示します。
なお，以下の記述例では，次に示す起動オプションが記述されています。

ターゲット・デバイス：	μPD70F3742
デバイス・ファイルの検索フォルダ：	C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥Device¥V850¥Devicefile
レジスタ・モード種別：	r26
システム情報テーブル・ファイル：	sit.s
システム情報ヘッダ・ファイル：	kernel_id.h
C コンパイラ・パッケージの種類：	NECEL
C コンパイラ・パッケージに対するコマンド検索パス：	C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥CA850¥V3.43¥bin
ヘッダ・ファイル情報の検索対象フォルダ名：	C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥RX850V4¥V4.30¥inc850, C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥SampleProjects¥V850ES_JX3 RX850V4 (CA850) V1.00¥appli¥include
C プリプロセッサの起動有無：	起動する
システム・コンフィギュレーション・ファイル：	sys.cfg

図 19 － 1　コマンド・ファイルの記述例

```
# Command File
-cpu f3742 -devpath="C:¥Program Files¥NEC Electronics
CubeSuite¥CubeSuite¥Device¥V850¥Devicefile" -reg26
-i sit.s -d kernel_id.h
-t NECEL -T "C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥CA850¥V3.43¥bin"
-I "C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥RX850V4¥V4.30¥inc850"
-I "C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥SampleProjects¥V850ES_JX3
RX850V4 (CA850) V1.00¥appli¥include"
sys.cfg
```

19.2.4　コマンド入力例

以下に，CF850V4 のコマンド入力例を示します。
ただし，入力例中の“C>”はコマンド・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）システム・コンフィギュレーション・ファイル *sys.cfg* をカレント・フォルダから，品種指定名 f3742 に対応したデバイス・ファイルを C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥Device¥V850¥Devicefile フォルダから入力ファイルとして読み込んだあと，システム情報テーブル・ファイル *sit.s*，システム情報ヘッダ・ファイル *kernel_id.h*，エントリ・ファイル *entry.s* を 26 レジスタ・モードのファイル形式で出力します。
なお，-t で指定された C コンパイラ・パッケージの C プリプロセッサに対するコマンド検索処理は，以下に示した順序で行われ，CF850V4 によるシステム・コンフィギュレーション・ファイルの構文解析処理完了時に，該当 C プリプロセッサが起動されます。

1. -T で指定された C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥CA850¥V3.43¥bin フォルダ
2. 環境変数（PATH など）で定義されているフォルダ

また，-I で指定されたフォルダに対するインクルード・ファイル検索処理は，以下に示した順序で行われます。

1. -Iで指定されたC:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥RX850V4¥V4.30¥inc850フォルダ
2. -I で指定された C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥SampleProjects¥V850ES_JX3 RX850V4 (CA850) V1.00¥appli¥include フォルダ

```
C> cf850v4.exe △ -cpu △ f3742 △ -devpath="C:¥Program Files¥NEC Electronics
   CubeSuite¥CubeSuite¥Device¥V850¥Devicefile" △ -reg26 △ -i △ sit.s △ -d △ kernel_id.h △ -e △
   entry.s △ -t △ NECEL △ -T △ "C:¥Program Files¥NEC Electronics
   CubeSuite¥CubeSuite¥CA850¥V3.43¥bin" △ -I △ "C:¥Program Files¥NEC Electronics
   CubeSuite¥CubeSuite¥RX850V4¥V4.30¥inc850" △ -I △ "C:¥Program Files¥NEC Electronics
   CubeSuite¥CubeSuite¥SampleProjects¥V850ES_JX3 RX850V4 (CA850) V1.00¥appli¥include" △
   sys.cfg ［Enter］
```

2）CF850V4 のバージョン情報を標準出力に出力します。

```
C> cf850v4.exe △ -V ［Enter］
```

3）CF850V4 の起動オプションに関する情報（種類，用途など）を標準出力に出力します。

```
C> cf850v4.exe △ -help ［Enter］
```

付録 A　ウインドウ・リファレンス

本付録では，CF850V4 の起動オプションを統合開発環境プラットホーム CubeSuite から設定する際に必要となるウインドウ／パネルについて説明しています。

A.1　説　　明

以下に，ウインドウ／パネルの一覧を示します。

表 A－1　ウインドウ／パネルの一覧

ウインドウ／パネル名	機能概要
メイン・ウインドウ	CubeSuite を起動した際，最初にオープンするウインドウ
プロジェクト・ツリー パネル	プロジェクトの構成要素のツリー表示
プロパティ パネル	プロジェクト・ツリー パネルで選択しているリアルタイム OS ノード，システム・コンフィギュレーション・ファイル等について，詳細情報の表示，および設定の変更

メイン・ウインドウ

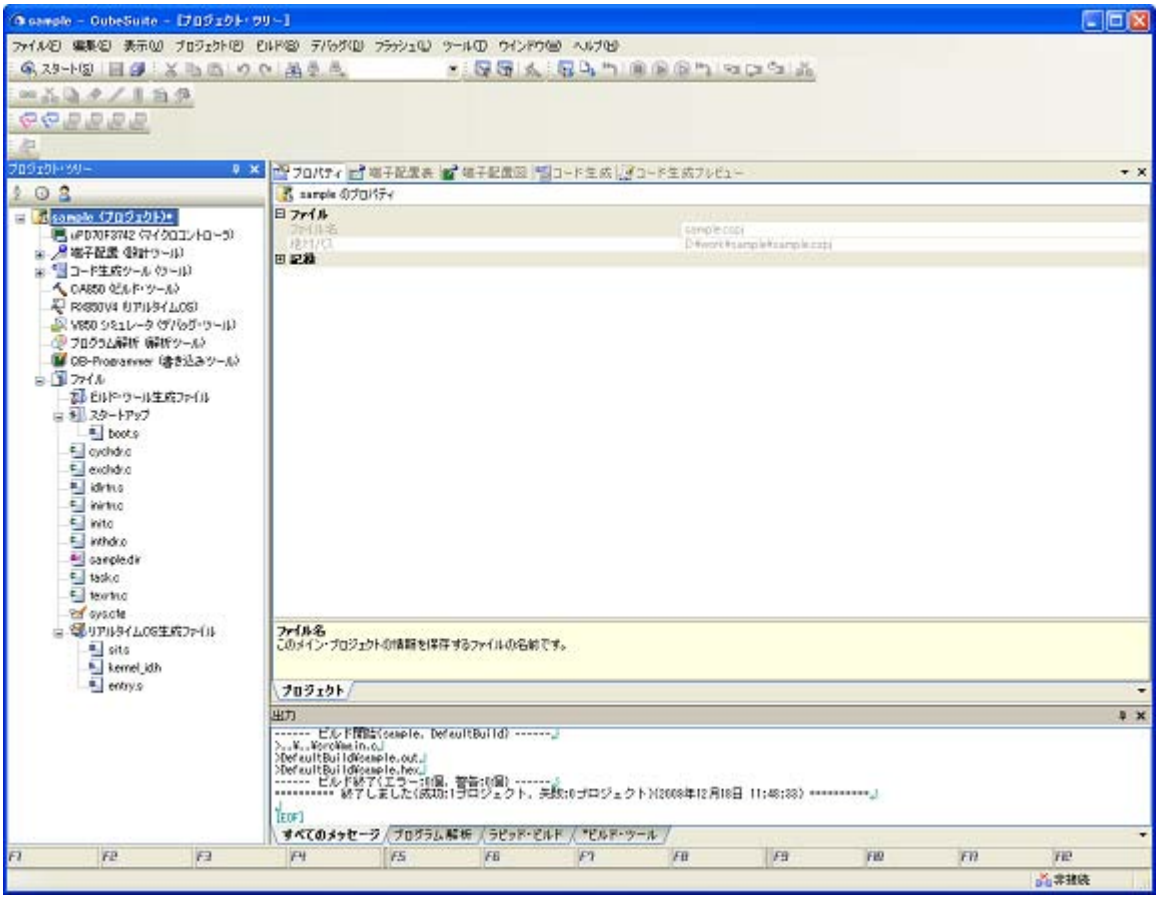
概要

CubeSuite を起動した際，最初にオープンするウインドウです。
ビルドを行う際は，本ウインドウからユーザ・プログラムの実行制御，および各パネルのオープンを行います。

なお，本ウインドウは，次の方法でオープンすることができます。

- Windows® の［スタート］→［すべてのプログラム］→［NEC Electronics CubeSuite］→［CubeSuite］を選択

表示イメージ



機能

1）メニューバー

リアルタイム OS 関連のメニューを示します。
なお，各メニューから引き出される項目は，ユーザ設定 ダイアログでカスタマイズすることができます。

- ［表示］

リアルタイム OS	リアルタイム OS の各ツールを起動するためのカスケード・メニューを表示します。
リソース情報	リアルタイム OS リソース情報 パネルをオープンします。 なお，本メニューは，デバッグ・ツールと切断時は無効となります。
実行解析	AZ850V4 ウインドウをオープンします。 なお，本メニューは，デバッグ・ツールと切断時は無効となります。

2）ツールバー

リアルタイム OS 関連のボタン群を示します。
なお，ツールバー上のボタンは，ユーザ設定 ダイアログでカスタマイズすることができます。また，同ダイアログにより，新規にツールバーを作成することもできます。

- リアルタイム OS ツールバー

	リアルタイム OS リソース情報 パネルをオープンします。 なお，本ボタンは，デバッグ・ツールと切断時は無効となります。

3）パネル表示エリア

以下のパネルを表示するエリアです。

- プロジェクト・ツリー パネル
- プロパティ パネル
- 出力 パネル

表示内容の詳細については，各パネルの項を参照してください。

備考　出力 パネルについての詳細は，「CubeSuite V850 ビルド編」，または「CubuSuite ビルド編（CX コンパイラ）」のユーザーズ・マニュアルを参照してください。

プロジェクト・ツリー パネル

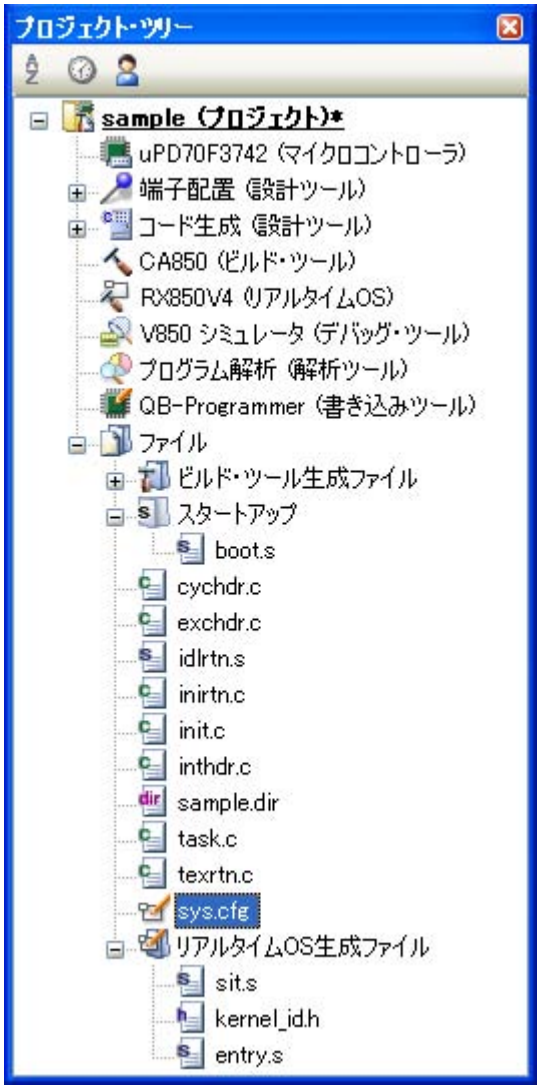
概要

プロジェクトを構成するリアルタイム OS ノード，システム・コンフィギュレーション・ファイル等の構成要素をツリー表示します。

なお，本パネルは，次の方法でオープンすることができます。

- ［表示］メニュー→［プロジェクト・ツリー］を選択

表示イメージ



機能

1）プロジェクト・ツリー エリア

プロジェクトの構成要素を以下のノードでツリー表示します。

ノード	説明
RX850V4(リアルタイム OS) （“リアルタイム OS ノード”と呼びます。）	使用するリアルタイム OS です。
xxx.cfg	システム・コンフィギュレーション・ファイルです。
リアルタイム OS 生成ファイル （“リアルタイム OS 生成ファイル・ノード”と呼びます。）	システム・コンフィギュレーション・ファイル追加時に作成されるノードで，以下の情報ファイルが直下に表示されます。 - システム情報テーブル・ファイル（.s） - システム情報ヘッダ・ファイル（.h） - エントリ・ファイル（.s） 本ノード，および本ノードに表示されているファイルを直接削除することはできません。 システム・コンフィギュレーション・ファイルをプロジェクトから外した場合，本ノード，および本ノードに表示されているファイルは表示されなくなります。

コンテキスト・メニュー

1）リアルタイム OS ノード，リアルタイム OS 生成ファイル・ノードを選択している場合

プロパティ	選択しているノードのプロパティをプロパティ パネルに表示します。

2）システム・コンフィギュレーション・ファイル，情報ファイルを選択している場合

アセンブル	選択しているアセンブラ・ソース・ファイルをアセンブルします。 なお，本メニューは，システム情報テーブル・ファイル，エントリ・ファイルを選択している場合のみ表示されます。 ただし，ビルド・ツールが実行中の場合は無効となります。
開く	ファイルの拡張子に割り当てられたアプリケーションで選択しているファイルをオープンします。 なお，本メニューは，複数のファイルを選択している場合は無効となります。
内部エディタで開く ...	エディタ パネルで選択しているファイルをオープンします。 なお，本メニューは，複数のファイルを選択している場合は無効となります。
アプリケーションを指定して開く ...	プログラムから開く ダイアログをオープンし，指定したアプリケーションで選択しているファイルをオープンします。 ただし，複数のファイルを選択している場合は無効となります。
エクスプローラでフォルダを開く	選択しているファイルが存在しているフォルダをエクスプローラでオープンします。
追加	プロジェクトにファイル，カテゴリ・ノードを追加するためのカスケード・メニューを表示します。

既存のファイルを追加 ...	既存のファイルを追加 ダイアログをオープンし，選択したファイルをプロジェクトに追加します。
新しいファイルを追加 ...	ファイル追加 ダイアログをオープンし，選択した種類でファイルを作成し，プロジェクトに追加します。
新しいカテゴリを追加	選択しているファイルと同じレベルにカテゴリ・ノードを追加し，カテゴリ名が編集可能な状態になります。 なお，本メニューは，ビルド・ツールが実行中の場合，およびカテゴリのネスト数が 20 の場合は無効となります。
プロジェクトから外す	選択しているファイルをプロジェクトから外します。 ファイル自体はファイル・システム上からは削除されません。 なお，本メニューは，ビルド・ツールが実行中の場合は無効となります。
コピー	選択しているファイルをクリップ・ボードにコピーします。 ファイル名を編集中の場合は，選択している文字列をクリップ・ボードにコピーします。
貼り付け	本メニューは常に無効です。
名前の変更	選択しているファイルの名前が編集可能な状態になります。 実際のファイル名も変更されます。
プロパティ	選択しているファイルのプロパティをプロパティ パネルに表示します。

プロパティ パネル

概要

プロジェクト・ツリー パネルで選択しているリアルタイム OS ノード，システム・コンフィギュレーション・ファイル等について，カテゴリ別に詳細情報の表示，および設定の変更を行います。

なお，本パネルは，次の方法でオープンすることができます。

- プロジェクト・ツリー パネル上において，リアルタイム OS ノード，システム・コンフィギュレーション・ファイル等を選択したのち，［表示］メニュー→［プロパティ］を選択，またはコンテキスト・メニュー→［プロパティ］を選択

備考　すでにプロパティ パネルがオープンしている場合，プロジェクト・ツリー パネル上において，リアルタイム OS ノード，システム・コンフィギュレーション・ファイル等を選択することで，選択した項目の詳細情報を表示します。

表示イメージ

機能

1）対象名エリア

プロジェクト・ツリー パネルで選択しているノードの名称を表示します。
複数のノードを選択している場合，本エリアは空欄となります。

2）詳細情報表示／変更エリア

プロジェクト・ツリー パネルで選択しているリアルタイム OS ノード，システム・コンフィギュレーション・ファイル等の詳細情報を，カテゴリ別のリスト形式で表示し，設定の変更を直接行うことができるエリアです。
⊟マークは，そのカテゴリ内に含まれているすべての項目が展開表示されていることを示し，また，⊞マークは，カテゴリ内の項目が折りたたみ表示されていることを示します。展開／折りたたみ表示の切り替えは，このマークのクリック，またはカテゴリ名のダブルクリックにより行うことができます。
カテゴリ，およびそれに含まれる項目の表示内容／設定方法についての詳細は，該当するタブの項を参照してください。

3）プロパティの説明エリア

詳細情報表示／変更エリアで選択したカテゴリや項目の簡単な説明を表示します。

4）タブ選択エリア

タブを選択することにより，詳細情報を表示するカテゴリが切り替わります。
本パネルには，次のタブが存在します（各タブ上における表示内容／設定方法についての詳細は，該当するタブの項を参照してください）。

- プロジェクト・ツリー パネルでリアルタイム OS ノードを選択している場合
  - ［RX850V4］タブ
- プロジェクト・ツリー パネルでシステム・コンフィギュレーション・ファイルを選択している場合
  - ［システム・コンフィギュレーション・ファイル関連情報］タブ
  - ［ファイル情報］タブ
- プロジェクト・ツリー パネルでリアルタイム OS 生成ファイル・ノードを選択している場合
  - ［カテゴリ情報］タブ
- プロジェクト・ツリー パネルでシステム情報テーブル・ファイル，エントリ・ファイルを選択している場合
  - ［ビルド設定］タブ
  - ［個別アセンブル・オプション］タブ
  - ［ファイル情報］タブ
- プロジェクト・ツリー パネルでシステム情報ヘッダ・ファイルを選択している場合
  - ［ファイル情報］タブ

備考 1　［ファイル情報］タブ，［カテゴリ情報］タブ，［ビルド設定］タブ，［個別アセンブル・オプション］タブについての詳細は，「CubeSuite V850 ビルド編」，または「CubuSuite ビルド編（CX コンパイラ）」のユーザーズ・マニュアルを参照してください。

備考 2　プロジェクト・ツリー パネルで複数の構成要素を選択している場合は，その構成要素に共通するタブのみ表示されます。プロパティの値の変更は，選択している複数の構成要素に共通に反映されます。

［編集］メニュー（プロパティ パネル専用部分）

元に戻す	直前に行ったプロパティの値の編集作業を取り消します。
切り取り	プロパティの値を編集中の場合，選択している文字列を切り取ってクリップ・ボードに移動します。
コピー	選択しているプロパティの値文字列をクリップ・ボードにコピーします。
貼り付け	プロパティの値を編集中の場合，クリップ・ボードの内容を挿入します。
削除	プロパティの値を編集中の場合，選択している文字列を削除します。
すべて選択	プロパティの値を編集中の場合，選択しているプロパティの値文字列をすべて選択します。

コンテキスト・メニュー

元に戻す	直前に行ったプロパティの値の編集作業を取り消します。
切り取り	プロパティの値を編集中の場合，選択している文字列を切り取ってクリップ・ボードに移動します。
コピー	選択しているプロパティの値文字列をクリップ・ボードにコピーします。
貼り付け	プロパティの値を編集中の場合，クリップ・ボードの内容を挿入します。
削除	プロパティの値を編集中の場合，選択している文字列を削除します。
すべて選択	プロパティの値を編集中の場合，選択しているプロパティの値文字列をすべて選択します。

デフォルトに戻す	選択している項目の設定値をプロジェクトに設定しているデフォルト値に戻します。 ただし，［個別アセンブル・オプション］タブにおいては，全体オプションの設定値に戻します。
すべてデフォルトに戻す	現在表示しているタブの設定値をすべてプロジェクトに設定しているデフォルト値に戻します。 ただし，［個別アセンブル・オプション］タブにおいては，全体オプションの設定値に戻します。

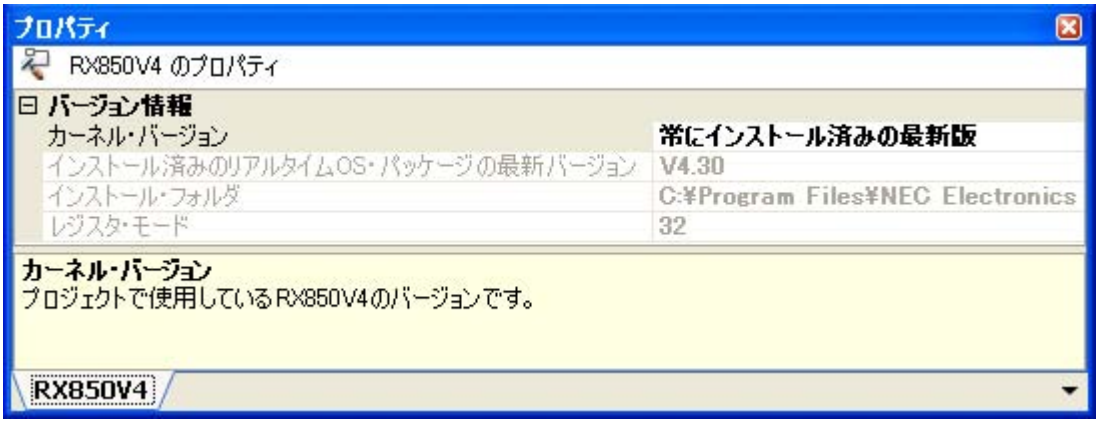
［RX850V4］タブ

概要

本タブでは，使用する RX850V4 に対して，次に示すカテゴリごとに詳細情報の表示を行います。

- バージョン情報

表示イメージ



機能

1）［バージョン情報］

RX850V4 のバージョンに関する詳細情報の表示を行います。

カーネル・バージョン	使用する RX850V4 のバージョンを選択します。 プロジェクト作成時には，デフォルトで［常にインストール済みの最新版］が選択されます。 バージョンの変更は，プロジェクト作成後に行うことができます。 なお，V850E2M のデバイスに対応した RX850V4 は V4.30 以降となります。 V850E2M のデバイスを指定したプロジェクトにおいて，本プロパティで V4.30 より前のバージョンを選択した場合は，選択前のバージョンに戻ります。		
	デフォルト	常にインストール済みの最新版	
	変更方法	ドロップダウン・リストによる選択	
	指定可能値	常にインストール済みの最新版	インストールしている RX850V4 のうち，最新バージョンを使用します。
		インストールしている RX850V4 のバージョン	選択したバージョンの RX850V4 を使用します。
インストール済みのリアルタイム OS・パッケージの最新バージョン	［カーネル・バージョン］プロパティで［常にインストール済みの最新版］を選択した場合に使用する RX850V4 のバージョンを表示します。 なお，本プロパティは，［カーネル・バージョン］プロパティで［常にインストール済みの最新版］を選択した場合のみ表示します。		
	デフォルト	*インストールしている RX850V4 の最新バージョン*	
	変更方法	変更不可	

インストール・フォルダ	使用する RX850V4 がインストールされているフォルダを絶対パスで表示します。	
	デフォルト	*使用する RX850V4 がインストールされているフォルダ*
	変更方法	変更不可
レジスタ・モード	プロジェクトで設定しているレジスタ・モードを表示します。 ビルド・ツールの［レジスタ・モードの選択］プロパティで選択しているレジスタ・モードと同じ値が表示されます。	
	デフォルト	*ビルド・ツールでのプロパティで選択しているレジスタ・モード*
	変更方法	変更不可

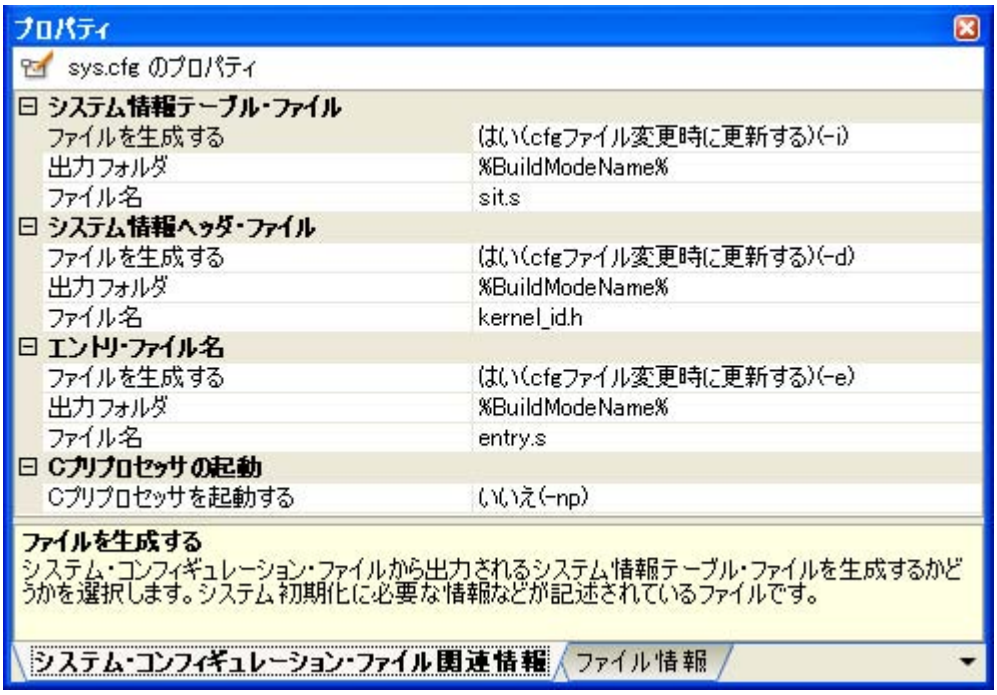
［システム・コンフィギュレーション・ファイル関連情報］タブ

概要

本タブでは，使用するシステム・コンフィギュレーション・ファイルに対して，次に示すカテゴリごとに詳細情報の表示，および設定の変更を行います。

- システム情報テーブル・ファイル
- システム情報ヘッダ・ファイル
- エントリ・ファイル
- C プリプロセッサの起動

表示イメージ



機能

1) ［システム情報テーブル・ファイル］

システム情報テーブル・ファイルに関する詳細情報の表示，および設定の変更を行います。

<table>
<tr><td rowspan="6">ファイルを生成する</td><td colspan="3">システム情報テーブル・ファイルを生成するかどうか，およびシステム・コンフィギュレーション・ファイルを変更した場合にシステム情報テーブル・ファイルを更新するかどうかを選択します。</td></tr>
<tr><td>デフォルト</td><td colspan="2">はい (.cfg ファイル変更時に更新する)(-i)</td></tr>
<tr><td>変更方法</td><td colspan="2">ドロップダウン・リストによる選択</td></tr>
<tr><td rowspan="3">指定可能値</td><td>はい (.cfg ファイル変更時に更新する)(-i)</td><td>システム情報テーブル・ファイルを新規生成し，プロジェクト・ツリーに表示します。
システム情報テーブル・ファイルがすでに存在するときにシステム・コンフィギュレーション・ファイルを変更した場合は，システム情報テーブル・ファイルを更新します。</td></tr>
<tr><td>はい (.cfg ファイル変更時に更新しない)(-ni)</td><td>システム・コンフィギュレーション・ファイルを変更しても，システム情報テーブル・ファイルを更新しません。
システム情報テーブル・ファイルが存在しないときに本項目を選択した場合は，ビルド時にエラーとなります。</td></tr>
<tr><td>いいえ (プロジェクトに登録しない)(-ni)</td><td>システム情報テーブル・ファイルの生成を行わず，プロジェクト・ツリーにも表示しません。
システム情報テーブル・ファイルが存在するときに本項目を選択しても，ファイル自体の削除は行いません。</td></tr>
<tr><td rowspan="4">出力フォルダ</td><td colspan="3">システム情報テーブル・ファイルを出力するフォルダを指定します。
相対パスで指定した場合は，プロジェクト・フォルダを基点とします。
絶対パスで指定した場合は，プロジェクト・フォルダを基点とした相対パスに変換されます（ドライブが異なる場合を除く）。
埋め込みマクロとして次のマクロ名があります。
%BuildModeName% : ビルド・モード名に置換します。
空欄にした場合は，マクロ名“%BuildModeName%”が表示されます。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-ni)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td colspan="2">%BuildModeName%</td></tr>
<tr><td>変更方法</td><td colspan="2">テキスト・ボックスによる直接入力，または［...］ボタンをクリックし，フォルダの参照 ダイアログによる編集</td></tr>
<tr><td>指定可能値</td><td colspan="2">247 文字までの文字列</td></tr>
<tr><td rowspan="4">ファイル名</td><td colspan="3">システム情報テーブル・ファイル名を指定します。
ファイル名を変更すると，プロジェクト・ツリーに表示しているファイル名も変更します。
拡張子は“.s”を指定してください。拡張子が異なる場合や省略した場合は，“.s”が自動的に付加されます。
［エントリ・ファイル］カテゴリの［ファイル名］プロパティと同じファイル名を指定することはできません。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-ni)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td colspan="2">sit.s</td></tr>
<tr><td>変更方法</td><td colspan="2">テキスト・ボックスによる直接入力</td></tr>
<tr><td>指定可能値</td><td colspan="2">259 文字までの文字列</td></tr>
</table>

2)［システム情報ヘッダ・ファイル］

システム情報ヘッダ・ファイルに関する詳細情報の表示，および設定の変更を行います。

<table>
<tr><td rowspan="6">ファイルを生成する</td><td colspan="3">システム情報ヘッダ・ファイルを生成するかどうか，およびシステム・コンフィギュレーション・ファイルを変更した場合にシステム情報ヘッダ・ファイルを更新するかどうかを選択します。</td></tr>
<tr><td>デフォルト</td><td colspan="2">はい (.cfg ファイル変更時に更新する)(-d)</td></tr>
<tr><td>変更方法</td><td colspan="2">ドロップダウン・リストによる選択</td></tr>
<tr><td rowspan="3">指定可能値</td><td>はい (.cfg ファイル変更時に更新する)(-d)</td><td>システム情報ヘッダ・ファイルを生成し，プロジェクト・ツリーに表示します。
システム情報ヘッダ・ファイルがすでに存在するときにシステム・コンフィギュレーション・ファイルを変更した場合は，システム情報ヘッダ・ファイルを更新します。</td></tr>
<tr><td>はい (.cfg ファイル変更時に更新しない)(-nd)</td><td>システム・コンフィギュレーション・ファイルを変更しても，システム情報ヘッダ・ファイルを更新しません。
システム情報ヘッダ・ファイルが存在しないときに本項目を選択した場合は，ビルド時にエラーとなります。</td></tr>
<tr><td>いいえ (プロジェクトに登録しない)(-nd)</td><td>システム情報ヘッダ・ファイルの生成を行わず，プロジェクト・ツリーにも表示しません。
システム情報ヘッダ・ファイルが存在するときに本項目を選択しても，ファイル自体の削除は行いません。</td></tr>
<tr><td rowspan="4">出力フォルダ</td><td colspan="3">システム情報ヘッダ・ファイルを出力するフォルダを指定します。
相対パスで指定した場合は，プロジェクト・フォルダを基点とします。
絶対パスで指定した場合は，プロジェクト・フォルダを基点とした相対パスに変換されます（ドライブが異なる場合を除く）。
埋め込みマクロとして次のマクロ名があります。
%BuildModeName% : ビルド・モード名に置換します。
空欄にした場合は，マクロ名“%BuildModeName%”が表示されます。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-nd)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td colspan="2">%BuildModeName%</td></tr>
<tr><td>変更方法</td><td colspan="2">テキスト・ボックスによる直接入力，または［...］ボタンをクリックし，フォルダの参照 ダイアログによる編集</td></tr>
<tr><td>指定可能値</td><td colspan="2">247 文字までの文字列</td></tr>
<tr><td rowspan="4">ファイル名</td><td colspan="3">システム情報ヘッダ・ファイル名を指定します。
ファイル名を変更すると，プロジェクト・ツリーに表示しているファイル名も変更します。
拡張子は“.h”を指定してください。拡張子が異なる場合や省略した場合は，“.h”が自動的に付加されます。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-nd)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td colspan="2">kernel_id.h</td></tr>
<tr><td>変更方法</td><td colspan="2">テキスト・ボックスによる直接入力</td></tr>
<tr><td>指定可能値</td><td colspan="2">259 文字までの文字列</td></tr>
</table>

3）［エントリ・ファイル］

エントリ・ファイルに関する詳細情報の表示，および設定の変更を行います。

<table>
<tr><td rowspan="5">ファイルを生成する</td><td colspan="3">エントリ・ファイルを生成するかどうか，およびシステム・コンフィギュレーション・ファイルを変更した場合にエントリ・ファイルを更新するかどうかを選択します。</td></tr>
<tr><td>デフォルト</td><td colspan="2">はい (.cfg ファイル変更時に更新する)(-e)</td></tr>
<tr><td>変更方法</td><td colspan="2">ドロップダウン・リストによる選択</td></tr>
<tr><td rowspan="3">指定可能値</td><td>はい (.cfg ファイル変更時に更新する)(-e)</td><td>エントリ・ファイルを生成し，プロジェクト・ツリーに表示します。
エントリ・ファイルがすでに存在するときにシステム・コンフィギュレーション・ファイルを変更した場合は，エントリ・ファイルを更新します。</td></tr>
<tr><td>はい (.cfg ファイル変更時に更新しない)(-ne)</td><td>システム・コンフィギュレーション・ファイルを変更しても，エントリ・ファイルを更新しません。
エントリ・ファイルが存在しないときに本項目を選択した場合は，ビルド時にエラーとなります。</td></tr>
<tr><td>いいえ (プロジェクトに登録しない)(-ne)</td><td>エントリ・ファイルの生成を行わず，プロジェクト・ツリーにも表示しません。
エントリ・ファイルが存在するときに本項目を選択しても，ファイル自体の削除は行いません。</td></tr>
<tr><td rowspan="4">出力フォルダ</td><td colspan="3">エントリ・ファイルを出力するフォルダを指定します。
相対パスで指定した場合は，プロジェクト・フォルダを基点とします。
絶対パスで指定した場合は，プロジェクト・フォルダを基点とした相対パスに変換されます（ドライブが異なる場合を除く）。
埋め込みマクロとして次のマクロ名があります。
%BuildModeName%：ビルド・モード名に置換します。
空欄にした場合は，マクロ名“%BuildModeName%”が表示されます。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-ne)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td colspan="2">%BuildModeName%</td></tr>
<tr><td>変更方法</td><td colspan="2">テキスト・ボックスによる直接入力，または［...］ボタンをクリックし，フォルダの参照 ダイアログによる編集</td></tr>
<tr><td>指定可能値</td><td colspan="2">247 文字までの文字列</td></tr>
<tr><td rowspan="4">ファイル名</td><td colspan="3">エントリ・ファイルを指定します。
ファイル名を変更すると，プロジェクト・ツリーに表示しているファイル名も変更します。
拡張子は“.s”を指定してください。拡張子が異なる場合や省略した場合は，“.s”が自動的に付加されます。
［システム情報テーブル・ファイル］カテゴリの［ファイル名］プロパティと同じファイル名を指定することはできません。
なお，本プロパティは，［ファイルを生成する］プロパティで［いいえ (プロジェクトに登録しない)(-ne)］を選択した場合は表示されません。</td></tr>
<tr><td>デフォルト</td><td colspan="2">entry.s</td></tr>
<tr><td>変更方法</td><td colspan="2">テキスト・ボックスによる直接入力</td></tr>
<tr><td>指定可能値</td><td colspan="2">259 文字までの文字列</td></tr>
</table>

4) ［C プリプロセッサの起動］

C プリプロセッサの起動に関する詳細情報の表示，および設定の変更を行います。

C プリプロセッサを起動する	コンフィギュレータ起動前に，システム・コンフィギュレーション・ファイルに対して C プリプロセッサを起動するかどうかを選択します。 システム・コンフィギュレーション・ファイルでマクロ定義を行っている場合は，［はい (-T)］を選択してください。		
	デフォルト	いいえ (-np)	
	変更方法	ドロップダウン・リストによる選択	
	指定可能値	はい (-T)	C プリプロセッサを起動します。 C プリプロセッサ起動時に参照するインクルード・パスは，C コンパイラで設定しているインクルード・パスを使用します。
		いいえ (-np)	C プリプロセッサを起動しません。

付録 B　浮動小数点演算機能【CX】

RX850V4 の CX 版では，V850E2M コアの持つ浮動小数点演算機能に対応しています。これにより，各処理プログラム（タスク，周期ハンドラ，割り込みハンドラなど）内で浮動小数点演算を行うことができます。

RX850V4 が浮動小数点演算に関連して操作するレジスタは，“レジスタ・バンク選択レジスタ BSEL”と“浮動小数点演算の設定 ／ステータス・レジスタ FPSR”です。これらの値はユーザが必要に応じて，各処理プログラム内で設定値を変更することができます。

基本的には BSEL と FPSR の値は各処理プログラムで独立しており，その値が処理プログラム間で継承されることはありません。

ただし，以下の場合は，BSEL と FPSR の値が処理プログラム間で継承されます。

- タスク上でタスク例外処理ルーチンが起動した場合は，タスク上の BSEL と FPSR の値がタスク例外処理ルーチンに継承されます。タスク例外処理ルーチン終了後は，起動前の値に戻ります。
- 直接起動割り込みハンドラと拡張サービス・コール・ルーチンの起動時や終了時には，RX850V4 から BSEL と FPSR の操作が行なわれません。このため，直接起動割り込みハンドラと拡張サービス・コール・ルーチンにおける BSEL と FPSR は，起動前の値を継承し，処理プログラム上で変更された値は処理プログラム終了後もそのままの値となります。

各処理プログラム初期起動時のレジスタ値に関しては，表 B － 1 を参照してください。

表 B － 1　各処理プログラム起動時のレジスタ値

処理プログラム名	BSEL 初期値	FPSR 初期値
タスク	00000000H	ユーザ設定値
タスク例外処理ルーチン	起動前の値を継承	起動前の値を継承
周期ハンドラ	00000000H	ユーザ設定値
割り込みハンドラ	00000000H	ユーザ設定値
直接起動割り込みハンドラ	起動前の値を継承	起動前の値を継承
拡張サービス・コール・ルーチン	起動前の値を継承	起動前の値を継承
CPU 例外ハンドラ	00000000H	ユーザ設定値
初期化ルーチン	00000000H	ユーザ設定値
アイドル・ルーチン	00000000H	ユーザ設定値

備考 1　BSEL の設定値 00000000H は，システム・レジスタ・バンクのグループ番号が CPU 機能グループ，バンク番号が基本バンクという意味です。

備考 2　タスク中断後にそのタスクへ復帰した場合，BSEL，FPSR はタスク中断前の値に復帰されます。

備考 3　FPSR のユーザ設定値は，システム・コンフィギュレーション・ファイルの“DEF_FPSR(FPSR 初期値)”で設定した値です。

付録 C　索　　引

A

C

D

E

F

G

I

L

P

R

S

T

U

V

W

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

た

ち

て

と

【発行】NEC**エレクトロニクス株式会社（**http://www.necel.co.jp/**）**
【問い合わせ先】http://www.necel.com/contact/ja/