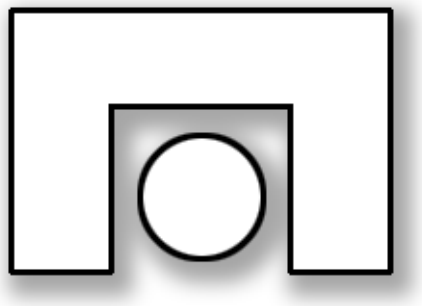
ITRON デバッギングインタフェース仕様

- ITRON Debugging Interface Specification -

Version 1.00.00

社団法人 トロン協会 ITRON 部会
ITRON デバッギングインタフェース仕様ワーキンググループ



ITRONデバッギングインタフェース仕様（Ver. 1.00.00）

本仕様ならびに本仕様書に関する問い合わせ先は，下記の通りです.

(社) トロン協会 ITRON部会
〒108-0073 東京都港区三田1丁目3番39号 勝田ビル5階
TEL: 03-3454-3191　FAX: 03-3454-3224

まえがき

μITRON 仕様は，組込みシステム用リアルタイムカーネルのデファクト標準仕様に発展してきたが，その弱点の一つに，μITRON 仕様カーネルに対応した開発環境が不十分であることが以前より指摘されてきた．μITRON 仕様カーネルとデバッグツール（デバッガや ICE など）との間のインタフェースを標準化することは，この弱点を補うための有力なアプローチである．実際，この標準化の必要性は 10 年以上前から認識され，一部で標準化に向けての取組みも行われたが，そもそもどのインタフェースを標準化すべきかが難しく，これまで成果を挙げることができなかった．

そのような状況の中で，1999 年 6 月には，ITRON プロジェクトの第 2 フェーズの成果の一つとして，ITRON カーネル仕様の決定版ともいうべき μITRON4.0 仕様を公開した．それを受けて，長い間懸案となっていたデバッグツールに関する標準化検討を本格化させることになった．実際には，μITRON4.0 仕様の内容がほぼ固まった 1999 年 2 月に，μITRON4.0 仕様研究会 デバッギングインタフェース仕様 WG の活動を開始した．

本書は，この WG での約 2 年に渡る検討を通じて作成された ITRON デバッギングインタフェース仕様の仕様書である．ITRON デバッギングインタフェース仕様は，デバッグツールが μITRON4.0 仕様に準拠したリアルタイムカーネルをサポートするための，標準的なインタフェースを規定するものである．その標準化アプローチとして，リアルタイムカーネルに新たな機能を追加するのではなく，デバッグツール側に埋め込んでリアルタイム OS のサポート機能を実現するモジュールを標準化するという独特のアプローチを採っている．

独特の標準化アプローチを採ったことから，実装を通じて仕様の妥当性を検証することが不可欠と考えられた．そこで昨年の 6 月には，その時点までの成果をまとめて暫定仕様書を公開し，各社にそれに準拠した製品の開発を促した．その結果現在までに，いくつかのメンバ社に，仕様に準拠した製品の開発ないしはその予備検討を開始していただくことができた．その結果をフィードバックして，仕様を完成度が高まったのは言うまでもない．今後，この仕様に準拠したデバッグツールならびに μITRON 仕様カーネルの開発が活発化することが期待される．

とは言え，組込みシステム向けには多種多様なデバッグツールが開発されており，この仕様がそれらのすべてに効率よく対応できることを完全に検証できたわけではない．ITRON カーネル仕様も，最初に検討を開始して以来，バージョンを上げる度に完成度を上げたきたことを考えると，ITRON デバッギングインタフェース仕様についても，まだまだ完成度を上げる余地があると考えられる．

最後に，この暫定仕様書を作成するにあたってご協力いただいたデバッギングインタフェース仕様 WG のメンバ各位に感謝の意を表したい．特に，幹事として議論を引っ張ってくださった山中勝（レッドハット），石井秀弘（YDC），権藤正樹（エルグ）の各氏，仕様書の執筆を担当した若林隆行君（豊橋技術科学大学）の貢献なくしては，この仕様は完成しなかったと思われる．深く感謝する次第である．

2001 年 5 月

高田 広章

(社)トロン協会 ITRON 部会 幹事／
豊橋技術科学大学 情報工学系

I. 目次

II. 表目次

III. 図目次

IV. 関数目次

1. 本文書内での書式

1.1 表記

本文書中では，表記の通りに記述する必要があるものは，下で示すように表記される (太字斜体ゴシック). このように表記されるものとして，コマンド名，構造体名，定数名などがある.

% ***command, T_RSBRK, rif***_xxx_yyy など

この文書内でプログラムコードは次のように表記される.

Program source
```
Program code
```
Program source

サービスコールは次のように表記される.

名称　　機能の概要　　[区分] マーク

プロトタイプ

引数の型　　引数の名称
　　引数の意味

(返値) 返値の型　　返値が ER 以外である場合，その意味に相当する名称
　　返値の意味

説明

機能に関する説明

補足説明

パラメータやサービスコールの補足説明

フラグ

フラグ名称　　フラグに関する説明
フラグ名称　　デフォルトフラグに関する説明 **(default)**

Extension

このようなバナーが上下につけられている機能は拡張機能である．TIF を構成する必須関数 [R] の説明の中で，拡張機能として扱われる部分を示している.

Extension

キー

キー名　　意味 (後述の情報取得キーコードに関する書式を参照)

エラー

エラー定数名　エラーの説明

配列および配列のメンバは次のように表記される.

配列の型名 {
　型名 名称　　: 説明
　型名 名称　　: 説明 (実行で値が書き換えられる可能性のある変数)
}

区分は RIF は機能単位ごとに [OBJ][CTX][SVC][BRK][CND][LOG], TIF は要求レベルに応じて [R][E] と区分している. 各インタフェースのコールバックも区分の中に記述され, [xxx:callback] と表現される. 区分の詳細に関しては 4.2 [レベル表示] を参照のこと.

本仕様書では 2 つの記号 (マーク) を用いて, API を実装しなければならない個所を明確にする. 記号「○」は RIM 側での実装を意味し, 記号「□」はデバッグツール側での実装を意味する.

情報取得キーコードに関しては, 次のような表記を行う. キーコードに関する詳細は 3.6 [情報取得構造体と取得キー] にて述べる.

第 1 キー **値 [型]**
　このキーで取得できる情報の説明
. 第 2 キー **値 [型]**
　このキーで取得できる情報の説明
. 第 3 キー **値 [型]**
　このキーで取得できる情報の説明
. 第 4 キー **値 [型]**
　このキーで取得できる情報の説明

型は次のような書式を持つ.

Table 1: **書式とキーコードの型**

表記	型
W	32 ビット符号付整数
S	文字列
T	構造体をはじめとする特殊な型
1	ブール値 (FALSE → 0, TRUE → 0 以外) (実際は 32 ビット符号付整数 [W])

1.2 命名規約

1.2.1 変数名 引数名

デバッギングインタフェースに含まれる関数内で用いられている構造体内変数名および関数の引数名は, 次のような基準に基づいて名前がつけられている.

変数名は次のようになっており，全て小文字で表現する．

変数名 := [* 接頭語 "_"] ((補足説明 * 説明 [* 接尾語]) | (固有名))

定数名は次のように表記され，全て大文字で表現する．

定数名 := *(種別 "_") < 意味が特定できるような文字列 >

種別には次のようなものがある．

ACS	アクセス方法を設定するためのフラグ
FLG	複数の関数で用いられる共通のフラグ
OPT	関数の機能にヒントを与えるためのオプション定数
OBJ	オブジェクトの種別を特定するフラグ
BRK	ブレーク関連の定数
E	エラーコード
ET	ターゲット上のエラーコード
DSP	ディスパッチャに関する定数
EV	イベントコード
LOG	ログ

構造体名は次のように表記され，全て大文字で表現する．

構造体名 = "***T***_" ([インタフェース] < 関数名 xxx_yyy の最初の一文字 > 説明) | < 理解可能な文字列 >

他の構造体のメンバとして利用される構造体 (入れ子の構造体) は，次のように表記される．

構造体名 = < この構造体を内包する構造体名 > **"**_" < メンバにつけられた名称の大文字表記 >

また構造体のタグ名として，構造体名の小文字表記で表現する．具体的には，構造体 ***T_ROSEM*** のタグ名は ***t_rosem*** である．

1.2.2 接頭語

次の接頭語で始まる変数は，その構造や用法に意味があることを示す．

Table 2: **接頭語**

文字	意味
p	この変数には値が格納され，内容が変更される
pk	構造体の実体である
str	NULL 終端文字列である

1.2.3 補足説明

名称の直前につけられる補足説明文字は，対象となる変数が持つ意味を補足する．

Table 3: **補足説明**

文字	意味
w	待ち状態
s	送信
r	受信
f	空き
c	呼び出し (ランデブポート)
a	受け付け (ランデブポート)
run	実行中

1.2.4 説明

次の文字は，その変数が持つ意味を示す．これには Table 8: [xxx と yyy の表記] で示す語句を用いる場合もある．

Table 4: **説明**

文字	意味
id	ID 番号
blk	ブロック
stat	状態
pri	優先度
obj	μITRON オブジェクト
sem	セマフォ
tsk	タスク
type	型情報フラグ
opt	オプション項目
ptn	ビットパターン
dtq	データキュー
msg	メッセージ
mbf	メッセージバッファ
sz	長さ

Table 4: **説明**

文字	意味
fn	機能コード
prm	パラメータ
ptr	ポインタ

長さを示す ***len*** と ***sz*** の違いは，単位である．***len*** は各項目要素の大きさを単位とし，***sz*** はバイトを単位とする．

接尾語 - 次の文字で終了する変数は，その用法やデータそのものに意味があることを示す．

Table 5: **接尾語**

文字	意味
adr	アドレス
cnt	個数を格納する
lst	リストを格納する
ptr	情報を格納するポインタ
ofs	オフセット
len	長さ

接尾語 ***adr*** と接尾語 ***ptr*** の違いは，その変数のもつ意味にある．接尾語 ***adr*** を持つ変数は，アドレス自体に意味がある項目に付けられる．具体的には，ブレーク位置 (***brkadr***) がある．一方，アドレス自体に意味はなく，そのアドレスの指し示す先の情報に意味がある項目に付けられた変数名には ***ptr*** が付けられる．具体的には，バッファへのポインタ (***bufptr***) がある．

接尾語 ***cnt***,***lst*** の対は，関数 ***rif_ref_obj*** では特殊な機能を持つ．詳しくは 5.2 [オブジェクト状態の取り出し] を参照されたい．

固有名 - 次の名前はそれらが固有の意味を持つ変数であることを示す

Table 6: **固有名**

文字	意味
result	結果を格納している変数
storage	データを保持する領域など (主として書き込み用)
param	パラメータ
flags	フラグ変数 / 引数

Table 6: **固有名**

文字	意味
name	名称
length	長さ (構造体内で単一の場合)

インタフェース識別文字 - 次の文字はその構造体が用いられるインタフェースを明確化する

Table 7: **インタフェース識別文字**

文字	意味
R	RTOS アクセスインタフェース
T	ターゲットアクセスインタフェース

ただし，次のような場合に限り，インタフェース識別文字は省略される．

- **両インタフェースで共通な構造体である**
- **両インタフェースに属さない，独立した構造体である**

1.2.5 関数名

デバッギングインタフェースに含まれる関数は全て www_xxx_yyy の形 (ソフトウェア部品の名付け基準) をとる．www はインタフェースごとにその名称が入り，RTOS アクセスインタフェース上の関数には ***rif*** が，ターゲットアクセスインタフェース上の関数には ***tif*** が入る．また，RIF および TIF のどちらにも属さない関数については ***dbg*** を用いる．
xxx, yyy に関しては次の表を参照されたい．

Table 8: xxx **と** yyy **の表記**

略字	本来の意味	意味
alc	allocate	割当
brk	break	中断
cal	call	呼び出し
can	cancel	取り消し
cfg	configure	構成
fin	finalize	終了処理
fre	free	開放
get	get	取得
hok	hook	フック関数の登録
ini	initialize	初期化

Table 8: xxx と yyy の表記

略字	本来の意味	意味
pol	poll	ポーリング
ref	refer (forward)	参照
req	request	要求
rep	report	報告 (含 コールバック)
rrf	refer (backward)	逆参照
rst	reset	リセット
set	set	設定
sta	start	開始
stp	stop	停止
bls	block set	メモリブロックセット
brk	break point	ブレークポイント
cfg	configuration	設定情報
cnd	condition	条件
ctx	context	コンテキスト
dgb	debug tool	デバッグツール
fnc	function	関数
log	trace log	トレースログ
mbh	memory block on host	ホスト側のメモリブロック
mbt	memory block on target	ターゲット側のメモリブロック
mem	memory on target	ターゲット上のメモリ
rdt	register set description table	レジスタ情報テーブル
reg	register	レジスタ
rim	RTOS interface module	RTOS インタフェースモジュール
stp	stop by break point	ブレークによる停止
svc	service call	サービスコール
sym	symbol	シンボル
tgt	target	ターゲット

www_***rep***_yyy は特殊な意味を持ち，これらはインタフェース www におけるコールバック関数として扱われる．

実装者が独自に追加している関数，および本仕様で定義されていない特殊な関数などの場合，接頭語 v を xxx の前につけ，それが独自であることを明記することを推奨する．(µITRON4.0 と同様) (例：***tif_vcal_svc***)

1.3 用語

本文章および本仕様では次のような用語を用いる．

Table 9: **用語一覧**

単語	意味
ターゲット	デバッグ対象となるプログラム およびそのプログラムを格納しているハードウェア
デバッグツール	デバッグを行うためのハードウェア / ソフトウェアを指す．(ホストコンピュータ , プローブ , デバッグ機能を提供するアプリケーションなど)
ガイドライン	仕様中 それに従う必要を強く望むものではないが，そうあるのが望ましいもの．弱い標準規約という意味合い．
エージェント	ある目的のために導入されるターゲット上のサポートプログラムのこと

1.4 名称の省略

本文章では，長い名称や頻繁に現れる名称に対して省略形を用いて表記する．

Table 10: **省略形**

省略形	元の意味
RIF	RTOS アクセスインタフェース
TIF	ターゲットアクセスインタフェース
RIM	RTOS インタフェースモジュール
レジスタテーブル	レジスタセットディスクリプションテーブル

2. 概要

2.1 背景

現在，さまざまなものにコンピュータが利用されている．その大部分を占める組込みの分野では，その製品の数もさることながら，より高度な機能を搭載すべく徐々にソフトウェアの規模も大きなものになっている．一方，製品の開発から市場投入までの期間 (Time to market) は短くなり, 短期間で大規模なアプリケーションソフトウェアの作成が要求されている．
短期間で大規模なソフトウェアを開発するためには，開発環境の向上，特にデバッグ環境の向上が必須である．全工程の大部分を占めるソフトウェアのテスト / デバッグに必要な時間は正確に把握することは難しく，一度問題が発見された場合，その問題を修正するために費やされる時間は，デバッグを行う者の経験とツールの性能に大きく左右される．
アプリケーションが OS を利用する場合，デバッグツールが OS をサポートするか否かは重要な要素である．デバッグを行う者にとって，現在注目しているコードとは無関係である OS 内部のコードへのステップインや，タスク状態などを人間に理解可能な形で表示できないことは，生産性を低下させる要因となる．
組込み応用の分野において，一般の OS の機能に加え，リアルタイム性を重視した Real-Time Operating System(以下 RTOS) が広く用いられている．Table 11: [最近開発した組込み機器に使用した OS] は，RTOS のシェアに関する 99 年度の調査結果である．日本国内において ITRON 仕様 OS は全体の 30% を超えるシェアを占めている．

Table 11: **最近開発した組込み機器に使用した** OS

区分	割合
市販の ITRON 仕様 OS	18.8%
自社製 ITRON/BTRON 仕様 OS	12.0%
CTRON 仕様 OS	1.0%
その他 市販の独自仕様 OS	40.4%
問題があるので用いていない	3.5%
必要が無いので用いていない	24.3%

単一の ITRON 仕様 OS をサポートするデバッグツールを作成するのはそれほど難しいことではなく，既に製品が存在する．しかし，全ての ITRON 仕様 OS をサポートするのは容易ではない．これは，ITRON 仕様というものが API を定めた仕様でなので, それぞれの OS を実装するための手法によって内部の構造が変化するからである．内部構造に依存するデバッグツールは，新しい ITRON 仕様 OS がリリースされる度に，RTOS に関連するモジュールを書き直さなければならない恐れがある．
ITRON 仕様 OS の開発環境には，もうひとつの問題が存在する．それは，ITRON 仕様 OS は組込み用チップを作成したチップメーカから提供されるのに対し，OS 用デバッグツールはツールの作成を専門に行うツールベンダから提供されるという点である．そのため，ツールと OS の整合性を保つことが難しい．単一の会社が OS から

デバッグツールまでを提供するのであれば問題ないが，異なる会社に属する部門間で行うとなると協調して開発することは困難である．このことより，ITRON 仕様 OS 上で動作するプログラムに高いポータビリティがあっても，ユーザーはデバッグ環境が変わることを恐れ，最新の ITRON 仕様 OS に手を出しにくい．
上記のような問題を持つ ITRON 仕様 OS に対して，標準的なデバッギング手法を提供し続けることは困難であった．そのため，ITRON 仕様 OS は開発環境が不足していると指摘されてきた．日本国内における 1999 年度のアンケートでは，国内で 30% 近いシェアを保持していながら，20% 以上の技術者がこの問題を指摘している (Table 12).

Table 12: ITRON **仕様** OS **の短所**

指摘点	**割合**
開発環境 / ツールの不足	22.9%
依存性が高く，移植性が悪い	12.9%
ソフトウェア部品が少ない	11.5%
技術者不足	7.8%
機能不足	4.4%
OS の要求リソースが大きい	4.4%
その他	18.9%
目立った短所は無い	17.2%

この問題を解決するためには，RTOS とデバッグツール間の橋渡し役としてのインタフェースを標準化することが必要である．これにより，複数のデバッグツールと複数の ITRON 仕様 OS との如何なる組合せも可能となり，自由度の高い RTOS レベルのデバッギング環境を提供することが出来る．
本書で述べる ITRON デバッギングインタフェース仕様は，1999 年 2 月から開始された ITRON デバッギングインタフェース仕様ワーキンググループの活動の成果である．

2.2 標準化の目的と目標

ITRON デバッギングインタフェース仕様ワーキンググループの主たる目的は, デバッガに RTOS サポート機能を追加するためのインタフェース制定である．
またこの目的を達成するにあたり，特に考慮すべき項目を目標として掲げている．以下にその目標を示す．

- **高いスケーラビリティを持たせる**
  8bit の低速プロセッサから 32bit の高速プロセッサまでに対応
- **様々なデバッグ環境を想定して仕様を作成する**
  ソフトデバッガ , ICE, JTAG エミュレータ , ソフトウェアエミュレータなどでも共通に使える
- **できる限り ITRON 仕様 OS に限定しないようインタフェースを作成する**
  他の RTOS やソフトウェア部品もデバッグ可能にする

2.3 標準化のアプローチ

実際にインタフェース仕様を定めるに当たり，さまざまな方向からインタフェース仕様を検討した．ここでは，仕様検討の際に討議が行われた本インタフェース仕様のアプローチ案とその利点および欠点，ならびに採択理由などについて述べる．

2.3.1 アプローチ案

アプローチ 1 : オブジェクト情報を固定する

アプローチ 1 は μITRON 仕様で定められているオブジェクトの状態を保持しているコントロールブロックを，現在の名称のみの標準化から，より強いバイナリレベルでの標準で縛り，そのブロックの格納場所やアライメントなどを一意に定めることにより，複数の OS 間での互換を図る方法である．

- **利点**
  デバッグツール上にほとんど改造を加えることなく実装が可能となる
- **欠点**
  現在市販されている OS は ほぼ非対応となる
  CPU アーキテクチャに依存する
  各社の独自性を失わせる

アプローチ 2 : ターゲット側にサポート機能を実装する

この方法は，RTOS ごとに異なる情報をターゲットから取得する時点で標準化してしまう方法である．その機能が実装される場所によって次の 3 種類に分けることができる．

タスクとして実装
サポート機構がタスクとして導入される．ターゲット内にメモリ管理ユニット (以下 MMU) があると, サポート機能がタスクで実現されているため OS 内部の情報は読めないなどの弊害があるが，他のタスクに影響を及ぼしにくい．

RTOS 内部に実装
サポート機構が RTOS 内部に直接導入される．詳細な情報を得ることができ，MMU の影響も受けないが，他のタスクに影響を及ぼしやすい．

スタブを拡張する
ターゲットのデバッグに用いるスタブを拡張する方法．RTOS の動作にも影響を及ぼしやすい．

- **利点**
  既存の OS を含め，応用できる OS の範囲が広い
- **欠点**
  ターゲットに負担をかける (CPU/ メモリリソースとも)
  ターゲット上に MMU や保護機構などが入った場合，カーネル内にサポート関数などを設ける必要があり，構造が複雑化する

アプローチ 3 : サポートモジュールをデバッグツール内に導入する

この方法はデバッグツール内にRTOS情報を取得するための機能を持ったモジュールを組み込み，それに伴う一連の機能を標準化する手法である．

- **利点**
  既存のものを含め，さまざまなOSに対応可能

  ターゲット上の負担が最小限になる
- **欠点**
  デバッグツールにモジュールを内包できる柔軟さが必要

2.3.2 アプローチの選定理由

上記 3 つのアプローチのうち，ITRON デバッギングインタフェース仕様ワーキンググループでは，上記アプローチ 3「サポートモジュールをデバッグツール内に導入する」という手法を採択した．その最たる理由は，従来の設計から本仕様を利用した設計へ移行することが容易であるためである．

これまで作成されてきているデバッグ環境の大抵のものは，ターゲット側にRTOSレベルのデバッグサポート機構を多く盛り込むことにより(前出のアプローチ 2), RTOSレベルのデバッグを可能にしている．
このことを踏まえると，上記アプローチ 3 の場合に，新たにサポートモジュールをデバッグツール内部に組み込まれなければならないとしても，デバッグツールは今までターゲット上にあった関数がホストに移っただけに等しい．一方，RTOS メーカは記述先がRTOSカーネル内部やデバッグサポートタスクからサポートモジュールに移るに過ぎない．そのため，デバッグツール /RTOS 両ベンダともに，これまで培われて資産を無駄にせず新しい環境に移行できると考えている．

このアプローチは従来のアプローチと競合しないため，RTOS メーカは必要に応じてこの機構を搭載することも可能である．すべての機能をサポートモジュールだけで行うことが不可能である場合や，導入によって高いスケーラビリティを実現することが可能な場合などは，ターゲット内部にサポート機構を設けることになる．
このような場合においても，アプローチ 3 はホスト - ターゲット間の情報転送量の低減にも効力を発揮する．アプローチ 2 では，ターゲットからデバッグツールに至るまでの間に情報が標準化されていなければならないという問題がある．しかし，本手法を用いることで，情報はホストコンピュータ内部に存在するサポートモジュール前後で標準化されいれば良い事になる．よって，ターゲット内に存在するデバッグサポート機構は，必要最小限の情報のみをデバッグツールと交換するだけである．最終的にサポートモジュールが内部情報を展開し，標準形に整形すれば，ターゲットコンピュータへの最小限の転送量と最小限の負荷で既存のデバッガと同様の操作が行える．

以上の点より，ITRON デバッギングインタフェース仕様ワーキンググループではアプローチ 3 を主体とすることとした．

2.4 概念

ITRONデバッギングインタフェース仕様は，μITRON仕様OSを利用したアプリケーションにおけるデバッギング環境改善のための仕様である．

デバッギングインタフェースの概念図を下に示す.

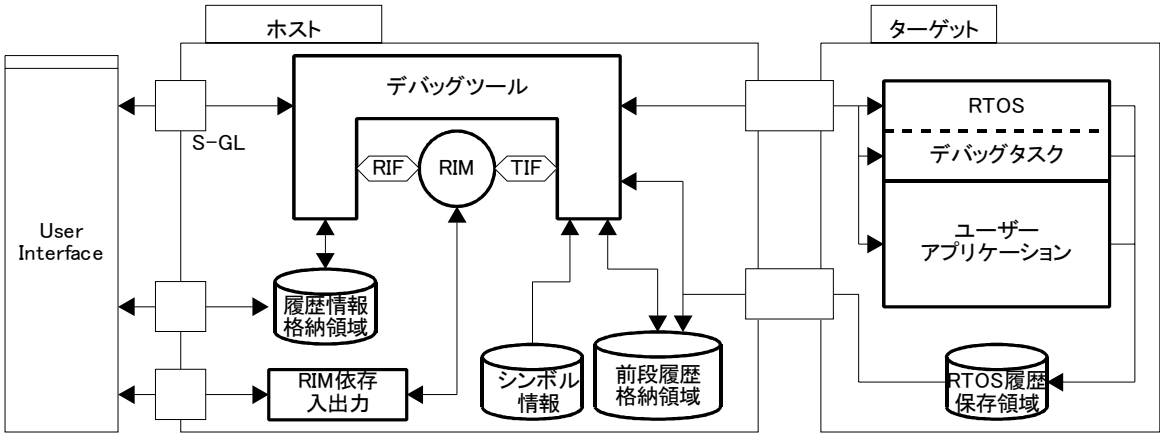


Figure 1: ITRON **デバッギングインタフェース仕様 概念図**

ITRON デバッギングインタフェース仕様では, ホスト側がターゲット上の RTOS 依存情報を容易に取得できるように，1 つのモジュールの概念と，それに関連する 2 つのインタフェースと 1 つのガイドラインの規定を設けている.

- **サポート機能ガイドライン**
  デバッグツールに搭載される "RTOS サポート機能 " に関する機能や，その詳細を決めるガイドライン．このガイドラインによって本仕様をサポートするデバッグツール間で用語や，類似した機能を統一し，ユーザーに対して最低限の機能を保証することができる．後述の 2 つのインタフェース (RIF，TIF) もこのガイドラインに沿って制定されている.

- **RTOS インタフェースモジュール (RIM)**
  RTOS の内部情報をデバッグツールに通知し，デバッグツールが理解できない RTOS 依存命令をデバッグツールが理解可能な命令へと翻訳するモジュール．RTOS メーカによって提供され，デバッグツール本体内部に組み込まれる. (提供の方法として C 言語ソースプログラム, および Windows-DLL などがある) 本仕様の中核を成すモジュール.

- **RTOS アクセスインタフェース (RIF)**
  デバッグツールが RIM の機能を利用して，RTOS に依存するようなデバッグを行うとき，利用されるインタフェースが RIF である．デバッグツールに RTOS の現状を知る手法を提供する．C 言語 API の形で定められた合計 21 個の関数 および コールバック関数から構成されている. "RTOS オブジェクトの詳細取得 " や，" コンテキストの取得 " などの機能を提供する.

- **ターゲットアクセスインタフェース (TIF)**
  RIM は, RIF を通じて発行されたデバッグツールの要求を実現するために, デバッグツールの機能を用いてターゲットや RTOS 内部にアクセスする必要がある．ターゲットアクセスインタフェースはそのような状況に対処するため，デバッグツールが持つ機能を RIM に提供するための，基本的なデバッグツールの機能を定めた合計 31 個の関数 および コールバック関数からなるインタフェースである．" メモリ読み出し / 書き出し " " 実行 / 中断 " などの機能を提供する.

その他のモジュールに関しては次の通り

- **前段履歴格納領域**
  トレースログを取得している最中に，一時的にログ情報を格納しておくための領域.
- **標準情報格納領域**
  トレースログなどをITRONデバッギングインタフェース標準フォーマットで格納している領域．デバッグツールの支援がなくとも，標準フォーマット対応ビューワのみで閲覧が可能となる．
- **RIM 依存入出力**
  RTOS が高度なデバッグオプションなどを持っている場合や，独自の実装依存情報を扱うような場合，デバッグツールが与える標準的な情報の入力だけでは不十分となることもある．このときに利用されるのがこの RIM 依存入出力である．この RIM 依存入出力ではデバッグツールのユーザーインタフェースの一部を標準化し，RIM が独自にユーザーと対話することを可能とする．(現仕様ではサポートしていない)

ITRON デバッギングインタフェースでは，デバッグツールと，デバッグツールに内包されるRTOSインタフェースモジュール(RIM)との通信によって,目的である"RTOSをサポートしないデバッグツールの RTOS 対応 " を達成する．以下に例を挙げてその動作原理を紹介する．

2.4.1 動作例

ここで，例として "ID1 のタスクの状態取得 " を挙げる．

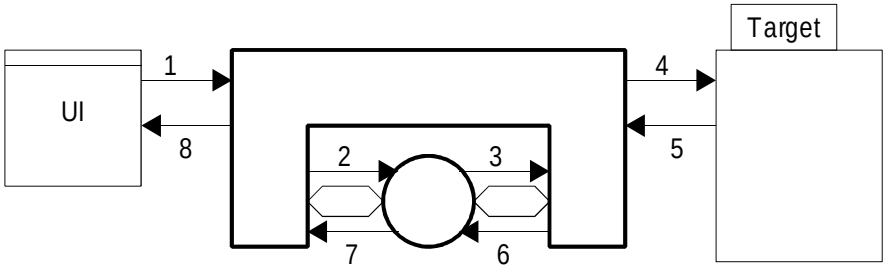


Figure 2: ID1 **のタスクに対する状態取得の様子**

1. ユーザから "ID1 のタスク状態を取得したい " という要求が発生する
2. RTOS 内部状態を参照するため，デバッグツールは RIM に "ID1 のタスク状態取得 " を要求する
3. RIM はタスク ID1 に相当する TCB(タスク制御ブロック) のアドレスをシンボルテーブルなどから参照し，該当するターゲット上のメモリを読み出すようデバッグツールに要求する
4. デバッグツールは既存の機能を用いて，指定されたターゲット上のメモリを読み出す
5. 読み出されたメモリの内容がデバッグツールに通知される
6. デバッグツールは、3 の結果として読み出した内容を RIM に伝える

7. RIM は TCB の内容をもとにその内容をデコードし，標準化された形式でデバッグツールに通知する
8. 得られた結果を基に，"ID1 のタスク状態 " を画面に表示する

RTOS と RIM をセットで提供することで，デバッグツールは RTOS の内部構造の詳細を知ることなく，RTOS 依存情報にアクセスすることが可能となる

2.5 特徴

この節では, ITRON デバッギングインタフェース仕様で特徴的な部分を取り上げ, その詳細を述べる.

2.5.1 2 種類のタスク ID によるブレーク手法

デバッグツールの RTOS サポート機能において，ある特定のタスク ID を持つタスクがある特定の動作を行ったときにブレークを行うという，特定タスクに対するブレーク機能は重要である．特に同一のモジュールを複数のタスクで共有するような場合，タスクの数が増えるにしたがって，特定のタスクとなるまで人間が実行再開を行うという本質的でない作業が多くなる．
ITRON デバッギングインタフェース仕様では，目標のひとつである複数のデバッグツールを対象にするという要求を満たすために，2 種類の特定タスクに対するブレーク機能を搭載している．これは，高機能のデバッグツールが低機能デバッグツールを想定して標準化されている機能を利用することで，本来の機能を発揮できず，低機能化してしまうことを防止する目的も含まれている．

手法 1: RIM を利用した，コールバックルーチンによるブレーク

タスク ID などの RTOS 依存情報をとることのできないデバッグツールを，RIM によって RTOS サポートデバッグツールにするのは前に述べたとおりである．この手法はインタフェースの中核である RIM のブレーク発生コールバックを使ってタスク ID 依存ブレークを行う．

この機能を利用するデバッグツールは，次のような種類のものを想定している．

- **特定アドレスに対する実行ブレーク機能を持っている**
- **条件ブレーク機構はもっていない**
- **RTOS に依存する機構はもっていない**

実例を挙げて本機能の詳細を説明する．ここではタスク ID が 1 であるタスクがアドレス 0x12345678 を実行したときにブレークが発生するように設定したとする．

1. デバッグツールは RIM に対して, タスク ID1 に対してアドレス 0x12345678 の実行ブレークを設定するように要求する
2. RIM はデバッグツールに対してアドレス 0x12345678 に対する実行ブレークを設定する
3. ユーザーによりプログラムが実行される
4. プログラムがアドレス 0x12345678 を実行し，デバッグツールによりブレークする
5. コールバック関数により，ブレークしたことがデバッグツールから RIM に通知される

6. RIM は現在実行中のタスク ID を格納している領域などを調べ，停止するべきなのかを判断し，デバッグツールに通知する
7. RIM が条件を満たしていると通知すれば，デバッグツールはブレーク動作を完了させ，ユーザに通知する．そうでなければブレーク動作を中断し，再度プログラムを再開する

このブレーク手法の特徴は，デバッグツールがそれほど高機能でなくともタスク ID によるブレークを可能にできる点と，RIM がブレーク判断を行うことで，ほぼ全ての RTOS に対応できる点である．
また同時に欠点として，大量のブレークを設定した場合，コールバックの回数が増え，ホスト側にかかる負担が大きくなるという問題がある．特にシリアルポートなどを使ったリモートデバッグなどの際，ブレークするたびにタスク ID の確認動作が入るため，かなりのオーバーヘッドとなる．
ブレーク通知の際に，RIM は停止すべきと判断しなかったとしても，その判断中ターゲットは停止状態となる．このことは時間制約の厳しいプログラムではこの余計なブレーク時間が引き金となって誤動作を招いたり，逆にデバッグすべきエラーが見つからなくなるなどの危険性がある．

手法 2: ブレーク条件を取得し，デバッグツール本体による条件ブレーク

この手法はデバッグツールが条件ブレーク機構をすでに搭載している場合に用いる手法である．RIM は要求に対し，それと等価となる条件をデバッグツールに通知する．

この機能を利用するデバッグツールは次のような種類のものを想定している．

- **特定アドレスに対する実行ブレーク機能を持っている**
- **条件ブレーク機構を搭載している**
- **RTOS に依存する機構はもっていない**

この手法では，RIM は自分でブレークポイントを設定しない．このとき RIM が行うのは，先ほどの例では自分が行っていた条件判断と等価な条件ブレークの条件式を生成して，これをデバッグツールに返す動作のみである．デバッグツールはこれによって得られた条件に，さらに必要に応じて条件式を追加し，デバッグツールにより直接条件ブレークを設定する．
この方法では先ほどの例とは異なり，ブレークヒット時にもコールバックは行われないため，非常に高度な条件ブレーク機構をもったデバッグツールでは，ほぼ余分なオーバーヘッドを生むことなく RTOS 情報に依存するようなブレーク機構を実現することが可能となる．

現在この機能で利用できる条件は次のようなものである．

- **メモリアドレス**
- **データのバイト長**
- **値**
- **条件 (等価，大小，不一致)**

Figure 3: [2 種類のブレーク手法と動作フローの違い] に，2 つのブレーク種別のプログラムフローの違いを示す．実線で示された部分は，異なるプログラム間 (RIM - デ

バッグツール間，ターゲット - デバッグツール間) での処理を示し，この本数が多いほどプログラムのオーバーヘッドが大きいと予測できる．実線部分は同一プログラム内を示す．矢印の番号はそれぞれブレークポイントを設定するときの流れ，停止すべきかを判定するときのプログラムの流れ，ブレークヒットと判断し，ブレークヒットをユーザーに通知するまでの流れを示している．

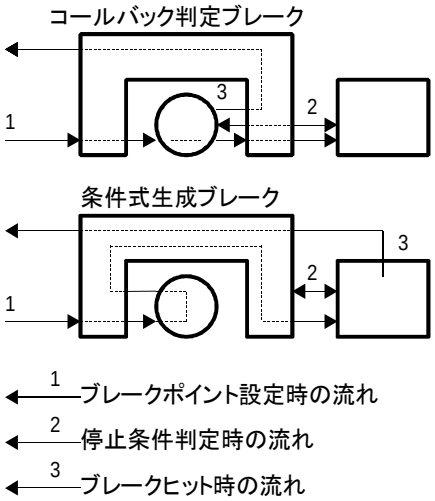


Figure 3: 2 **種類のブレーク手法と動作フローの違い**

2.5.2 スケーラブルなデバッグ環境

組込み応用の分野で特徴的なのが，状況によって発生しないバグが多いことである．たとえば，タイムクリティカルなタスクがある特定のタイミングで実行されたときのみバグが発生する状況がありうる．I/O 書込み後の I/O 読込みウェイト無視などがこれに該当し，読み出し直前にブレークポイントを設定し，ステップ動作で読込みを行った場合，ブレークによって I/O 読出しまでに十分なウェイトが置かれてしまうためにエラーとならないが，実環境と同様に実行した場合，これはエラーとなる．
このように時間で左右されるバグでは，デバッグツールの機能が豊富であることが裏目に出るときもある．ITRON デバッギングインタフェースでも安易な RIM 実装を行うことにより，前述のブレークサポート機能などでは，判定のために RIM とターゲットの間を往復する時間によって，これらのバグが出現しない可能性がある．
これに対し ITRON デバッギングインタフェースのアーキテクチャでは，RIM と RTOS が共に RTOS メーカから提供されるために，状況に応じて複数の RIM と RTOS のセットを提供し，利用者の環境に合わせた組み合わせを選択することも可能である．
RIM と RTOS のどちらにデバッグサポート機能を多く搭載するかで，まったく同じ機能を提供するとしても，次のような特性を持たせることができる．

•RIM 側重視の実装

RIM に多くの機能を搭載することで，リリース時に限りなく近い環境でアプリケーションのデバッグを行うことができ，ターゲットにかかる負荷を低くすることができる．

•RTOS 側重視の実装

RTOS 側に多くの機能を搭載することで，デバッグツールとターゲットとの間の通信時間を限りなく減らすことができるため，より高速なレスポンスが可能となる．

RIM と RTOS がソースコードで提供される場合，アプリケーションに応じて自由に再構成可能な RTOS に対して，RIM 自体も再構成することも手法の一つとしてあげることができる．RIM に存在する使わない機能や，RTOS 内部の余計なデバッグサポート機能にメモリを割くことを避けるため，必要な機能だけをサポートするよう RTOS 自体を再構築すると同時に，この RTOS をデバッグするのに最適な，余計な機能をもたない RIM を生成すれば，デバッグ時には無用なオーバーヘッドを最小限にすることができる．

具体例として，高速なブレークをサポートしたい場合を挙げる．高速なブレークを行うためには，現行デバッグ環境のボトルネックであるターゲットとデバッグツール間の通信量を，極力減らすことが求められる．本仕様において全機能を RIM 側重視で実装した場合，RTOS に依存する条件を判定するのはデバッグツール内のモジュールであるため，ターゲット - デバッグツール間の通信がボトルネックとなる．高速なブレークを行う際，通常の方法ではこの問題のために目的を達成できない．
この問題を解消するためには，RIM に搭載されていたブレークヒット判定ルーチンを RTOS 内部に組み込むという手法がある．この手法では，RTOS 内部にタスク ID 判別ルーチンを埋め込んでしまい，本来ブレークを置くべき場所ではなく，RTOS 内部にブレークポイントを設定する．本来ブレークポイントを置くべきであった場所には，そのルーチンを呼び出す命令に置き換える．これにより，ホスト - ターゲット間の通信量が劇的に減少し，より高速なデバッグが可能になる．ただしその一方で，RTOS 内部にこのようなルーチンが入ることにより，ターゲットには通常の手法よりも大きなオーバーヘッドが発生する．
複数ある手法のうちのどの手法を用いるかは，デバッグを行う人の判断によることになる．

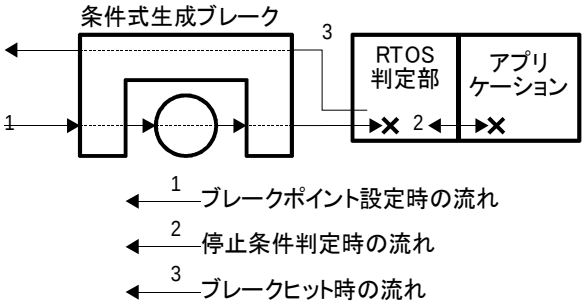


Figure 4: **特殊なブレークルーチンを搭載した場合の動作フロー**

3. 共通規定

本章では共通規定と題して，ITRON デバッギングインタフェース仕様内部で共通の概念などについて説明する．

3.1 インタフェース関数の登録 / 削除

ITRON デバッギングインタフェース仕様では，インタフェース関数ポインタを格納する構造体 ***T_INTERFACE*** に目的とする関数へのポインタを登録することにより，デバッグツール または RIM から利用できるように規定している．
本デバッギングインタフェースで提供する関数の中で ***dbg_ini_inf*** を除く全関数の呼び出しは，このインタフェース構造体から関数へのポインタを取得し，呼び出さなければならない．関数が静的にバインドされる場合を除き，このインタフェース構造体に登録されている関数ポインタの内容は変化する可能性があるため，ローカルコピーを作成して利用するなど，これらの変化が反映されないような処理を行ってはならない．

構造体 ***T_INTERFACE*** は全てのインタフェース関数へのポインタを格納する．構造体メンバは本仕様書の記載順で整列され，それに続いて仕様書外の関数へのポインタが順不同に続く．構造体メンバのうち，存在しない または サポートしないすべての関数へのポインタには，***NULL***(=0) が格納されていなければならない．デバッグツールは，インタフェース初期化関連関数 ***dbg_ini_inf***, ***dbg_ini_rim*** を呼び出す前に，インタフェースに含まれている全てのサポートされない関数と，RIM が所有するすべての関数へのポインタを ***NULL*** で埋めなければならない．
下にデバッグツール側におけるデバッギングインタフェース初期化ルーチンの例を示す．

Program source

```
  /* インタフェース構造体の初期化 */
ZeroMemory(&interface,sizeof(T_INTERFACE));
  /* TIF 関数の登録 */
interface.tif_xxx_yyy = xxx_yyy;
  /* インタフェース初期化 */
dbg_ini_inf(...);
  /* RIM 初期化 */
if( interface->dbg_ini_rim != (void *)0l)
    (*interface->dbg_ini_rim)(...);
```

Program source

3.2 一貫性

「一貫性」とは，「ひとつの操作を通じて，内容が変化しないこと」を言い，「一貫性を保証する」とは，「ひとつの操作を通じて，内容がターゲット上における情報と相違ないことを保証する」とする．
RTOS に依存する情報読み出しなどの実行時に，システムが不定な状態であった場合 (OS のクリティカルセクション内など)，「操作は一貫性を保てない」と判断する．

ターゲットを停止させてから処理を行うようなデバッグツールでは，すべての関数は一貫性を保証していると考えてよい．ただし，停止時にターゲットが OS のクリティカルセクションに入っているような場合があれば，それは一貫性を保証できていないと判断する．

例として，機能と一貫性を保証するための項目を挙げる．

- **単一メモリブロックの読み出し (tif_get_mem)**
  指定されたメモリブロックの読み出し操作時に対象メモリブロックに書き込みが行われないこと．
- **複数メモリブロックの読み出し (tif_get_bls)**
  指定されたメモリブロックの読み出し操作時に対象となる全メモリブロックに書き込みが行われないこと (単一メモリブロックを複数回呼び出した場合，各ブロック間で一貫性は保証されない).
- **タスクの状態取り出し (rif_ref_obj)**
  現在実行位置が OS のクリティカルセクション内でないこと．かつ TCB へのポインタ，および TCB 本体が一貫性を保証して読み出しが行わること．かつ 読み出された情報から構築された内容に矛盾がないこと．

3.3 ターゲット停止の禁止

ターゲット停止の禁止とは，「ITRON デバッギングインタフェース仕様で規定されているターゲット動作関連操作を利用してはならない」ことを意味する．すべての操作においてターゲットを停止させないことが要求された場合，これらの関数を呼び出してはならない．

対象となる ITRON デバッギングインタフェース仕様 ターゲット動作関連操作は次の通り

- **ターゲットの実行**
  tif_sta_tgt
- **ターゲットの停止**
  tif_stp_tgt
- **ターゲットの実行中断**
  tif_brk_tgt
- **ターゲットの実行再開**
  tif_cnt_tgt

関数によっては，一貫性の保証とターゲット停止の許可が併用される場合がある．このときターゲットの停止を伴わずして一貫性を保証することができない場合，それが恒久的なものであれば RIM は ***E_NOSPT*** エラーを返し，それが一時的なものであれば RIM は ***E_FAIL*** を返さなければならない．

3.4 型

ITRON デバッギングインタフェース仕様で定められている独自の型を Table 13: [独自の型] に示す.

Table 13: **独自の型**

型名	意味
BITMASK	ビットマスク (詳細は後述)
ER	エラーコードを格納する 16bit 以上の整数
FLAG	32bit 符号なし整数
ER_ID	ID と ER のどちらか大きな整数 正数は ID を，負数は ER を示す
DT_xxx	ITRON カーネル仕様で xxx と定義されている変数を 格納するのに十分な大きさの同系型
ID	デバッギングインタフェース上のオブジェクト番号 を格納するのに十分な大きさの符号なし整数
INT	ホスト上における自然な長さの符号付整数
UINT	ホスト上における自然な長さの符号なし整数
VP	ホスト上における型無しポインタ
VP_INT	VP, INT を格納するのに十分な大きさの型
LOGTIM	ログ時間を示す整数 (単位に関しては実装定義)

また，RIM およびデバッグツール内でターゲットの ITRON カーネル仕様で定義された型の変数を格納するために，接頭辞 ***DT_*** で始まる型を定義する．この型は「ターゲットのデータを格納するのに十分な大きさを持った変数」であり，ターゲットの型の大きさと一致しない場合もある．(ターゲットの ***INT*** が 16bit であるとき，***DT_INT*** が 32bit であってもかまわない)

基本的にデバッギングインタフェースでは，ITRON カーネル仕様と同名の型名が定義されていたとしても，それは ITRON カーネル仕様の型ではないものとする．具体的には，***ER*** や ***ID*** は ITRON カーネル仕様でも定義されているが，本仕様内で独自に意味付けがなされている．

3.5 ビットマスク

ITRON デバッギングインタフェースでは, 有効および無効の設定などのために, ビットマスクを利用する．ビットマスクは 1 ビットのフラグの集合である．

ビットマスクでは，ビットマスクの第一項目が最下位ビットに相当する．そのため n 番目のフラグの状態は，C 言語で表現した場合，次のようにして取得できるように格納されなければならない．

Program source
```
(( bitmask >> n) & 1)
```
Program source

ビットマスクはその長さに応じて複数の型がある.

BITMASK 仕様内で利用される最大数を超える自然な長さ

BITMASK_8 1 バイト (8 個)

BITMASK_16 2 バイト (16 個)

BITMASK_32 4 バイト (32 個)

BITMASK_64 8 バイト (64 個)

64 個を超えない任意長のビットマスクを作成する場合, 要求を満たす最小限の規定型を用いることを原則とする.

64 個を超える場合, またはビットマスクの長さが一意に定まらないような場合, ビットマスクを 1 バイトビットマスクの配列 ***BITMASK_8***[] として定義する. その場合 n 番目の項目は, C 言語で表記すると, 次のような構文で読み出すことができるよう格納されなければならない.

Program source
```
(( bitmask [n>>3] >> (n & 7)) & 1)
```
Program source

3.6 情報取得構造体と取得キー

ITRON デバッギングインタフェース仕様では, 情報の取得に特殊な構造体と, 対象となる情報を特定するためのキーを用いる.

本規定が適用される情報を取得する関数を挙げる.

- ***rif_ref_cfg* : カーネルコンフィギュレーションの取得**
- ***dbg_ref_dbg* : デバッグツール関連情報の取得**
- ***dbg_ref_rim* : RIM に関連する情報の取得**

取得する情報を特定するため, ITRON デバッギングインタフェース仕様ではキーコードと呼ばれる 4 個の 8 ビット整数を用いる. 本文章内において, キーコードは次のように表記される. またプログラム中などで, 各キーには情報取得キーであることを明確にするために接頭辞 ***INF_*** が付加される場合もある.

キーコード := **第 1 キー** ["." **第 2 キー** ["." **第 3 キー** ["." **第 4 キー**]]]

(例 : ***BREAK.CONDITION.MAX***, ***INF_HOST.INF_NAME***)

キーコードは第 2 キー以降は省略可能である. このとき, 省略されたキーは ***DEFAULT***(=0) として扱われる.

情報取得用構造体 ***T_INFO*** の詳細を下に示す.

```
typedef struct    t_info_result_buf
{
    UINT sz             : バッファのサイズ
```

```
    VP ptr          : 文字列 または 特殊な型を格納する領域へのポインタ
}   T_INFO_RESULT_BUF;

typedef union   t_info_result
{
    INT value       : 32bit 符号付整数
    T_INFO_RESULT_BUF buf
                    : 特殊な型の値
}   T_INFO_RESULT;

typedef struct  t_info
{
    char key[4]     : 情報を特定するためのキー
    T_INFO_RESULT result
                    : キーに対応する値
}   T_INFO;
```

取得可能な情報には 32 ビット整数，文字列または特殊な型の 2 種がある．情報の型は末尾キーの最上位ビットから類推することが可能である．先ほどの例で挙げられた ***BREAK.CONDITION.MAX*** の場合では，第 3 キー ***MAX*** から取得できる．下に末尾キーの最上位ビットと型の関係を示す．

Table 14: **末尾キーの最上位ビットと取得した情報の型**

最上位ビット	取得した型
0	32 ビット整数
1	文字列 または特殊な型

T_INFO::result.buf.sz は文字列取得用バッファの長さを保持する変数である．文字列や特殊な型を読み出した場合，その長さが ***T_INFO::result.buf.sz*** に格納される．整数値の読み出しを行った場合，値は不定となる．
文字列と特殊な型の取得の場合，格納領域を呼び出す側が別に用意しなければならない．呼出し側は十分な大きさの格納領域を確保し、確保した領域へのポインタを ***T_INFO::result.buf.ptr*** に，確保したサイズを ***T_INFO::result.buf.sz*** に格納する．呼び出された側は，転送する長さがサイズを超えないように，指定された領域へ文字列情報や特殊な型の情報を格納する．文字列には常に終端記号が付加されるため，***T_INFO::result.buf.sz*** に 1 を指定した場合には，関数が正常終了しても内容は何も取得できないことに注意されたい．また ***T_INFO::result.buf.sz*** に 0 を指定した場合，***E_PAR*** エラーとなる．
特殊な型の読み出しを行うにあたり，バッファの大きさが転送すべきデータ長に満たない場合の振る舞いは実装定義である．ただし ***T_INFO::result.buf.sz*** が 0 であった場合は ***E_PAR*** エラーとなる．

同時に複数の情報を一括して読み出した場合，引数として指定された ***T_INFO*** のうちのどれかに無効なキーコード[1]が含まれていた場合，関数はエラーとなる．このとき，

無効なキーコード以外の情報が正しく読み込めているかどうかに関して，関数は報告も保証もしない．

キーコードの代入は第 1 キーが配列 ***T_INFO::key*** の 1 番目の項目となるように代入される．動作を明確にするため，下にキーコード生成用関数の実装例を示す．(コードは C++)

Program source

```
static char StringBuffer[MAX_STRBUF_LENGTH];

static inilne void MAKE_KEYCODE
    ( T_INFO * info, char key1, char key2=0, char key3=0,char key4=0)
{
    info->key[0] = key1;
    info->key[1] = key2;
    info->key[2] = key3;
    info->key[3] = key4;
    info->result.buf.sz = MAX_STRBUF_LENGTH;
    info->result.buf.ptr = StringBuffer;
}
```

Program source

キーコード ***0.0.0.0*** は特殊な意味を持つ．キーコードとして ***0.0.0.0*** が渡された場合，情報取得関数は直前に取得したキーコードに後続するキーコードを返却する．また，引数として渡されてたキーコード配列の先頭の要素が ***0.0.0.0*** であった場合は先頭のキーコードを意味し，情報取得関数はキーコードの順序関係のうち，もっとも小さいものを取得する．
つまり，キーコードが ***0.0.0.0*** である 5 つの ***T_INFO*** 配列からは，先頭から 5 番目までのそれぞれに対応した情報が取得できる．ただし，***0.0.0.0*** による連続取得によって，以降の関数実行時の動作が変化することはない．そのため，すべてが ***0.0.0.0*** からなる ***T_INFO*** 配列を用いて複数回関数を実行したとしても，取得できる情報は同一であることに注意されたい．

Program source

```
    //RIF の各機能単位がサポートされているかどうかを調べる
T_INFO support[6];

MAKE_KEYCODE(&support[0], INF_RIF, INF_UNIT, INF_OBJ, 0);
for(i=1;i<6;i++)
    MAKE_KEYCODE(&support[i], 0, 0, 0, 0);

    // これで RIF.UNIT.OBJ 〜 RIF.UNIT.CTX までが取得できる
dbg_ref_rim(support,6,0);
```

Program source

この情報取得構造体をサポートする関数が複数あるため，場合によっては異なる関数で異なる情報キーコードが用いられることも考えられる (例 : ***dbg_ref_dbg*** に対して ***CFG*** キーを使用する)．このとき関数がエラーを返すか，該当する値を返すか，または無効な値を返すかは実装定義である．したがって，呼び出し側は関数とキーコードが一致しなくとも情報が取得できることを仮定してはならないし，そのような操作でエラーが報告されることを期待してもならない．

1. ここでいう無効なキーコードとは，ITRON デバッギングインタフェース仕様で定義していないもので，独自仕様にも含まれないものを指す．ITRON デバッギングインタフェース仕様で定義されているものはすべて何らかの値を持たなければならない．

情報取得キーは ITRON デバッギングインタフェース仕様で定めるもの以外に，各ベンダが自由に作成することも可能である．ただしその場合は上位第 2 ビットならびに上位第 3 ビットが共に 1 となるキー[1] を利用することを強く勧める．ITRON デバッギングインタフェース仕様では今後この範囲に重なるキーを定義しないことを保証する．

3.7 エラーコード

3.7.1 E_xxx エラーと ET_xxx エラー

ITRON デバッギングインタフェース仕様では，ITRON カーネル仕様において ***E***_xxx で定義されるエラーコードを ***ET***_xxx と表現する．この ***ET***_xxx というエラーはターゲット操作に伴うエラーを表現するために用いられ, 一方 ***E***_xxx はホスト上で発生したエラーを表現するために用いられる．例えば，***E_NOMEM*** はホスト上でメモリ不足が起ったことを意味し，***ET_NOMEM*** はターゲット上でメモリ不足が起ったことを意味する．

ターゲット上のエラーを示す ***ET***_xxx エラーは，ITRON カーネル仕様と同じ値を持つ．一方，それと対になるターゲット上のエラーを示す ***E***_xxx は 128 ずれた位置に定義されている．例として ***ET_ID*** は -18 であり，***E_ID*** は -146 となっている．

3.7.2 共通エラー

共通エラーとは，ITRON デバッギングインタフェース仕様で定義する全ての関数に共通して発生するエラーのことである．

E_OK

処理は正常に終了した．

E_NOMEM

メモリ不足のために，ホスト上にメモリを確保できなかった．

E_NOSPT

関数はその機能をサポートしていない．フラグによって指定された機能を実装していない場合や，実現不可能である場合などに発生するエラー．

E_FAIL

関数は何らかの要因により要求を達成できなかったものの，ターゲットプログラムなどの実行に支障を与えない範囲であった．再度要求を発行すれば正しく要求を実行できる可能性がある．弱い意味での一般エラー．

関数がこのエラーを返す場合，関数が内部で行った変更を，関数開始時のと意味が変化しないレベル[2] まで復帰させなければならない．また, デバッ

1. 0x60-0x7f, 0xe0-0xff の計 64 個
2. 関数がエラー終了した場合に無効となる引数などは，中身を完全に戻さなくても，引数そのものが無効となるため意味は等価となる．確保したメモリの開放などもある瞬間にまとめて行うような実装の場合，内部で利用されないことが保証されるのであればその時点で開放しなくても良い．

グツールがエラー発生直後に，再度エラーの発生した関数を同一のパラメータで呼び出すこと (リトライ) を仮定してもならない.

(***E_FAIL*** エラーの例 : ***rif_ref_obj*** 発行時点で現在実行位置がカーネルのクリティカルセクション内であるため，キューを正しく処理できなかった. ***tif_set_reg*** 発行時点で全レジスタに書き込みが行えなかった．)

E_SYS

関数が何らかの要因により要求を達成できず，またターゲットの実行に支障を与える可能性がある．再度要求を発行しても正常に処理できない．強い意味での一般エラー. RTOS アクセスインタフェースの関数の場合は, 関数内部で使用しているターゲットアクセスインタフェースの関数が予期せず ***E_SYS*** で終了した場合は，必ず ***E_SYS*** を返さなければならない.

(例 : ***rif_ref_obj***発行時点でデバッグツールのメモリ読み出し機構が正常に動作しておらず，情報を取得できない. ***tif_set_reg*** でレジスタに書き込みを行っている途中で書き込みが行えなくなり，一部のレジスタの値が更新されていない．)

E_SYS エラーが発生した場合, デバッグツールはユーザーに対して致命的なエラーが発生したことを通知し, デバッグ環境 (ターゲットおよびデバッグツール) が不安定な状況で動作していることを明確にすることが望ましい.

3.7.3 よく似たエラー

本節では，ITRON デバッギングインタフェース仕様内で定義されているエラーのうち，よく似ているものを取り上げ，その違いを説明する.

E_ID と E_NOID

- ***E_ID***

  指定された ID の範囲は有意の範囲を超えている．その ID 番号を用いる限り，エラーは再発する.

- ***E_NOID***

  自動割当を行うための ID 空間が不足している．自動割当 ID 空間内のオブジェクトが破棄され, 空間内に未割当 ID 番号ができるまで, エラーは再発する.

ET_OBJ と ET_NOEXS と ET_OACV

- ***ET_OBJ***

  ターゲット上に指定した ID に割当てられたオブジェクトは存在するが, 操作に失敗した. 失敗要因が取り除かれない限り, エラーも取り除かれない.

  (例として, オブジェクトが現在カーネル内で操作中であるときは***ET_OBJ*** エラーとなり，クリティカルセクションから脱出するまでエラーは取り除かれない)

- ***ET_NOEXS***
  ターゲット上には指定 ID のオブジェクトは存在しない．指定 ID に割当てられたオブジェクトを生成するまで，エラーは取り除かれない．
- ***ET_OACV***
  ターゲット上に指定した ID に割当てられたオブジェクトは存在するが, 特権違反などの原因により，カーネルが対象オブジェクトへのアクセスを拒否した．呼出元および呼び出し先の特権レベルなどが変わらなければ，エラーは取り除かれない．

3.8 可変長格納領域

ITRON デバッギングインタフェース仕様では，タスク ID リストなどの可変長の情報を取得するために，2 種類の方法を用いている．

- **別空間可変長領域 (接尾辞 *-lst*)**
  可変長情報を格納する領域を情報取得用構造体とは別に確保する．同一構造体に可変長情報が複数ある場合など．
- **同一空間可変長領域 (接尾辞 *-ary*)**
  情報取得構造体に連続する領域を可変長情報取得領域として利用する．

以下にその詳細を説明する．

3.8.1 別空間可変長領域

別空間可変長領域は 2 種類の固有な接尾辞を持つ変数と，可変長データを格納する領域から構成される．

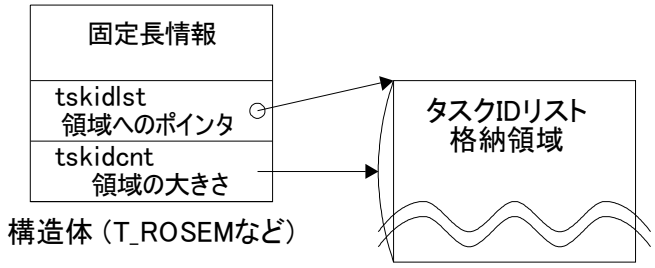


Figure 5: **別空間可変長領域 (タスク** ID)

- **接尾辞 *lst***
  可変長データを格納する領域の先頭を示すポインタを格納するための変数
- **接尾辞 *cnt***
  可変長データを格納する領域の大きさを格納するための変数 (項目単位)

仕様書中では接尾辞 ***cnt*** を持つ変数に連続する，接尾辞 ***lst*** を持つポインタ変数として表記されている．

3.8.2 同一空間可変長領域

同一空間可変長領域は格納領域の大きさを示す変数と，可変長データの格納領域として利用される構造体に連続する領域から構成される．

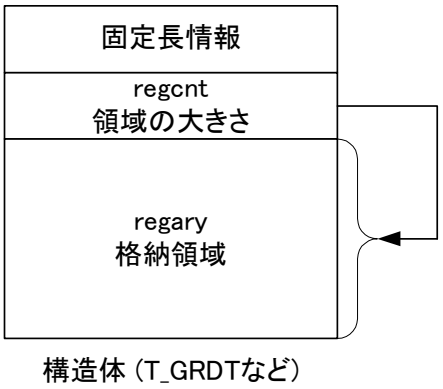


Figure 6: **同一空間可変長領域**

- **接尾辞 *ary***
  可変長データを格納する配列
- **接尾辞 *cnt***
  可変長データを格納する配列の大きさを格納するための変数 (項目単位)

仕様書中では接尾辞 ***cnt*** を持つ変数に連続する，接尾辞 ***ary*** を持つ配列 またはポインタ変数として表記されている．

3.9 識別番号 (ID)

ITRON デバッギングインタフェース仕様では，ブレークポイントやメモリポーリング (ウォッチポイント) などを設定すると，その設定項目を一意に識別するための識別番号 (以下 ID) を割り当てる．ただし，ここでいう識別番号 (ID) は，ITRON カーネル仕様で定義された識別番号とは異なる．

ID が割り当てられる機能を次に示す．

- **ブレークポイント (RIF)**
- **ブレークポイント (TIF)**
- **ポーリング (ウォッチポイント)**
- **ログ (RIF)**
- **ログ (TIF)**

ID は次のような特徴をもつ

- **値は 1 以上の正数である**
  0 以下に関しては基本的に扱えないものとし，通常の手法で 0 以下の ID を持つ設定項目を操作することはできない．

- **値は連続しない場合がある**
  自動番号割当機能などを用いて連続して ID を割り当てた場合でも, これらに割り当てられた値は連続していない場合がある.
- **値は再利用される場合がある**
  一度開放された後であれば，ID は再利用される場合がある．ただし，同一の機能で，同時に同一 ID を持つオブジェクトが複数存在することはない.
- **値は機能ごとに独立である**
  ID は機能ごとに与えられるため, 機能間 (ブレークポイント (TIF) とブレークポイント (RIF) など) で同一の ID を持つ設定項目がある場合がある．ただし，機能が異なれば，ID が同一であっても示している設定項目の実体は異なる.

上記 5 つの機能に割り当てられる ID は，***ID*** 型として宣言されている．一方，ITRON カーネル仕様で定義される識別番号は ***DT_ID*** 型として宣言されている．ITRON 仕様で定義されている ***ID*** 型 (***DT_ID*** 型) の変数の取扱に関しては ITRON カーネル仕様書などを参考のこと.

3.10 レジスタ名称

ITRON デバッギングインタフェース仕様では，ターゲットコンピュータのレジスタを識別するために文字列を用いている．このとき利用されるレジスタを識別する文字列には，次のような規定を設ける.

- **構成文字**
  レジスタ名称を構成する文字列は, 英字大文字 (A-Z) および数字 (0-9) からなる文字列とする
- **文字数制限**
  レジスタ名称は 8 文字 (終端含む) を超えてはならない
- **一意な名称**
  レジスタ名称は，ターゲットチップのハードウェアマニュアルで記述された名称 (省略形) か，ターゲットチップメーカが作成したアセンブラで用いられる名前かのいずれかでなければならない．また同一のレジスタを示す名称が複数存在する場合，別名を受け入れるように作成することを推奨するが，基本的にはその別名のうちレジスタの特徴を強く示す名称で統一することとする

レジスタ名称を利用する関数

- ***rif_get_rdt* : ディスクリプションテーブルの取得**

3.11 フラグ

ITRON デバッギングインタフェース仕様では，全ての関数に機能指定用のフラグを設けている (コールバックおよび一部サポート関数を除く) ．このフラグは一貫性保証や自動番号割当てなど，ITRON デバッギングインタフェース仕様で定められた機能を使用する際のパラメータの一部として利用される.

フラグの各ビットには次のような意味がある.

Table 15: **フラグの内容**

ビットマスク	意味
0xFF000000	***FLG_*** で始まる共通フラグ
0x00FF0000	予約
0x0000FF00	RIM およびデバッグツールが自由に定義できるフラグ領域
0x000000FF	各機構ごとに設けられたオプション (***OPT_*** で始まる)

一切フラグを指定しない状態を ***FLG_DEFAULT***(=0) で表現する.

このフラグは，各ベンダによって新しい項目を増やすことが可能である．そのとき，下位 8 〜 15 ビットまでの領域を使うことを強く勧める．ITRON デバッギングインタフェース仕様では，この領域に新たなフラグを定義しないことを保証する.
ITRON デバッギングインタフェース仕様で定める全ての関数は，その関数が処理できるフラグ以外のフラグの到来に対して ***E_NOSPT*** エラーを返却する.

3.12 レジスタセットディスクリプションテーブル

レジスタセットディスクリプションテーブルは，レジスタ値の格納位置やレジスタ名称などの情報から構成されるテーブルである．RIM およびデバッグツールは，このレジスタセットディスクリプションテーブルに記述された情報を基に, レジスタ および コンテキストを操作する.

レジスタセットディスクリプションテーブルを扱う関数を次に列挙する.

rif_get_rdt	ディスクリプションテーブルの取得
rif_get_ctx	タスクコンテキストの取得
rif_set_ctx	タスクコンテキストの設定
tif_get_reg	レジスタ内容の読み出し
tif_set_reg	レジスタ内容の書き出し

レジスタセットディスクリプションテーブル構造体 ***T_GRDT*** を次に示す.

```
typedef struct    t_grdt_regary
{
    char * strname      : レジスタ名称を指すポインタ
    UINT length         : 長さ ( バイト単位 )
    UINT offset         : 格納オフセット位置
}    T_GRDT_REGARY;

typedef struct    t_grdt
{
    UINT regcnt         : レジスタ本数
    UINT ctxcnt         : コンテキストに含まれる可能性のあるレジスタの数
```

```
    T_GRDT_REGARY regary[]
                        : レジスタ情報
} T_GRDT;
```

レジスタセットディスクリプションテーブルは次のような特徴を持つ.

- **レジスタ名称とサイズ, および格納位置を保持する**
  レジスタセットディスクリプションテーブルのレジスタ情報 (***T_GRDT::regary***) は，レジスタ名称，レジスタの長さ (バイト長)，およびレジスタ関連操作を行った際にレジスタ値が格納されるオフセット位置を持つ．レジスタ長とオフセット位置は，取得や設定などを行う際に必要となる．これらの値は，レジスタ値を保持している領域の中で，対象となるレジスタの値が格納されているオフセット位置とデータ長を定めている.
- **コンテキストと RIM が操作するレジスタを保持する**
  レジスタセットディスクリプションテーブルは，コンテキスト情報とレジスタ情報の 2 つを格納している．レジスタセットディスクリプションテーブルの前半には，対象となる OS のコンテキストとなりうるレジスタを列挙し，後半には RIM が操作する可能性のある全てのレジスタを列挙する.

  T_GRDT::ctxcnt は，対象となる OS のコンテキストの数を保持している．***T_GRDT::regary*** の先頭から ***ctxcnt*** 個のレジスタは，OS のタスクコンテキストである．一方，***T_GRDT::regcnt*** は，レジスタセットディスクリプションテーブルに記述された全てのレジスタの個数を保持している.
- **全てのレジスタ操作に用いられる**
  レジスタセットディスクリプションテーブルは，ターゲットアクセスインタフェース上で提供されているレジスタ関連操作関数で，暗黙のうちに利用される. そのため, 基本的には, レジスタセットディスクリプションテーブルに記述されていないレジスタの操作を行なう事はできない.
- **プログラムの実行を通じて不変である**
  RIM が提供するレジスタセットディスクリプションテーブルは，プログラムの実行[1] を通じて不変である．RIM は実行中にテーブルの内容を書き換えてはならない．また，デバッグツールは ***rif_get_rdt*** を利用して RIM から取得したテーブルの内容を書き換えてはならない.

レジスタセットディスクリプションテーブルを利用する関数のうち, ***rif_get_rdt*** を除いた 4 関数は, 引数として有効 / 無効識別情報 (***BITMASK_8*** * ***valid***) を持つ. ***valid*** は ***regary*** の各要素に対応し，対応するビットが非ゼロである要素が有効となる．また，操作結果の有効 / 無効識別情報が再度 ***valid*** に格納される.
valid が NULL である場合，全てのレジスタが操作対象となり，また操作結果は格納されない.

Table 16: [レジスタセットディスクリプションテーブルの具体例] に，具体的な例を示す．この例では，タスクコンテキストは，プログラムカウンタ (PC) とスタックポ

1. ここで言うプログラムの実行とは，dbg_ini_rim から dbg_fin_rim までを指す。

インタ (SP) のみを持つ. そして RIM は PC と SP に加え, ステータスレジスタ (SR) と汎用レジスタ (R14) を操作する可能性がある.

Table 16: **レジスタセットディスクリプションテーブルの具体例**

フィールド	内容
regcnt	4
ctxcnt	2
regary[0]	{“PC”, 4, 0}
regary[1]	{ “SP”, 4, 4}
regary[2]	{ “SR”, 4, 8}
regary[3]	{ “R14”, 4, 12}

取得用関数 ***rif_get_ctx*** および ***tif_get_reg*** は，レジスタセットディスクリプションテーブルの情報に基づいて，タスクコンテキストおよびレジスタ値を指定された領域に格納する．前述の例で示したレジスタセットディスクリプションテーブルを利用し，次のプログラムを実行した場合の例を次に示す.

Program source

```
char buffer[16];
BITMASK_8 valid = 0xa;
tif_get_reg(buffer, &valid, FLG_DEFAULT);
```

Program source

上記のプログラムを実行し，すべての操作が正常に終了した場合，関数 ***tif_get_reg*** は変数 ***buffer*** に次のようにレジスタ値を格納する.

Table 17: **レジスタ格納の様子**

オフセット	内容
0 〜 3	何も格納されない
4 〜 7	スタックポインタ (SP)
8 〜 11	何も格納されない
12 〜 15	汎用レジスタ (R14)

3.13 特殊なブロッキングモード

ITRON デバッギングインタフェース仕様では，本来ノンブロッキングで動作する関数を対象に，特殊なブロッキングモードでの実行が行えるようになっている．特殊ブロッキングモードは，実行にそれほど多くの時間がかからない処理を行うときに用いることを想定しており，この機能によりプログラムの実装が簡単になる.

特殊ブロッキングモードでは，関数の実行により，実行終了までプログラムがブロックされる．ただし，関数内でプログラムが停止してしまうことを避けるため，ある程

度の時間が経過すると，自動的にタイムアウトする．関数はタイムアウトによって処理が中断された場合，***E_FAIL*** を返す．

ブロッキングにタイムアウトを設けたのは，RIM の処理がブロックすることで，デバッグツールのユーザインタフェースの更新などが停止してしまうことを避けるためである．そのため，ブロッキングのタイムアウト時間は，利用者が耐え得る時間[1]を目処とする．具体的な時間は実装依存とする．

特殊ブロッキングモードはオプションフラグ ***OPT_BLOCKING*** を指定することで，利用できる．特殊ブロッキングモードをサポートしている関数を，以下に列挙する．

- ***rif_cal_svc*** **: サービスコールの発行**
- ***tif_set_pol*** **: メモリ内容変更通知の設定**
- ***tif_cal_fnc*** **: 関数の発行**

1. 通常，ユーザが通知無しに耐え得る処理待ち時間は数秒と言われている．しかし，処理前や処理中にユーザに時間がかかることを通知すれば，タイムアウト時間を長くしても構わない．ただし，無限に待たせることはあってはならない．

4.RTOS サポート機能ガイドライン

4.1 搭載される機能の統一化

サポート機能ガイドラインは,「ITRON デバッギングインタフェース仕様 対応デバッグツール」に搭載され RTOS サポート機能を統一化する．以下に ITRON デバッギングインタフェース仕様における RTOS サポート機能の機能要素を挙げる．

•ITRON オブジェクトの状態取得

• タスクコンテキストの操作

• サービスコールの発行

•OS に依存したブレーク・トレース

•OS に依存した実行履歴 (サービスコール・タスク遷移・デバッグログなど)

次に，それぞれの機能と実現方法について簡単に述べる．

ITRON オブジェクトの状態取得

この機能は，RTOS 内部のオブジェクト (タスク・セマフォなど) や，普段参照できない内部情報 (レディキュー・CPU 状態など) の情報を取り出し，ユーザーに提示するための機能である．

得られる情報の例

- **タスク**
  優先度 , スタック , 待ち要因 , 待ちオブジェクト など
- **同期オブジェクト**
  コントロールブロックの内容 , 待ちタスク など
- **レディーキュー**
  現在実行中のタスク ID, 実行可能状態のタスクリスト
- **システム関連**
  現在のコンテキストモード , カーネル内部状態

タスクコンテキストの操作

タスクコンテキストの操作では，現在起動しているタスクのレジスタ内容，スタックポインタ, 実行位置などを操作するための機能である. タスクの実効状態に関わらず, 適切な領域からコンテキスト情報を取得，または設定することが可能となる．

サービスコールの発行

サービスコール発行では，外部からターゲット内部のカーネルサービスコール (システムコール) をパラメータと共に発行することを可能にする．外部からセマフォの返却を行うなどの機能が利用可能となる.
またシステムコールだけではなく，ソフトウェア部品を組み込んだ際の，ソフトウェア部品のサービスコール発行も想定しているので，部品を組み合わせた時のデバッグにも威力を発揮する．

OS に依存したブレーク・トレース

ITRON デバッギングインタフェースでは,以下のような RTOS に依存するブレーク機能をサポートする.

- **タスク関連ブレーク**
  タスク ID を指定してブレーク
  他のタスクの実行を止めずに指定タスクだけブレーク
  指定タスクがサービスコールを発行するとブレーク
- **オブジェクト関連ブレーク**
  指定したオブジェクトを操作するとブレーク
- **システム関連ブレーク**
  コンテキストスイッチでブレーク
  指定したタスクにディスパッチするときブレーク

また，各ブレークに対してターゲットを停止させる方法も選ぶことができる.

- **システム全体を停止**
- **対象タスクのみを停止し，システム自体は実行継続 (要 RTOS のサポート)**

OS に依存した実行履歴

この機能は，システムの挙動を監視するための実行履歴を取得する機能である．本仕様では以下のような機能をサポートしている.

- **ディスパッチ履歴**
  タスクの実行遷移履歴を取得する
- **サービスコール発行履歴**
  発行したサービスコールのパラメータやエラーコードなどの履歴を取得する
- **ユーザーイベント履歴**
  ユーザが記述したコメントなどの履歴を取得する

4.2 レベル表示

サービスコールにはそれぞれレベル分けが成されている.「ITRON デバッギングインタフェース仕様 対応」を明言するデバッグツールは，どの範囲のレベルまでをサポートするのかを明言しなければならない．これによってユーザは自分が利用できる機能を知ることができる.
ITRON デバッギングインタフェース仕様では, RIF と TIF にそれぞれ独立なレベル表記が用いられる．次節から，RIF および TIF それぞれのレベル表記を示す.

4.2.1 RIF レベル表記

RIM (RIF) のレベルは，RTOS サポート機能を提供する関数の集合である機能単位ごとに表示される.

機能単位は次のとおり (括弧内はそれぞれの略称)

- **オブジェクト状態取得 [OBJ]**

- **コンテキスト操作 [CTX]**
- **サービスコール発行 [SVC]**
- **ブレーク設定 [BRK]**
- **ブレーク条件取得 [CND]**
- **実行履歴 [LOG]**

RIM はそれぞれの機能単位ごとに実行可能かどうかを表記しなければならない．
デバッグツールは，各機能単位への接続機構 (ユーザインタフェースなど) の搭載状況を，RIF のレベルとして表記しなければならない．

レベル表記の例を下に示す．Table18 のような組み合わせの場合，エンドユーザは「オブジェクト状態取得」「コンテキスト操作」「サービスコール発行」「ブレーク設定」「実行履歴[1]」の各項目の最低限の機能は利用できる．

Table 18: **レベル表記の一例**

機能単位	RIM	デバッグツール
OBJ	○	○
CTX	○	○
SVC	○ (tif_alc_mbt 必須)	○ (tif_cal_svc は未対応)
BRK	○	○
CND	○	× (条件ブレーク非対応)
LOG	○ (RIM 単体でも可)	× (UI は提供)
その他	一部拡張あり	TIF レベル [R]

4.2.2 TIF レベル表記

TIF のレベル表記は，エンドユーザに利用可能な機能を知らせると同時に，RIM を実装するものにとっての指標である．
TIF のレベルは関数ごとに設けられ，主に 2 種類に大別される．(括弧内はそれぞれの略称)

- **必須機能 [R]**

  ITRONデバッギングインタフェース仕様のデバッグツールとして必ず搭載されなければならない関数である．RIM 実装者は，必須機能とされている関数が存在するかどうかをチェックしなくても良い．ただし必須機能である関数のオプションの一部などが拡張機能になる場合もある．必須機能のうちの拡張部分に関しては，その機能を利用した際，関数が ***E_NOSPT*** を返す場合がある．

1. デバッグツールは実行履歴の取得に対応していないが，RIM 単体でも実行可能であるため，機能は利用可能である．厳密には，TIF のログ機構が利用できないという意味である．

- **拡張機能 [E]**
  拡張機能は，主として利便性を考慮されて定義された機能 (条件ブレーク設定など) である．これらの機能はデバッグツールによっては搭載することが不可能な場合があるため，RIM 作成者は拡張機能とされている関数が存在するかどうかの判定なしに呼び出すことがあってはならない．

4.2.3 その他のインターフェース

RIF の「***rif_ref_cfg:*** カーネルコンフィギュレーションの取得」と，関数名が ***dbg_*** で始まる一部の関数に関しては，TIF と同様に必須 [R] と拡張 [E] で表記を行う．これらの関数は RIM およびデバッグツール作成者に対する情報としての意味合いが濃いため，実装しても表記する必要はない．

4.3 用語と説明

4.3.1 デバッグツール

ITRON デバッギングインタフェース仕様では, ターゲットプログラムおよびターゲットハードウェアが正常に動作しているかを監視するためのツールをまとめて**デバッグツール**と呼んでいる．このデバッグツールには，ホストコンピュータ，デバッグ支援プログラム，ICE やプローブなどのハードウェアなどが含まれる．デバッグツールにはターゲットおよびターゲットプログラムは含まれないが，ツールメーカが提供するデバッグに必要なデバッグエージェント (後述) を含むこともある．

これらをあえてデバッガと呼ばない理由は，通常デバッガとはプログラムを指す場合が多く，周辺環境までの全てを含めたものもデバッガと呼ぶことによりターゲットとデバッガの境が不明瞭になってしまうことを防ぐためである．

4.3.2 デバッグエージェント

ITRON デバッギングインタフェース仕様では，デバッグ環境を構成したとき，ターゲットハードウェア上のプログラムとしてデバッグ支援を行うソフトウェアを総じて**デバッグエージェント**と呼ぶ．ただし OS 内の一部である場合や，タスクの一部である場合などの実装形態は問わない．

4.4 ブレーク機構

ITRON デバッギングインタフェース仕様では, RTOS を意識したブレークをサポートするための機能と関数群が用意されている．ここではこれらブレーク機構に関しての詳細な説明を行う．

ブレーク機構を構成する関数群は次の通り

rif_set_brk	ブレークポイントの設定要求
rif_del_brk	ブレークポイントの解除
rif_rep_brk	ブレークヒットの通知
rif_ref_brk	設定したブレーク情報の取得
rif_ref_cnd	ブレーク条件の取得
tif_set_brk	ブレークポイントの設定

tif_del_brk ブレークポイントの解除
tif_rep_brk ブレーク通知

次にブレーク機構で特徴的な部分を取り上げ，詳細の説明を行う．

4.4.1 コールバック判定

デバッグツールは，ターゲットアクセスインターフェース上の関数 ***tif_set_brk*** で設定されたブレークポイントに達すると，コールバック関数 ***tif_rep_brk*** を呼び出し，停止すべきかどうかの判断を RIM に委ねる．またそれと同様に，RTOS アクセスインターフェース上の関数 ***rif_set_brk*** で設定されたブレークポイントに達すると，RIM はコールバック関数 ***rif_rep_brk*** を呼び出し，停止したことを通知する．
tif_rep_brk が ***E_TRUE*** を返した場合，デバッグツールはブレーク動作を継続させるが，***E_FALSE*** を返した場合，デバッグツールはブレーク動作を中断させ，再度ターゲットの実行を再開させる．ただし ***rif_rep_brk*** はブレーク中断の判定を下すことはできない．

一連のブレーク動作はこのシーケンスをもって実行される．

具体的に例を挙げて説明する．

1. デバッグツールは ***rif_set_brk*** を用い，ユーザのブレークポイント設定要求を RIM に伝え，RIM は ***tif_set_brk*** 関数を用いてある特定のアドレスにブレークポイントを設定する．

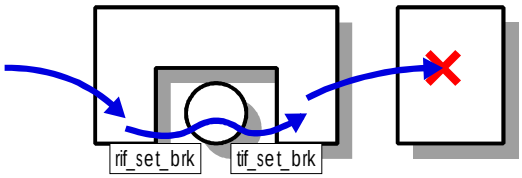


Figure 7: **ブレークポイント設定**

2. RIM またはユーザによってターゲットの実行が開始され，ターゲットプログラムがブレークポイントの設定条件を満たした時，デバッグツールの機能により，ターゲットはブレーク状態となる．この時点からターゲットプログラムの実行は中断された状態にある．

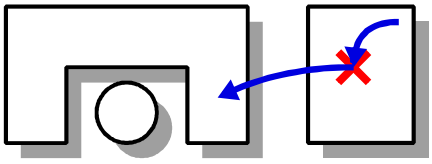


Figure 8: **ブレークヒット**

3. デバッグツールは停止した原因となったブレークポイントが ***tif_set_brk*** によって設定されたものであった場合，コールバック関数 ***tif_rep_brk*** を呼び

出してブレークしたことを通知する．このとき，引数としてブレークポイント ID と ***tif_set_brk*** で設定されたブレーク時パラメータを渡す．

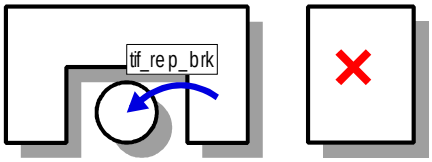


Figure 9: tif_rep_brk **の呼出**

4. RIM は TIF 上の関数を駆使し，設定したブレークによってターゲットの実行を停止させてよいか判断するのに十分な情報を収集し，判断を行う．ブレーク処理は，停止すべきと判断した場合は 5 へ，停止すべきではないと判断した場合は 5' へと続く．

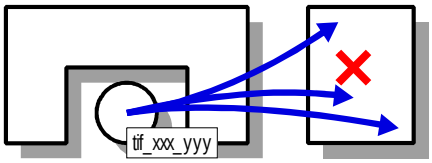


Figure 10: **情報収集**

5. 情報収集の結果 RIM は停止すべきと判断し，かつそのブレークポイントが ***rif_set_brk*** によって設定されたブレークポイントである場合, RIF 上の関数 ***rif_rep_brk*** を呼び出す．その後ブレーク成立をデバッグツールに通知するため，***tif_set_brk*** は ***E_TRUE*** を返す．

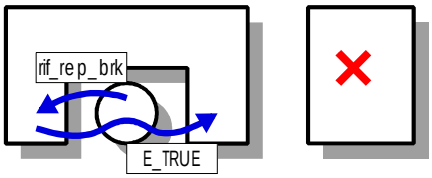


Figure 11: **条件一致時の** tif_rep_brk **の動作**

6. デバッグツールはブレーク動作を継続させた後，ユーザにブレークポイントで停止したことを通知する．

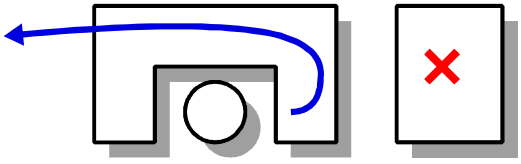


Figure 12: **ブレーク動作の継続**

5'. 一方情報収集の結果，RIM は停止すべきではないと判断した場合，直ちに ***E_FALSE*** を返す.

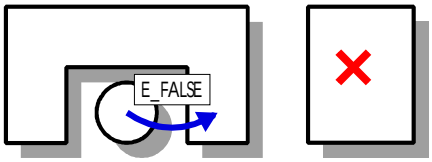


Figure 13: **条件不一致時の** tif_rep_brk **の動作**

6'. デバッグツールは項目 2. で停止したターゲットプログラムの実行を再開させる.

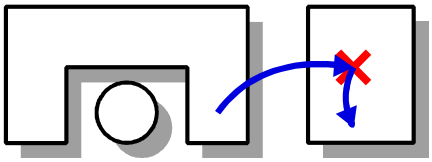


Figure 14: **ブレーク動作の中断とターゲットプログラムの実行再開**

4.4.2 条件取得型ブレーク

ITRON デバッギングインタフェース仕様では，条件のみを取得するようなブレークサポート機構も用意している. (2.5.1 [2 種類のタスク ID によるブレーク手法] を参照) この機能を使うにあたり，デバッグツールは条件ブレーク機構を持っていることが前提となる.

条件取得型ブレーク機構を構成する関数を下に示す.

rif_ref_cnd ブレーク条件の取得

動作フローを以下に示す.

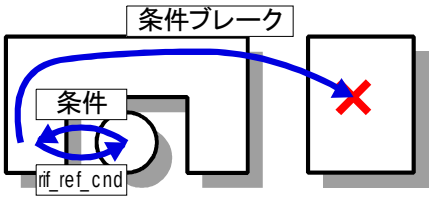


Figure 15: **条件取得型ブレーク**

1. ユーザからの要求に基づいて，デバッグツールは ***rif_ref_cnd*** を呼び出し，RTOS 依存の条件を生成するよう要求する.
2. RIM は要求を満たすような条件を生成し，デバッグツールに返却する.
3. デバッグツールは生成された条件を基に，ユーザが要求するアドレスにブレークポイントを設定する.

条件取得型ブレーク機構では，ブレークポイントの設定はデバッグツール側が行う．そのため，この手法によって設定されたブレークポイントが原因となって***rif_rep_brk***が呼ばれることはない．

4.5 トレースログ機構

ITRON デバッギングインタフェース仕様では，OS に依存した実行履歴の取得をサポートするために，トレースログ機構と呼ばれる一連の機能と関数群が用意されている．ここではこれらトレースログ機構の詳細を説明する．

トレースログ機構は次の関数から成る．

rif_set_log	トレースログの設定
rif_del_log	トレースログ設定の解除
rif_sta_log	トレースログ機能の開始要求
rif_stp_log	トレースログ取得の解除要求
rif_get_log	トレースログの取得

トレースログ機構の操作は，大きく分けると「設定」「開始」「実行」「取得」「終了」「解除」の 6 つに分割できる．トレースログ機構の処理の流れは，1 回のログ機構の使用に当たり，1 回の設定，複数回の「開始・実行・終了」の一連の作業，複数回の「取得」，そして 1 回の「解除」となる．

次ではこれら各項目に分けて詳細を説明する．

4.5.1 設定

トレースログの設定では，関数 ***rif_set_log*** を用いて必要な数だけトレースログの設定を行う．

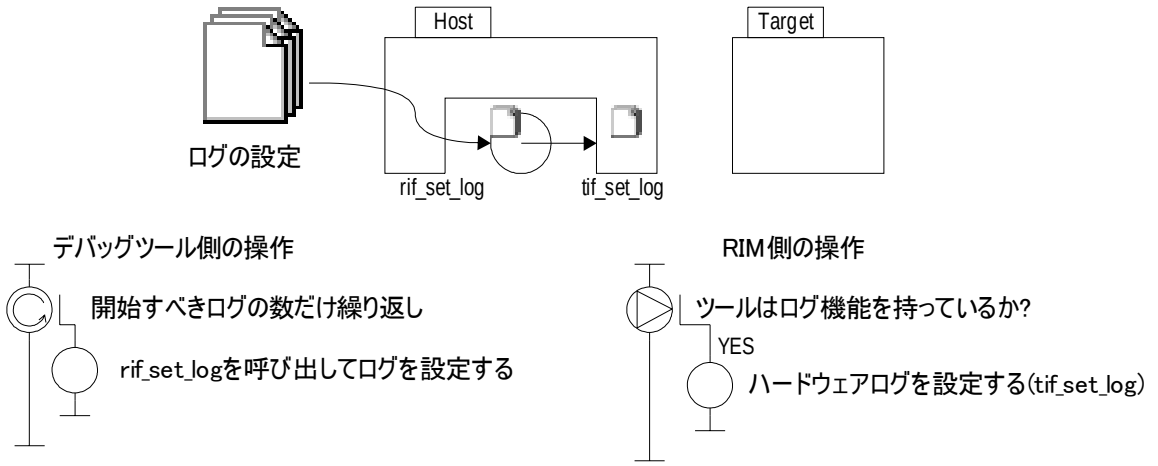


Figure 16: **トレースログ 設定**

この時点ではまだターゲットの動作に影響を与えるような操作を行うことはない[1].
デバッグツールは RIM に設定項目を与え，RIM はこの情報を基に，後の開始に備えてさまざまな処理を行う (場合によっては過去に設定された項目とマージするなどの最適化を施すこともありうる).
デバッグツールがメモリアクセスログなどを取得する機能を持っていた場合, RIM は ***tif_set_log*** 関数を用いてこれらの設定を同時に行う.

4.5.2 開始

トレースログの開始では，設定したトレースログを実際に取得するための処理を行う.

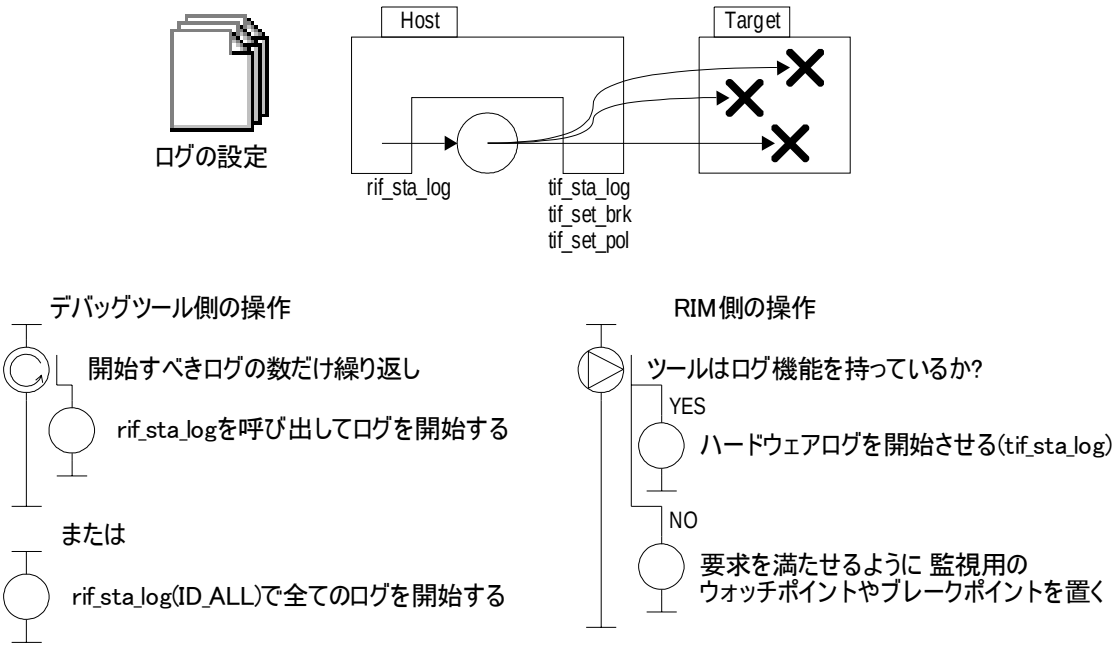


Figure 17: **トレースログ 開始**

設定内容に基づいてログを取得するための準備を行う.
デバッグツールがログ機構をもっている場合は, ***tif_sta_log*** 関数を用いてこれらの設定を有効にする．そうでない場合は，各条件を満たすようにブレークポイントやウォッチポイントを設定しておき，後にプログラムを実行したときに呼ばれる各種コールバックの中からメモリ読み出しなどの命令を発行し，ログに必要な情報を回収することになる.

1. ユーザの立場から見た場合という意味．実装によってはターゲットに多少の操作を加えることもありうるが，それはユーザが意識できるものであってはならない.

4.5.3 実行

実行ではプログラムを動作させてログ情報を取得する.

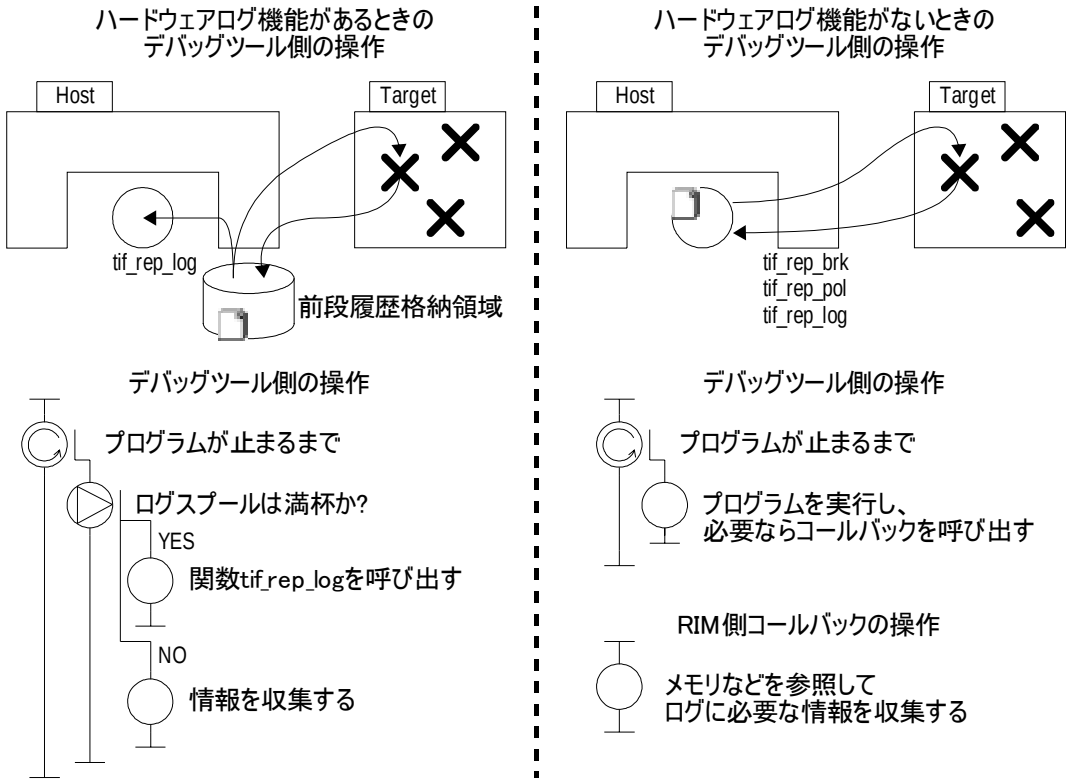


Figure 18: **トレースログ 実行**

デバッグツールがハードウェアログ機構をもっている場合, ログ取得に関する機構は全てハードウェアに依存した形を取る. RIM はハードウェアログ機構のログバッファ (本仕様では**前段履歴格納領域**と呼んでいる) が満杯になったときと, ログ取得が終了したときに発生するコールバックの処理だけを行えばよい.
一方, 相当するハードウェアが存在しない場合, RIM はウォッチポイントやブレークポイントのコールバック関数を用いて, ログに必要な情報を収集することになる.

4.5.4 取得

ログ取得のためのプログラム実行が終了した後，デバッグツールはログの取得を行う．

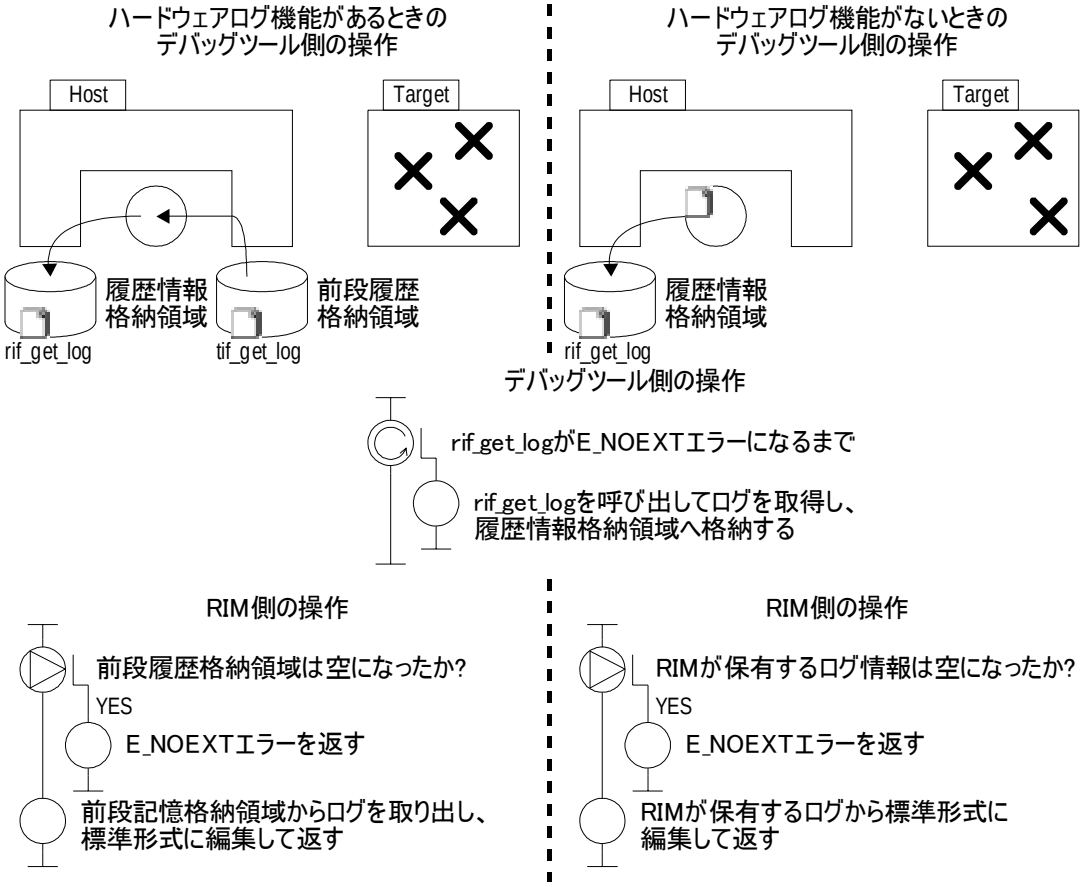


Figure 19: **トレースログ 取得**

ハードウェアログをサポートしているデバッグツールを利用している場合, RIM はデバッグツールが取得したログが格納されている前段履歴格納領域から, ***tif_get_log*** 関数を用いてログを取得し，そのログ情報から要求を満たすような情報を再生成し，デバッグツールに返す．
ハードウェアログをサポートしていないデバッグツールを利用している場合, RIM は自ら取得したログ情報から情報を再構成し，デバッグツールに返す．
デバッグツールは ***rif_get_log*** を呼び出し，一項目分のログを取得する．ログは取得した順[1] に取り出され，履歴情報格納領域に格納される．履歴情報格納領域は ITRON デバッギングインタフェース仕様で定められた標準ログ形式で格納されるため，デバッグツールとは独立したログビューアなどで閲覧することも可能である．
取得はログ機構が作動すればいつでも行うことができる．通常考えうる取得動作の実行位置には，次のような場合がある．

- **ログスプールが満杯になったとき**
- **プログラムが終了したとき**

1. 取得した順は格納された順であり，厳密な意味でのログ取得時間順ではない．

4.5.5 終了

ログ取得が終了した後，デバッグツールはログを終了させる．

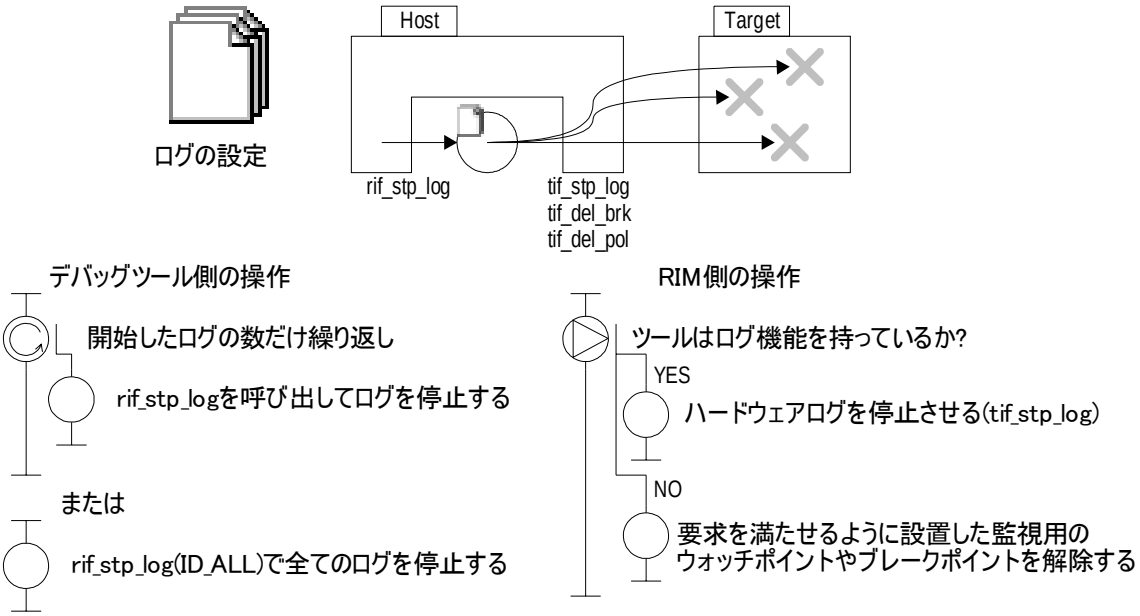


Figure 20: **トレースログ 終了**

設定内容に基づいてログを取得するために準備したリソースの破棄を行う．
デバッグツールがログ機構をもっている場合は，***tif_sta_log*** 関数で有効にしたハードウェアログ機構を ***tif_stp_log*** 関数を用いて解除する．そうでない場合は，情報取得用に設定したブレークポイントやウォッチポイントを解除する．いずれの場合も，同時にログ格納領域として RIM が取得していたメモリも解放する．

4.5.6 解除

全てのログ取得が完了し，ログの設定が不要となったとき，デバッグツールはログ設定の解除を行う．

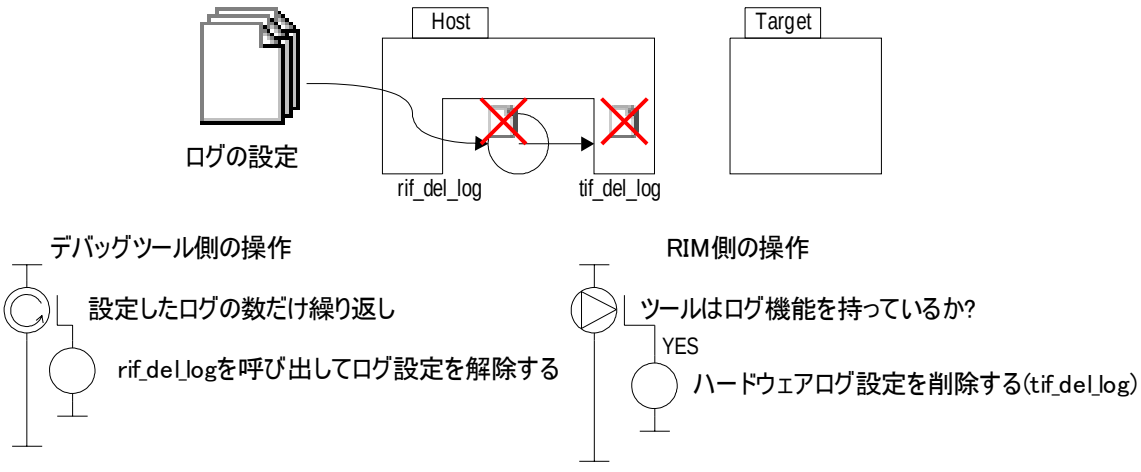


Figure 21: **トレースログ 解除**

トレースログ解除時には，RIM が内部に持つログキューから設定を削除し，デバッグツールがハードウェアログ機能を持っていた場合に設定したハードウェアログ設定も削除する．

5.RTOS アクセスインターフェース

5.1 機能単位

RTOS アクセスインターフェース上の関数は，すべて機能単位というグループによってまとめられ，この機能単位ごとに利用可能 / 不可能を決める．
各機能単位が利用可能であるとは，その機能単位を構成するすべての関数が実装されており，かつその機能単位を識別するキーコードが存在することである．ただし関数の一部の機能が実装されておらず，***E_NOSPT*** エラーを返すこともある．

機能単位はそれぞれ次のとおり (括弧内はそれぞれの略称)

- **オブジェクト状態取得 [OBJ]**
- **コンテキスト取得 [CTX]**
- **サービスコール発行 [SVC]**
- **ブレーク設定 [BRK]**
- **ブレーク条件取得 [CND]**
- **実行履歴 [LOG]**

キー

RIF 4h
.UNIT 20h
.OBJ 1h[1]
機能単位 [オブジェクト状態取得] をサポートする
.LOG 2h[1]
機能単位 [実行履歴取得] をサポートする
.SVC 3h[1]
機能単位 [サービスコール発行] をサポートする
.BRK 4h[1]
機能単位 [ブレーク設定] をサポートする
.CND 5h[1]
機能単位 [ブレーク条件取得] をサポートする
.CTX 6h[1]
機能単位 [コンテキスト取得] をサポートする

5.2 オブジェクト状態の取り出し

rif_ref_obj　オブジェクトの状態取りだし [OBJ] ○

ER rif_ref_obj
(VP p_result, UINT objtype, DT_ID objid, FLAG flags)

VP p_result
結果格納場所

UINT objtype
オブジェクトの種別

DT_ID objid
取り出し対象のオブジェクト ID

FLAG flags
諸フラグ

この関数は，現在 RTOS 上に存在するオブジェクトの状態を取得する．

オブジェクト状態の取り出しには，オブジェクトの種別を特定するためのフラグ (***objtype***) と，状態を格納する結果パケットを利用する．このとき，「待ちタスク ID リスト」など，type ***** identifier-***lst*** で表記される可変長データは，それと対になる ***UINT*** identifier-***cnt*** で表記される「個数パラメータ」に上限値を設定して本関数を呼ぶことで，その読み出し上限が決定する．このとき，操作後に「個数パラメータ」に代入される値は，操作前に代入した「読み出し上限値」と実際の可変長データ個数のどちらかすくない方となり，読み出し上限値が実際の可変長データ個数を下回った場合は，この上限値を超えて可変長データの転送は行われない．

下の Table 19: [10 個のタスクが待ちリストにあるセマフォに対する操作] に，10 個のタスクが待ちリストにあるセマフォに対して ***rif_ref_obj*** を発行したときの，***R_ROSEM.wtskcnt*** と ***wtsklst*** が指す領域に格納される内容の対応を示す．

Table 19: 10 個のタスクが待ちリストにあるセマフォに対する操作

T_ROSEM.wtskcnt		wtsklst が指す領域に格納される内容
実行前	実行後	
0	0	何も格納されない
1	1	待ちキューの先頭にいるタスク ID
2	2	先頭待ちタスク ID 第 2 位待ちタスク ID
10	10	先頭待ちタスク ID … 末尾待ちタスク ID
11	10	同上

rif_ref_obj が返すパケットの先頭には, 後続のフィールドが有効であるか無効であるかを示すビットマスクが配置されている．ビットマスクの第一候補は ***valid*** に続く構造体メンバである．構造体メンバが，他の構造体へのポインタであった場合，ポインタが示す先の構造体メンバの有効 / 無効識別情報が格納され，他の構造体へのポインタ自体の有効 / 無効識別情報は格納されない．ポインタ自体が無効である場合は，ポインタが示す構造体の全メンバを無効にする．ビットマスクの詳細に関しては，3.5 [ビットマスク] ビットマスクを参照されたい．
例として，固定長メモリプール (***OBJ_FMEMPOOL***) の情報を格納する構造体 ***T_ROMPF*** の各フィールドと，対応するビットマスクのビット位置を示す．

Table 20: T_ROMPF **のメンバとビットマスクのビット位置の対応**

ビット位置	構造体メンバ
0	***T_ROMPF::mpfatr***
1	***T_ROMPF::blksz***
2	***T_ROMPF::fblkcnt***
3	***T_ROMPF::blkcnt***
4	***T_ROMPF::ablkcnt***
5	***T_ROMPF_BLKLST::htskid***
6	***T_ROMPF_BLKLST::blkadr***
7	***T_ROMPF::wtskcnt***
8	***T_ROMPF::wtsklst***

オブジェクトを判別するためのフラグ群 (***ObjType***) と結果パケットを以下に示す. オブジェクト名の後に NC とあるものは, その項目や引数が無効であることを意味する.

•*OBJ_SEMAPHORE* (0x80) : セマフォ

```
typedef struct    t_rosem
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR sematr     : セマフォ属性
    DT_UINT isemcnt   : 初期セマフォカウント
    DT_UINT maxsem    : セマフォ資源の最大値
    DT_UINT semcnt    : セマフォカウント値
    DT_UINT wtskcnt   : 待ちタスク数 (wtsklst の上界を兼ねる )
    DT_ID * wtsklst   : 待ちタスク ID リストを格納する領域へのポインタ
}     T_ROSEM;
```

セマフォに関する情報を取得する．実行前に ***wtsklst*** と ***wtskcnt*** を初期化すること．

•*OBJ_EVENTFLAG* (0x81) : イベントフラグ

```
typedef struct    t_roflg_wflglst
{
```

```
    DT_ID wtskid        : 待ちタスク ID
    DT_FLGPTN wflgptn: タスクの待ちフラグパターン
    DT_UINT wflgmode : タスクの待ちモード
}   T_ROFLG_WFLGLST;

typedef struct   t_roflg
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR flgatr    : フラグ属性
    DT_FLGPTN iflgptn : 初期フラグパターン
    DT_FLGPTN flgptn : フラグパターン
    DT_UINT wflgcnt  : 待ちタスク数 (wflglst の上界を兼ねる )
    T_ROFLG_WFLGLST * wflglst
                     : このフラグを待つタスクの情報へのポインタ
}   T_ROFLG;
```

イベントフラグに関する情報を取得する．実行前に ***wtskcnt***, ***wtsklst, wflgptn, wflgmode*** を初期化すること．

•*OBJ_DATAQUEUE* (0x82) : データキュー

```
typedef struct   t_rodtq
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR dtqatr    : データキュー属性
    DT_UINT dtqcnt   : データキュー容量
    DT_UINT stskcnt  : 送信待ちタスク数 (wstsklst の上界を兼ねる )
    DT_ID * stsklst  : 送信待ちタスク ID リストを格納する領域へのポインタ
    DT_UINT rtskcnt  : 受信待ちタスク数 (wrtsklst の上界を兼ねる )
    DT_ID * rtsklst  : 受信待ちタスク ID リストを格納する領域へのポインタ
    DT_UINT itemcnt  : キュー内データ数 (itemlst の上界を兼ねる )
    DT_VP_INT * itemlst
                     : 全内容のリストを格納する領域へのポインタ
}   T_RODTQ;
```

データキューに関する情報を取得する．実行前に ***stskcnt*** と ***wstsklst***, ***rtskcnt*** と ***wrtsklst***, ***itemcnt*** と ***itemlst*** をそれぞれ初期化すること．

•*OBJ_MAILBOX* (0x83) : メールボックス

```
typedef struct   t_rombx
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR mbxatr    : メールボックス属性
    DT_PRI maxmpri   : 優先度の最大値
    DT_UINT wtskcnt  : 待ちタスク数 (wtsklst の上界を兼ねる )
    DT_ID * wtsklst  : 待ちタスク ID リストを格納する領域へのポインタ
    DT_UINT msgcnt   : メッセージヘッダ数 (msglst の上界を兼ねる )
```

```
    DT_T_MSG ** msglst
                        : 全メッセージリストを格納する領域へのポインタ
}   T_ROMBX;
```

メールボックスに関する情報を取得する．実行前に ***wtskcnt*** と ***wtsklst***，***msgcnt*** と ***msglst*** を初期化すること．

•*OBJ_MUTEX* (0x84) : ミューテックス

```
typedef struct   t_romtx
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR mtxatr    : ミューテックス属性
    DT_PRI ceilpri   : 上限優先度
    DT_ID htskid     : ミューテックスをロックするタスクの ID
    DT_UINT wtskcnt  : 待ちタスク数 (wtsklst の上界を兼ねる )
    DT_ID * wtsklst  : 待ちタスク ID リストを格納する領域へのポインタ
}   T_ROMTX;
```

ミューテックスに関する情報を取得する．実行前に ***wtskcnt*** と ***wtsklst*** を初期化すること．

•*OBJ_MESSAGEBUFFER* (0x85): メッセージバッファ

```
typedef struct   t_rombf_msglst
{
    DT_VP msgadr : メッセージのアドレス
    DT_UINT msgsz : メッセージの長さ
}   T_ROMBF_MSGLST;

typedef struct   t_rombf
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR mbfatr    : メッセージバッファ属性
    DT_UINT maxmsz   : メッセージ最大サイズ
    DT_SIZE mbfsz    : バッファ領域のサイズ
    DT_UINT stskcnt  : 送信待ちタスク数 (stsklst の上界を兼ねる )
    DT_ID * stsklst  : 待ちタスク ID リストを格納する領域へのポインタ
    DT_UINT rtskcnt  : 受信待ちタスク数 (rtsklst の上界を兼ねる )
    DT_ID * rtsklst  : 待ちタスク ID リストを格納する領域へのポインタ
    DT_SIZE fmbfsz   : 空き領域の大きさ
    DT_UINT msgcnt   : メッセージ数 (msglst の上界を兼ねる )
    T_ROMBF_MSGLST * msglst
                        : 各メッセージの情報へのポインタ
}   T_ROMBF;
```

メッセージバッファに関する情報を取得する．実行前に ***stskcnt*** と ***wtsklst***，***msgcnt*** と ***msglst*** ならびに ***msgszlst*** のそれぞれを初期化すること．

•OBJ_RENDEZVOUSPORT (0x86) : ランデブポート

```
typedef struct    t_ropor
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR poratr     : ランデブポート属性
    DT_UINT maxcmsz : 呼び出しメッセージの最大サイズ
    DT_UINT maxrmsz : 呼応メッセージの最大サイズ
    DT_UINT ctskcnt   : 呼び出し待ちタスク数 (ctsklst の上界を兼ねる )
    DT_ID * ctsklst   : 全呼び出し待ちタスク ID を格納する領域へのポインタ
    DT_UINT atskcnt   : 受付待ちタスク数 (atsklst の上界を兼ねる )
    DT_ID * atsklst   : 全受付待ちタスク ID を格納する領域へのポインタ
}    T_ROPOR;
```

ランデブポートに関する情報を取得する．実行前に ***ctskcnt, atskcnt*** と ***ctsklst***, ***atsklst*** を初期化すること．

•OBJ_RENDEZVOUS (0x87) : ランデブ

```
typedef struct    t_rordv
{
    BITMASK valid     : 有効フィールド情報
    DT_ID tskid       : ランデブ待ち状態のタスク ID
}    T_RORDV;
```

ランデブに関する情報を取得する．

•OBJ_FMEMPOOL (0x88) : 固定長メモリプール

```
typedef struct    t_rompf_blklst
{
    DT_ID htskid      : ブロックを獲得しているタスクの ID 番号
    DT_VP blkadr      : ブロック開始アドレス
}    T_ROMPF_BLKLST;

typedef struct    t_rompf
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR mpfatr     : 固定長メモリプール属性
    DT_SIZE blksz     : ブロックサイズ
    DT_UINT fblkcnt   : 残り固定長メモリブロック数
    DT_UINT blkcnt    : 全メモリブロック数
    DT_UINT ablkcnt   : 獲得済みブロック数 (blklst の上界 )
    T_ROMPF_BLKLST * ablklst
                      : 各ブロックの詳細情報へのポインタ
    DT_UINT wtskcnt   : 獲得待ちタスクの数 (wtsklst の上界 )
    DT_ID * wtsklst   : 獲得待ちタスクの ID を格納する領域へのポインタ
}    T_ROMPF;
```

固定長メモリプールに関する情報を取得する．実行前に ***ablkcnt*** と ***ablklst***，***wtskcnt*** と ***wtsklst*** を初期化すること．

•*OBJ_VMEMPOOL* (0x89) : 可変長メモリプール

```
typedef struct    t_rompl_blklst
{
    DT_SIZE blksz      : ブロックの大きさ
    DT_ID htskid       : ブロックを獲得しているタスクの ID 番号
    DT_VP blkadr       : ブロック開始アドレス
}    T_ROMPL_BLKLST;

typedef struct    t_rompl
{
    BITMASK valid      : 有効フィールド情報
    DT_ATR mplatr      : 可変長メモリプール属性
    DT_SIZE mplsz      : 可変長メモリプール領域のサイズ
    DT_UINT fblksz     : 獲得可能な最大サイズ
    DT_UINT ablkcnt    : 獲得済みブロック数 (blklst の上界 )
    T_ROMPL_BLKLST * ablklst
                       : 各ブロックの詳細情報へのポインタ
    DT_UINT wtskcnt    : 獲得待ちタスクの数 (wtsklst の上界 )
    DT_ID * wtsklst    : 獲得待ちタスクの ID を格納する領域へのポインタ
}    T_ROMPL;
```

可変長メモリプールに関する情報を取得する．実行前に ***ablkcnt*** と ***ablklst***，***wtskcnt*** と ***wtsklst*** を初期化すること．

•*OBJ_TASK* (0x8a) : タスク

```
typedef struct    t_rotsk
{
    BITMASK valid      : 有効フィールド情報
    DT_ATR tskatr      : タスク属性
    DT_VP_INT exinf    : 拡張情報
    DT_FP task         : 起動番地
    DT_PRI itskpri     : 初期優先度
    DT_VP stk          : 初期スタックの先頭番地
    DT_SIZE stksz      : スタックサイズ
    DT_STAT tskstat    : タスク状態
    DT_PRI tskpri      : タスクの現在優先度
    DT_PRI tskbpri     : タスクのベース優先度
    DT_STAT tskwait    : タスクの待ち要因
    DT_ID wobjid       : 待ち対象のオブジェクト ID
    DT_TMO lefttmo     : タイムアウトまでの時間
    DT_UINT actcnt     : 起動要求キューイング数
    DT_UINT wupcnt     : 起床要求キューイング数
    DT_UINT suscnt     : 強制待ち要求ネスト数
```

} **T_ROTSK;**

タスクに関する情報を取得する.

•*OBJ_READYQUEUE* (0x8b) : レディーキュー (NC : objid)

typedef struct t_rordq
{
BITMASK valid : 有効フィールド情報
DT_ID runtskid : 現在実行中のタスク ID
DT_UINT tskcnt : レディー(含 RUNNING) 状態にあるタスク数 (tsklst の上界)
DT_ID * tsklst : 実行可能状態にある全タスク ID の格納領域へのポインタ
} **T_RORDQ;**

レディーキューに関する情報を取得する. 実行可能なタスクが存在しない場合, ***runtskid*** および ***tskcnt*** には 0 が返る. このとき ***tsklst*** の内容には変化はない.
実行前に ***tskcnt*** と ***tsklst*** を初期化すること.

•*OBJ_TIMERQUEUE* (0x8c) : タイマーキュー (NC : objid)

typedef struct t_rotmq_quelst
{
UINT objtype : 待ちオブジェクトの種別を格納する領域へのポインタ
DT_ID wobjid : 待ちオブジェクトの ID を格納する領域へのポインタ
DT_TMO lefttmo : 残り待ち時間を格納する領域へのポインタ
} **T_ROTMQ_QUELST;**

typedef struct t_rotmq
{
BITMASK valid : 有効フィールド情報
DT_SYSTIM systim : 情報取得時のシステム時刻
DT_UINT quecnt : タイマーキュー内の待ちオブジェクト数 (quelst の上界)
T_ROTMQ_QUELST * quelst
: タイマーキュー内の待ちオブジェクト情報へのポインタ
} **T_ROTMQ;**

タイマーキューに関する情報を取得する.
タイマーキュー情報にはタイムイベントによって起動する全てのイベント (周期起動ハンドラ, アラームハンドラ, オーバーランハンドラ, タスク) の種別と, そのイベントの発生予定時刻が含まれる. ただし周期起動ハンドラに関しては, 今後全ての起動位置ではなく, 次の起動位置のみ取得できる.
待ちオブジェクトの種別は ***rif_ref_obj*** でオブジェクトを選定するときに用いられる ***OBJ_xxx*** で示される定数が格納される.
実行前に ***quecnt*** と ***objtyplst*** ならびに ***wobjidlst*** および ***lefttmolst*** を初期化すること.

•*OBJ_CYCLICHANDLER* (0x8d) : 周期起動ハンドラ

typedef struct t_rocyc
{
BITMASK valid : 有効フィールド情報
DT_ATR cycatr : 属性

```
    DT_VP_INT exinf   : 拡張情報
    DT_FP cychdr      : 起動番地
    DT_RELTIM cyctim : 周期
    DT_RELTIM cycphs : 起動位相
    DT_STAT cycstat   : 周期起動ハンドラの起動状態
    DT_RELTIM lefttim : 残り時間
}   T_ROCYC;
```

周期起動ハンドラに関する情報を取得する.

•OBJ_ALARMHANDLER (0x8e): アラームハンドラ

```
typedef struct    t_roalm
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR almatr     : 属性
    DT_VP_INT exinf   : 拡張情報
    DT_FP almhdr      : 起動番地
    DT_STAT almstat   : アラームハンドラの起動状態
    DT_RELTIM lefttim : 残り時間
}   T_ROALM;
```

アラームハンドラに関する情報を取得する.

•OBJ_OVERRUNHANDLER (0x8f): オーバーランハンドラ

```
typedef struct    t_roovr
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR ovratr     : 属性
    DT_FP ovrhdr      : 起動番地
    DT_STAT ovrstat   : ハンドラの起動状態
    DT_OVRTIM lefttmo
                      : 残りプロセッサ時間
}   T_ROOVR;
```

オーバーランハンドラに関する情報を取得する.

•OBJ_ISR (0x90) : 割込サービスルーチン

```
typedef struct    t_roisr
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR isratr     : 属性
    DT_VP_INT exinf   : 拡張情報
    DT_FP isrfnclst   : 登録済みルーチンの開始番地
    DT_INHNO inhno    : 対応づけられた割込みハンドラ番号
}   T_ROISR;
```

割込みサービスルーチンに関する情報を取得する.

•*OBJ_KERNELSTATUS* (0x91) : カーネル情報 (NC : objid)

```
typedef struct    t_roker
{
    BITMASK valid     : 有効フィールド情報
    BOOL actker       : カーネル起動状態 (TRUE= 起動している )
    BOOL inker        : カーネルコードを実行している (TRUE= 実行中 )
    BOOL ctxstat      : コンテキスト状態 (sns_ctx)
    BOOL loccpu       : CPU ロック状態 (sns_cpu)
    BOOL disdsp       : ディスパッチ禁止状態 (sns_dsp)
    BOOL dsppnd       : ディスパッチ保留状態 (sns_dpn)
    DT_SYSTIM systim : システム時刻
    DT_VP intstk      : 非タスクコンテキスト時のスタック
    DT_SIZE intstksz  : 非タスクコンテキスト時のスタックのサイズ
}     T_ROKER;
```

カーネル状態に関する情報を取得する.
actker はカーネルの起動状況を表す変数である. ターゲット起動時は ***FALSE*** であり, カーネル初期化が終了しシステムコールが発行可能となった時点で ***TRUE*** となる.
inker は現在実行しているのがカーネルコードであるかどうかを示す変数である.[1]

•*OBJ_TASKEXCEPTION* (0x92) : タスク例外ハンドラ

```
typedef struct    t_rotex
{
    BITMASK valid     : 有効フィールド情報
    DT_TEXPTN pndptn : 保留状態にある例外要因
    DT_FP texrtn      : 例外ハンドラ起動番地
}     T_ROTEX;
```

タスク例外ハンドラに関する情報を取得する.

•*OBJ_CPUEXCEPTION* (0x93) :CPU 例外ハンドラ (*objid* は例外要因に相当)

```
typedef struct    t_roexc
{
    BITMASK valid     : 有効フィールド情報
    DT_FP excrtn      : 例外ハンドラ起動番地
}     T_ROEXC;
```

CPU 例外に関する情報を取得する.

補足説明

実装依存情報は各構造体に後続する構造体メンバとして定義される.

1. 次の状態はカーネル動作と判断する. 例外発生からハンドラ起動まで. ハンドラ終了からディスパッチャ終了まで. ターゲット起動から最初のアプリケーションタスク起動まで.

独自のオブジェクトを定義する場合，オブジェクト識別定数は 0 から 255 までの値を採ってはならない．また独自のオブジェクトの操作を行う場合，その事を明確にするようフラグを設けることを薦める．

フラグ

OPT_GETMAXCNT (1)
: 上界が可変長データ数を下回った場合でも，データ個数だけは完全に追跡，取得する

OPT_VENDORDEPEND (2)
: 実装依存情報を取得する

FLG_NOCONSISTENCE (10000000h) : 非一貫性フラグ
: このフラグが指定されたときは，取得した内容に一貫性を持たせる必要はない．例として，「タスク待ち要因がないのにタスク状態は待ちから解除されていない」など．

FLG_NOSYSTEMSTOP (20000000h) : 明示的なシステム停止を許さない
: このフラグが指定された場合，関数内で ***tif_brk_tgt*** を利用してシステムを停止するような操作を行ってはならない．このフラグをサポートしない場合は，***E_NOSPT*** エラーとなる．

キー

RIF 4h
- **.RIF_REF_OBJ** 1h
  - **.FLG_NOCONSISTENCE** 1h[1]
    フラグ ***FLG_NOCONSISTENCE*** は利用可能である
  - **.FLG_NOSYSTEMSTOP** 2h[1]
    フラグ ***FLG_NOSYSTEMSTOP*** は利用可能である
  - **.OPT_VENDORDEPEND** 10h[1]
    オプション ***OPT_VENDORDEPEND*** は利用可能である
  - **.OPT_GETMAXCNT** 11h[1]
    オプション ***OPT_GETMAXCNT*** は利用可能である
  - **.STATICPARAMETER** 12h
    - **.OBJ_SEMAPHORE** 80h[T]
      セマフォ情報のうち静的に決定する情報をもつ構造体
    - **.OBJ_EVENTFLAG** 81h[T]
      イベントフラグ情報のうち静的に決定する情報をもつ構造体
    - **.OBJ_DATAQUEUE** 82h[T]
      データキュー情報のうち静的に決定する情報をもつ構造体
    - **.OBJ_MAILBOX** 83h[T]
      メールボックス情報のうち静的に決定する情報をもつ構造体
    - **.OBJ_MUTEX** 84h[T]
      ミューテックス情報のうち静的に決定する情報をもつ構造体
    - **.OBJ_MESSAGEBUFFER** 85h[T]
      メッセージボックス情報のうち静的に決定する情報をもつ構造体
    - **.OBJ_RENDEZVOUSPORT** 86h[T]
      ランデブポート情報のうち静的に決定する情報をもつ構造体

.OBJ_RENDEZVOUS 87h[T]
ランデブ情報のうち静的に決定する情報をもつ構造体
.OBJ_FMEMPOOL 88h[T]
固定長メモリプール情報のうち静的に決定する情報をもつ構造体
.OBJ_VMEMPOOL 89h[T]
可変長メモリプール情報のうち静的に決定する情報をもつ構造体
.OBJ_TASK 8Ah[T]
タスク情報のうち静的に決定する情報をもつ構造体
.OBJ_READYQUEUE 8Bh[T]
レディーキュー情報のうち静的に決定する情報をもつ構造体
.OBJ_TIMERQUEUE 8Ch[T]
タイマーキュー情報のうち静的に決定する情報をもつ構造体
.OBJ_CYCLICHANDLER 8Dh[T]
周期起動ハンドラ情報のうち静的に決定する情報をもつ構造体
.OBJ_ALARMHANDLER 8Eh[T]
アラームハンドラ情報のうち静的に決定する情報をもつ構造体
.OBJ_OVERRUNHANDLER 8Fh[T]
オーバーランハンドラ情報のうち静的に決定する情報をもつ構造体
.OBJ_ISR 90h[T]
割込みサービスルーチン情報のうち静的に決定する情報をもつ構造体
.OBJ_KERNELSTATUS 91h[T]
カーネル情報のうち静的に決定する情報をもつ構造体
.OBJ_TASKEXCEPTION 92h[T]
タスク例外情報のうち静的に決定する情報をもつ構造体
.OBJ_CPUEXCEPTION 93h[T]
CPU 例外情報のうち静的に決定する情報をもつ構造体

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_CONSIST (-225)
一貫性を保証できなかった (ただし ***FLG_NOCONSISTENCE*** が設定されていた場合に限ってはエラーとならない)

ET_NOEXS (-42)
ターゲット上に対象となるオブジェクトが存在しない

E_PAR (-145)
パラメータが不正な値であった

ET_OBJ (-41)
ターゲット上の対象オブジェクトは操作できない

ET_ID (-18)
指定されたカーネルオブジェクト ID は無効

ET_OACV (-27)
不正なターゲット上のオブジェクトに対するアクセス (tskid < 0)

5.3 タスクコンテキストの取り出し

5.3.1 レジスタセットディスクリプションテーブルの取得

rif_get_rdt　ディスクリプションテーブルの取得　[CTX] ○

ER　　rif_get_rdt (const T_GRDT ** ppk_pgrdt, FLAG flags)

const T_GRDT **　ppk_prgrdt
　　レジスタセットディスクリプションテーブル構造体へのポインタを格納する領域へのポインタ

FLAG　　flags
　　フラグ

この命令は，対象となるタスクのコンテキスト情報を持つレジスタテーブルへのポインタを取得する．このテーブルの本体は RIM 内部にあり，その内容は不変である．デバッグツールが内容を修正するような場合，デバッグツール側で複製を作成し，複製した領域上の内容を修正する．

レジスタテーブルは，タスクスイッチ時に退避しなければならないレジスタの詳細を持つ．以下に構造体 ***T_GRDT*** の詳細を示す．レジスタテーブルに関しては 3.12 [レジスタセットディスクリプションテーブル] を参照のこと．

```
typedef struct    t_grdt_regary
{
    char * strname      : レジスタ名称を指すポインタ
    UINT length         : 長さ ( バイト単位 )
    UINT offset         : 格納オフセット位置
}    T_GRDT_REGARY;

typedef struct    t_grdt
{
    UINT regcnt         : レジスタ本数
    UINT ctxcnt         : コンテキストに含まれる可能性のあるレジスタの数
    T_GRDT_REGARY regary[]
                        : レジスタ情報
}    T_GRDT;
```

補足説明

レジスタセットディスクリプションテーブルはコンテキストを構成するすべてのレジスタと，RIM が操作対象とする全てのレジスタを含んでいる．そのうち，タスクコンテキストは，***T_GRDT::regary*** の先頭から ***T_GRDT::ctxcnt*** で示された個数の要素が対象となる．また ***T_GRDT::regcnt*** は，RIM が操作対象とするレジスタの数を示している．

キー

RIF 04h
.RIF_GET_RDT 02h
.REGISTER 2h
.SIZE 04h[W]
レジスタの格納に十分な領域のサイズ (バイト単位)
.CONTEXT 12h
.SIZE 04h[W]
コンテキストの格納に十分な領域のサイズ (バイト単位)

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_PAR (-145)
パラメータが不正な値であった

5.3.2 タスクコンテキストの取得

rif_get_ctx　タスクコンテキストの取得 [CTX] ○

ER rif_get_ctx

(VP p_ctxblk, BITMASK_8 * p_valid, DT_ID tskid, FLAG flags)

VP p_ctxblk
: 取得したコンテキストを格納する領域を指す先頭ポインタ

BITMASK_8 * p_valid
: レジスタテーブルの各要素の有効 / 無効情報へのポインタ
(NULL : 全てのコンテキストを対象)

DT_ID tskid
: 対象となるタスク ID

FLAG flags
: フラグ

この関数は，***rif_get_rdt*** によって得られるレジスタテーブルをもとに，タスクコンテキストの取得，格納を行う．この関数を利用することにより，デバッグツールは現在のタスク状態に関わらず，常に適切な領域からコンテキストを取得することができる．

変数 ***p_ctxblk*** は，この関数の実行によって取得されるコンテキストを格納するバッファへのポインタである．デバッグツールはこの関数を実行する前に，コンテキストの格納に十分なサイズを持った領域を作成しなければならない．このバッファのサイズは，情報取得キーコード ***RIF.RIF_GET_RDT.CONTEXT.SIZE*** を利用することで取得できる．また，関数 ***rif_get_rdt*** によって取得したレジスタテーブルからも算出可能である．算出によって領域の大きさを求める場合，レジスタテーブルのうち，コンテキスト部のみを格納するのに十分な大きさの領域を用意すればよいことに注意されたい．

格納は，***rif_get_rdt*** で取得したレジスタテーブルに記載された格納オフセット位置とレジスタ長を基に行われる．レジスタテーブルに関しては 3.12 [レジスタセットディスクリプションテーブル] を参照のこと．

p_valid は各レジスタの有効 / 無効を指定する．関数の引数として与えた際，無効となったレジスタの格納は行われない．またこの関数は，対象となったレジスタ取得の結果を ***p_valid*** に格納する．ただし，取得できなかったレジスタが対象タスクのコンテキストに必須であった場合，状況に応じて ***ET_MACV*** などのエラーが返る[1]．取得できなかったレジスタに対応する領域に格納される情報は，実装依存である．また，コンテキストを構成するレジスタの数 (***T_GRDT::ctxcnt***) を超えて有効 / 無効情報が与えられても，その数を超えたレジスタの取得は行われない．

1. 例えば浮動小数点レジスタなどは，浮動小数点演算を行わないタスクにとっては不要である．このような場合では，浮動小数点レジスタがレジスタテーブルに含まれていても，関数は浮動小数点レジスタを取得せずに E_OK を返す場合がある．

p_valid に NULL が指定された場合，全てのコンテキストが取得対象となり，詳細な結果の格納は行われない．

フラグ ***OPT_APPCONTEXT*** が指定された場合，アプリケーションレベルでのコンテキストを取得する．タスクがカーネル内部で停止している場合，RIM は現在のスタックフレームなどからカーネルコードに入る前のコンテキストを生成し，返却する．

フラグ

OPT_APPCONTEXT (1)
: アプリケーションレベルのコンテキストを対象とする

FLG_NOCONSISTENCE (10000000h) : 非一貫性フラグ
: このフラグが指定されたときは，取得した内容に一貫性を持たせる必要はない．例として，「タスク待ち要因がないのにタスク状態は待ちから解除されていない」など．

FLG_NOSYSTEMSTOP (20000000h) : 明示的なシステム停止を許さない
: このフラグが指定された場合，関数内で ***tif_brk_tgt*** を利用してシステムを停止するような操作を行ってはならない．このフラグをサポートしない場合は，***E_NOSPT*** エラーとなる．

キー

RIF 04h
- **.RIF_GET_CTX** 03h
  - **.FLG_NOCONSISTENCE** 01h[1]
    フラグ ***FLG_NOCONSISTENCE*** は利用可能である
  - **.FLG_NOSYSTEMSTOP** 02h[1]
    フラグ ***FLG_NOSYSTEMSTOP*** は利用可能である
  - **.OPT_APPCONTEXT** 10h[1]
    オプション ***OPT_APPCONTEXT*** は利用可能である

エラー

E_OK (0)
: 正常に終了した

E_NOSPT (-137)
: その操作はサポートしない

E_NOMEM (-161)
: ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
: 何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
: 何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_CONSIST (-225)
: 一貫性を保証できなかった (ただし ***FLG_NOCONSISTENCE*** が設定されていた場合に限ってはエラーとならない)

ET_OBJ (-41)
: ターゲット上の対象オブジェクトは操作できない

ET_OACV (-27)
不正なターゲット上のオブジェクトに対するアクセス (tskid < 0)
ET_ID (-18)
指定されたカーネルオブジェクト ID は無効
E_PAR (-145)
パラメータが不正な値であった
ET_NOEXS (-42)
ターゲット上に対象となるオブジェクトが存在しない

5.3.3 タスクコンテキストの設定

rif_set_ctx　タスクコンテキストの設定 [CTX] ○

ER rif_set_ctx

(VP p_ctxblk, BITMASK_8 * valid, FLAG flags)

VP p_ctxblk
設定するコンテキストを格納する領域の先頭を指すポインタ

BITMASK_8 * p_valid
レジスタテーブルの各要素の有効 / 無効情報へのポインタ
(NULL : 全てのコンテキストを対象)

FLAG flags
フラグ

この関数は, ***rif_get_rdt*** によって得られるレジスタテーブルをもとに, タスクコンテキストの設定を行う. この関数を利用することにより, デバッグツールは現在のタスク状態に関わらず, 常に適切なコンテキストを設定することができる.

設定は ***rif_get_rdt*** で取得できるレジスタテーブルの情報を基に行われる. 変数 ***p_ctxblk*** はこの関数の実行によって設定されるコンテキストを格納するバッファへのポインタである. デバッグツールはこの関数を実行する前に, ***rif_get_rdt*** によって取得できるレジスタテーブルに従って, 設定すべきコンテキスト内容を領域に格納しなければならない. レジスタテーブルに関しては 3.12 [レジスタセットディスクリプションテーブル] を参照のこと.

p_valid は各レジスタの有効 / 無効を指定する. 関数の引数として与えた際, 無効となったレジスタの設定は行われない. またこの関数は, 対象となったレジスタ取得の結果を ***p_valid*** に格納する. ただし, 設定できなかったレジスタが, 対象タスクのコンテキストにとって必須であった場合, 状況に応じて ***ET_MACV*** 等のエラーが返る[1]. また, コンテキストを構成するレジスタの数 (***T_GRDT::ctxcnt***) を超えて有効 / 無効情報が与えられても, その数を超えたレジスタの設定は行われない.
p_valid に ***NULL*** が指定された場合, 全てのコンテキストが設定対象となり, 詳細な結果の格納は行われない.

フラグ ***OPT_APPCONTEXT*** が指定された場合, アプリケーションレベルでのコンテキストを設定する.

フラグ

OPT_APPCONTEXT (1)
アプリケーションレベルのコンテキストを対象とする

1. 例えば浮動小数点レジスタなどは, レジスタテーブルに含まれていても, 浮動小数点演算を行わないタスクにとっては不要である. このような場合では, 関数は浮動小数点レジスタが設定対象であっても設定せずに E_OK を返す場合がある.

FLG_NOSYSTEMSTOP (20000000h) : 明示的なシステム停止を許さない

このフラグが指定された場合，関数内で ***tif_brk_tgt*** を利用してシステムを停止するような操作を行ってはならない．このフラグをサポートしない場合は，***E_NOSPT*** エラーとなる．

キー

RIF 04h

.RIF_SET_CTX 13h

.FLG_NOSYSTEMSTOP 02h[1]

フラグ ***FLG_NOSYSTEMSTOP*** は利用可能である

.OPT_APPCONTEXT 10h[1]

オプション ***OPT_APPCONTEXT*** は利用可能である

エラー

E_OK (0)

正常に終了した

E_NOSPT (-137)

その操作はサポートしない

E_NOMEM (-161)

ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)

何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)

何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_CONSIST (-225)

一貫性を保証できなかった (ただし ***FLG_NOCONSISTENCE*** が設定されていた場合に限ってはエラーとならない)

ET_OBJ (-41)

ターゲット上の対象オブジェクトは操作できない

ET_OACV (-27)

不正なターゲット上のオブジェクトに対するアクセス (tskid < 0)

ET_ID (-18)

指定されたカーネルオブジェクト ID は無効

E_PAR (-145)

パラメータが不正な値であった

ET_NOEXS (-42)

ターゲット上に対象となるオブジェクトが存在しない

5.4 サービスコールの発行

5.4.1 サービスコールの発行

rif_cal_svc　サービスコールの発行 [SVC] ○

ER　　rif_cal_svc (T_RCSVC * pk_psvc , FLAG flags)

T_RCSVC *　　pk_psvc
　　発行情報

FLAG　　flags
　　フラグ

この関数はサービスコールの発行を行う．発行はノンブロッキングで行われるため，この関数の終了がサービスコールの終了を意味しているわけではない．ただし，明示的に ***OPT_BLOCKING*** が設定され場合に限り，特殊ブロッキングモードで発行する．特殊ブロッキングモードで実行した場合，処理は有限時間でタイムアウトする．

T_RCSVC の内容を次に示す．

```
typedef struct    t_rcsvc
{
    DT_FN svcfn       : 発行する機能コード
    BOOL tskctx       : タスクコンテキストで実行する (= TRUE)
    DT_ID tskid       : 対象となるタスク ID (tskctx = TRUE 時 )
    UINT prmcnt       : パラメータの数
    VP_INT prmary[]   : 全パラメータリストを格納する配列
}    T_RCSVC;
```

補足説明

本関数はノンブロッキングで行われるため，本関数の終了と発行したサービスコールの終了は同義ではない．ただし，実装の仕方によってノンブロッキングが不可能である場合，ブロッキングで実装することで，本関数の終了がサービスコール終了とみなすことは可能である．その場合，***dbg_ref_rim*** 関数はそのことをデバッグツール側に伝えるよう ***RIF.RIF_CAL_SVC.NONBLOCKING*** の項目を ***FALSE*** として実装しなければならない．

T_RCSVC::tskctx に ***FALSE*** を設定すると非タスクコンテキストで実行する．

OPT_BLOCKING が指定された場合でも，***FLG_NOREPORT*** が指定されていなければ，コールバック関数 ***rif_rep_svc*** が呼び出される．

rif_cal_svc を特殊ブロッキングモードで実行した場合，サービスコールが厳密な意味で終了するまで本関数は制御を返さない場合がある．厳密な終了とは，関数終了時のスタックフレームが ***rif_cal_svc*** にて関数を呼出した時のスタックフレームと等価となる場合を指す．具体的には，関数内で待ち状態になるなどディスパッチを伴うようなサービスコールを実行した場合，再度同一タスクへのディスパッチが起って対象サービスコールが完了するまで本関数は制御を返さない．また対象関数内で再帰的に同一関数が実行された場合，呼び出した回数だけ終了しなければ本関数は制御を返さ

ない．ただし，特殊ブロッキングモードで実行した場合は，厳密に終了しなくとも，有限時間でタイムアウトする．詳細は 3.13 [特殊なブロッキングモード] を参照されたい．
T_RCSVC::prmary にはパラメータとして渡すべき値を格納する．パラメータの渡し方は µITRON4.0 仕様のサービスコール ***cal_svc*** に準ずる (構造体などは構造体へのポインタを格納する).

フラグ

FLG_NOREPORT (80000000h) : 通知関数の無効化
対になるコールバック関数の呼出しを行わない
OPT_BLOCKING (1)
ブロッキングモードで実行する

キー

RIF 04h
.RIF_CAL_SVC 04h
.FLG_NOREPORT 03h[1]
フラグ ***FLG_NOREPORT*** 利用可能である
.OPT_BLOCKING 10h[1]
フラグ ***OPT_BLOCKING*** 利用可能である
.NONBLOCKING 12h[1]
ノンブロッキング SVC 発行に対応している

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_PAR (-145)
パラメータが不正な値であった
E_EXCLUSIVE (-226)
すでに要求が発行されており，終了するまで受け付けることができない
ET_OBJ (-41)
ターゲット上の対象オブジェクトは操作できない
ET_OACV (-27)
不正なターゲット上のオブジェクトに対するアクセス (tskid < 0)
ET_ID (-18)
指定されたカーネルオブジェクト ID は無効

ET_NOEXS (-42)

ターゲット上に対象となるオブジェクトが存在しない

5.4.2 発行したサービスコールの取り消し

rif_can_svc 発行したサービスコールの取り消し [SVC] ○

ER rif_can_svc (FLAG flags)

FLAG flags
フラグ

この関数は直前に発行したサービスコールを取り消す．ただし，発行によって失ったフォーカスを取得するのが目的であり，完全にサービスコールの影響を取り消すことが出来るわけではない．

フラグ

OPT_CANCEL (0)
発行したサービスコールの影響に関しては考慮しない **(default)**

OPT_UNDO (1)
完全に発行前の状態に戻す

RIF 04h
.RIF_CAN_SVC 05h[1]
rif_can_svc を実装している
.OPT_CANCEL 10h[1]
オプション ***OPT_CANCEL*** は利用可能である
.OPT_UNDO 11h[1]
オプション ***OPT_UNDO*** は利用可能である

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_OBJ (-169)
操作対象は存在しない または 操作できない

5.4.3 サービスコールの終了通知

rif_rep_svc サービスコールの終了通知 [SVC:callback] □

void rif_rep_svc (DT_ER result)

DT_ER result

最後に発行されたサービスコールのエラーコード

rif_cal_svc で発行されたサービスコールが終了したとき，デバッグツールはコールバック関数 ***rif_rep_svc*** を呼び出すことで，サービスコールの終了を RIM に通知する．***rif_rep_svc*** は，最後に発行されたサービスコールのエラーコードを受け取るためのコールバック関数でもある．ただし，***rif_cal_svc*** でサービスコールを発行する際に ***FLG_NOREPORT*** フラグを指定した場合，この関数による終了通知は行われない．

補足説明

引数 ***result*** にはカーネル仕様のエラーコード (***ET_xxx***) が格納される．

本関数が呼び出されるタイミングはサービスコール終了と同時であるため，***rif_cal_svc*** を脱出する前に終了通知が行われる可能性がある．そのため，次のようなコードを書いてはならない．

Program source

```
volatile int flag;
rif_rep_svc(err)
{  flag = 1; }

foo()
{
    rif_cal_svc(…);
        // フラグをクリア ( すでにこの時点で通知されているかもしれない )
    flag = 0;
        // サービスコールが終わるまでブロッキング
    while(flag == 0);
}
```

Program source

キー

RIF 04h

.RIF_CAL_SVC 06h

エラー

本関数は値を返さない．

5.4.4 機能コードの取得

rif_ref_svc 機能コードの取得 [SVC] ○

ER rif_ref_svc (DT_FN * p_svcfn, char * strsvc, FLAG flags)

DT_FN * p_svcfn

サービスコール名称に対応する機能コードを格納する領域へのポインタ

char * strsvc

対象となるサービスコール名称

rif_ref_svc はサービスコール関数名から機能コードを取得する．この関数によって得られた機能コードは，***rif_cal_svc***, ***rif_set_brk***, ***rif_set_log*** など機能コードをパラメータに持つ関数で利用することができる．

補足説明

この関数の引数となる ***strsvc***(対象となるサービスコール名称) は µITRON 規格で定められた API 名称に相当する．サービスコール名称に C 言語における接頭語 "_" や，C++ 言語における接尾辞 (パラメータの型やバイト数など) が付加された場合，関数は正常に動作することを保証しない. 同様に API 名称に続く括弧や括弧内にパラメータ部が指定されている場合も保証しない.

キー

RIF 04h

.RIF_REF_SVC 07h

エラー

E_OK (0)

正常に終了した

E_NOSPT (-137)

その操作はサポートしない

E_NOMEM (-161)

ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)

何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)

何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_OBJ (-169)

操作対象は存在しない または 操作できない

E_PAR (-145)

パラメータが不正な値であった

5.4.5 サービスコール名称の取得

rif_rrf_svc　サービスコール名称の取得 [SVC] ○

ER　　rif_rrf_svc
　　(char * p_strsvc, UINT bufsz, DT_FN svcfn, FLAG flags)

　　char *　　p_strsvc
　　サービスコール名称を格納する領域の先頭をさすポインタ

　　UINT　　bufsz
　　名称を格納するバッファのサイズ (含 終端記号)

　　DT_FN　　svcfn
　　対象となるサービスコールの機能コード

　　FLAG　　flags
　　フラグ

rif_rrf_svc は機能コードを基にサービスコール名称を取得する.

補足説明

この関数の返値となる ***p_strsvc***(サービルコール名称) は μITRON 規格で定められた API 名称である．C 言語における接頭語 "_" や C++ 言語における接尾辞 (パラメータの型やバイト数など) は付加されない．同様に API 名称に続く括弧や括弧内にパラメータ部なども付加されない.
引数 ***bufsz*** には，***p_strsvc*** に指定されたバッファ領域のサイズをバイト単位で指定する．このとき，***bufsz*** は終端記号も含まれるために，完全に取り出すためにはサービスコール名称の長さ +1 以上が指定されなくてはならない．これに満たない場合，終端記号を含めこれを超えない長さまでのサービスコール名称が格納される．***bufsz*** が 1 である場合は終端記号のみが格納され正常終了となるが，***bufsz*** が 0 である場合は ***E_PAR*** エラーとなる.

キー

RIF　04h
　.RIF_RRF_SVC　08h

エラー

E_OK (0)
　正常に終了した
E_NOSPT (-137)
　その操作はサポートしない
E_NOMEM (-161)
　ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
　何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_OBJ (-169)
操作対象は存在しない または 操作できない
E_PAR (-145)
パラメータが不正な値であった

5.5 ブレークポイントの設定

5.5.1 ブレークポイントの設定

rif_set_brk　ブレークポイントの設定要求　[BRK] ○

ER_ID　rif_set_brk (ID brkid, T_RSBRK * pk_rsbrk , FLAG flags)

ID　brkid
　ブレークポイント ID

T_RSBRK *　pk_rsbrk
　設定するブレークの情報を持つ構造体へのポインタ

FLAG　flags
　フラグ

(返値) ID　brkid
　割り当てられたブレークポイント ID

この関数は，RTOS に依存するようなブレークを設定する機能を提供する．ブレークポイントには，それぞれブレークポイント ID が割り当てられる．ブレークポイント ID は非 0 の正数で表現され，解除やヒット通知の際にこの番号が用いられる．

T_RSBRK の構造は次の通り

```
typedef struct    t_rsbrk
{
    UINT brktype      : ブレークの種別
    UINT brkcnt       : ブレークするまでの回数
    DT_ID tskid       : タスク ID
    DT_ID objid       : オブジェクト ID
    UINT objtype      : オブジェクトの種別
    VP_INT brkprm     : コールバック関数へのパラメータ
    DT_VP brkadr      : ブレークを設定するアドレス
    DT_FN svcfn       : 機能コード
}    T_RSBRK;
```

brktype は後述の " 停止条件 " から 1 つ，" 付加条件 " から任意個，" 停止操作 " から 1 つの組み合わせで行われる．括弧内は利用するパラメータを意味する．ただし，***brkcnt*** はどの組み合わせであっても有効である．

停止条件

- ***BRK_EXECUTE* (1)**
  実行ブレークを設定する (***brkadr***, ***tskid***)

- ***BRK_ACCESS* (2)**
  アクセスブレークを設定する (***brkadr***, ***tskid***)
- ***BRK_DISPATCH* (3)**
  タスクディスパッチャ (実行後) にブレークを設定する (***tskid***)
- ***BRK_SVC* (4)**
  サービスコールの発行時にブレークする (***tskid, objid***, ***svcfn***)

付加条件

- ***BRK_ENTER* (00h)**
  開始位置にブレークを置く (***BRK_DISPATCH***, ***BRK_SVC***)
- ***BRK_LEAVE* (80h)**
  脱出位置にブレークを置く (***BRK_DISPATCH***, ***BRK_SVC***)

停止操作

- ***BRK_SYSTEM* (0h)**
  ブレーク時は全体が停止する
- ***BRK_TASK* (40h)**
  ブレーク時はタスク単体が停止する
- ***BRK_REPORT* (20h)**
  通知のみを行う (実際にブレークは行わない)

また，各パラメータには特殊な値が設けられている．詳細は次の通り．

Table 21: **ブレーク設定で使える特殊なパラメータ値**

パラメータ	値	意味
tskid	***ID_ALL (-1)***	全てのタスクをブレーク対象とする
objid	***ID_ALL (-1)***	全オブジェクトをブレーク対象とする
svcfn	***ID_ALL (-1)***	全ての SVC で停止する
brkcnt	***BRK_NOCNT (1)***	カウントを利用しない

ブレークはこれらの組み合わせによって設定する．

例 : タスク 2 に 10 回目に切り替わるときにブレーク

Program source

```
T_RSBRK {
    brktype     : BRK_DISPATCH
    brkcnt:     : 10
    tskid       : 2
}
```

Program source

例 : タスク 5 がセマフォ 2 を獲得しようとしたときにブレーク

Program source
```
T_RSBRK {
    brktype     : BRK_SVC
    brkcnt      : BRK_NOCNT
    tskid       : 5
    objtype     : OBJ_SEMAPHORE
    objid       : 2
    ext.svcfn   : -0x25 (wai_sem)
}
```
Program source

オプションの選択によっては無視されるパラメータの部分に関しては，基本的には考慮しない．ただしベンダが特殊なブレークを設定する機能を提供する場合，この引数部を利用したり，パラメータを追加してもかまわないが，そのことを明示するため次のフラグを ***flags*** に設定すること．

OPT_EXTPARAM (2)
拡張パラメータ指定

タスクディスパッチャに設定した場合, RIM はカーネル内でタスクディスパッチの起こる可能性がある全ての場所にブレークを設定する．

補足説明

この関数はデバッグツールから呼ばれるが，このときデバッグツールは自分がサポートしないブレークを設定してはならない (例 : アクセスブレークをサポートしないデバッグツールがこの関数によってアクセスブレークの設定を要求する) .

自動番号割当てフラグ ***FLG_AUTONUMBERING*** を指定した場合, 関数の実行に成功すると，ブレークポイントに割り当てられた 1 以上の値 (ID 値) を返却する．自動割当フラグを指定しなかった場合であっても同様である．

フラグ

OPT_NOCNDBREAK (1)
ブレークの設定に条件ブレークを用いてはならない

OPT_EXTPARAM (2)
拡張パラメータ指定

FLG_NOREPORT (80000000h) : 通知関数の無効化
対になるコールバック関数の呼出しを行わない

FLG_AUTONUMBERING (40000000h) : ID 自動割当
ID の自動割当を行う．引数に ID を指定した場合，関数はこれを無視する．成功した場合，関数は自動割当された ID を返す．

キー

RIF 04h
.RIF_SET_BRK 09h
.FLG_NOREPORT 03h[1]
フラグ ***FLG_NOREPORT*** は利用可能である
.FLG_AUTONUMBERING 04h[1]
フラグ ***FLG_AUTONUMBERING*** は利用可能である

.OPT_NOCNDBREAK 10h[1]
オプション ***OPT_NOCNDBREAK*** は利用可能である
.OPT_EXTPARAM 11h[1]
オプション ***OPT_EXTPARAM*** を利用できる

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_PAR (-145)
パラメータが不正な値であった
E_ID (-146)
指定されたオブジェクト ID は無効
E_NOID (-162)
自動割当用 ID が不足している
ET_OBJ (-41)
ターゲット上の対象オブジェクトは操作できない
ET_OACV (-27)
不正なターゲット上のオブジェクトに対するアクセス (tskid < 0)
ET_ID (-18)
指定されたカーネルオブジェクト ID は無効
ET_NOEXS (-42)
ターゲット上に対象となるオブジェクトが存在しない

5.5.2 ブレークポイントの解除

rif_del_brk ブレークポイントの解除 [BRK] ○

ER rif_del_brk (ID brkid , FLAG flags)

ID brkid
解除するブレークポイント ID

FLAG flags
フラグ

この関数は，RTOS に依存するようなブレークを解除するよう RIM に要求する．
brkid に ***ID_ALL***(=-1) を指定すると，全てのブレークポイントが解除される．

キー

RIF 04h
.RIF_DEL_BRK 0Ah

フラグ

特になし

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_OBJ (-169)
操作対象は存在しない または 操作できない
E_ID (-146)
指定されたオブジェクト ID は無効

5.5.3 ブレークヒットの通知

rif_rep_brk ブレークヒットの通知 [BRK:callback] □

void rif_rep_brk (ID brkid, VP_INT exinf)

ID brkid
ヒットしたブレークの ID

VP_INT exinf
拡張パラメータ

rif_set_brk でセットしたブレークに到達 , 停止したときに, RIM が停止を通知するために利用するコールバック．通常はデバッグツールから RIM へのコールバック関数 ***tif_rep_brk*** 関数がこの関数を呼び出す.
関数には拡張パラメータを渡すことができる. このパラメータは***rif_set_brk***でブレークポイントを設定するとき，***T_RSBRK::brkprm*** に設定された値が用いられる.

キー

RIF	04h
.RIF_REP_BRK	0Bh

エラー

本関数は返値を持たない

5.5.4 設定したブレーク情報の取得

rif_ref_brk 設定したブレーク情報の取得 [BRK] ○

ER rif_ref_brk (ID brkid, T_RSBRK * ppk_rsbrk, FLAG flags)

ID brkid
ブレークポイント ID

T_RSBRK * ppk_rsbrk
ブレーク情報を格納する領域へのポインタ

FLAG flags
フラグ

指定されたブレークポイント ID に対応するブレークポイント情報を取得する．成功した場合は ***ppk_rsbrk*** が示す領域へ指定したブレークポイント ID に関する情報が格納される．

キー

RIF	04h
.RIF_REF_BRK	0Ch

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_OBJ (-169)
操作対象は存在しない または 操作できない

E_ID (-146)
指定されたオブジェクト ID は無効

E_PAR (-145)
パラメータが不正な値であった

5.5.5 ブレーク条件の取得

rif_ref_cnd ブレーク条件の取得 [CND] ○

ER rif_ref_cnd
(T_RRCND_DBG * ppk_dbg, T_RRCND_RTOS * pk_rtos,
FLAG flags)

T_RRCND_DBG * ppk_dbg
設定した条件を知るための情報を格納するする領域へのポインタ

T_RRCND_RTOS *pk_rtos
取得したい条件を格納する領域へのポインタ

FLAG flags
フラグ

この関数は，デバッグツールが独自の機能のみを用いて RTOS に依存するようなブレークを行いたい時に必要となる RTOS 依存の条件を取得する時に利用する．

T_RRCND_RTOS には RTOS 依存となるような条件が入力される．内容は次の通り．

```
typedef struct    t_rrcnd_rtos
{
    FLAG type          : 調べる内容
    DT_ID objid        : 条件となる ID
}    T_RRCND_RTOS;
```

T_RRCND_RTOS::type に設定できる値は次の通り

- ***CND_CURTSKID* (0)**
  現在実行しているタスク ID が ***T_RRCND_RTOS::objid*** と等しくなる条件

T_RRCND_DBG は設定した条件を調べる方法が返される．内容は次の通り．

```
typedef struct    t_rrcnd_dbg
{
    DT_VP execadr      : 実行番地 (NULL : NC)
    DT_VP valadr       : アドレス (NULL : NC)
    UINT vallen        : 内容の長さ ( バイト長 :1,2,4)
    VP_INT value       : 内容またはポインタ値
}    T_RRCND_DBG;
```

T_RRCND_DBG で生成される条件は，「プログラムカウンタが ***execadr*** に達したとき，メモリ番地 ***valadr*** から ***vallen*** バイトの値が ***value*** であったとき」と表現することができる．***execadr*** に ***NULL*** が格納された場合，この式はプログラムカウンタに依存しない条件式となり，***valadr*** が省略された場合，この式はメモリ内容に依存しない

条件式となる．ただしこの関数が ***execadr*** および ***valadr*** ともに ***NULL*** になるような条件を生成した場合，それは無効と判断する．
T_RRCND_DBG::value には比較対象となる値が格納される．このとき，値が ***VP_INT*** よりも大きい場合，***value*** には比較対象となる値が格納されている領域へのポインタを格納しなければならない．

補足説明

関数は指定された ID の範囲が有効であるかどうかを判断するが，タスクなどが存在するかどうかは確認しない．

キー

RIF	04h
.RIF_REF_CND	0Dh

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_PAR (-145)
パラメータが不正な値であった
E_CND (-228)
その条件は設定できない
ET_ID (-18)
指定されたカーネルオブジェクト ID は無効

5.6 実行履歴 (トレースログ)

5.6.1 トレースログの設定

rif_set_log　トレースログの設定 [LOG] ○

ER_ID　rif_set_log

(ID logid, UINT logtype, VP pk_rslog , FLAG flags)

ID　logid
設定するログに割り当てる ID 番号

UINT　logtype
設定するログの種別

VP　pk_rslog
トレースログの設定情報を格納する領域へのポインタ

FLAG　flags
フラグ

(返値) ID　logid
設定されたログを識別する一意な値

RIM に対してトレースログ取得のための設定情報を渡し，開始を要求する.

logtype には次の値を用いることができる.

- ***LOG_TYP_INTERRUPT* (1)**
  割込
- ***LOG_TYP_ISR* (2)**
  割込サービスルーチン
- ***LOG_TYP_TIMERHDR* (3)**
  タイムイベントハンドラ
- ***LOG_TYP_CPUEXC* (4)**
  CPU 例外
- ***LOG_TYP_TSKEXC* (5)**
  タスク例外
- ***LOG_TYP_TSKSTAT* (6)**
  タスク状態
- ***LOG_TYP_DISPATCH* (7)**
  タスクディスパッチ
- ***LOG_TYP_SVC* (8)**
  サービスコール

- ***LOG_TYP_COMMENT* (9)**
  コメント (文字列のみからなるログ．主としてユーザが記述)

追加フラグとして起動および開始を指定するときに利用する ***LOG_ENTER (0h)*** と，終了を指定するときに利用する ***LOG_LEAVE (80h)*** が存在する．位置指定子が要求されているにもかかわらず，これらが省略された場合，***LOG_ENTER*** が指定されたものとして判断する．
(例 ***LOG_TYP_TSK*** | ***LOG_ENTER*** : タスクの起動に対してログを取得する．)

上記の各種別に対し，***pk_rslog*** には次の構造体が割り当てられる．各パラメータのうち "***ID_ALL*** 可 " とあるものは，***ID_ALL(=-1)*** を指定することで全ての ID が対象となる．必要に応じてそれぞれの型にキャストして代入を行うこと．

LOG_TYP_INTERRUPT (1) : 割込 (起動 , 終了)

```
typedef struct    t_rslog_interrupt
{
    DT_INTNO intno    : 割込番号 (ID_ALL 可 )
}    T_RSLOG_INTERRUPT;
```

LOG_TYP_ISR (2) : 割込サービスルーチン (起動 , 終了)

```
typedef struct    t_rslog_isr
{
    DT_ID isrid          : 割込サービスルーチン ID (ID_ALL 可 )
    DT_INTNO intno    : 割込番号 (ID_ALL 可 )
}    T_RSLOG_ISR;
```

※ ***intno*** が ***ID_ALL*** であった場合，***isrid*** は自動的に ***ID_ALL*** となる

LOG_TYP_TIMERHDR (3) : タイムイベントハンドラ (起動 , 終了)

```
typedef struct    t_rslog_timerhdr
{
    UINT type          : ハンドラの種別 (OBJ_ALL 可 )
          (rif_ref_obj::objtype で利用する定数 OBJ_xxx を格納する )
          (OBJ_ALL(=ID_ALL) を指定すると全ての種別を対象とする )
    DT_ID hdrid        : ハンドラ ID(ID_ALL 可 )
}    T_RSLOG_TIMERHDR;
```

LOG_TYP_CPUEXC (4) : CPU 例外 (起動 , 終了)

```
typedef struct    t_rslog_cpuexc
{
    DT_EXCNO excno : CPU 例外番号 (ID_ALL 可 )
}    T_RSLOG_CPUEXC;
```

LOG_TYP_TSKEXC (5) : タスク例外 (起動 , 終了)

```
typedef struct    t_rslog_tskexc
{
    DT_ID tskid         : タスク番号 (ID_ALL 可 )
}    T_RSLOG_TSKEXC;
```

LOG_TYP_TSKSTAT (6) : タスク状態

```
typedef struct    t_rslog_tskstat
{
    DT_ID tskid         : タスク番号 (ID_ALL 可 )
}    T_RSLOG_TSKSTAT;
```

※タスク状態は実行状態と実行可能状態を区別せず，実行可能状態として扱う

LOG_TYP_DISPATCH (7) : タスクディスパッチ開始

```
typedef struct    t_rslog_dispatch
{
    DT_ID tskid         : タスク番号 (ID_ALL 可 )
}    T_RSLOG_DISPATCH;
```

LOG_TYP_SVC (8) : システムコール (開始 , 終了)

```
typedef struct    t_rslog_svc
{
    DT_FN svcfn         : 機能コード (ID_ALL 可 )
    DT_ID objid         : 対象オブジェクト ID
          ( 対象がない SVC の場合は無視 , ID_ALL 可 )
    DT_ID tskid         : タスク ID (ID_ALL 可 )
    BITMASK param    : 取得対象パラメータ (ID_ALL 可 )
}    T_RSLOG_SVC;
```

※ ***tskid*** に ***ID_NONTSKCTX(=0)*** を指定すると非タスクコンテキストが対象となる．***ID_ALL*** はタスク / 非タスクコンテキストの両方を意味する．***param*** はログとして取得するパラメータを指定し，それぞれ 1 となっているビット位置に対応するパラメータをログとして取得する．***LOG_ENTER*** 指定時には最左引数が第 1 パラメータに相当し，***LOG_LEAVE*** 指定時には返値が第 1 パラメータとなり，第 2 パラメータ以降が引数となる．

LOG_TYP_COMMENT (9) : コメント

```
typedef struct    t_rslog_comment
{
    UINT length         : コメント文字列の長さ
}    T_RSLOG_COMMENT;
```

補足説明

一部のログは，それぞれ出力する順番が決まっている．以下に出力順序が決まっているログ種別を示す．それぞれ左側に置かれたものの方が先に表示されるべきログを示す．

- ***LOG_TYP_DISPATCH | LOG_LEAVE, LOG_TYP_TSKEXC***
- ***LOG_TYP_DISPATCH | LOG_ENTER, LOG_TYP_TSKSTAT***

LOG_TYP_SVC|LOG_LEAVE では，次のサービスコールの終了は検出しない．

- ***ext_tsk***
- ***exd_tsk***

LOG_TYP_TSKEXC|LOG_LEAVE は，次のような状況では検出されない．

- タスク例外ハンドラからの大域脱出[1]

LOG_TYP_TSKSTAT は実行可能状態 (READY) と実行状態 (RUNNING) を区別せず，双方とも実行可能状態として識別する．実行可能状態と実行状態に関しては ***LOG_TYP_DISPATCH*** で取得する．

自動番号割当てフラグ ***FLG_AUTONUMBERING*** を指定した場合，関数の実行に成功すると，ログ項目に割り当てられた 1 以上の値 (ID 値) を返却する．自動割当フラグを指定しなかった場合であっても同様である．

フラグ

FLG_AUTONUMBERING (40000000h) : ID 自動割当

ID の自動割当を行う．引数に ID を指定した場合，関数はこれを無視する．成功した場合，関数は自動割当された ID を返す．

キー

RIF	04h
.RIF_SET_LOG	0Eh
.FLG_AUTONUMBERING	04h[1]
フラグ FLG_AUTONUMBERING は利用可能である	
.OPT_BUFFUL_STOP	10h[1]
オプション OPT_BUFFUL_STOP は利用可能である	
.OPT_BUFFUL_FORCEEXEC	11h[1]
オプション OPT_BUFFUL_FORCEEXEC は利用可能である	

エラー

E_NOSPT (-137)

その操作はサポートしない

---

1. longjmp, setjmp などを用い，関数の実行順序によることなく強制的に特定の関数へ処理を移す処理を指す

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_ID (-146)
指定されたオブジェクト ID は無効

E_NOID (-162)
自動割当用 ID が不足している

E_OBJ (-169)
操作対象は存在しない または 操作できない

ET_ID (-18)
指定されたカーネルオブジェクト ID は無効

E_PAR (-145)
パラメータが不正な値であった

5.6.2 トレースログ設定の解除

rif_del_log　トレースログ設定の解除 [LOG] ○

ER　　rif_del_log (ID logid ,FLAG flags)

ID　　logid
　　解除対象となるトレースログ ID

FLAG　　flags
　　フラグ

rif_set_log で設定したトレースログ設定を破棄する．***logid*** に ***ID_ALL***(=-1) を指定した場合，全てのログ設定を削除する．

補足説明

rif_sta_log で有効になっているトレースログ項目を解除することはできない．

キー

RIF　　04h
.RIF_DEL_LOG　　0Fh

エラー

E_OK (0)
　　正常に終了した
E_NOSPT (-137)
　　その操作はサポートしない
E_NOMEM (-161)
　　ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
　　何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
　　何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_ID (-146)
　　指定されたオブジェクト ID は無効
E_OBJ (-169)
　　操作対象は存在しない または 操作できない
E_EXCLUSIVE (-226)
　　すでに要求が発行されており，終了するまで受け付けることができない

5.6.3 トレースログ機能開始要求

rif_sta_log　トレースログ機能の開始要求 [LOG] ○

ER　　rif_sta_log (ID logid, FLAG flags)

ID　　logid
　　開始したトレースログに割り当てられた ID 番号

FLAG　　flags
　　フラグ

rif_set_log で設定された項目に従い，トレースログ機能を開始する．***ID_ALL***(=-1) が指定された場合は設定された全てのログを有効にする．

補足説明
トレースログの取得はノンブロッキングで行われるため，本関数の終了がトレースログ取得終了を意味しているわけではないことに注意されたい．実際は本関数を呼び出した後，プログラム実行再開などでプログラムが実行されている間にログの取得が行われる．

補足説明
同一 ID のログ設定に対して複数回開始した場合でも，関数は ***E_OK*** を返却する．ただし，複数回開始しても一回の停止操作で指定されたログ設定は停止する．

キー
RIF 04h
.RIF_STA_LOG 10h

エラー
E_OK (0)
　　正常に終了した
E_NOSPT (-137)
　　その操作はサポートしない
E_NOMEM (-161)
　　ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
　　何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
　　何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_ID (-146)
　　指定されたオブジェクト ID は無効
E_OBJ (-169)
　　操作対象は存在しない または 操作できない

5.6.4 トレースログの終了要求

rif_stp_log　トレースログ取得の解除要求 [LOG] ○

ER　　rif_stp_log (ID logid, FLAG flags)

ID　　logid
　　対象となるトレースログ ID

FLAG　　flags
　　フラグ

設定されたトレースログを終了させる．***logid*** に ***ID_ALL***(=-1) を指定した場合，全ログが対象となる．

補足説明
本関数はトレースログ取得のために設定されたブレークポイントなどの解除が目的であり，トレースログ設定の破棄を行うものではない．
この関数の前後で ***rif_set_log*** で設定された内容は保存されていることを保証しなければならない．

補足説明
すでに停止状態にあるログ設定に対して本関数を実行した場合でも, 関数は ***E_OK*** を返却する．

キー
RIF 04h
.RIF_STP_LOG 11h

エラー
E_OK (0)
　　正常に終了した
E_NOSPT (-137)
　　その操作はサポートしない
E_NOMEM (-161)
　　ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
　　何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
　　何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_ID (-146)
　　指定されたオブジェクト ID は無効
E_OBJ (-169)
　　操作対象は存在しない または 操作できない

5.6.5 トレースログの取得

rif_get_log　トレースログの取得 [LOG] ○

ER　　rif_get_log (T_RGLOG * ppk_rglog, FLAG flags)

T_RGLOG *　　ppk_rglog
　　標準トレースログ情報を格納する領域へのポインタ

FLAG　　flags
　　フラグ

rif_get_log は，RIM に蓄積したログの取得を要求する．RIM は必要に応じて ***tif_get_log*** を発行し，プリミティブなログ情報を取得し，加工してこの関数の返値とする．高機能なデバッグツールにおいては，RIM は ***tif_get_log*** で取得したデータを加工することなく返値とできる場合もある．
rif_get_log は 1 項目分のログを取得すると，次のログ項目へ読み出し位置を移動させる．全てのログを取得するために，デバッグツールは複数回この関数を呼び出す必要がある．残り項目数が 0 になると，***rif_get_log*** は ***E_OBJ*** エラーを返す．

T_RGLOG の内容は次の通り

```
typedef struct    t_rglog
{
    UINT logtype      : ログの種別
    LOGTIM logtim     : 発生時刻
    BITMASK valid     : 有効フィールドビットマップ
    UINT bufsz        : バッファ領域 buf の大きさ ( バイト単位 )
    char buf[]        : 情報を格納するバッファ領域 ( 詳細は後述 )
}    T_RGLOG;
```

T_RGLOG は必須項目である ***logtype***, ***logtim***, ***valid*** に加え，後述の「種別とバッファに格納される内容」を格納するのに十分な領域を持たなければならない．

T_RGLOG::type には発生したログの種別が入る．***T_RGLOG::buf*** には指定した種別に対応する情報が格納される．起動および終了が指定できるものに関しては，指定し ***LOG_ENTER*** および ***LOG_LEAVE*** が ***T_RGLOG::type*** にセットされる．以下にログ種別と格納される情報の詳細を示す．なお ***LOG_TYP_DISPATCH*** のみ起動時と終了時のログで取得できる情報が異なるので注意されたい．

LOG_TYP_INTERRUPT (1): 割込ハンドラ

```
typedef struct    t_rglog_interrupt
{
    DT_INHNO inhno  : 割込ハンドラ番号
}    T_RGLOG_INTERRUPT;
```

LOG_TYP_ISR (2) : 割込サービスルーチン

```
typedef struct    t_rglog_isr
{
    DT_ID isrid         : 割込みサービスルーチン ID
    DT_INHNO inhno      : 割込みハンドラ番号
}    T_RGLOG_ISR;
```

LOG_TYP_TIMERHDR (3) : タイムイベントハンドラ

```
typedef struct    t_rglog_timerhdr
{
    UINT type           : タイマーの種別
          (rif_ref_obj::objtype で利用する定数 OBJ_xxx が格納される )
    DT_ID hdrid         : タイムイベントハンドラの ID
    DT_VP_INT exinf     : 拡張情報
}    T_RGLOG_TIMERHDR;
```

LOG_TYP_CPUEXC (4) : CPU 例外

```
typedef struct    t_rglog_cpuexc
{
    DT_ID tskid         : 対象となるタスク ID
}    T_RGLOG_CPUEXC;
```

CPU 例外の発生要因がタスク外にあった場合，tskid は 0 となる.

LOG_TYP_TSKEXC (5) : タスク例外

```
typedef struct    t_rglog_tskexc
{
    DT_ID tskid         : 対象となるタスク ID
}    T_RGLOG_TSKEXC;
```

LOG_TYP_TSKSTAT (6) : タスク状態

```
typedef struct    t_rglog_tskstat
{
    DT_ID tskid         : タスク ID
    DT_STAT tskstat     : 遷移先タスク状態
    DT_STAT tskwait     : 待ち状態
    DT_ID wobjid        : 待ち対象のオブジェクト ID
}    T_RGLOG_TSKSTAT;
```

LOG_TYP_DISPATCH | LOG_ENTER (7) : タスクディスパッチ開始

```
typedef struct    t_rglog_dispatch_enter
{
```

```
    DT_ID tskid         : これまで実行状態にあったタスクの ID
    UINT disptype       : ディスパッチ種別
}   T_RGLOG_DISPATCH_ENTER;
```

ディスパッチ種別は次のとおり.

DSP_NORMAL (0)
タスクコンテキストからのディスパッチ

DSP_NONTSKCTX (1)
割込み処理および CPU 例外からのディスパッチ

LOG_TYP_DISPATCH | LOG_LEAVE (135) : タスクディスパッチ終了

```
typedef struct  t_rglog_dispatch_leave
{
    DT_ID tskid         : これから実行状態になるタスクの ID
}   T_RGLOG_DISPATCH_LEAVE;
```

LOG_TYP_SVC (8) : サービスコール

```
typedef struct  t_rglog_svc
{
    DT_FN fncno         : 機能コード
    UINT prmcnt         : パラメータ数
    DT_VP_INT prmary[]: パラメータ
}   T_RGLOG_SVC;
```

LOG_TYP_COMMENT (9) : コメント (文字列のみからなるログ)

```
typedef struct  t_rglog_comment
{
    UINT length         : 文字列の長さ
    char strtext []     : 文字列 (NULL 終端 ) - 中断あり
}   T_RGLOG_COMMENT;
```

デバッグツールはこの関数を呼び出す前に，***T_RGLOG*** 構造体のメンバ ***bufsz*** に ***T_RGLOG::buf*** が示すバッファ領域のバイト単位のサイズを格納しなければならない.

補足説明

「中断あり」と説明されているログ項目 (***LOG_TYP_COMMENT::strtext***) は, バッファの領域が足りなくとも転送が可能な部位まで転送を行う．ただし，バッファ領域が足りず，転送が途中で中断しなければならない場合でも，その意味を成す最小単位を保

証しなければならない.[1] また中断されたパラメータの有効 / 無効を示すビットマップ (後述) は有効とし，返り値は ***E_NOMEM*** エラーとなる.

T_RGLOG::valid は ***T_RGLOG::buf*** に格納される項目の有効フィールドを示す．各項目の上から順にビットマップの 0,1,2... とマップされていく．例として，***LOG_TYP_SVC|ENTER*** では，***fncno*** が最下位ビット，***prmcnt*** が下位第 2 ビット，***prmary[n]*** が下位第 3+n ビットに割り当てられ，項目が有効である場合は 1，無効である場合は 0 が格納される．ただし ***T_RGLOG_COMMENT::strtext*** はそれぞれの文字単位ではなく，文字列全体を単位として扱われる．項目と対応関係のないビットは常に 0 である.

ログ種別 - サービスコール開始 (***LOG_TYP_SVC|LOG_ENTER***) によって得られた ***T_RGLOG_SVC::prmcnt*** には取得できたパラメータの最大数が格納されている．また ***T_RGLOG_SVC::prmary*** のうち，正常に取得できた部分に関しては最左引数を第 1 番目として ***T_RGLOG::valid*** の対応するビットが 1 となる．パラメータ取得が部分的にしか行われなかった場合などは関数の引数の数と ***T_RGLOG_SVC::prmcnt*** が一致しないので注意されたい.
ログ種別 - サービスコール終了 (***LOG_TYP_SVC|LOG_LEAVE***) によって得られた ***T_RGLOG_SVC::prmcnt*** には，返値を含めて取得できたパラメータの最大数が格納されている．また ***T_RGLOG_SVC::prmary*** のうち，正常に取得できた部分に関しては返値を第 1 番目，関数の最左引数を第 2 番目として ***T_RGLOG::valid*** の対応するビットが 1 となる.

一部のログは，それぞれ出力する順番が決まっている．以下に出力順序が決まっているログ種別を示す．それぞれ左側に置かれたものの方が先に表示されるべきログを示す.

- ***LOG_TYP_DISPATCH | LOG_LEAVE, LOG_TYP_TSKEXC***
- ***LOG_TYP_DISPATCH | LOG_ENTER, LOG_TYP_TSKSTAT***

LOG_TYP_SVC|LOG_LEAVE では，次の関数の終了は検出しない.

- ***ext_tsk***
- ***exd_tsk***

LOG_TYP_TSKEXC|LOG_LEAVE は，次のような状況では検出されない.

- タスク例外ハンドラからの大域脱出

LOG_TYP_TSKSTAT は実行可能状態 (READY) と実行状態 (RUNNING) を区別せず，双方とも実行可能状態として識別する．実行可能状態と実行状態に関しては ***LOG_TYP_DISPATCH*** で取得する.

1. strtext は NULL 終端文字列であるので, NULL 終端文字列として意味を成すために残バッファ量が 1 バイトになった時点で終端記号を付加し，中断しなければならない.

オプション

OPT_PEEK (1)

トレースログをスプールから削除せずに取り出す

キー

RIF 04h

.RIF_GET_LOG 12h

.OPT_PEEK 10h[1]

オプション ***OPT_PEEK*** は利用可能である

.STRUCT_SVC 11h[1]

LOG_TYP_SVC の開始 / 終了に対してそれぞれ専用の構造体を利用する

エラー

E_OK (0)

正常に終了した

E_NOSPT (-137)

その操作はサポートしない

E_NOMEM (-161)

ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)

何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)

何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_OBJ (-169)

操作対象は存在しない または 操作できない

E_PAR (-145)

パラメータが不正な値であった

5.6.6 トレースログ機構の構成変更

rif_cfg_log　トレースログ機構の構成変更 [LOG] ○

ER　　rif_cfg_log (T_RCLOG * pk_rclog, FLAG flags)

T_RCLOG *　　pk_rclog
　　トレースログ構成情報を格納するパケットへのポインタ

FLAG　　flags
　　フラグ

トレースログ機構の構成を変更する.

トレースログ構成情報を格納する構造体 ***T_RCLOG*** の詳細を次に示す.

```
typedef struct    t_rclog
{
    UINT type          : トレースログ構成情報
    DT_VP bufptr       : トレースログバッファへのポインタ
    DT_SIZE bufsz      : トレースログバッファのサイズ
}    T_RCLOG;
```

T_RCLOG::type は，トレースログ機構の設定情報が格納される．設定情報にはバッファ取得方法，およびログバッファ満載時を指定でき，それぞれ次の値を用いることができる (サポートしない場合は ***E_NOSPT*** エラーとなる).

バッファ取得方法

- ***LOG_HARDWARE*** **(0)**
  TIF を利用したハードウェアログ機構を利用して取得する.
- ***LOG_SOFTWARE*** **(1)**
  RIM 単体で実行するソフトウェア主体のログ機構を利用して取得する.

バッファ満載時の動作

- ***LOG_BUFFUL_STOP*****(0)**
  バッファ満載時にトレース取得を停止する
- ***LOG_BUFFUL_FORCEEXEC*** **(4)**
  バッファ満載時に最も古いものを破棄し，取得を継続する

T_RCLOG::bufptr と ***T_RCLOG::bufsz*** は, RIM およびデバッグツールが RTOS 履歴保存領域を作成する時の目安を設定する．ログ取得時には，指定された領域がログバッファとして利用される.

補足説明

これらの設定を行わずにログ機構を使用した場合の動作は実装定義とする.

LOG_HARDWARE が指定され, RIM がキーコード ***DEBUGGER.LOG.NUM*** を調べた結果ハードウェアログ機構をもたないと判断した場合, 関数は ***E_NOSPT*** を返さなければならない.
ログバッファ領域がプログラム領域 (データ領域またはコード領域) と重複していた場合，または存在しないメモリ空間を指定していた場合，RIM は ***ET_MACV*** エラーを返す.

キー

RIF	04h
.RIF_CFG_LOG	13h

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_PAR (-145)
パラメータが不正な値であった

ET_MACV (-26)
ターゲット上の不正なメモリ領域に対するアクセスが発生した

5.7 その他 RTOS 関連情報の取得 / 操作

5.7.1 カーネルコンフィギュレーションの取得

rif_ref_cfg　カーネルコンフィギュレーションの取得　[R] ○

ER　　rif_ref_cfg

(T_INFO * p_information, UINT packets, FLAG flags)

T_INFO *　　p_information
　　情報取得構造体の配列の先頭を指すポインタ

UINT　　packets
　　p_information が示す情報取得構造体配列の長さ

FLAG　　flags
　　フラグ

カーネルコンフィギュレーションを取得する[1].

本関数の情報取得には情報取得用関数 ***T_INFO*** とキーコードを用いる. 詳細に関しては 3.6 [情報取得構造体と取得キー] を参照のこと. ***rif_ref_cfg*** では ***INF_CFG*** キーを頂点とするキーコードを取得することが可能である.

キー

CFG 7h
- **.CPUEXCEPTION** 17h
  - **.MIN** 1h[W]
    カーネルが利用する内部例外要因のうち最小の数
  - **.MAX** 2h[W]
    カーネルが利用する内部例外要因のうち最大の数
  - **.NUM** 3h[W]
    カーネルが利用する内部例外要因の数
- **.SYSTIM** 20h
  - **.TICK_D** 1h[W]
    タイマ分解能を千分の一秒 (ms) で表現したときの分母
  - **.TICK_N** 2h[W]
    タイマ分解能を千分の一秒 (ms) で表現したときの分子
  - **.UNIT_D** 3h[W]
    タイマの単位を千分の一秒 (ms) で表現したときの分母
  - **.UNIT_N** 4h[W]
    タイマの単位を千分の一秒 (ms) で表現したときの分子

1. ITRON デバッギングインタフェース仕様では, カーネルを再構成した場合に変化する情報をカーネルコンフィギュレーションと定義している. 新しい情報項目を追加する際, 後述の ***dbg_ref_rim*** との間でどちらの項目とするか迷った場合は, このことを参考にしてほしい.

.LOGTIM 21h
- **.TICK_D** 1h[W]
  ログ時刻分解能を千分の一秒 (ms) で表現したときの分母
- **.TICK_N** 2h[W]
  ログ時刻分解能を千分の一秒 (ms) で表現したときの分子
- **.UNIT_D** 3h[W]
  ログ時刻の単位を千分の一秒 (ms) で表現したときの分母
- **.UNIT_N** 4h[W]
  ログ時刻の単位を千分の一秒 (ms) で表現したときの分子

.INTERRUPT 22h
- **.MIN** 1h[W]
  カーネルが利用する外部割込み要因のうち最小の数
- **.MAX** 2h[W]
  カーネルが利用する外部割込み要因のうち最大の数
- **.NUM** 3h[W]
  カーネルが利用する外部割込み要因の数

.ISR 25h
- **.MIN** 1h[W]
  カーネルが提供する ISR のうち最小の番号
- **.MAX** 2h[W]
  カーネルが提供する ISR のうち最大の番号
- **.NUM** 3h[W]
  カーネルが提供する ISR の数

.MAKER 23h[W]
メーカーコード

.PRIORITY 24h
- **.MIN** 1h[W]
  カーネルで利用できる優先度のうち最小の数
- **.MAX** 2h[W]
  カーネルで利用できる優先度のうち最大の数

.OBJ_SEMAPHORE 80h
- **.MIN** 1h[W]
  割当て可能な ID のうち最小の数
- **.MAX** 2h[W]
  割当て可能な ID のうち最大の数

.OBJ_EVENTFLAG 81h
- **.MIN** 1h[W]
  割当て可能な ID のうち最小の数
- **.MAX** 2h[W]
  割当て可能な ID のうち最大の数

.OBJ_DATAQUEUE 82h
- **.MIN** 1h[W]
  割当て可能な ID のうち最小の数
- **.MAX** 2h[W]
  割当て可能な ID のうち最大の数

.OBJ_MAILBOX 83h
- **.MIN** 1h[W]
  割当て可能な ID のうち最小の数

.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_MUTEX 84h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_MESSAGEBUFFER 85h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_RENDEZVOUSPORT 86h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_RENDEZVOUS 87h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_FMEMPOOL 88h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_VMEMPOOL 89h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_TASK 8Ah
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_CYCLICHANDLER 8Dh
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_ALARMHANDLER 8Eh
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数

.PRVER A0h[S]
カーネル自身のバージョン番号
.SPVER A1h[S]
ITRON 仕様のバージョン番号

上記キーコードで，***.MAX*** が 0 かつ ***.MIN*** が 0 である場合，その機能はサポートされていないことを意味する．
.MIN はシステムが利用しているオブジェクト ID 等の下限を押さえるためのものである．デバッグツールがこれらシステムオブジェクトの表示を行わない場合，これらの値はユーザが利用できるオブジェクト ID の最小値 1 で置き換え可能である．[1]

補足説明

存在しない情報取得キーコードを指定した場合, また 大きさ 0 のバッファと共に呼び出された場合，関数は ***E_PAR***(パラメータエラー) を返す．

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_PAR (-145)
パラメータが不正な値であった
E_OBJ (-169)
操作対象は存在しない または 操作できない

1. ITRON 仕様の慣習により，システムオブジェクトは負のオブジェクト ID を持つ．一方，ユーザタスクは正のオブジェクト ID しか利用できないので，システムオブジェクトを表示しないのであれば，INF_MIN の値は特に重要ではない．

6. ターゲットアクセスインターフェース

6.1 メモリ関連操作

6.1.1 メモリ確保 (ホスト上)

tif_alc_mbh メモリ確保 (ホスト上) [R] □

ER tif_alc_mbh (VP * p_blk, UINT blksz, FLAG flags)

VP * p_blk
確保したブロックの開始ポインタを格納する領域へのポインタ

UINT blksz
ブロックサイズ

FLAG flags
フラグ

メモリ読み出し等の作業領域を作成するために RIM がメモリを確保する手段を与える．ホストで C ライブラリが利用できるのであればデバッグツールは malloc 関数を呼ぶだけでよい．ただし，RIM はデバッグツールが動くホストで C ライブラリが搭載されていることを仮定してはならないため，RIM 内部で malloc 関数を呼ぶことはあってはならない．

キー

TIF	05h
.TIF_ALC_MBH	01h

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_PAR (-145)
パラメータが不正な値であった

6.1.2 メモリ確保 (ターゲット上)

tif_alc_mbt メモリ確保 (ターゲット上) [E] □

ER tif_alc_mbt (DT_VP * p_blk, DT_SIZE blksz, FLAG flags)

DT_VP * p_blk
: 確保したメモリ領域の先頭を指すポインタを格納する領域

DT_SIZE blksz
: 確保するメモリ領域の大きさ (バイト単位)

FLAG flags
: フラグ

デバッグツールがターゲット上のメモリを管理することが可能である場合[1]，RIM がターゲット上にグルールーチン[2] を書く等の作業を行うためにターゲット上のメモリを確保する際，この関数を実行する．

メモリを動的に割り当てることのできない場合，この機能をサポートする必要はない．その際は RIM が自身の手で領域を確保することが必要となる．

キー

TIF 05h

.TIF_ALC_MBT 02h[1]
: この関数をサポートする

エラー

E_OK (0)
: 正常に終了した

E_NOSPT (-137)
: その操作はサポートしない

E_NOMEM (-161)
: ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
: 何らかの原因により，操作が失敗した (継続して動作は可能である)

E_PAR (-145)
: パラメータが不正な値であった

E_SYS (-133)
: 何らかの原因で復帰不可能な (致命的な) エラーが発生した

ET_NOMEM (-33)
: ターゲット上におけるメモリ不足のため，要求を実行できない

---

1. ICE やターゲットシミュレータなどに搭載されている，物理的にメモリが存在しない空間に，メモリがあるかのようにエミュレートする機能などがある場合を想定している．
2. RIM が SVC 発行などの際に，対象関数を呼び出すための一時的なプログラムを生成する場合がある．このプログラムをグルールーチンと呼んでいる．

6.1.3 メモリ解放 (ホスト上)

tif_fre_mbh メモリ解放 (ホスト上) [R] □

ER tif_fre_mbh (VP blk, FLAG flags)

VP blk
解放するブロックの先頭を指すポインタ

FLAG flags
フラグ

ホスト上に取得したメモリの解放を行う. 大抵のホストでは C ライブラリの free 関数に対応づけられると考えている.

補足説明

blk は, 閉区間 [ブロック開始位置 , ブロック開始位置 + ブロック長 -1] に含まれていれば正常に解放できる.

キー

TIF	05h
.TIF_FRE_MBH	03h

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため, 要求を実行できない

E_FAIL (-227)
何らかの原因により, 操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_PAR (-145)
パラメータが不正な値であった

E_OBJ (-169)
操作対象は存在しない または 操作できない

6.1.4 メモリ解放 (ターゲット上)

tif_fre_mbt メモリ解放 (ターゲット上) [E] □

ER tif_fre_mbt (DT_VP blk, FLAG flags)

DT_VP blk
解放するブロックの先頭を指すポインタ

FLAG flags
フラグ

ターゲット上に確保したメモリを解放する.

補足説明

blk は, 閉区間 [ブロック開始位置 , ブロック開始位置 + ブロック長 -1] に含まれていれば正常に解放できる.

キー

TIF 05h
.TIF_FRE_MBT 04h[1]
この関数をサポートする

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため, 要求を実行できない
E_FAIL (-227)
何らかの原因により, 操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_PAR (-145)
パラメータが不正な値であった
ET_NOMEM (-33)
ターゲット上におけるメモリ不足のため, 要求を実行できない
ET_OBJ (-41)
ターゲット上の対象オブジェクトは操作できない

6.1.5 メモリ読み出し (メモリブロック)

tif_get_mem メモリ読み出し [R] □

ER tif_get_mem
(VP p_result, DT_VP memadr, DT_SIZE memsz, FLAG flags)

VP p_result
格納場所の先頭を指すポインタ

DT_VP memadr
読み出し開始アドレス

DT_SIZE memsz
読み出す長さ (バイト単位)

FLAG flags
フラグ

tif_get_mem は, ***memadr*** から始まる ***memsz*** の長さのターゲットメモリの内容を読み出す．関数を呼び出す前に，RIM は ***memsz*** で指定した以上の長さを持つバッファを作成し，***p_result*** に設定する．デバッグツールは読み出したメモリの内容を，バイト列として ***p_result*** で示された格納領域へ格納する．

Extension

拡張機能として，これらの動作を行うことも出来る．

フラグ

FLG_NOCONSISTENCE (10000000h) : 非一貫性フラグ
このフラグが指定されたときは, 取得した内容に一貫性を持たせる必要はない．例として,「タスク待ち要因がないのにタスク状態は待ちから解除されていない」など.

FLG_NOSYSTEMSTOP (20000000h) : 明示的なシステム停止を許さない
このフラグが指定された場合，関数内で ***tif_brk_tgt*** を利用してシステムを停止するような操作を行ってはならない．このフラグをサポートしない場合は，***E_NOSPT*** エラーとなる．

Extension

補足説明

読み出し時のアクセスサイズはデバッグツール側が決定する．

キー

TIF 05h
.TIF_GET_MEM 05h
.FLG_NOCONSISTENCE 01h[1]
フラグ ***FLG_NOCONSISTENCE*** をサポートする

.FLG_NOSYSTEMSTOP 02h[1]
フラグ ***FLG_NOSYSTEMSTOP*** をサポートする

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

ET_MACV (-26)
ターゲット上の不正なメモリ領域に対するアクセスが発生した

E_PAR (-145)
パラメータが不正な値であった

E_CONSIST (-225)
一貫性を保証できなかった (ただし ***FLG_NOCONSISTENCE*** が設定されていた場合に限ってはエラーとならない)

6.1.6 メモリ読み出し (ブロックセット)

tif_get_bls　ブロックセット単位でのメモリ読み出し [R] □

ER tif_get_bls
(VP p_result , T_BLKSET * blkset , FLAG flags)

VP p_result
読み出した結果を格納する領域へのポインタ

T_BLKSET * blkset
読み出し場所を指示する構造体

FLAG flags
諸フラグ

ターゲットメモリの内容を，ブロックセットを単位として読み出す．ブロックセットは，メモリアドレスとバイト長からなり，複数のターゲットメモリ空間上の位置を保持する．***tif_get_bls*** では，ブロックセットによって示されたターゲットメモリ空間を一括して読み出すことができる．

構造体 ***T_BLKSET*** は読み出し単位となるメモリブロックを格納する集合である．

```
typedef struct    t_blkset
{
    UINT blkcnt         : ブロック数
    T_MEMBLK blkary []
                        : ブロックの配列
}    T_BLKSET;

typedef struct    t_memblk
{
    DT_VP blkptr        : メモリブロックデータを格納するポインタ
    DT_SIZE blksz       : メモリブロックデータのバイト数
}    T_MEMBLK;
```

ターゲットメモリを読み出した内容は，ブロックセットで指定した順で，連続して ***p_result*** で示されるメモリ空間に格納される．つまり，次のようなブロックセットを用いてメモリ読出しを行った場合，データは Table 22: [ブロックセットとデータ配置の関係] のように格納される．

T_BLKSET pk_blkset = { 3, { { 0x1000, 128} , { 0x2000, 1} , { 0x3000, 64} } }

Table 22: **ブロックセットとデータ配置の関係**

開始オフセット	0	128	129
データ長	128 バイト	1 バイト	64 バイト
データ番地	0x1000 ～ 0x1080	0x2000	0x3000 ～ 0x3040

補足説明

本関数が ***E_OK*** を返すとき，要求されたブロックセットは要求された条件のもとに，すべて正常に読み取れたことを保証する．複数ブロックの要求に対してひとつでも正常に読み出しのできないブロックがあった場合，***E_MACV*** エラーとなる．また，後述の ***FLG_NOCONSISTENC*** が設定されていない場合, ***tif_get_mem*** を連続して実行したときとは異なり, ***tif_get_bls*** ではメモリブロック毎ではなく全ての領域に一貫性が保証できなければ ***E_CONSIST*** エラーとなる．

読み込み時のアクセスサイズはデバッグツール側が決定する．

関数を呼び出す前に, RIM は結果を格納するのに十分な大きさを持つバッファを作成し，***p_result*** に格納しなければならない．

Extension

拡張機能として，これらの動作を行うことも出来る．

フラグ

FLG_NOCONSISTENCE (10000000h) : 非一貫性フラグ

このフラグが指定されたときは, 取得した内容に一貫性を持たせる必要はない．例として,「タスク待ち要因がないのにタスク状態は待ちから解除されていない」など．

FLG_NOSYSTEMSTOP (20000000h) : 明示的なシステム停止を許さない

このフラグが指定された場合，関数内で ***tif_brk_tgt*** を利用してシステムを停止するような操作を行ってはならない．このフラグをサポートしない場合は，***E_NOSPT*** エラーとなる．

Extension

キー

TIF 05h
- **.TIF_GET_BLS** 06h
  - **.FLG_NOCONSISTENCE** 01h[1]
    フラグ ***FLG_NOCONSISTENCE*** をサポートする
  - **.FLG_NOSYSTEMSTOP** 02h[1]
    フラグ ***FLG_NOSYSTEMSTOP*** をサポートする

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

ET_MACV (-26)
ターゲット上の不正なメモリ領域に対するアクセスが発生した

E_PAR (-145)
パラメータが不正な値であった

E_CONSIST (-225)
一貫性を保証できなかった (ただし ***FLG_NOCONSISTENCE*** が設定されていた場合に限ってはエラーとならない)

6.1.7 メモリ書き込み (メモリブロック)

tif_set_mem メモリ書き込み [R] □

ER tif_set_mem
(VP storage , DT_VP memadr, DT_SIZE memsz , FLAG flags)

VP storage
書き出す内容を保持する領域の先頭を指すポインタ

DT_VP memadr
書き出し先となるターゲット上のアドレス

DT_SIZE memsz
書き出す長さ (バイト単位)

FLAG flags
フラグ

この関数はメモリブロックを単位として，***storage*** に格納された内容を元に，ターゲット上のメモリへの書き出しを行う．詳細は 6.1.5 [メモリ読み出し (メモリブロック)] を参照のこと．

Extension

拡張機能として，これらの動作を行うことも出来る．

フラグ

FLG_NOCONSISTENCE (10000000h) : 非一貫性フラグ
このフラグが指定されたときは, 取得した内容に一貫性を持たせる必要はない．例として,「タスク待ち要因がないのにタスク状態は待ちから解除されていない」など．

FLG_NOSYSTEMSTOP (20000000h) : 明示的なシステム停止を許さない
このフラグが指定された場合，関数内で ***tif_brk_tgt*** を利用してシステムを停止するような操作を行ってはならない．このフラグをサポートしない場合は，***E_NOSPT*** エラーとなる．

Extension

補足説明

書き込み時のアクセスサイズはデバッグツール側が決定する．

キー

TIF 05h
.TIF_SET_MEM 07h
.FLG_NOCONSISTENCE 01h[1]
フラグ ***FLG_NOCONSISTENCE*** をサポートする
.FLG_NOSYSTEMSTOP 02h[1]
フラグ ***FLG_NOSYSTEMSTOP*** をサポートする

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

ET_MACV (-26)
ターゲット上の不正なメモリ領域に対するアクセスが発生した

E_PAR (-145)
パラメータが不正な値であった

E_CONSIST (-225)
一貫性を保証できなかった (ただし ***FLG_NOCONSISTENCE*** が設定されていた場合に限ってはエラーとならない)

6.1.8 メモリ書き込み (ブロックセット)

tif_set_bls　ブロックセット単位でのメモリ書き込み [R] □

ER tif_set_bls (VP storage, T_BLKSET * blkset , FLAG flags)

VP storage
書き出す内容を格納している領域を指すポインタ

T_BLKSET * blkset
書き出し先を指示する構造体へのポインタ

FLAG flags
フラグ

この関数はブロックセットを単位として，ターゲット上のメモリへの書き出しを行う．詳細は ***tif_get_bls*** を参照のこと．

この関数は ***tif_get_bls*** と対称関係にある関数である．次のような操作を行った場合，そのメモリの内容は変化しないことを保証しなければならない (リアルタイム性のある空間，動的に内容が変化する空間は除く).

Program source

```
{
        // 読み出した内容をそのまま書き出す
if( get_bls(buffer,blkset,0) == E_OK)
        set_bls(buffer,blkset,0);
}
```

Program source

補足事項

設定されたブロックセットのうちのひとつでも失敗した場合，関数は ***E_MACV*** で終了する．このとき ***tif_set_bls*** は，与えられた ***blkset*** のうちのどこまでが書き込まれているかについて，これを報告も保証もしない．

書き込み時のアクセスサイズはデバッグツール側が決定する．

Extension

拡張機能として，これらの動作を行うことも出来る．

フラグ

FLG_NOCONSISTENCE (10000000h) : 非一貫性フラグ

このフラグが指定されたときは, 取得した内容に一貫性を持たせる必要はない. 例として,「タスク待ち要因がないのにタスク状態は待ちから解除されていない」など.

FLG_NOSYSTEMSTOP (20000000h) : 明示的なシステム停止を許さない

このフラグが指定された場合, 関数内で ***tif_brk_tgt*** を利用してシステムを停止するような操作を行ってはならない. このフラグをサポートしない場合は, ***E_NOSPT*** エラーとなる.

Extension

キー

TIF	05h
.TIF_SET_BLS	08h
.FLG_NOCONSISTENCE	01h[1]
フラグ ***FLG_NOCONSISTENCE*** をサポートする	
.FLG_NOSYSTEMSTOP	02h[1]
フラグ ***FLG_NOSYSTEMSTOP*** をサポートする	

エラー

E_OK (0)

正常に終了した

E_NOSPT (-137)

その操作はサポートしない

E_NOMEM (-161)

ホスト上のメモリ不足のため, 要求を実行できない

E_FAIL (-227)

何らかの原因により, 操作が失敗した (継続して動作は可能である)

E_SYS (-133)

何らかの原因で復帰不可能な (致命的な) エラーが発生した

ET_MACV (-26)

ターゲット上の不正なメモリ領域に対するアクセスが発生した

E_PAR (-145)

パラメータが不正な値であった

E_CONSIST (-225)

一貫性を保証できなかった (ただし ***FLG_NOCONSISTENCE*** が設定されていた場合に限ってはエラーとならない)

6.1.9 変更通知の設定

tif_set_pol　メモリ内容変更通知の設定 [E] □

ER_ID　tif_set_pol

(ID polid, DT_VP adr, DT_INT value, UINT length, FLAG flags)

ID　polid
　ポーリング ID

DT_VP　adr
　変更を検出するメモリアドレス

DT_INT　value
　比較対象となる値

UINT　length
　対象となるメモリブロックのバイト長 (1,2,4,8)

FLAG　flags
　フラグ

(返値) ID　polid
　このポーリング設定を識別する一意な値

デバッグツールによるポーリングの設定をする．デバッグツールは，あるメモリ番地の内容をポーリングにより監視し，変更があった場合，コールバック関数によって変更を通知する．ただし ***tif_set_pol*** では，高速に内容が変化するような場合，追従できない可能性がある．
OPT_CMPVALUE を指定した場合，デバッグツールは ***value*** と現在のメモリ内容を比較し，これらが異なっている場合に ***tif_rep_pol*** を呼び出す．***OPT_CMPVALUE*** が指定されない場合は ***tif_set_pol*** を設定したときのメモリ内容を保存し，比較を行って，内容が異なっている場合に通知を行う．

補足説明

tif_set_pol はアクセスブレークとは異なり，内容が変化しなければ通知を行わない．

メモリ内容の更新と関数 ***tif_rep_pol*** の呼び出しタイミングは同時ではない．

自動番号割当てフラグ ***FLG_AUTONUMBERING*** を指定した場合，関数の実行に成功すると，設定項目に割り当てられた 1 以上の値 (ID 値) を返却する．自動割当フラグを指定しなかった場合であっても同様である．

フラグ

OPT_CMPVALUE (2)
比較対象となる値を設定する

FLG_AUTONUMBERING (40000000h) : ID 自動割当
ID の自動割当を行う．引数に ID を指定した場合，関数はこれを無視する．成功した場合，関数は自動割当された ID を返す．

キー

TIF 05h
.TIF_SET_POL 09h[1]
この関数をサポートする
.FLG_AUTONUMBERING 04h[1]
フラグ ***FLG_AUTONUMBERING*** をサポートする
.OPT_CMPVALUE 10h[1]
オプション ***OPT_CMPVALUE*** をサポートする

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

ET_MACV (-26)
ターゲット上の不正なメモリ領域に対するアクセスが発生した

E_PAR (-145)
パラメータが不正な値であった

E_ID (-146)
指定されたオブジェクト ID は無効

E_NOID (-162)
自動割当用 ID が不足している

E_OBJ (-169)
操作対象は存在しない または 操作できない

6.1.10 変更通知の設定解除

tif_del_pol 変更通知の設定解除 [E] □

ER tif_del_pol (ID polid, FLAG flags)

ID polid
解除対象となる ID

FLAG flags
フラグ

tif_set_pol で設定した通知変更 (ポーリング) を解除する．***ID_ALL***(=-1) を指定した場合，全ての変更通知が解除される．

補足説明

本関数は通知関数 ***tif_rep_pol*** 内から呼び出すことも可能である．

キー

TIF 05h
.TIF_DEL_POL 0Ah[1]
この関数をサポートする

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_ID (-146)
指定されたオブジェクト ID は無効
E_OBJ (-169)
操作対象は存在しない または 操作できない

6.1.11 変更通知

tif_rep_pol　メモリ内容変更の通知 [E:callback] ○

void tif_rep_pol (ID polid, DT_INT value, FLAG flags)

ID polid
ポーリング ID

DT_INT value
変更後のメモリ内容の値

FLAG flags
フラグ

tif_set_pol によって設定されたポーリングによって, デバッグツールが変更を検出したとき，この関数によって変更が通知される.

キー

TIF	05h
.TIF_REP_POL	0Bh

エラー

この関数は返値を持たない.

6.2 レジスタ関連操作

6.2.1 レジスタ内容の読み出し

tif_get_reg　レジスタ内容の読み出し [R] □

ER tif_get_reg (VP r_result, BITMASK_8 * p_valid, FLAG flags)

VP r_result
レジスタ値を格納する領域の先頭を指すポインタ

BITMASK_8 * p_valid
レジスタテーブルの各要素の有効 / 無効情報へのポインタ
(NULL : 全てのコンテキストを対象)

FLAG flags
フラグ

この関数は，レジスタセットディスクリプションテーブルの内容に従って，現在のターゲットのレジスタ内容を取得する．

変数 ***p_result*** はこの関数の実行によって取得されるレジスタ内容を格納するバッファへのポインタである．デバッグツールはこの関数を実行する前に，レジスタ内容の格納に十分なサイズを持った領域を作成しなければならない．このバッファのサイズは情報取得キーコード ***RIF.RIF_GET_RDT.REGISTER.SIZE*** を利用する．または，関数 ***rif_get_rdt*** によって取得したレジスタテーブルからも算出可能である．その際には，レジスタテーブルが示す全レジスタを格納するのに十分な大きさの領域を用意しなければならない．

p_valid は各レジスタの有効 / 無効を指定する．関数の引数として与えた際，無効となったレジスタの取得は行われない．またこの関数は，対象となったレジスタ取得の結果を ***p_valid*** に格納する．基本的に，対象レジスタの全てが正常に取得できなかった場合，関数は状況に応じて ***ET_SYS*** などのエラーを返す．取得できなかったレジスタに対応する領域に格納される情報は，実装依存である．また，レジスタの数 (***T_GRDT::regcnt***) を超えて有効 / 無効情報が与えられても，その数を超えたレジスタの取得は行われない．
p_valid に ***NULL*** が指定された場合，全てのレジスタが取得対象となり，詳細な結果の格納は行われない．

Extension

拡張機能として，これらの動作を行うことも出来る．

フラグ

FLG_NOCONSISTENCE (10000000h) : 非一貫性フラグ

このフラグが指定されたときは, 取得した内容に一貫性を持たせる必要はない. 例として,「タスク待ち要因がないのにタスク状態は待ちから解除されていない」など.

FLG_NOSYSTEMSTOP (20000000h) : 明示的なシステム停止を許さない

このフラグが指定された場合, 関数内で ***tif_brk_tgt*** を利用してシステムを停止するような操作を行ってはならない. このフラグをサポートしない場合は, ***E_NOSPT*** エラーとなる.

Extension

補足説明

読み出されたレジスタの値はターゲットのエンディアンに従って格納される.

存在しないレジスタの読出しが指定された場合, 関数は ***E_PAR*** エラーを返す.

キー

TIF 05h
- **.TIF_GET_REG** 0Ch
  - **.FLG_NOCONSISTENCE** 01h[1]
    フラグ ***FLG_NOCONSISTENC*** をサポートする
  - **.FLG_NOSYSTEMSTOP** 02h[1]
    フラグ ***FLG_NOSYSTEMSTOP*** をサポートする

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため, 要求を実行できない

E_FAIL (-227)
何らかの原因により, 操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_CONSIST (-225)
一貫性を保証できなかった (ただし ***FLG_NOCONSISTENCE*** が設定されていた場合に限ってはエラーとならない)

E_PAR (-145)
パラメータが不正な値であった

ET_MACV (-26)
ターゲット上の不正なメモリ領域に対するアクセスが発生した

6.2.2 レジスタ書き込み

tif_set_reg　レジスタ内容の書き出し [R] □

ER　　tif_set_reg (VP storage , BITMASK_8 * p_valid, FLAG flags)

VP　　storage
　　書き込む内容を保持しているポインタ

BITMASK_8 *　　p_valid
　　レジスタテーブルの各要素の有効 / 無効情報へのポインタ
　　(***NULL*** : 全てのコンテキストを対象)

FLAG　　flags
　　フラグ

この関数はターゲット上のレジスタの値を変更する.

補足説明

レジスタに書き込む値は, ターゲットのエンディアンに従って格納されていなければならない.

存在しないレジスタへの書き込みが指定された場合, 関数は ***E_PAR*** エラーを返す.

キー

TIF	05h
.TIF_SET_REG	0Dh

エラー

E_OK (0)
　　正常に終了した
E_NOSPT (-137)
　　その操作はサポートしない
E_NOMEM (-161)
　　ホスト上のメモリ不足のため, 要求を実行できない
E_FAIL (-227)
　　何らかの原因により, 操作が失敗した (継続して動作は可能である)
E_SYS (-133)
　　何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_CONSIST (-225)
　　一貫性を保証できなかった (ただし ***FLG_NOCONSISTENCE*** が設定されていた場合に限ってはエラーとならない)
E_PAR (-145)
　　パラメータが不正な値であった
ET_MACV (-26)
　　ターゲット上の不正なメモリ領域に対するアクセスが発生した

6.3 ターゲット動作関連操作

6.3.1 ターゲットの実行

tif_sta_tgt　ターゲットの実行 [R] □

ER　　tif_sta_tgt (DT_VP staadr, FLAG flags)

DT_VP　　staadr
　　開始アドレス

FLAG　　flags
　　フラグ

この関数は指定されたアドレスからターゲットを実行させる．このとき，レジスタ内容など現在の状態を保持した状態で，指定されたアドレスからターゲットを実行させる．

Extension

フラグ

OPT_RESTART (1)
　　ターゲットを再起動する (引数 ***staadr*** は無視される)

Extension

補足説明

ターゲットが停止または中断状態でなければ本関数は実行できない．このような状態で関数が実行できなかった場合，関数は ***E_EXCLUSIVE*** エラーを返す．

キー

TIF	05h
.TIF_STA_TGT	0Eh
.OPT_RESTART	10h[B]
OPT_RESTART は利用可能である	

エラー

E_OK (0)
　　正常に終了した
E_NOSPT (-137)
　　その操作はサポートしない
E_NOMEM (-161)
　　ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
　　何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
　　何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_PAR (-145)
パラメータが不正な値であった

E_EXCLUSIVE (-226)
すでに要求が発行されており, 終了するまで受け付けることができない

6.3.2 ターゲットの実行停止

tif_stp_tgt　ターゲットの実行中止 [E] □

ER　　tif_stp_tgt (FLAG flags)

FLAG　　flags
フラグ

発行時刻をもって，ターゲットを停止させる.

補足説明

ターゲットが停止状態または中断状態である場合でも，本関数の実行でターゲットは停止状態へ移行する.
停止状態からの再開に関しては実装定義である.

キー

TIF 05h
.TIF_STP_TGT 0Fh[1]
この関数をサポートする

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

6.3.3 ターゲットの実行中断

tif_brk_tgt　ターゲットの実行中断 [E] □

ER　　tif_brk_tgt (FLAG flags)

FLAG　　flags
フラグ

ターゲットを再開可能な状態で停止させる.

キー

TIF 05h
.TIF_BRK_TGT 10h[1]
この関数をサポートする

補足説明

ターゲットが停止状態で本関数が実行された場合, ***E_EXCLUSIVE*** エラーとなる (中断状態では ***E_OK*** となる).

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため, 要求を実行できない
E_FAIL (-227)
何らかの原因により, 操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_EXCLUSIVE (-226)
すでに要求が発行されており, 終了するまで受け付けることができない

6.3.4 ターゲットの実行再開

tif_cnt_tgt　ターゲットの実行再開 [R] □

ER　　tif_cnt_tgt (FLAG flags)

FLAG　　flags
フラグ

実行中断状態にあるターゲットの実行を再開する.

補足説明

ターゲットが中断状態以外で本関数が実行された場合，***E_EXCLUSIVE*** エラーとなる.

キー

TIF 05h
.TIF_CNT_TGT 11h

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_EXCLUSIVE (-226)
すでに要求が発行されており, 終了するまで受け付けることができない

6.4 ハードウェアブレーク関連操作

6.4.1 ブレークポイントの設定

tif_set_brk　ブレークポイントの設定 [R] □

ER_ID　tif_set_brk (ID brkid, T_TSBRK * pk_tsbrk, FLAG flags)

ID　brkid
　ブレークポイント ID

T_TSBRK *　pk_tsbrk
　ブレークポイントの情報を持つ構造体へのポインタ

FLAG　flags
　フラグ

(返値) ID　brkid
　割り当てられたブレークポイント ID

ターゲット上にブレークポイントを設定すると同時に，このブレークに関するコールバックルーチンの設定を行う．

T_TSBRK の内容は次の通り

```
typedef struct    t_tsbrk
{
    UINT brktype      : ブレークの種類
    DT_VP brkadr      : ブレークを設定するアドレス
    VP_INT brkprm     : コールバックルーチンの通知フラグ
}    T_TSBRK;
```

パラメータ ***brktype*** が取りうる値と意味は次の通り

- ***BRK_EXECUTE*** **(1)**
  実行ブレーク

補足説明

自動番号割当てフラグ ***FLG_AUTONUMBERING*** を指定し，関数の実行に成功した場合，関数は設定項目に割り当てられた 1 以上の値 (ID 値) を返却する．自動割当フラグを指定しなかった場合であっても同様である．

フラグ

FLG_NOREPORT (80000000h) : 通知関数の無効化
　対になるコールバック関数の呼出しを行わない

Extension

拡張機能として，これらの動作を行うことも出来る．

パラメータ ***brktype*** にはさらに次の値を設定することが出来る．

- ***BRK_ACCESS* (2)**
  アクセスブレーク

アクセスブレークを指定した場合，次のアクセス指定子のうち，少なくともひとつを設定しなければならない．ただし，複数設定することも可能である．

- ***ACS_READ*(0x100)**
  対象アドレスに対する読出し時にブレーク
- ***ACS_WRITE*(0x200)**
  対象アドレスに対する書込み時にブレーク
- ***ACS_MODIFY*(0x400)**
  対象アドレスに対する修正時にブレーク

デバッグツールが ITRON デバッギングインタフェース仕様推奨条件ブレーク機能をサポートする場合, ***flags*** パラメータに ***OPT_CNDBRK*** を設定することで, ***T_TSBRK*** に代わりに次の ***T_TSBRK_CND*** を利用することができる．***T_TSBRK_CND*** を利用する場合, RIM は ***T_TSBRK_CND*** 型の変数を ***T_TSBRK*** 型でキャストし, ***tif_set_brk*** に渡す必要がある．

```
typedef struct    t_tsbrk_cnd
{
    UINT brktype      : ブレークの種類
    DT_VP brkadr      : プレークを設定するアドレス
    VP_INT brkprm     : コールバックルーチンの通知フラグ
    DT_VP cndadr      : 条件ブレークに設定するアドレス
    VP_INT cndval     : 条件ブレークに設定する値
    UINT cndlen       : 条件ブレークに設定する値のバイト長 (1,2,4)
}    T_TSBRK_CND;
```

この構造体と ***OPT_CNDBRK*** を用いた場合，通常のブレーク条件に加えて，条件式 (*cndadr == cndval) が付け加えられる. この 2 条件を満たしたときのみ仮ブレークヒットと判定され，必要に応じて ***tif_rep_brk*** が呼び出される．

フラグ

OPT_CNDBREAK (4)
デバッグツールの条件ブレーク機構を利用する

FLG_AUTONUMBERING (40000000h) : ID 自動割当
ID の自動割当を行う．引数に ID を指定した場合，関数はこれを無視する. 成功した場合，関数は自動割当された ID を返す.

Extension

キー

TIF 05h
.TIF_SET_BRK 13h
.FLG_AUTONUMBERING 04h[1]
フラグ ***FLG_AUTONUMBERING*** をサポートする
.OPT_CNDBREAK 10h[1]
オプション ***OPT_CNDBREAK*** をサポートする
.BRK_ACCESS 11h[1]
アクセスブレークが利用可能である

エラー

E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_NOID (-162)
自動割当用 ID が不足している
E_OBJ (-169)
操作対象は存在しない または 操作できない
ET_ID (-18)
指定されたカーネルオブジェクト ID は無効
E_PAR (-145)
パラメータが不正な値であった
ET_MACV (-26)
ターゲット上の不正なメモリ領域に対するアクセスが発生した

6.4.2 ブレークポイントの解除

tif_del_brk ブレークポイントの解除 [R] □

ER tif_del_brk (ID brkid, FLAG flags)

ID brkid
ブレークポイント ID

FLAG flags
フラグ

指定した ID に対応するブレークポイントを解除する.

解除時の ID 指定には，次の特殊パラメータが設定できる.

- ***ID_ALL* (-1)**
  全てのブレークポイントを消去する

キー

TIF	05h
.TIF_DEL_BRK	14h

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_ID (-146)
指定されたオブジェクト ID は無効

E_OBJ (-169)
操作対象は存在しない または 操作できない

6.4.3 ブレーク通知

tif_rep_brk ブレーク通知 [R:callback] ○

ER tif_rep_brk (ID brkid, VP_INT param)

ID brkid
ブレークポイント ID

VP_INT param
通知パラメータ (***tif_set_brk*** を参照)

tif_rep_brk で設定したブレークによってターゲットが停止したことを報告する. RIM はこのコールバック関数内で，このブレークの条件が満たされているかを判定し，実際にシステムをブレークさせるかを判定する．この関数が条件一致と判断したとき，所定の操作を行い関数から脱出した後，デバッグツールはターゲットを停止させる処理を続行する．そうでない場合はターゲットの停止を取り消し，実行を再開する．

ブレークに関する一連の流れは次の通り

1. ***rif_set_brk*** により，RIM にブレーク設定要求が渡る.
2. RIM は要求を満たすのに必要な箇所に ***tif_set_brk*** でブレークポイントを設定する.
3. ブレークポイントにヒットしたとき，デバッグツールはそれが ***tif_set_brk*** で設定された物かどうかを調べる.
4. そうであれば，デバッグツールはブレーク ID と通知フラグを引数として ***tif_rep_brk*** を実行する.
5. コールバック関数は通知パラメータ，ブレーク ID と ***tif_set_brk*** の引数をもとに，現在停止した状況が要求されたブレーク設定条件を満たすかどうかをチェックする (満たす場合は 6 へ，満たさない場合は 6' へ移行する).
6. 要求を満たす場合，***tif_rep_brk*** は ***rif_rep_brk*** を呼び出す.
7. ***rif_rep_brk*** が必要な処理を行い，その後 ***tif_rep_brk*** は ***E_TRUE*** を返す.
8. デバッグツールはブレークしたとユーザに報告する
(※ 3 以降，ターゲットはブレークよって停止した状態となっている).

6’. 要求を満たさなかった場合，は ***E_FALSE*** を返す.
7’. デバッグツールはターゲットの動作を再開させる.

補足説明

デバッグツールはこの関数からの返値が ***E_TRUE*** であった場合，ブレーク動作を継続する．一方，***E_FALSE*** を受け取ったときはブレーク動作を中断し，ターゲットの実行を停止させる．ただし，対象となるブレークポイントの停止時動作として ***BRK_REPORT*** が指定されていた場合，本関数が ***E_TRUE*** を返却してもブレーク動作は継続されない.
この関数がこれら判定動作を行っている間，ターゲットは実行中断状態にある. ただし，対象となるブレークポイントの停止時動作として ***BRK_REPORT*** が指定されていた場合は除く.

キー

TIF 05h

.TIF_REP_BRK 12h

この関数をサポートする

.FLG_AUTONUMBERING 04h[1]

フラグ ***FLG_AUTONUMBERING*** をサポートする

エラー

E_TRUE (0)

判定ルーチン用リターンパラメータ **(TRUE)**

ブレークヒットと判断し，ブレーク処理を続行させる

E_FALSE (-229)

判定ルーチン用リターンパラメータ **(FALSE)**

条件は偽であるとし，ターゲット実行を続行させる

E_NOSPT (-137)

その操作はサポートしない

E_NOMEM (-161)

ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)

何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)

何らかの原因で復帰不可能な (致命的な) エラーが発生した

6.5 シンボルテーブル関連操作

6.5.1 シンボルテーブルの値参照

tif_ref_sym シンボルテーブルの値参照 [R] □

ER tif_ref_sym (INT * p_value , char * strsym , FLAG flags)

INT * p_value
シンボルが示す値を格納する領域へのポインタ

char * strsym
シンボル名 (NULL terminated string)

FLAG flags
フラグ

strsym で指定したシンボルテーブルの値を取得する.

補足説明

tif_ref_sym で取得できるのは, シンボルの値 (アドレス) のみであり, 式の評価は原則的に行えないものとする. 具体的には, 算術および論理演算, 配列 (***dummy[n]***), 間接演算子 (****dummy***), アドレス演算子 (***&dummy***), メンバ選択式 (***a.b***, ***c->d***) などは利用できない.

キー

TIF 05h
.TIF_REF_SYM 15h

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため, 要求を実行できない
E_FAIL (-227)
何らかの原因により, 操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_OBJ (-169)
操作対象は存在しない または 操作できない
E_PAR (-145)
パラメータが不正な値であった

6.5.2 シンボルテーブルのシンボル参照

tif_rrf_sym シンボルテーブルのシンボル参照 [E] □

ER tif_rrf_sym
(char * p_sym , UINT maxlen , INT value , FLAG flags)

char * p_sym
対応するシンボルが格納される

UINT maxlen
シンボルを格納する領域の最大量 (終端コード含む)

INT value
逆引きのキーとなる値

FLAG flags
フラグ

値からそれに最も近いシンボルを検索する.

シンボル検索時には次のようなフラグが排他的に利用できる.

OPT_SEARCH_COMPLETELY (0)
完全一致する物だけを対象とする **(default)**

OPT_SEARCH_FORWARD (1)
前方 (アドレスが増加する側) で，指定された値に最も近いシンボルを探す

OPT_SEARCH_BACKWARD (2)
後方 (アドレスが減少する側) で，指定された値に最も近いシンボルを探す

補足説明

OPT_SEARCH_FORWARD や ***OPT_SEARCH_BACKWARD*** を指定した場合，アドレス空間の始端または終端に達した時点で検索を終了するものとする.
OPT_SEARCH_FORWARD や ***OPT_SEARCH_BACKWARD*** は，RIM が現在実行しているサービスコール名称を取得する場合などで利用することを想定している.
検索した値に複数のシンボルが存在した場合の動作は実装依存であるが，上記の理由より，コード領域では関数名等を優先し，データ領域ではグローバル変数名などを優先するのが望ましい.

maxlen はシンボル名を格納するバッファのサイズを示している．***maxlen*** は終端文字を含んだ長さであるため，***maxlen*** が 1 であった場合，文字列は空となり ***E_OK*** が返る．***maxlen*** が 0 であった場合，***E_PAR*** エラーとなる.

キー

TIF 05h

.TIF_RRF_SYM 16h[1]

この関数をサポートする

.OPT_SEARCH_FORWARD 10h[1]

オプション ***OPT_SEARCH_FORWARD*** は利用可能である

.OPT_SEARCH_BACKWARD 11h[1]

オプション ***OPT_SEARCH_BACKWARD*** は利用可能である

.OPT_SEARCH_COMPLETELY 12h[1]

オプション ***OPT_SEARCH_COMPLETELY*** は利用可能である

エラー

E_OK (0)

正常に終了した

E_NOSPT (-137)

その操作はサポートしない

E_NOMEM (-161)

ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)

何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)

何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_OBJ (-169)

操作対象は存在しない または 操作できない

E_PAR (-145)

パラメータが不正な値であった

6.6 関数の実行

6.6.1 関数の実行

tif_cal_fnc 関数の発行 [E] □

ER tif_cal_fnc (T_TCFNC * pk_tcfnc, FLAG flags)

T_TCFNC * pk_tcfnc
発行するサービスコール情報を格納した構造体へのポインタ

FLAG flags
フラグ

デバッグツールの持つ機能を使って関数を呼び出す. 関数はノンブロッキングで行われることを基本とし, 実行が終了した場合コールバック関数 ***tif_rep_fnc*** が呼ばれる.

構造体 ***T_TCFNC*** の内容は次の通り

```
typedef struct    t_tcfnc_prmary
{
    UINT prmsz          : パラメータのサイズ ( バイト単位 )
    VP prmptr           : パラメータを格納する領域へのポインタ
}    T_TCFNC_PRMARY;

typedef struct    t_tcfnc
{
    DT_VP fncadr        : 関数のアドレス
    DT_VP stkadr        : 関数発行時のスタックポインタ
    UINT retsz          : 結果を格納する領域の大きさ ( バイト単位 )
    VP retptr           : 実行結果を格納する領域へのポインタ
    UINT prmcnt         : パラメータの数
    T_TCFNC_PRMARY prmary[]
                        : パラメータ
}    T_TCFNC;
```

関数の返値を格納するため RIM はバッファを作成し, ***T_TCFNC::resultptr*** にバッファ領域へのポインタを, ***T_TCFNC::resultsz*** にはバッファ領域のサイズを格納する. ***tif_cal_fnc*** は関数を実行した後, 関数の返値をこのバッファへと格納する. デバッグツールは, 返値がこのサイズで格納できないと判断した場合, 発行前にエラーとする. ただし, デバッグツールが返値の型チェックを行うか否かは実装依存である.

デバッグツールはパラメータを渡す際, ***T_TCFNC::param[0]*** が実行する関数の最左パラメータとなるように展開する. デバッグツールはパラメータをそのままターゲットスタックなどに積む場合があるため, 実際に実行する関数が要求するサイズに比べ設定したサイズが満たない場合, 2 つのパラメータが結合されることがある.

実装上ノンブロッキングが不可能である場合は情報取得キーコードの項目 ***TIF.TIF_CAL_FNC.NONBLOCKING*** を ***FALSE***(=0) としなければならない．このとき関数は ***OPT_BLOCKING*** が指定されずに本関数が実行された場合，***E_NOSPT*** を返す．また，この関数がノンブロッキング実行された場合でもコールバック関数 ***tif_rep_fnc*** が呼ばれる．

補足説明

tif_cal_fnc をブロッキング実行した場合, 実装によっては呼び出した関数が厳密な意味で終了するまで本関数は制御を返さない．厳密な終了とは，関数終了時のスタックフレームが ***tif_cal_fnc*** にて関数を呼出した時のスタックフレームと等価となる場合を指す．具体的には，呼び出した関数内でディスパッチが発生し別のタスクに制御が移ったとしても本関数はこれを終了と見なさず，コンテキストの状態にはよらないが対象関数から脱出するまで本関数は制御を返さない場合もある．
tif_cal_fnc の終了通知 ***tif_rep_fnc*** も上記に準ずる．

フラグ

FLG_NOREPORT (80000000h) : 通知関数の無効化
対になるコールバック関数の呼出しを行わない

OPT_BLOCKING (1)
ブロッキングモードで実行する

キー

TIF 05h
.TIF_CAL_FNC 17h[1]
この関数をサポートする
.FLG_NOREPORT 03h[1]
フラグ ***FLG_AUTONUMBERING*** をサポートする
.OPT_BLOCKING 11h[1]
オプション ***OPT_NONBLOCKING*** をサポートする
.NONBLOCKING 12h[1]
ノンブロッキング関数呼び出しに対応している

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_EXCLUSIVE (-226)
すでに要求が発行されており，終了するまで受け付けることができない

E_PAR (-145)
パラメータが不正な値であった

ET_MACV (-26)
ターゲット上の不正なメモリ領域に対するアクセスが発生した

ET_NOMEM (-33)
ターゲット上におけるメモリ不足のため，要求を実行できない

6.6.2 関数実行の終了通知

tif_rep_fnc 関数実行の終了通知 [E:callback] ○

void tif_rep_fnc (FLAG flags)

FLAG flags
フラグ

tif_cal_fnc でノンブロッキング発行した関数の終了を通知する．返値は ***tif_cal_fnc*** で設定した領域に格納される．

キー

TIF 05h
.TIF_REP_FNC 18h[1]
この関数をサポートする

エラー

本関数は返値を持たない

6.7 トレースログ関連操作

6.7.1 トレースログの設定

tif_set_log　トレースログの設定 [E] □

ER_ID　tif_set_log (ID logid, T_TSLOG * pk_tslog , FLAG flags)

ID　logid
　設定したログ情報に割り当てられた ID

T_TSLOG *　pk_tslog
　トレースログの設定情報を格納した構造体へのポインタ

FLAG　flags
　フラグ

(返値) ID　logid
　割り当てられたログ ID (***rif_set_log*** とは独立な ID)

TIF トレースログの設定を行う.

構造体 ***T_TSLOG*** の内容は次の通り

```
typedef struct    t_tslog
{
    UINT logtype      : ログ種別フラグ
    DT_VP staadr      : 開始アドレス
    DT_VP endadr      : 終了アドレス ( 範囲を指定しない場合は NULL)
    DT_VP valptr      : 読出し開始位置 ( NULL : イベント発生位置 )
    DT_SIZE valsz     : データ長 ( バイト単位 )
}    T_TSLOG;
```

T_TSLOG::logtype に設定できる値は次の通り

次の値は排他的に用いることができる.

- ***LOG_INSTRUCTION*** **(0)**
  命令 **(default)**
- ***LOG_DATA*** **(1)**
  データ

logtype に ***LOG_DATA*** を指定した場合，次に示す動作オプションを少なくとも 1 つ指定しなければならない．ただし，次の値は同時に設定することも可能である.

- ***ACS_READ*** **(0x100)**
  読み込み

- ***ACS_WRITE* (0x200)**
  書き込み
- ***ACS_MODIFY* (0x400)**
  修正 (Read Modify Write)

ログ取得用のバッファが満載になったときの動作を，次のなかから排他的に設定できる．

- ***LOG_BUFFUL_STOP* (0)**
  バッファ満載時にトレース取得を停止する **(default)**
- ***LOG_BUFFUL_CALLBACK* (2)**
  バッファ満載時にコールバック関数を実行する
- ***LOG_BUFFUL_FORCEEXEC* (4)**
  バッファ満載時に最も古いものを破棄し，取得を継続する

上記オプションの有効範囲は，この関数の実行によって設定されたログ単体である．例としてオプションを設定しなかったログ (ID:1) と，***OPT_BUFFUL_CALLBACK*** を設定したログ (ID:2) と，***FLG_NOREPORT*** を設定したログ (ID:3) があり，これらのログが起動されているとする．その後，ターゲットプログラムの実行によってバッファが満杯となり，現在取得したログイベントが格納できないとデバッグツールが判断したとき，***OPT_BUFFUL_STOP*** がデフォルト設定されている ID1 および ID3 のログに対して，強制終了が発行され，***tif_rep_log*** は ID1 の終了イベント (***EV_STOP***) を受け取る．ID3 は ***FLG_NOREPORT*** が設定されているので報告は行われない．また ID2 は ***OPT_BUFFUL_CALLBACK*** が設定されているので，***tif_rep_log*** が ***EV_REPORT*** で呼び出される．このとき，***tif_rep_log*** の中でバッファ読出しなどの適切な処理が行われず，再度バッファが満載になった場合，***tif_rep_log*** が現存する全てのログに対して ***EV_BUFFER_FULL*** で呼び出される．

補足説明

変数 ***T_TSLOG::staadr*** と ***T_TSLOG::endadr*** はログイベント発生対象となるメモリ領域を定める．このときの領域は閉区間 [***staadr***, ***endadr***] であり，アドレス ***endadr*** は対象となる．***staadr***>***endadr*** の場合の動作は，***E_PAR*** エラーとする．

変数***T_TSLOG::endadr***はイベント発生対象となるメモリの範囲を規定するものであり, 変数 ***T_TSLOG::valsz*** イベント発生時に読み出すべきメモリ長を規定するものである．***T_TSLOG::valsz*** に 0 を設定した場合，イベントのみが記録される．
変数 ***T_TSLOG::valptr*** はイベント発生時に，読出しの起点となるアドレスを指定する．ある特定のアドレスが設定されていた場合，閉区間 [***staadr***,***endadr***] 間で発生したログイベントに対して，***T_TSLOG::valptr*** から ***T_TSLOG::valsz*** バイト読み出したものを記録する．一方 ***T_TSLOG::valptr*** に ***NULL*** を設定した場合，イベントが発生したアドレスが起点となる．このとき閉区間 [***staadr***, ***endadr***] 間のあるアドレス (***evtadr*** とする) に対するアクセスなどでイベントが発生した場合, ***evtadr*** から ***length*** バイト読み出したものを記録する．

自動番号割当てフラグ***FLG_AUTONUMBERING***を指定し,関数の実行に成功下場合,関数は設定項目に割り当てられた 1 以上の値 (ID 値) を返却する．自動割当フラグを指定しなかった場合であっても同様である.

フラグ

FLG_NOREPORT (80000000h) : 通知関数の無効化
対になるコールバック関数の呼出しを行わない

FLG_AUTONUMBERING (40000000h) : ID 自動割当
ID の自動割当を行う．引数に ID を指定した場合，関数はこれを無視する．成功した場合，関数は自動割当された ID を返す.

OPT_BUFFUL_STOP (0)
バッファが満杯になったらトレース取得を停止する **(default)**

OPT_BUFFUL_FORCEEXEC (1)
バッファが満杯になったら最も古いものを破棄し，継続する

OPT_BUFFUL_CALLBACK (2)
バッファが満杯になったら ***tif_rep_log*** を実行する

キー

TIF 05h
.TIF_SET_LOG 19h[1]
この関数をサポートする
.FLG_NOREPORT 03h[1]
フラグ ***FLG_NOREPORT*** は利用可能である
.FLG_AUTONUMBERING 04h[1]
フラグ ***FLG_AUTONUMBERING*** をサポートする
.OPT_BUFFUL_FORCEEXEC 11h[1]
オプション ***OPT_BUFFUL_FORCEEXEC*** は利用可能である
.OPT_BUFFUL_CALLBACK 12h[1]
オプション ***OPT_BUFFUL_CALLBACK*** は利用可能である
.LOG_INSTRUCTION 13h[1]
ログ種別 ***LOG_INSTRUCTION*** は利用可能である
.LOG_DATA 14h[1]
ログ種別 ***LOG_DATA*** は利用可能である
.LOG_READ 15h[1]
LOG_READ は利用可能である
.LOG_WRITE 16h[1]
LOG_WRITE は利用可能である
.LOG_MODIFY 17h[1]
LOG_MODIFY は利用可能である

エラー

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_ID (-146)
指定されたオブジェクト ID は無効
E_NOID (-162)
自動割当用 ID が不足している
E_OBJ (-169)
操作対象は存在しない または 操作できない
ET_MACV (-26)
ターゲット上の不正なメモリ領域に対するアクセスが発生した
E_PAR (-145)
パラメータが不正な値であった

6.7.2 トレースログ設定の解除

tif_del_log　トレースログ設定の解除 [E] □

ER　　tif_del_log (ID logid, FLAG flags)

ID　　logid
　　解除するログ ID

FLAG　　flags
　　フラグ

この関数は，前出の ***tif_set_log*** で設定したログ項目の一部または全部を取り消す.

補足説明

logid に ***ID_ALL***(=-1) を用いると，全てのログが対象となる．また，この ***logid*** は ***tif_set_log*** で与えられたものであり,***rif_set_log*** で用いられるログ ID とは独立な ID 番号である.

キー

TIF　　05h
.TIF_DEL_LOG　　1Ah[1]
　　この関数をサポートする

エラー

E_OK (0)
　　正常に終了した
E_NOSPT (-137)
　　その操作はサポートしない
E_NOMEM (-161)
　　ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
　　何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
　　何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_ID (-146)
　　指定されたオブジェクト ID は無効
E_OBJ (-169)
　　操作対象は存在しない または 操作できない
E_EXCLUSIVE (-226)
　　すでに要求が発行されており, 終了するまで受け付けることができない

6.7.3 トレースログ取得開始

tif_sta_log　トレースログの開始 [E] □

ER　　tif_sta_log (ID logid, FLAG flags)

ID　　logid
　　起動対象となるログ ID

FLAG　　flags
　　フラグ

tif_set_log で設定した内容でトレースログの取得を開始する．***logid*** に ***ID_ALL***(=-1) を指定した場合，***tif_set_log*** で設定した全てのログ設定を有効にする．

補足説明

すでに開始されたログ設定に対して再度本関数を実行しても，関数は正常終了する．ただし，複数回開始しても一度の停止操作で指定されたログ設定は停止する．

キー

TIF 05h
.TIF_STA_LOG 1Bh[1]
　　この関数をサポートする

エラー

E_OK (0)
　　正常に終了した
E_NOSPT (-137)
　　その操作はサポートしない
E_NOMEM (-161)
　　ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
　　何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
　　何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_ID (-146)
　　指定されたオブジェクト ID は無効
E_OBJ (-169)
　　操作対象は存在しない または 操作できない

6.7.4 トレースログ取得停止

tif_stp_log　トレースログの停止 [E] □

ER　　tif_stp_log (ID logid, FLAG flags)

FLAG　　flags
フラグ

指定された現在取得中のトレースログを停止させる.

補足説明

本関数はターゲットの実行状況に関与しない.
すでに停止したログ設定に対して再度本関数を実行しても，関数は正常終了する．ただし，複数回停止しても一度の開始操作で指定されたログ設定は開始する.

キー

TIF 05h
.TIF_STP_LOG 1Ch[1]
この関数をサポートする

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_ID (-146)
指定されたオブジェクト ID は無効
E_OBJ (-169)
操作対象は存在しない または 操作できない

6.7.5 トレースログ関連コールバック

tif_rep_log　トレースログ関連コールバック [E:callback] ○

void　tif_rep_log (ID logid, UINT event, FLAG flags)

ID　logid
　発生原因となったログの ID

UINT　event
　この関数が呼ばれた原因

FLAG　flags
　フラグ

トレースログ関連操作によって発生した何からの原因，またはトレースログの取得 ***tif_set_log*** によってコールバックを行うと設定されたとき，この関数が呼ばれ，発生原因に沿った適切な処理を行う．またバッファフルなどでログが解除される場合の処理などを行う．

発生事象は次の通り

EV_BUFFER_FULL (1)
　トレースバッファが満杯になった

EV_STOP (2)
　トレースログ機能が停止された

EV_REPORT (4)
　tif_set_log で設定された通知条件を満たした

補足説明

バッファフルなどでログが強制停止された場合，RIM は対象となる ID の ***tif_stp_log*** を呼び出す必要はない．
本コールバックが呼び出されている最中にターゲット上でトレースログを取得すべき状況が起こった場合，本関数はトレースログの内容が取得されることを保証しない．
tif_set_log で ***OPT_BUFFUL_CALLBACK*** が指定されたログに関して，一回目のバッファフルは ***EV_REPORT*** として報告される．その後，***OPT_BUFFUL_CALLBACK*** を指定したログが複数存在する場合，それら全てに対して ***EV_REPORT*** が発行された後，適切な処理が行われず ***EV_REPORT*** のきっかけとなったバッファフルが解消されてない場合，全ての残存ログに復帰不可能なエラーとして ***EV_BUFFER_FULL*** が呼ばれる．このバッファフルに対しても適切な処理が行われなかった場合，デバッグツールは全てのログを強制停止し，***EV_STOP*** を通知して処理を終了させる．

キー

TIF　05h
.TIF_REP_LOG　1Dh[1]
　この関数をサポートする

エラー

この関数は返値を持たない

6.7.6 トレースログの取得

tif_get_log　トレースログソースの取得 [E] □

ER　　tif_get_log (VP p_result, FLAG flags)

VP　　p_result
　　トレースログソースを格納する領域へのポインタ

FLAG　　flags
　　フラグ

デバッグツールが保持しているトレースログソースを取得する．このトレースログソースは，デバッグツールがログ情報として取り込んだターゲット上のメモリ内容である．RIM がデバッグツールに頼らず，ターゲット上のデバッグタスクなどにより直接メモリやディスクなどにログを記述していく場合，この関数でログを取得することはできない．
tif_get_log は 1 項目分のログを取得すると，次のログ項目へ読み出し位置を移動させる．全てのログを取得するために，RIM は複数回この関数を呼び出す必要がある．残り項目数が 0 になると，***tif_get_log*** は ***E_OBJ*** エラーを返す．

ログ取得用構造体 ***T_TGLOG*** の内容は次の通り

```
typedef struct    t_tglog
{
    ID logid            : 対応するログ ID
    DT_VP staadr        : 設定した開始アドレス
    DT_VP endadr        : 設定した終了アドレス
    UINT logtype        : ログ種別情報
    LOGTIM logtim       : タイムスタンプ
    DT_SIZE bufsz       : バッファサイズ
    char buf[]          : 取得した値を格納する領域
}    T_TGLOG;
```

※ ***tlogid*** は無い場合 ***(tlogid=0)*** もある. その時はアドレスなどから判断しなければならない.
※ ***bufsz*** に設定された値が ***buf*** で取得できる最大長となる. 関数を実行すると, ***bufsz*** には格納したデータのサイズが格納される．詳細は ***rif_ref_obj*** を参照のこと

オプション

OPT_PEEK (1)
　　トレースログをスプールから削除せずに取り出す

キー

TIF	05h
.TIF_GET_LOG	1Eh[1]
この関数をサポートする	
.OPT_PEEK	10h[1]

オプション ***OPT_PEEK*** をサポートする

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_OBJ (-169)
操作対象は存在しない または 操作できない

E_PAR (-145)
パラメータが不正な値であった

7. その他のインターフェース

7.1 デバッグツール関連操作

7.1.1 デバッグツール情報の取得

dbg_ref_dbg デバッグツール関連情報の取得 [R] □

ER dgb_ref_dbg
(T_INFO * pk_rdbg, UINT packets, FLAG flags)

T_INFO * pk_rdbg
デバッグツール関連情報を格納する構造体の配列の先頭を指すポインタ

UINT packets
T_INFO 構造体配列の長さ

FLAG flags
フラグ

この関数は RIM がデバッグツールの種別，行える操作などを調べるために用いる．

本関数の情報取得には情報取得用関数 ***T_INFO*** とキーコードを用いる．詳細に関しては 3.6 [情報取得構造体と取得キー] を参照のこと．

T_INFO の内容は次の通り．

```
typedef struct    t_info_result_buf
{
    UINT sz             : バッファのサイズ
    VP ptr              : 文字列 または 特殊な型を格納する領域へのポインタ
}    T_INFO_RESULT_BUF;

typedef union     t_info_result
{
    INT value           : 32bit 符号付整数
    T_INFO_RESULT_BUF buf
                        : 特殊な型の値
}    T_INFO_RESULT;

typedef struct    t_info
{
    char key[4]         : 情報を特定するためのキー
    T_INFO_RESULT result
                        : キーに対応する値
}    T_INFO;
```

キー

DEBUGGER 1h
- **.CNDBREAK** 1h
  - **.NUM** 3h[W]
    設定可能な条件ブレークの数 (0:Not supported)
- **.LOG** 2h
  - **.NUM** 3h[W]
    設定可能なハードウェアログの数 (0:Not supported)
- **.NAME** 80h[S]
  デバッグツールを特定する一意な文字

HOST 2h
- **.ENDIAN** 1h[W]
  ホストコンピュータのエンディアン (0:Little 1:Big)
- **.NAME** 80h[S]
  ホストコンピュータを特定する一意な文字

TARGET 3h
- **.ENDIAN** 1h[W]
  ターゲットコンピュータのエンディアン (0:Little 1:Big)
- **.REGISTER** 2h
  - **.NUM** 3h[W]
    ターゲットコンピュータのレジスタの数
- **.NAME** 80h[S]
  ターゲット装置を識別する一意な文字

補足説明

本関数で取得できる情報には，6. [ターゲットアクセスインターフェース] で内で記述されている ***INF_TIF*** を第一キーに持つ全キーコードも含む

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_OBJ (-169)
操作対象は存在しない または 操作できない

E_PAR (-145)
パラメータが不正な値であった

7.2RIM 関連操作

7.2.1 RIM 初期化

dbg_ini_rim RIM 初期化 [R] ○

ER dbg_ini_rim(VP param)

VP param
デバッグツールから送られてくる実装依存のパラメータ

デバッグツールの起動時に RIM を初期化する．コールバック関数などもこの時点で登録を行う．この関数は後述の「インターフェース初期化」関数 ***dbg_ini_inf*** よりも後に実行される．そのため，この関数内において，デバッグツール側が提供している全てのインターフェース上の関数は利用可能であることが保証される．

パラメータの内容は特に定めないが，デバッギングインターフェース内のガイドライン (Windows-DLL ガイドラインなど) で標準化する場合もある．

補足説明

本関数が ***E_OK*** 以外のエラーを返した場合，初期化に失敗したと判断し，デバッグツールは RIM 側に所属する他のインターフェース関数を読み出してはならない．

エラー

E_OK (0)
正常に終了した

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

(本関数はこれ以外のエラー値を取らない)

7.2.2 RIM の終了処理

dbg_fin_rim RIM の終了処理 [R] ○

ER dbg_fin_rim (VP param)

VP param
デバッグツールから送られるパラメータ

RIM の終了処理を行う. デバッグツールはプログラム終了前にこの関数を呼ばなければならず, RIM は獲得した資源の全てをこの関数内で解放しなければならない.
パラメータの内容は特に定めないが, デバッギングインターフェース内のガイドライン (Windows-DLL ガイドラインなど) で標準化する可能性はある.

補足説明

デバッグツールはこの関数が ***E_OK*** 以外で終了した場合, これ以降 RIM が提供する全ての関数を呼び出してはならない.

エラー

E_OK (0)
正常に終了した

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

(本関数はこれ以外のエラー値を取らない)

7.2.3 RIM 情報の取得

dbg_ref_rim RIM に関連する情報の取得 [R] ○

ER dbg_ref_rim
(T_INFO * ppk_rrim, UINT packets, FLAG flags)

T_INFO * ppk_rrim
情報を格納する構造体の配列の先頭ポインタ

UINT packets
ppk_rrim が示す配列の長さ

この関数は RIM の持つ機能など，RIM に関する情報を取得する．デバッグツールは ***dbg_ref_rim*** で得られた情報をもとに，RTOS アクセスインターフェース上で用いることの出来る機能などを知ることが出来る．

本関数の情報取得には情報取得用関数 ***T_INFO*** とキーコードを用いる．詳細に関しては 3.6 [情報取得構造体と取得キー] を参照のこと．

キー

OS 8h
.NAME 80h[S]
ターゲット OS を識別する一意な文字 ("ITRON")

補足説明

本関数で取得できる情報には，5. [RTOS アクセスインターフェース] 内で記述されている ***INF_RIF*** を第一キーとする全キーコードも含む

エラー

E_OK (0)
正常に終了した
E_NOSPT (-137)
その操作はサポートしない
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_OBJ (-169)
操作対象は存在しない または 操作できない
E_PAR (-145)
パラメータが不正な値であった

7.3 インターフェース関連操作

7.3.1 インターフェース初期化

dbg_ini_inf インターフェースの初期化 [E] ○

ER dbg_ini_inf (T_INTERFACE * ppk_interface, VP param)

T_INTERFACE * ppk_interface
各関数へのエントリポイントを格納する領域へのポインタ

VP param
デバッグツール側が提供するパラメータ

dbg_ini_inf は，インターフェース関数へアクセスするための関数ポインタテーブルの位置を通達すると共に，その初期化を行うために，デバッグツール側から実行される関数である．この関数内で RIM は自身が提供するインターフェース上の関数へのポインタを ***ppk_interface*** に登録する．

この関数が実行される前にデバッグツールは次の関数へのポインタを提供しなければならない．

- ***dbg_ref_dbg***
- **TIF 上の関数**
  (注 この時点ではまだ RIF 上のコールバックを登録する必要はない)

この関数内で RIM は次の関数へのポインタを提供しなければならない．

- ***dbg_ini_rim***
- ***dbg_ref_rim***
- ***dbg_fin_rim***
- **RIM 上の関数**
  (注 この時点ではまだ TIF 上のコールバックを登録する必要はない)

T_INTERFACE は本デバッギングインターフェース仕様で提供される全ての関数ポインタを持つ構造体である．

全ての関数が静的にバインドされるような環境下ではこの関数を実行する必要はない．

エラー

E_OK (0)
正常に終了した

E_NOSPT (-137)
その操作はサポートしない

E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない

E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)

E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した

E_PAR (-145)
パラメータが不正な値であった

8. ガイドライン規定

本章では ITRON デバッギングインタフェース仕様の「ガイドライン規定」を説明する．ガイドライン規定は仕様規定とは異なり準拠する必要はないが，互換性などにかかわる項目を定めているため，より多くのデバッグツール /RIM をサポートするためにはこの規定に沿って実装が行われていることが望ましい．

8.1RIM 作成ガイドライン

8.1.1 RIM 動作ガイドライン

- **ターゲット初期化前の不定状態におけるアクセス**
  ターゲットが初期化されていない状態では，デバッグツールによるアクセスができない場合がある．そのような状態で操作が行われた場合，関数はシステムエラー ***E_SYS*** を返し，その際得られた情報は無効となる．

8.1.2 RIM 提供手法ガイドライン

RIM は各社のデバッグツール内に取り込まれるため, その提供方法には特定のガイドラインが適応される．

現在の仕様では，次のような提供方法を考慮している．

- **C 言語ソースプログラムを提供する**
- **ライブラリで提供する**

上記以外の提供法を用いる場合，RIM を作成する側はモジュール本体と C 言語インターフェースの間にサンクレイヤなどを導入し，これらの橋渡しを行う必要がある．

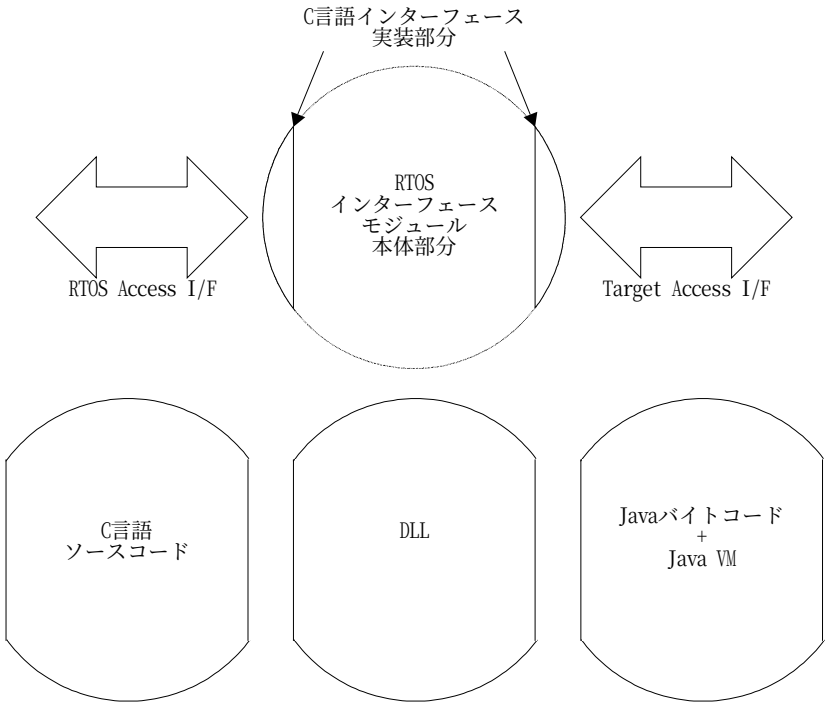


Figure 22: **特殊な** RIM **提供方法**

8.1.3 高速化とデバッグエージェント

ブレークポイントを設定するに当たり，現在のデバッギングインターフェースではコールバックを用いて判定を行っている．しかし，シミュレータ環境ではない実機を用いた環境では，得てしてホスト - ターゲット間はシリアルインターフェースによって情報転送を行っているものが多いため，コールバックが頻繁に発生することはデバッグツールの速度低下につながる．
そのため，RTOS メーカは高速にこれらを動かすためには「高速化を意図したデバッグタスク」を導入することが望ましい．(その機能は特に高速化を期待されるブレークとトレースログには効果的である)

このような目的のために利用されるデバッグタスクが持つ機能の一例を列挙する．

- **ブレークに関する機能**
- **ブレーク条件の一部をデバッグタスク内で解決する機能**
- **トレースログに関する機能**
- **デバッグツールの力を借りないターゲットで閉じたトレースログ取得機能**

これら以外に RTOS の特性などによってより効果的な機能を提供することや，より高速な動作を提供することも可能と考えている．

これらの機能をデバッグエージェントに導入し，提供することで，利用者は TPO に併せて 3 つ (場合によっては 2 つ) の環境を使い分けてデバッグを行うことができる．

- **大きなデバッグエージェントと小さな RIM を用い, 比較的見つけやすいバグを豊富な機能によって取り除くためのデバッグ環境**
- **小さなデバッグエージェントと大きな RIM を用い, 機能は限られるものの比較的ターゲットにかかる負荷を最小にすることで実環境に近づけ, 時間制約など見つけづらいバグを取り除くためのデバッグ環境**
- **RIM だけを用い，ターゲットにいっさい負荷をかけないデバッグ環境**

このように RIM とデバッグエージェントを 1 つのセットとして考え, デバッグの状況 (オーバーヘッドと機能とのトレードオフ) に応じて使い分けることができるよう複数のセットを提供できれば, 利用者のデバッグ事情はより改善されるものと期待している.

8.2 Windows-DLL 作成ガイドライン (32bitRIM)

8.2.1 型

Windows-DLL で提供する場合ホスト側の型は次のように固定される.

Table 23: 32bit RIM DLL **ホスト型**

型名	意味	ビット長
BOOL	真偽値	32bit
ER_ID	ID と ER のどちらか大きな整数 正数は ID を，負数は ER を示す	32bit
ID	デバッギングインターフェース上のオブジェクト番号を格納するのに十分な大きさの符号なし整数	32bit
INT	ホスト上における自然な長さの符号付整数	32bit
UINT	ホスト上における自然な長さの符号なし整数	32bit
VP	ホスト上における型無しポインタ	32bit
VP_INT	VP, INT を格納するのに十分な大きさの型	32bit
LOGTIM	ログ時間 (精度に関しては実装定義)	64bit

Table 24: 32bit RIM DLL **ターゲット型**

型名	意味	ビット長
DT_B, DT_UB, DT_VB	8 ビットデータ型	8bit
DT_H, DT_UH, DT_VH	16 ビットデータ型	16bit
DT_W, DT_UW, DT_VW	32 ビットデータ型	32bit
DT_D, DT_UD, DT_VD	64 ビットデータ型	64bit

Table 24: 32bit RIM DLL **ターゲット型**

型名	意味	ビット長
DT_SYSTIM, DT_RELTIM, DT_OVRTIM, DT_TMO	時刻関連型 (絶対時刻，相対時刻，相対時間型)	64bit
上記以外	上記以外の全ての型	32bit[a]

a. 64bit RIM DLL では，64bit データとして扱う

場合によってはこれらの型が Windows の規定する型と重複する場合がある. その場合は次のような手段によって回避することが可能である.

Program source

```
#define TYPE WINDOWS_TYPE
#include <windows.h>
#undef TYPE
    // 以後 TYPE は WINDOWS_TYPE として利用可能になる
```

Program source

8.2.2 構造体のビットアライメント

Windows-DLL として作成される RIM，およびその RIM-DLL を受け入れるデバッグツールは，デバッギングインターフェースで定義されるそれぞれの構造体を宣言するにあたり，Windows と同様に次のアライメント規約に従わなければならない.

Table 25: Windows DLL **作成ガイドライン ビットアライメント**

データ型	アライメント
DT_B*, *DT_UB	バイト境界に配置
DT_H*, *DT_UH	偶数バイト境界に配置
32bit データ型	32 ビット境界に配置
LOGTIM*, *DT_SYSTIM	64 ビット境界に配置
構造体	いずれかのメンバの最大アライメント要件に合わせる
共用体	最初のメンバのアライメント要件に合わせる

8.2.3 関数のエクスポート

RIM-DLL は次の関数のシンボルをエクスポートしなければならない.

- ***dbg_ini_inf* : インターフェースの初期化**

8.3 標準実行履歴ファイル形式

ITRON デバッギングインタフェース仕様では，取得した実行履歴をファイルの格納する際の標準形式を定める.

ファイルは ASCII 形式で格納され，各トークン間は 1 つ以上の空白文字[1] で区切られる．また記号 "." "|" ":" ";" は区切り子[2] とみなす．
以下に文法を示す．

文法のフォーマット

非終端記号	斜体
終端記号	太字ゴシック
コメント	# に続く文字列
文字列	文字列 xxx とコメントで表現

標準履歴ファイル
　　コンフィギュレーションデータ群　実行履歴データ群

コンフィギュレーションデータ群
　　コンフィギュレーションデータ　コンフィギュレーションデータ群
　　コンフィギュレーションデータ

コンフィギュレーションデータ
　　キーコード: *値リスト*;

キーコード
　　キー　後続キー　後続キー　後続キー
　　キー　後続キー　後続キー
　　キー　後続キー
　　キー

後続キー
　　. *キー*

キー
　　xxx　　# キーの名前

値リスト
　　値 値リスト
　　値

値
　　-　　# 値の設定をスキップする場合はハイフンで示す
　　整数値　　# 値の表記は C 言語に準拠 (10 進 16 進表記のみ)
　　文字列　　# 値の表記は C 言語に準拠

実行履歴データ群
　　実行履歴データ　実行履歴データ群
　　実行履歴データ

実行履歴データ
　　実行履歴ヘッダ　種別依存履歴データ;

---

1. スペース , キャリッジリターン , ラインフィード , タブ
2. 区切り子の前後の空白文字は省略可能

実行履歴ヘッダ
　　履歴種別 **：***履歴時刻*

履歴種別
　　xxx　　#LOG_TYP_xxx で示される全ログ種別名
　　xxx **| ENTER**　　#LOG_TYP_xxx|LOG_ENTER
　　xxx **| LEAVE**　　#LOG_TYP_xxx|LOG_LEAVE

履歴時刻
　　-　　# 値の設定をスキップする場合はハイフンで示す
　　整数値

種別依存履歴データ
　　値リスト　　# 各ログ種別のパラメータのメンバの数だけ

この文法から生成される言語の例

Program source
```
CFG.LOGTIM.TICK_N : 1;
CFG.LOGTIM.TICK_D : 1000;
INTERRUPT|ENTER : 0 4;
TASK|ENTER : 180 1;
COMMENT : 200 25 " プログラムが開始しました ";
```
Program source

9. リファレンス

9.1 構造体

•T_MEMBLK [tif_get_bls, tif_set_bls]

```
typedef struct    t_memblk
{
    DT_VP blkptr        : メモリブロックデータを格納するポインタ
    DT_SIZE blksz       : メモリブロックデータのバイト数
}    T_MEMBLK;
```

•T_BLKSET [tif_get_bls, tif_set_bls]

```
typedef struct    t_blkset
{
    UINT blkcnt         : ブロック数
    T_MEMBLK blkary []
                        : ブロックの配列
}    T_BLKSET;
```

•T_RCSVC [rif_cal_svc]

```
typedef struct    t_rcsvc
{
    DT_FN svcfn         : 発行する機能コード
    BOOL tskctx         : タスクコンテキストで実行する (= TRUE)
    DT_ID tskid         : 対象となるタスク ID (tskctx = TRUE 時 )
    UINT prmcnt         : パラメータの数
    VP_INT prmary[]     : 全パラメータリストを格納する配列
}    T_RCSVC;
```

•T_GRDT [rif_get_rdt, tif_get_reg, tif_set_reg]

```
typedef struct    t_grdt_regary
{
    char * strname      : レジスタ名称を指すポインタ
    UINT length         : 長さ ( バイト単位 )
    UINT offset         : 格納オフセット位置
}    T_GRDT_REGARY;

typedef struct    t_grdt
{
    UINT regcnt         : レジスタ本数
    UINT ctxcnt         : コンテキストに含まれる可能性のあるレジスタの数
    T_GRDT_REGARY regary[]
                        : レジスタ情報
}    T_GRDT;
```

•T_INFO [rif_ref_cfg, dbg_ref_dbg, dbg_ref_rim]

```
typedef struct    t_info_result_buf
{
    UINT sz             : バッファのサイズ
    VP ptr              : 文字列 または 特殊な型を格納する領域へのポインタ
}    T_INFO_RESULT_BUF;

typedef union     t_info_result
{
    INT value           : 32bit 符号付整数
    T_INFO_RESULT_BUF buf
                        : 特殊な型の値
}    T_INFO_RESULT;

typedef struct    t_info
{
    char key[4]         : 情報を特定するためのキー
    T_INFO_RESULT result
                        : キーに対応する値
}    T_INFO;
```

•T_RCLOG [rif_cfg_log]

```
typedef struct    t_rclog
{
    UINT type           : トレースログ構成情報
    DT_VP bufptr        : トレースログバッファへのポインタ
    DT_SIZE bufsz       : トレースログバッファのサイズ
}    T_RCLOG;
```

•T_RGLOG_COMMENT [rif_get_log]

```
typedef struct    t_rglog_comment
{
    UINT length         : 文字列の長さ
    char strtext []     : 文字列 (NULL 終端 ) - 中断あり
}    T_RGLOG_COMMENT;
```

•T_RGLOG_CPUEXC [rif_get_log]

```
typedef struct    t_rglog_cpuexc
{
    DT_ID tskid         : 対象となるタスク ID
}    T_RGLOG_CPUEXC;
```

•T_RGLOG_DISPATCH_ENTER [rif_get_log]

```
typedef struct    t_rglog_dispatch_enter
{
    DT_ID tskid         : これまで実行状態にあったタスクの ID
    UINT disptype       : ディスパッチ種別
}    T_RGLOG_DISPATCH_ENTER;
```

•T_RGLOG_DISPATCH_LEAVE [rif_get_log]

```
typedef struct    t_rglog_dispatch_leave
{
    DT_ID tskid         : これから実行状態になるタスクの ID
}    T_RGLOG_DISPATCH_LEAVE;
```

•T_RGLOG_INTERRUPT [rif_get_log]

```
typedef struct    t_rglog_interrupt
{
    DT_INHNO inhno  : 割込ハンドラ番号
}    T_RGLOG_INTERRUPT;
```

•T_RGLOG_ISR [rif_get_log]

```
typedef struct    t_rglog_isr
{
    DT_ID isrid         : 割込みサービスルーチン ID
    DT_INHNO inhno  : 割込みハンドラ番号
}    T_RGLOG_ISR;
```

•T_RGLOG_SVC [rif_get_log]

```
typedef struct    t_rglog_svc
{
    DT_FN fncno         : 機能コード
    UINT prmcnt         : パラメータ数
    DT_VP_INT prmary[]: パラメータ
}    T_RGLOG_SVC;
```

•T_RGLOG_TIMERHDR [rif_get_log]

```
typedef struct    t_rglog_timerhdr
{
    UINT type           : タイマーの種別
          (rif_ref_obj::objtype で利用する定数 OBJ_xxx が格納される )
    DT_ID hdrid         : タイムイベントハンドラの ID
    DT_VP_INT exinf  : 拡張情報
}    T_RGLOG_TIMERHDR;
```

•T_RGLOG_TSKEXC [rif_get_log]

```
typedef struct    t_rglog_tskexc
{
    DT_ID tskid         : 対象となるタスク ID
}     T_RGLOG_TSKEXC;
```

•T_RGLOG_TSKSTAT [rif_get_log]

```
typedef struct    t_rglog_tskstat
{
    DT_ID tskid         : タスク ID
    DT_STAT tskstat     : 遷移先タスク状態
    DT_STAT tskwait     : 待ち状態
    DT_ID wobjid        : 待ち対象のオブジェクト ID
}     T_RGLOG_TSKSTAT;
```

•T_ROALM [rif_ref_obj]

```
typedef struct    t_roalm
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR almatr     : 属性
    DT_VP_INT exinf   : 拡張情報
    DT_FP almhdr      : 起動番地
    DT_STAT almstat   : アラームハンドラの起動状態
    DT_RELTIM lefttim : 残り時間
}     T_ROALM;
```

•T_ROCYC [rif_ref_obj]

```
typedef struct    t_rocyc
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR cycatr     : 属性
    DT_VP_INT exinf   : 拡張情報
    DT_FP cychdr      : 起動番地
    DT_RELTIM cyctim  : 周期
    DT_RELTIM cycphs  : 起動位相
    DT_STAT cycstat   : 周期起動ハンドラの起動状態
    DT_RELTIM lefttim : 残り時間
}     T_ROCYC;
```

•T_RODTQ [rif_ref_obj]

```
typedef struct    t_rodtq
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR dtqatr     : データキュー属性
    DT_UINT dtqcnt    : データキュー容量
```

```
    DT_UINT stskcnt    : 送信待ちタスク数 (wstsklst の上界を兼ねる )
    DT_ID * stsklst    : 送信待ちタスク ID リストを格納する領域へのポインタ
    DT_UINT rtskcnt    : 受信待ちタスク数 (wrtsklst の上界を兼ねる )
    DT_ID * rtsklst    : 受信待ちタスク ID リストを格納する領域へのポインタ
    DT_UINT itemcnt    : キュー内データ数 (itemlst の上界を兼ねる )
    DT_VP_INT * itemlst
                       : 全内容のリストを格納する領域へのポインタ
}   T_RODTQ;
```

•T_ROEXC [rif_ref_obj]

```
typedef struct    t_roexc
{
    BITMASK valid    : 有効フィールド情報
    DT_FP excrtn     : 例外ハンドラ起動番地
}   T_ROEXC;
```

•T_ROFLG[rif_ref_obj]

```
typedef struct    t_roflg_wflglst
{
    DT_ID wtskid     : 待ちタスク ID
    DT_FLGPTN wflgptn: タスクの待ちフラグパターン
    DT_UINT wflgmode : タスクの待ちモード
}   T_ROFLG_WFLGLST;

typedef struct    t_roflg
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR flgatr    : フラグ属性
    DT_FLGPTN iflgptn : 初期フラグパターン
    DT_FLGPTN flgptn : フラグパターン
    DT_UINT wflgcnt  : 待ちタスク数 (wflglst の上界を兼ねる )
    T_ROFLG_WFLGLST * wflglst
                     : このフラグを待つタスクの情報へのポインタ
}   T_ROFLG;
```

•T_ROISR [rif_ref_obj]

```
typedef struct    t_roisr
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR isratr    : 属性
    DT_VP_INT exinf  : 拡張情報
    DT_FP isrfnclst  : 登録済みルーチンの開始番地
    DT_INHNO inhno   : 対応づけられた割込みハンドラ番号
}   T_ROISR;
```

•T_ROKER [rif_ref_obj]

```
typedef struct    t_roker
{
    BITMASK valid    : 有効フィールド情報
    BOOL actker      : カーネル起動状態 (TRUE= 起動している )
    BOOL inker       : カーネルコードを実行している (TRUE= 実行中 )
    BOOL ctxstat     : コンテキスト状態 (sns_ctx)
    BOOL loccpu      : CPU ロック状態 (sns_cpu)
    BOOL disdsp      : ディスパッチ禁止状態 (sns_dsp)
    BOOL dsppnd      : ディスパッチ保留状態 (sns_dpn)
    DT_SYSTIM systim : システム時刻
    DT_VP intstk     : 非タスクコンテキスト時のスタック
    DT_SIZE intstksz : 非タスクコンテキスト時のスタックのサイズ
}    T_ROKER;
```

•T_ROMBX [rif_ref_obj]

```
typedef struct    t_rombx
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR mbxatr    : メールボックス属性
    DT_PRI maxmpri   : 優先度の最大値
    DT_UINT wtskcnt  : 待ちタスク数 (wtsklst の上界を兼ねる )
    DT_ID * wtsklst  : 待ちタスク ID リストを格納する領域へのポインタ
    DT_UINT msgcnt   : メッセージヘッダ数 (msglst の上界を兼ねる )
    DT_T_MSG ** msglst
                     : 全メッセージリストを格納する領域へのポインタ
}    T_ROMBX;
```

•T_ROMBF [rif_ref_obj]

```
typedef struct    t_rombf_msglst
{
    DT_VP msgadr : メッセージのアドレス
    DT_UINT msgsz : メッセージの長さ
}    T_ROMBF_MSGLST;

typedef struct    t_rombf
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR mbfatr    : メッセージバッファ属性
    DT_UINT maxmsz   : メッセージ最大サイズ
    DT_SIZE mbfsz    : バッファ領域のサイズ
    DT_UINT stskcnt  : 送信待ちタスク数 (stsklst の上界を兼ねる )
    DT_ID * stsklst  : 待ちタスク ID リストを格納する領域へのポインタ
```

```
    DT_UINT rtskcnt   : 受信待ちタスク数 (rtsklst の上界を兼ねる )
    DT_ID * rtsklst   : 待ちタスク ID リストを格納する領域へのポインタ
    DT_SIZE fmbfsz    : 空き領域の大きさ
    DT_UINT msgcnt    : メッセージ数 (msglst の上界を兼ねる )
    T_ROMBF_MSGLST * msglst
                      : 各メッセージの情報へのポインタ
}   T_ROMBF;
```

•T_ROMPF [rif_ref_obj]

```
typedef struct    t_rompf_blklst
{
    DT_ID htskid      : ブロックを獲得しているタスクの ID 番号
    DT_VP blkadr      : ブロック開始アドレス
}   T_ROMPF_BLKLST;

typedef struct    t_rompf
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR mpfatr     : 固定長メモリプール属性
    DT_SIZE blksz     : ブロックサイズ
    DT_UINT fblkcnt   : 残り固定長メモリブロック数
    DT_UINT blkcnt    : 全メモリブロック数
    DT_UINT ablkcnt   : 獲得済みブロック数 (blklst の上界 )
    T_ROMPF_BLKLST * ablklst
                      : 各ブロックの詳細情報へのポインタ
    DT_UINT wtskcnt   : 獲得待ちタスクの数 (wtsklst の上界 )
    DT_ID * wtsklst   : 獲得待ちタスクの ID を格納する領域へのポインタ
}   T_ROMPF;
```

•T_ROMPL [rif_ref_obj]

```
typedef struct    t_rompl_blklst
{
    DT_SIZE blksz     : ブロックの大きさ
    DT_ID htskid      : ブロックを獲得しているタスクの ID 番号
    DT_VP blkadr      : ブロック開始アドレス
}   T_ROMPL_BLKLST;

typedef struct    t_rompl
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR mplatr     : 可変長メモリプール属性
    DT_SIZE mplsz     : 可変長メモリプール領域のサイズ
    DT_UINT fblksz    : 獲得可能な最大サイズ
    DT_UINT ablkcnt   : 獲得済みブロック数 (blklst の上界 )
```

```
    T_ROMPL_BLKLST * ablklst
                        : 各ブロックの詳細情報へのポインタ
    DT_UINT wtskcnt  : 獲得待ちタスクの数 (wtsklst の上界 )
    DT_ID * wtsklst     : 獲得待ちタスクの ID を格納する領域へのポインタ
}     T_ROMPL;
```

•T_ROMTX [rif_ref_obj]

```
typedef struct     t_romtx
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR mtxatr     : ミューテックス属性
    DT_PRI ceilpri    : 上限優先度
    DT_ID htskid      : ミューテックスをロックするタスクの ID
    DT_UINT wtskcnt   : 待ちタスク数 (wtsklst の上界を兼ねる )
    DT_ID * wtsklst   : 待ちタスク ID リストを格納する領域へのポインタ
}     T_ROMTX;
```

•T_ROOVR [rif_ref_obj]

```
typedef struct     t_roovr
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR ovratr     : 属性
    DT_FP ovrhdr      : 起動番地
    DT_STAT ovrstat   : ハンドラの起動状態
    DT_OVRTIM lefttmo
                      : 残りプロセッサ時間
}     T_ROOVR;
```

•T_ROPOR [rif_ref_obj]

```
typedef struct     t_ropor
{
    BITMASK valid     : 有効フィールド情報
    DT_ATR poratr     : ランデブポート属性
    DT_UINT maxcmsz : 呼び出しメッセージの最大サイズ
    DT_UINT maxrmsz : 呼応メッセージの最大サイズ
    DT_UINT ctskcnt   : 呼び出し待ちタスク数 (ctsklst の上界を兼ねる )
    DT_ID * ctsklst   : 全呼び出し待ちタスク ID を格納する領域へのポインタ
    DT_UINT atskcnt   : 受付待ちタスク数 (atsklst の上界を兼ねる )
    DT_ID * atsklst   : 全受付待ちタスク ID を格納する領域へのポインタ
}     T_ROPOR;
```

•T_RORDV [rif_ref_obj]

```
typedef struct     t_rordv
{
```

```
    BITMASK valid    : 有効フィールド情報
    DT_ID tskid      : ランデブ待ち状態のタスク ID
}   T_RORDV;
```

•T_RORDQ [rif_ref_obj]

```
typedef struct   t_rordq
{
    BITMASK valid    : 有効フィールド情報
    DT_ID runtskid   : 現在実行中のタスク ID
    DT_UINT tskcnt   : レディー( 含 RUNNING) 状態にあるタスク数 (tsklst の上界 )
    DT_ID * tsklst   : 実行可能状態にある全タスク ID の格納領域へのポインタ
}   T_RORDQ;
```

•T_ROSEM [rif_ref_obj]

```
typedef struct   t_rosem
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR sematr    : セマフォ属性
    DT_UINT isemcnt  : 初期セマフォカウント
    DT_UINT maxsem   : セマフォ資源の最大値
    DT_UINT semcnt   : セマフォカウント値
    DT_UINT wtskcnt  : 待ちタスク数 (wtsklst の上界を兼ねる )
    DT_ID * wtsklst  : 待ちタスク ID リストを格納する領域へのポインタ
}   T_ROSEM;
```

•T_ROTEX [rif_ref_obj]

```
typedef struct   t_rotex
{
    BITMASK valid    : 有効フィールド情報
    DT_TEXPTN pndptn : 保留状態にある例外要因
    DT_FP texrtn     : 例外ハンドラ起動番地
}   T_ROTEX;
```

•T_ROTMQ [rif_ref_obj]

```
typedef struct   t_rotmq_quelst
{
    UINT objtype     : 待ちオブジェクトの種別を格納する領域へのポインタ
    DT_ID wobjid     : 待ちオブジェクトの ID を格納する領域へのポインタ
    DT_TMO lefttmo   : 残り待ち時間を格納する領域へのポインタ
}   T_ROTMQ_QUELST;

typedef struct   t_rotmq
{
    BITMASK valid    : 有効フィールド情報
```

```
    DT_SYSTIM systim : 情報取得時のシステム時刻
    DT_UINT quecnt   : タイマーキュー内の待ちオブジェクト数 (quelst の上界 )
    T_ROTMQ_QUELST * quelst
                     : タイマーキュー内の待ちオブジェクト情報へのポインタ
}    T_ROTMQ;
```

•T_ROTSK [rif_ref_obj]

```
typedef struct    t_rotsk
{
    BITMASK valid    : 有効フィールド情報
    DT_ATR tskatr    : タスク属性
    DT_VP_INT exinf  : 拡張情報
    DT_FP task       : 起動番地
    DT_PRI itskpri   : 初期優先度
    DT_VP stk        : 初期スタックの先頭番地
    DT_SIZE stksz    : スタックサイズ
    DT_STAT tskstat  : タスク状態
    DT_PRI tskpri    : タスクの現在優先度
    DT_PRI tskbpri   : タスクのベース優先度
    DT_STAT tskwait  : タスクの待ち要因
    DT_ID wobjid     : 待ち対象のオブジェクト ID
    DT_TMO lefttmo   : タイムアウトまでの時間
    DT_UINT actcnt   : 起動要求キューイング数
    DT_UINT wupcnt   : 起床要求キューイング数
    DT_UINT suscnt   : 強制待ち要求ネスト数
}    T_ROTSK;
```

•T_RRCND_DBG [rif_ref_cnd]

```
typedef struct    t_rrcnd_dbg
{
    DT_VP execadr    : 実行番地 (NULL : NC)
    DT_VP valadr     : アドレス (NULL : NC)
    UINT vallen      : 内容の長さ ( バイト長 :1,2,4)
    VP_INT value     : 内容またはポインタ値
}    T_RRCND_DBG;
```

•T_RRCND_RTOS [rif_ref_cnd]

```
typedef struct    t_rrcnd_rtos
{
    FLAG type        : 調べる内容
    DT_ID objid      : 条件となる ID
}    T_RRCND_RTOS;
```

•T_RSBRK [rif_set_brk]

```
typedef struct    t_rsbrk
{
    UINT brktype        : ブレークの種別
    UINT brkcnt         : ブレークするまでの回数
    DT_ID tskid         : タスク ID
    DT_ID objid         : オブジェクト ID
    UINT objtype        : オブジェクトの種別
    VP_INT brkprm       : コールバック関数へのパラメータ
    DT_VP brkadr        : ブレークを設定するアドレス
    DT_FN svcfn         : 機能コード
}    T_RSBRK;
```

•T_RGLOG [rif_get_log]

```
typedef struct    t_rglog
{
    UINT logtype        : ログの種別
    LOGTIM logtim       : 発生時刻
    BITMASK valid       : 有効フィールドビットマップ
    UINT bufsz          : バッファ領域 buf の大きさ ( バイト単位 )
    char buf[]          : 情報を格納するバッファ領域 ( 詳細は後述 )
}    T_RGLOG;
```

•T_RSLOG_CPUEXC [rif_set_log]

```
typedef struct    t_rslog_cpuexc
{
    DT_EXCNO excno : CPU 例外番号 (ID_ALL 可 )
}    T_RSLOG_CPUEXC;
```

•T_RSLOG_DISPATCH [rif_set_log]

```
typedef struct    t_rslog_dispatch
{
    DT_ID tskid         : タスク番号 (ID_ALL 可 )
}    T_RSLOG_DISPATCH;
```

•T_RSLOG_INTERRUPT [rif_set_log]

```
typedef struct    t_rslog_interrupt
{
    DT_INTNO intno    : 割込番号 (ID_ALL 可 )
}    T_RSLOG_INTERRUPT;
```

•T_RSLOG_ISR [rif_set_log]

```
typedef struct    t_rslog_isr
{
```

```
    DT_ID isrid         : 割込サービスルーチン ID (ID_ALL 可 )
    DT_INTNO intno      : 割込番号 (ID_ALL 可 )
}   T_RSLOG_ISR;
```

•T_RSLOG_SVC [rif_set_log]

```
typedef struct  t_rslog_svc
{
    DT_FN svcfn         : 機能コード (ID_ALL 可 )
    DT_ID objid         : 対象オブジェクト ID
        ( 対象がない SVC の場合は無視 , ID_ALL 可 )
    DT_ID tskid         : タスク ID (ID_ALL 可 )
    BITMASK param       : 取得対象パラメータ (ID_ALL 可 )
}   T_RSLOG_SVC;
```

```
typedef struct  t_rslog_svc
{
    DT_FN svcfn         : 機能コード (ID_ALL 可 )
    DT_ID objid         : 対象オブジェクト ID
        ( 対象がない SVC の場合は無視 , ID_ALL 可 )
    DT_ID tskid         : タスク ID (ID_ALL 可 )
    BITMASK param       : 取得対象パラメータ (ID_ALL 可 )
}   T_RSLOG_SVC;
```

```
typedef struct  t_T_RSLOG_TIMERHDR [rif_set_log]
typedef struct  t_rslog_timerhdr
{
    UINT type           : ハンドラの種別 (OBJ_ALL 可 )
        (rif_ref_obj::objtype で利用する定数 OBJ_xxx を格納する )
        (OBJ_ALL(=ID_ALL) を指定すると全ての種別を対象とする )
    DT_ID hdrid         : ハンドラ ID(ID_ALL 可 )
}   T_RSLOG_TIMERHDR;
```

•T_RSLOG_TSKEXC [rif_set_log]

```
typedef struct  t_rslog_tskexc
{
    DT_ID tskid         : タスク番号 (ID_ALL 可 )
}   T_RSLOG_TSKEXC;
```

•T_RSLOG_TSKSTAT [rif_set_log]

```
typedef struct  t_rslog_tskstat
{
    DT_ID tskid         : タスク番号 (ID_ALL 可 )
}   T_RSLOG_TSKSTAT;
```

•T_RSLOG_USEREVT [rif_set_log]

```
typedef struct    t_rslog_comment
{
    UINT length         : コメント文字列の長さ
}    T_RSLOG_COMMENT;
```

•T_TCFNC[tif_cal_fnc]

```
typedef struct    t_tcfnc_prmary
{
    UINT prmsz          : パラメータのサイズ ( バイト単位 )
    VP prmptr           : パラメータを格納する領域へのポインタ
}    T_TCFNC_PRMARY;

typedef struct    t_tcfnc
{
    DT_VP fncadr        : 関数のアドレス
    DT_VP stkadr        : 関数発行時のスタックポインタ
    UINT retsz          : 結果を格納する領域の大きさ ( バイト単位 )
    VP retptr           : 実行結果を格納する領域へのポインタ
    UINT prmcnt         : パラメータの数
    T_TCFNC_PRMARY prmary[]
                        : パラメータ
}    T_TCFNC;
```

•T_TGLOG [tif_get_log]

```
typedef struct    t_tglog
{
    ID logid            : 対応するログ ID
    DT_VP staadr        : 設定した開始アドレス
    DT_VP endadr        : 設定した終了アドレス
    UINT logtype        : ログ種別情報
    LOGTIM logtim       : タイムスタンプ
    DT_SIZE bufsz       : バッファサイズ
    char buf[]          : 取得した値を格納する領域
}    T_TGLOG;
```

•T_TSBRK [tif_set_brk]

```
typedef struct    t_tsbrk
{
    UINT brktype        : ブレークの種類
    DT_VP brkadr        : ブレークを設定するアドレス
    VP_INT brkprm       : コールバックルーチンの通知フラグ
}    T_TSBRK;
```

•T_TSBRK_CND [tif_set_brk]

```
typedef struct    t_tsbrk_cnd
{
    UINT brktype        : ブレークの種類
    DT_VP brkadr        : プレークを設定するアドレス
    VP_INT brkprm       : コールバックルーチンの通知フラグ
    DT_VP cndadr        : 条件ブレークに設定するアドレス
    VP_INT cndval       : 条件ブレークに設定する値
    UINT cndlen         : 条件ブレークに設定する値のバイト長 (1,2,4)
}    T_TSBRK_CND;
```

•T_TSLOG [tif_set_log]

```
typedef struct    t_tslog
{
    UINT logtype        : ログ種別フラグ
    DT_VP staadr        : 開始アドレス
    DT_VP endadr        : 終了アドレス ( 範囲を指定しない場合は NULL)
    DT_VP valptr        : 読出し開始位置 ( NULL : イベント発生位置 )
    DT_SIZE valsz       : データ長 ( バイト単位 )
}    T_TSLOG;
```

9.2 関数一覧

オブジェクトの状態取りだし [OBJ] ○

```
ER      rif_ref_obj
        ( VP p_result, UINT objtype, DT_ID objid, FLAG flags )
```

ディスクリプションテーブルの取得 [CTX] ○

```
ER      rif_get_rdt ( const T_GRDT ** ppk_pgrdt, FLAG flags)
```

タスクコンテキストの取得 [CTX] ○

```
ER      rif_get_ctx
        ( VP p_ctxblk, BITMASK_8 * p_valid, DT_ID tskid, FLAG flags)
```

タスクコンテキストの設定 [CTX] ○

```
ER      rif_set_ctx
        ( VP p_ctxblk, BITMASK_8 * valid, FLAG flags)
```

サービスコールの発行 [SVC] ○

```
ER      rif_cal_svc ( T_RCSVC * pk_psvc , FLAG flags)
```

発行したサービスコールの取り消し [SVC] ○

```
ER      rif_can_svc ( FLAG flags )
```

サービスコールの終了通知 [SVC:callback] □

```
void    rif_rep_svc ( DT_ER result )
```

機能コードの取得 [SVC] ○

```
ER      rif_ref_svc ( DT_FN * p_svcfn, char * strsvc, FLAG flags)
```

サービスコール名称の取得 [SVC] ○
ER rif_rrf_svc
(char * p_strsvc, UINT bufsz, DT_FN svcfn, FLAG flags)

ブレークポイントの設定要求 [BRK] ○
ER_ID rif_set_brk (ID brkid, T_RSBRK * pk_rsbrk , FLAG flags)

ブレークポイントの解除 [BRK] ○
ER rif_del_brk (ID brkid , FLAG flags)

ブレークヒットの通知 [BRK:callback] □
void rif_rep_brk (ID brkid, VP_INT exinf)

設定したブレーク情報の取得 [BRK] ○
ER rif_ref_brk (ID brkid, T_RSBRK * ppk_rsbrk, FLAG flags)

ブレーク条件の取得 [CND] ○
ER rif_ref_cnd
(T_RRCND_DBG * ppk_dbg, T_RRCND_RTOS * pk_rtos,
FLAG flags)

トレースログの設定 [LOG] ○
ER_ID rif_set_log
(ID logid, UINT logtype, VP pk_rslog , FLAG flags)

トレースログ設定の解除 [LOG] ○
ER rif_del_log (ID logid ,FLAG flags)

トレースログ機能の開始要求 [LOG] ○
ER rif_sta_log (ID logid, FLAG flags)

トレースログ取得の解除要求 [LOG] ○
ER rif_stp_log (ID logid, FLAG flags)

トレースログの取得 [LOG] ○
ER rif_get_log (T_RGLOG * ppk_rglog, FLAG flags)

トレースログ機構の構成変更 [LOG] ○
ER rif_cfg_log (T_RCLOG * pk_rclog, FLAG flags)

カーネルコンフィギュレーションの取得 [R] ○
ER rif_ref_cfg
(T_INFO * p_information, UINT packets, FLAG flags)

メモリ確保 (ホスト上) [R] □
ER tif_alc_mbh (VP * p_blk, UINT blksz, FLAG flags)

メモリ確保 (ターゲット上) [E] □
ER tif_alc_mbt (DT_VP * p_blk, DT_SIZE blksz, FLAG flags)

メモリ解放 (ホスト上) [R] □
ER tif_fre_mbh (VP blk, FLAG flags)

メモリ解放 (ターゲット上) [E] □
ER tif_fre_mbt (DT_VP blk, FLAG flags)

メモリ読み出し [R] □
ER tif_get_mem
(VP p_result, DT_VP memadr, DT_SIZE memsz, FLAG flags)

ブロックセット単位でのメモリ読み出し [R] □
ER tif_get_bls
(VP p_result , T_BLKSET * blkset , FLAG flags)

メモリ書き込み [R] □
ER tif_set_mem
(VP storage , DT_VP memadr, DT_SIZE memsz , FLAG flags)

ブロックセット単位でのメモリ書き込み [R] □
ER tif_set_bls (VP storage, T_BLKSET * blkset , FLAG flags)

メモリ内容変更通知の設定 [E] □
ER_ID tif_set_pol
(ID polid, DT_VP adr, DT_INT value, UINT length, FLAG flags)

変更通知の設定解除 [E] □
ER tif_del_pol (ID polid, FLAG flags)

メモリ内容変更の通知 [E:callback] ○
void tif_rep_pol (ID polid, DT_INT value, FLAG flags)

レジスタ内容の読み出し [R] □
ER tif_get_reg (VP r_result, BITMASK_8 * p_valid, FLAG flags)

レジスタ内容の書き出し [R] □
ER tif_set_reg (VP storage , BITMASK_8 * p_valid, FLAG flags)

ターゲットの実行 [R] □
ER tif_sta_tgt (DT_VP staadr, FLAG flags)

ターゲットの実行中止 [E] □
ER tif_stp_tgt (FLAG flags)

ターゲットの実行中断 [E] □
ER tif_brk_tgt (FLAG flags)

ターゲットの実行再開 [R] □
ER tif_cnt_tgt (FLAG flags)

ブレークポイントの設定 [R] □
ER_ID tif_set_brk (ID brkid, T_TSBRK * pk_tsbrk, FLAG flags)

ブレークポイントの解除 [R] □
ER tif_del_brk (ID brkid, FLAG flags)

ブレーク通知 [R:callback] ○
ER tif_rep_brk (ID brkid, VP_INT param)

シンボルテーブルの値参照 [R] □
ER tif_ref_sym (INT * p_value , char * strsym , FLAG flags)

シンボルテーブルのシンボル参照 [E] □
ER tif_rrf_sym
(char * p_sym , UINT maxlen , INT value , FLAG flags)

関数の発行 [E] □
ER tif_cal_fnc (T_TCFNC * pk_tcfnc, FLAG flags)

関数実行の終了通知 [E:callback] ○
void tif_rep_fnc (FLAG flags)

トレースログの設定 [E] □
ER_ID tif_set_log (ID logid, T_TSLOG * pk_tslog , FLAG flags)

トレースログ設定の解除 [E] □
ER tif_del_log (ID logid, FLAG flags)

トレースログの開始 [E] □
ER tif_sta_log (ID logid, FLAG flags)

トレースログの停止 [E] □
ER tif_stp_log (ID logid, FLAG flags)

トレースログ関連コールバック [E:callback] ○
void tif_rep_log (ID logid, UINT event, FLAG flags)

トレースログソースの取得 [E] □
ER tif_get_log (VP p_result, FLAG flags)

デバッグツール関連情報の取得 [R] □
ER dgb_ref_dbg
(T_INFO * pk_rdbg, UINT packets, FLAG flags)

RIM 初期化 [R] ○
ER dbg_ini_rim(VP param)

RIM の終了処理 [R] ○
ER dbg_fin_rim (VP param)

RIM に関連する情報の取得 [R] ○
ER dbg_ref_rim
(T_INFO * ppk_rrim, UINT packets, FLAG flags)

インターフェースの初期化 [E] ○
ER dbg_ini_inf (T_INTERFACE * ppk_interface, VP param)

9.3 オプションフラグ

9.3.1 共通フラグ

FLG_AUTONUMBERING (40000000h) : ID 自動割当
ID の自動割当を行う．引数に ID を指定した場合，関数はこれを無視する．成功した場合，関数は自動割当された ID を返す．

FLG_NOCONSISTENCE (10000000h) : 非一貫性フラグ
このフラグが指定されたときは，取得した内容に一貫性を持たせる必要はない．例として，「タスク待ち要因がないのにタスク状態は待ちから解除されていない」など．

FLG_NOREPORT (80000000h) : 通知関数の無効化
対になるコールバック関数の呼出しを行わない

FLG_NOSYSTEMSTOP (20000000h) : 明示的なシステム停止を許さない
このフラグが指定された場合，関数内で ***tif_brk_tgt*** を利用してシステムを停止するような操作を行ってはならない．このフラグをサポートしない場合は，***E_NOSPT*** エラーとなる．

9.3.2 固有フラグ

OPT_APPCONTEXT (1)
アプリケーションレベルのコンテキストを対象とする

OPT_BLOCKING (1)
ブロッキングモードで実行する

OPT_CANCEL (0)
発行したサービスコールの影響に関しては考慮しない **(default)**
OPT_CMPVALUE (2)
比較対象となる値を設定する
OPT_CNDBREAK (4)
デバッグツールの条件ブレーク機構を利用する
OPT_EXTPARAM (2)
拡張パラメータ指定
OPT_GETMAXCNT (1)
上界が可変長データ数を下回った場合でも，データ個数だけは完全に追跡，取得する
OPT_NOCNDBREAK (1)
ブレークの設定に条件ブレークを用いてはならない
OPT_NORDT (2)
レジスタセットディスクリプションテーブルを取得しない
OPT_PEEK (1)
トレースログをスプールから削除せずに取り出す
OPT_RESTART (1)
ターゲットを再起動する (引数 ***staadr*** は無視される)
OPT_SEARCH_BACKWARD (2)
後方 (アドレスが減少する側) で，指定された値に最も近いシンボルを探す
OPT_SEARCH_COMPLETELY (0)
完全一致する物だけを対象とする **(default)**
OPT_SEARCH_FORWARD (1)
前方 (アドレスが増加する側) で，指定された値に最も近いシンボルを探す
OPT_UNDO (1)
完全に発行前の状態に戻す
OPT_VENDORDEPEND (2)
実装依存情報を取得する

9.4 定数

9.4.1 オブジェクト識別定数

OBJ_SEMAPHORE (1)
セマフォ
OBJ_EVENTFLAG (2)
イベントフラグ
OBJ_DATAQUEUE (3)
データキュー
OBJ_MAILBOX (4)
メールボックス
OBJ_MUTEX (5)
ミューテックス
OBJ_MESSAGEBUFFER (6)
メッセージバッファ

OBJ_RENDEZVOUSPORT (8)
ランデブポート
OBJ_RENDEZVOUS (9)
ランデブ
OBJ_FMEMPOOL (10)
固定長メモリプール
OBJ_VMEMPOOL (11)
可変長メモリプール
OBJ_TASK (12)
タスク
OBJ_READYQUEUE (14)
レディキュー
OBJ_TIMERQUEUE (15)
タイマーキュー
OBJ_CYCLICHANDLER (17)
周期起動ハンドラ
OBJ_ALARMHANDLER (18)
アラームハンドラ
OBJ_OVERRUNHANDLER (19)
オーバランハンドラ
OBJ_ISR (20)
割込みサービスルーチン
OBJ_KERNELSTATUS (21)
カーネル関連情報
OBJ_TASKEXCEPTION (22)
タスク例外ハンドラ
OBJ_CPUEXCEPTION (23)
CPU 例外ハンドラ
OBJ_ALL (-1u)
全てのオブジェクトを意味する特殊な定数

9.4.2 エラー定数

E_CONSIST (-225)
一貫性を保証できなかった (ただし ***FLG_NOCONSISTENCE*** が設定されていた場合に限ってはエラーとならない)
E_EXCLUSIVE (-226)
すでに要求が発行されており, 終了するまで受け付けることができない
E_FAIL (-227)
何らかの原因により，操作が失敗した (継続して動作は可能である)
E_ID (-146)
指定されたオブジェクト ID は無効
E_NOID (-162)
自動割当用 ID が不足している
E_NOMEM (-161)
ホスト上のメモリ不足のため，要求を実行できない
E_NOSPT (-137)
その操作はサポートしない

E_OBJ (-169)
操作対象は存在しない または 操作できない
E_OK (0)
正常に終了した
E_PAR (-145)
パラメータが不正な値であった
E_SYS (-133)
何らかの原因で復帰不可能な (致命的な) エラーが発生した
E_TMOUT (-178)
処理はタイムアウトした (***OPT_BLOCKING*** 指定時)
ET_ID (-18)
指定されたカーネルオブジェクト ID は無効
ET_MACV (-26)
ターゲット上の不正なメモリ領域に対するアクセスが発生した
ET_NOEXS (-42)
ターゲット上に対象となるオブジェクトが存在しない
ET_NOMEM (-33)
ターゲット上におけるメモリ不足のため，要求を実行できない
ET_OACV (-27)
不正なターゲット上のオブジェクトに対するアクセス (tskid < 0)
ET_OBJ (-41)
ターゲット上の対象オブジェクトは操作できない

9.4.3 ブレーク定数

BRK_ACCESS (2)
アクセスブレークを設定する
BRK_DISPATCH (3)
タスクディスパッチャ (実行後) にブレークを設定する
BRK_ENTER (0)
開始位置にブレークを置く (***BRK_DISPATCH***, ***BRK_SVC***)
BRK_EXECUTE (1)
実行ブレークを設定する
BRK_LEAVE (128)
脱出位置にブレークを置く (***BRK_DISPATCH***, ***BRK_SVC***)
BRK_REPORT (32)
通知のみを行う (実際にブレークは行わない)
BRK_SVC (4)
SVC でブレークする
BRK_SYSTEM (0)
ブレーク時は全体が停止する
BRK_TASK (64)
ブレーク時はタスク単体が停止する

9.4.4 ログ定数

ログ種別 - オブジェクト

LOG_TYP_INTERRUPT (1)
割込
LOG_TYP_ISR (2)
割込サービスルーチン
LOG_TYP_TIMERHDR (3)
タイムイベントハンドラ
LOG_TYP_CPUEXC (4)
CPU 例外
LOG_TYP_TSKEXC (5)
タスク例外
LOG_TYP_TSKSTAT (6)
タスク状態
LOG_TYP_DISPATCH (7)
タスクディスパッチ
LOG_TYP_SVC (8)
サービスコール
LOG_TYP_COMMENT (9)
コメント (文字列のみからなるログ。主としてユーザが記述)

ログ種別 - ブレーク方式

LOG_INSTRUCTION (0)
命令
LOG_DATA (4)
データ

ログ種別 - ブレーク条件

LOG_READ (8)
読み込み
LOG_WRITE (16)
書き込み
LOG_MODIFY (32)
修正 (Read Modify Write)

ログ機構 - 構成 設定

LOG_HARDWARE (0)
TIF を利用したハードウェアログ機構を利用して取得する。
LOG_SOFTWARE (1)
RIM 単体で実行するソフトウェア主体のログ機構を利用して取得する。
LOG_BUFFUL_STOP (0)
バッファ満載時にログ取得を停止する
LOG_BUFFUL_CALLBACK (2)
バッファ満載時にコールバック関数を実行する
LOG_BUFFUL_FORCEEXEC(4)
バッファ満載時に最も古いものを破棄し、取得を継続する

通知イベント

EV_BUFFER_FULL (1)
トレースバッファが満杯になった
EV_STOP (2)
トレースログ機能が停止された
EV_REPORT (4)
tif_sta_log で設定された通知条件を満たした

ディスパッチ種別

DSP_NORMAL (0)
タスクコンテキストからのディスパッチ
DSP_NONTSKCTX (1)
割込み処理および CPU 例外からのディスパッチ

9.4.5 その他の定数

ADR_SYSTEMSTART(0)
ターゲットを再起動させる
CND_CURTSKID(0)
タスク ID を条件とする式を生成する
ID_ALL(-1)
全ての ID を対象にする
ID_NONTSKCTX (-127)
非タスクコンテキストを対象とする

9.5 情報取得用キーコード一覧

第 1 キー **値 [型]**
このキーで取得できる情報の説明
. 第 2 キー **値 [型]**
このキーで取得できる情報の説明
. 第 3 キー **値 [型]**
このキーで取得できる情報の説明
. 第 4 キー **値 [型]**
このキーで取得できる情報の説明

RIF 4h
.UNIT 20h
.OBJ 1h[1]
機能単位 [オブジェクト状態取得] をサポートする
.LOG 2h[1]
機能単位 [実行履歴取得] をサポートする
.SVC 3h[1]
機能単位 [サービスコール発行] をサポートする
.BRK 4h[1]
機能単位 [ブレーク設定] をサポートする
.CND 5h[1]
機能単位 [ブレーク条件取得] をサポートする

.CTX 6h[1]
機能単位 [コンテキスト取得] をサポートする

RIF 4h

.RIF_REF_OBJ 1h

.FLG_NOCONSISTENCE 1h[1]
フラグ ***FLG_NOCONSISTENCE*** は利用可能である

.FLG_NOSYSTEMSTOP 2h[1]
フラグ ***FLG_NOSYSTEMSTOP*** は利用可能である

.OPT_VENDORDEPEND 10h[1]
オプション ***OPT_VENDORDEPEND*** は利用可能である

.OPT_GETMAXCNT 11h[1]
オプション ***OPT_GETMAXCNT*** は利用可能である

.STATICPARAMETER 12h

.OBJ_SEMAPHORE 80h[T]
セマフォ情報のうち静的に決定する情報をもつ構造体

.OBJ_EVENTFLAG 81h[T]
イベントフラグ情報のうち静的に決定する情報をもつ構造体

.OBJ_DATAQUEUE 82h[T]
データキュー情報のうち静的に決定する情報をもつ構造体

.OBJ_MAILBOX 83h[T]
メールボックス情報のうち静的に決定する情報をもつ構造体

.OBJ_MUTEX 84h[T]
ミューテックス情報のうち静的に決定する情報をもつ構造体

.OBJ_MESSAGEBUFFER 85h[T]
メッセージボックス情報のうち静的に決定する情報をもつ構造体

.OBJ_RENDEZVOUSPORT 86h[T]
ランデブポート情報のうち静的に決定する情報をもつ構造体

.OBJ_RENDEZVOUS 87h[T]
ランデブ情報のうち静的に決定する情報をもつ構造体

.OBJ_FMEMPOOL 88h[T]
固定長メモリプール情報のうち静的に決定する情報をもつ構造体

.OBJ_VMEMPOOL 89h[T]
可変長メモリプール情報のうち静的に決定する情報をもつ構造体

.OBJ_TASK 8Ah[T]
タスク情報のうち静的に決定する情報をもつ構造体

.OBJ_READYQUEUE 8Bh[T]
レディーキュー情報のうち静的に決定する情報をもつ構造体

.OBJ_TIMERQUEUE 8Ch[T]
タイマーキュー情報のうち静的に決定する情報をもつ構造体

.OBJ_CYCLICHANDLER 8Dh[T]
周期起動ハンドラ情報のうち静的に決定する情報をもつ構造体

.OBJ_ALARMHANDLER 8Eh[T]
アラームハンドラ情報のうち静的に決定する情報をもつ構造体

.OBJ_OVERRUNHANDLER 8Fh[T]
オーバーランハンドラ情報のうち静的に決定する情報をもつ構造体

.OBJ_ISR 90h[T]
割込みサービスルーチン情報のうち静的に決定する情報をもつ構造体

.OBJ_KERNELSTATUS 91h[T]
カーネル情報のうち静的に決定する情報をもつ構造体

.OBJ_TASKEXCEPTION 92h[T]
タスク例外情報のうち静的に決定する情報をもつ構造体

.OBJ_CPUEXCEPTION 93h[T]
CPU 例外情報のうち静的に決定する情報をもつ構造体

RIF 04h

.RIF_GET_RDT 02h

.REGISTER 2h

.SIZE 04h[W]
レジスタの格納に十分な領域のサイズ (バイト単位)

.CONTEXT 12h

.SIZE 04h[W]
コンテキストの格納に十分な領域のサイズ (バイト単位)

RIF 04h

.RIF_GET_CTX 03h

.FLG_NOCONSISTENCE 01h[1]
フラグ ***FLG_NOCONSISTENCE*** は利用可能である

.FLG_NOSYSTEMSTOP 02h[1]
フラグ ***FLG_NOSYSTEMSTOP*** は利用可能である

.OPT_APPCONTEXT 10h[1]
オプション ***OPT_APPCONTEXT*** は利用可能である

RIF 04h

.RIF_SET_CTX 13h

.FLG_NOSYSTEMSTOP 02h[1]
フラグ ***FLG_NOSYSTEMSTOP*** は利用可能である

.OPT_APPCONTEXT 10h[1]
オプション ***OPT_APPCONTEXT*** は利用可能である

RIF 04h

.RIF_CAL_SVC 04h

.FLG_NOREPORT 03h[1]
フラグ ***FLG_NOREPORT*** 利用可能である

.OPT_BLOCKING 10h[1]
フラグ ***OPT_BLOCKING*** 利用可能である

.NONBLOCKING 12h[1]
ノンブロッキング SVC 発行に対応している

RIF 04h
.RIF_CAN_SVC 05h[1]
rif_can_svc を実装している
.OPT_CANCEL 10h[1]
オプション ***OPT_CANCEL*** は利用可能である
.OPT_UNDO 11h[1]
オプション ***OPT_UNDO*** は利用可能である
RIF 04h
.RIF_CAL_SVC 06h
RIF 04h
.RIF_REF_SVC 07h
RIF 04h
.RIF_RRF_SVC 08h
RIF 04h
.RIF_SET_BRK 09h
.FLG_NOREPORT 03h[1]
フラグ ***FLG_NOREPORT*** は利用可能である
.FLG_AUTONUMBERING 04h[1]
フラグ ***FLG_AUTONUMBERING*** は利用可能である
.OPT_NOCNDBREAK 10h[1]
オプション ***OPT_NOCNDBREAK*** は利用可能である
.OPT_EXTPARAM 11h[1]
オプション ***OPT_EXTPARAM*** を利用できる
RIF 04h
.RIF_DEL_BRK 0Ah
RIF 04h
.RIF_REP_BRK 0Bh
RIF 04h
.RIF_REF_BRK 0Ch
RIF 04h
.RIF_REF_CND 0Dh
RIF 04h
.RIF_SET_LOG 0Eh
.FLG_AUTONUMBERING 04h[1]
フラグ FLG_AUTONUMBERING は利用可能である
.OPT_BUFFUL_STOP 10h[1]
オプション OPT_BUFFUL_STOP は利用可能である
.OPT_BUFFUL_FORCEEXEC 11h[1]
オプション OPT_BUFFUL_FORCEEXEC は利用可能である

RIF 04h
.RIF_DEL_LOG 0Fh
RIF 04h
.RIF_STA_LOG 10h
RIF 04h
.RIF_STP_LOG 11h
RIF 04h
.RIF_GET_LOG 12h
.OPT_PEEK 10h[1]
オプション ***OPT_PEEK*** は利用可能である
.STRUCT_SVC 11h[1]
LOG_TYP_SVC の開始 / 終了に対してそれぞれ専用の構造体を利用する
RIF 04h
.RIF_CFG_LOG 13h
CFG 7h
.CPUEXCEPTION 17h
.MIN 1h[W]
カーネルが利用する内部例外要因のうち最小の数
.MAX 2h[W]
カーネルが利用する内部例外要因のうち最大の数
.NUM 3h[W]
カーネルが利用する内部例外要因の数
.SYSTIM 20h
.TICK_D 1h[W]
タイマ分解能を千分の一秒 (ms) で表現したときの分母
.TICK_N 2h[W]
タイマ分解能を千分の一秒 (ms) で表現したときの分子
.UNIT_D 3h[W]
タイマの単位を千分の一秒 (ms) で表現したときの分母
.UNIT_N 4h[W]
タイマの単位を千分の一秒 (ms) で表現したときの分子
.LOGTIM 21h
.TICK_D 1h[W]
ログ時刻分解能を千分の一秒 (ms) で表現したときの分母
.TICK_N 2h[W]
ログ時刻分解能を千分の一秒 (ms) で表現したときの分子
.UNIT_D 3h[W]
ログ時刻の単位を千分の一秒 (ms) で表現したときの分母
.UNIT_N 4h[W]
ログ時刻の単位を千分の一秒 (ms) で表現したときの分子
.INTERRUPT 22h
.MIN 1h[W]
カーネルが利用する外部割込み要因のうち最小の数

.MAX 2h[W]
カーネルが利用する外部割込み要因のうち最大の数
.NUM 3h[W]
カーネルが利用する外部割込み要因の数
.ISR 25h
.MIN 1h[W]
カーネルが提供する ISR のうち最小の番号
.MAX 2h[W]
カーネルが提供する ISR のうち最大の番号
.NUM 3h[W]
カーネルが提供する ISR の数
.MAKER 23h[W]
メーカーコード
.PRIORITY 24h
.MIN 1h[W]
カーネルで利用できる優先度のうち最小の数
.MAX 2h[W]
カーネルで利用できる優先度のうち最大の数
.OBJ_SEMAPHORE 80h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_EVENTFLAG 81h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_DATAQUEUE 82h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_MAILBOX 83h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_MUTEX 84h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数

.OBJ_MESSAGEBUFFER 85h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_RENDEZVOUSPORT 86h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_RENDEZVOUS 87h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_FMEMPOOL 88h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_VMEMPOOL 89h
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_TASK 8Ah
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_CYCLICHANDLER 8Dh
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.OBJ_ALARMHANDLER 8Eh
.MIN 1h[W]
割当て可能な ID のうち最小の数
.MAX 2h[W]
割当て可能な ID のうち最大の数
.PRVER A0h[S]
カーネル自身のバージョン番号
.SPVER A1h[S]
ITRON 仕様のバージョン番号

TIF 05h
.TIF_ALC_MBH 01h
TIF 05h
.TIF_ALC_MBT 02h[1]
この関数をサポートする
TIF 05h
.TIF_FRE_MBH 03h
TIF 05h
.TIF_FRE_MBT 04h[1]
この関数をサポートする
TIF 05h
.TIF_GET_MEM 05h
.FLG_NOCONSISTENCE 01h[1]
フラグ ***FLG_NOCONSISTENCE*** をサポートする
.FLG_NOSYSTEMSTOP 02h[1]
フラグ ***FLG_NOSYSTEMSTOP*** をサポートする
TIF 05h
.TIF_GET_BLS 06h
.FLG_NOCONSISTENCE 01h[1]
フラグ ***FLG_NOCONSISTENCE*** をサポートする
.FLG_NOSYSTEMSTOP 02h[1]
フラグ ***FLG_NOSYSTEMSTOP*** をサポートする
TIF 05h
.TIF_SET_MEM 07h
.FLG_NOCONSISTENCE 01h[1]
フラグ ***FLG_NOCONSISTENCE*** をサポートする
.FLG_NOSYSTEMSTOP 02h[1]
フラグ ***FLG_NOSYSTEMSTOP*** をサポートする
TIF 05h
.TIF_SET_BLS 08h
.FLG_NOCONSISTENCE 01h[1]
フラグ ***FLG_NOCONSISTENCE*** をサポートする
.FLG_NOSYSTEMSTOP 02h[1]
フラグ ***FLG_NOSYSTEMSTOP*** をサポートする
TIF 05h
.TIF_SET_POL 09h[1]
この関数をサポートする
.FLG_AUTONUMBERING 04h[1]
フラグ ***FLG_AUTONUMBERING*** をサポートする
.OPT_CMPVALUE 10h[1]
オプション ***OPT_CMPVALUE*** をサポートする

TIF 05h
.TIF_DEL_POL 0Ah[1]
この関数をサポートする
TIF 05h
.TIF_REP_POL 0Bh
TIF 05h
.TIF_GET_REG 0Ch
.FLG_NOCONSISTENCE 01h[1]
フラグ ***FLG_NOCONSISTENC*** をサポートする
.FLG_NOSYSTEMSTOP 02h[1]
フラグ ***FLG_NOSYSTEMSTOP*** をサポートする
TIF 05h
.TIF_SET_REG 0Dh
TIF 05h
.TIF_STA_TGT 0Eh
.OPT_RESTART 10h[B]
OPT_RESTART は利用可能である
TIF 05h
.TIF_STP_TGT 0Fh[1]
この関数をサポートする
TIF 05h
.TIF_BRK_TGT 10h[1]
この関数をサポートする
TIF 05h
.TIF_CNT_TGT 11h
TIF 05h
.TIF_SET_BRK 13h
.FLG_AUTONUMBERING 04h[1]
フラグ ***FLG_AUTONUMBERING*** をサポートする
.OPT_CNDBREAK 10h[1]
オプション ***OPT_CNDBREAK*** をサポートする
.BRK_ACCESS 11h[1]
アクセスブレークが利用可能である
TIF 05h
.TIF_DEL_BRK 14h
TIF 05h
.TIF_REP_BRK 12h
この関数をサポートする
.FLG_AUTONUMBERING 04h[1]
フラグ ***FLG_AUTONUMBERING*** をサポートする

TIF 05h
.TIF_REF_SYM 15h
TIF 05h
.TIF_RRF_SYM 16h[1]
この関数をサポートする
.OPT_SEARCH_FORWARD 10h[1]
オプション ***OPT_SEARCH_FORWARD*** は利用可能である
.OPT_SEARCH_BACKWARD 11h[1]
オプション ***OPT_SEARCH_BACKWARD*** は利用可能である
.OPT_SEARCH_COMPLETELY 12h[1]
オプション ***OPT_SEARCH_COMPLETELY*** は利用可能である
TIF 05h
.TIF_CAL_FNC 17h[1]
この関数をサポートする
.FLG_NOREPORT 03h[1]
フラグ ***FLG_AUTONUMBERING*** をサポートする
.OPT_BLOCKING 11h[1]
オプション ***OPT_NONBLOCKING*** をサポートする
.NONBLOCKING 12h[1]
ノンブロッキング関数呼び出しに対応している
TIF 05h
.TIF_REP_FNC 18h[1]
この関数をサポートする
TIF 05h
.TIF_SET_LOG 19h[1]
この関数をサポートする
.FLG_NOREPORT 03h[1]
フラグ ***FLG_NOREPORT*** は利用可能である
.FLG_AUTONUMBERING 04h[1]
フラグ ***FLG_AUTONUMBERING*** をサポートする
.OPT_BUFFUL_FORCEEXEC 11h[1]
オプション ***OPT_BUFFUL_FORCEEXEC*** は利用可能である
.OPT_BUFFUL_CALLBACK 12h[1]
オプション ***OPT_BUFFUL_CALLBACK*** は利用可能である
.LOG_INSTRUCTION 13h[1]
ログ種別 ***LOG_INSTRUCTION*** は利用可能である
.LOG_DATA 14h[1]
ログ種別 ***LOG_DATA*** は利用可能である
.LOG_READ 15h[1]
LOG_READ は利用可能である
.LOG_WRITE 16h[1]
LOG_WRITE は利用可能である
.LOG_MODIFY 17h[1]
LOG_MODIFY は利用可能である

TIF 05h
.TIF_DEL_LOG 1Ah[1]
この関数をサポートする
TIF 05h
.TIF_STA_LOG 1Bh[1]
この関数をサポートする
TIF 05h
.TIF_STP_LOG 1Ch[1]
この関数をサポートする
TIF 05h
.TIF_REP_LOG 1Dh[1]
この関数をサポートする
TIF 05h
.TIF_GET_LOG 1Eh[1]
この関数をサポートする
.OPT_PEEK 10h[1]
オプション ***OPT_PEEK*** をサポートする
DEBUGGER 1h
.CNDBREAK 1h
.NUM 3h[W]
設定可能な条件ブレークの数 (0:Not supported)
.LOG 2h
.NUM 3h[W]
設定可能なハードウェアログの数 (0:Not supported)
.NAME 80h[S]
デバッグツールを特定する一意な文字
HOST 2h
.ENDIAN 1h[W]
ホストコンピュータのエンディアン (0:Little 1:Big)
.NAME 80h[S]
ホストコンピュータを特定する一意な文字
TARGET 3h
.ENDIAN 1h[W]
ターゲットコンピュータのエンディアン (0:Little 1:Big)
.REGISTER 2h
.NUM 3h[W]
ターゲットコンピュータのレジスタの数
.NAME 80h[S]
ターゲット装置を識別する一意な文字
OS 8h
.NAME 80h[S]
ターゲット OS を識別する一意な文字 ("ITRON")

A. その他

A.1 メンバーリスト

仕様策定にあたりご助力いただいた方々に敬意を表し，ここに ITRON デバッギングインターフェースワーキンググループメンバーの名前を記す．(敬称略 , 名前順)

Table 26: **メンバーリスト**

所属	名前
日本電気マイコンテクノロジー株式会社	安部 孔栄
緑屋電気	庵 一行
NEC ソフトウエアデザイン研究所	伊賀 徳寿
株式会社 ワイ・ディ・シー	石井 秀弘
富士通デバイス株式会社	稲光 一豊
エルグ株式会社	岩田 茂人
松下電器産業株式会社 半導体開発本部	内田 和之
株式会社 松下電器情報システム広島研究所	衛藤 眞一郎
日本電気マイコンテクノロジー株式会社	金子 善則
株式会社エーアイコーポレーション	河合 孝夫
沖電気工業株式会社	川上 雅弘
三菱電機セミコンダクタシステム株式会社	岸谷 素子
富士通デバイス株式会社	工藤 健治
株式会社ソフィアシステムズ	黒田 久也
株式会社 東芝 青梅工場 COS 開発センタ	小泉 義行
株式会社アドバンスド データ コントロールズ	行田 雅彦
富士通デバイス株式会社	小島 史郎
富士通 株式会社	小林 康浩
エルグ 株式会社	権藤 正樹
富士通デバイス株式会社	桜庭 正明
トヨタ自動車株式会社 第 2 電子技術部	佐々木 茂
日本電気マイコンテクノロジー株式会社	佐藤 貴子
ファームウェアシステム株式会社	柴田 信次

Table 26: **メンバーリスト**

所属	名前
三菱電機マイコン機器ソフトウエア株式会社	宿口 雅弘
株式会社デンソークリエイト	高木 哲郎
豊橋技術科学大学	高田 広章
株式会社 ワイ・ディ・シー	武井 千春
社団法人 トロン協会	竹内 透
テスコ 株式会社	塚田 雄一
松下電器産業株式会社 半導体開発本部	永田 昭二
松下電器産業株式会社 半導体開発本部	永峰 聡
三菱電機株式会社	南角 茂樹
ビットラン株式会社	野本 幸雄
株式会社 アクセス	橋本 真一
富士通デバイス株式会社	長谷川 泰嗣
株式会社東芝 COS 開発センター	早貸 真一
株式会社 松下電器情報システム広島研究所	正木 忠勝
沖電気工業株式会社 LSI 事業部	水越 幸弘
緑屋電気	緑川 聡
三菱電機セミコンダクタシステム 株式会社	村木 宏行
富士通デバイス株式会社	本木 潔
株式会社 ワイ・ディ・シー	森本 登志子
株式会社 日立製作所	山田 真二郎
レッドハット株式会社	山中 勝
モトローラ 株式会社	山田 達雄
株式会社 ライトウェル	山本 一郎
株式会社 東芝 システム LSI 技術研究所	横澤 彰
三菱電機セミコンダクタシステム株式会社	吉田 宗宏
エルグ株式会社	吉村 美代子
豊橋技術科学大学	若林 隆行

B. 索引

Numerics

A

B

C

D

E

F

H

I

P

R

S

T

U

V

W

あ