RTA-TRACE

コンフィギュレーションガイド

著作権について

Document TD00008-002

目次

1 本書について 5
- **1.1** 本書の対象ユーザー 5
- **1.2** 表記上の規約 5

2 RTA-OSEK のコンフィギュレーション 6
- **2.1** 一般的なコンフィギュレーション（'Configuration' ペーン） 6
- **2.2** トレースポイント（'Tracepoints' ペーン） 9
- **2.3** タスクトレースポイント（'Task Tracepoints' ペーン） 9
- **2.4** インターバル（'Intervals' ペーン） 9
- **2.5** カテゴリ（'Categories' ペーン） 9
- **2.6** 列挙（'Enumerations' ペーン） 10
- **2.7** フィルタ（'Filter' ペーン） 10
- **2.8** osTraceStopwatch 11

3 ERCOS^{EK} コンフィギュレーションファイル 12
- **3.1** ターゲットのコンフィギュレーション 12
  - **3.1.1** BUFFER_SIZE 12
  - **3.1.2** TIME_SIZE 12
  - **3.1.3** COMPACT 12
- **3.2** トレースのコンフィギュレーション 12
  - **3.2.1** TASKS_AND_ISRS 13
  - **3.2.2** EXCLUDE_TASK_OR_ISR 13
  - **3.2.3** PROCESSES 13
  - **3.2.4** STARTUP_AND_SHUTDOWN 13
  - **3.2.5** ACTIVATIONS 14

3.2.6 RESOURCES 14
3.2.7 INTERRUPT_LOCKS 14
3.2.8 ERRORS 14
3.2.9 EXPLICIT_STATE_MESSAGES 14
3.2.10 IMPLICIT_STATE_MESSAGES 14
3.2.11 OSEK_MESSAGES 15
3.2.12 MESSAGE_DATA 15
3.2.13 ALARMS 15
3.2.14 TIMETABLES 15
3.2.15 SWITCHING_OVERHEADS 15
3.2.16 TRACEPOINTS 16
3.2.17 TASK_TRACEPOINTS 16
3.2.18 INTERVALS 16
3.2.19 STACK 16
3.2.20 CATEGORY 16
3.3 トレース表示の調整 17
3.3.1 MESSAGE 17
3.3.2 TRACEPOINT 17
3.3.3 TASK_TRACEPOINT 17
3.3.4 INTERVAL 17
3.3.5 COUNTER 18
3.3.6 ENUM 18

4 フォーマット文字列（'Format Strings'） 19
4.1 フォーマット規則 19
4.2 フォーマットの例 20

5 お問い合わせ先 21

索引 23

1 本書について

RTA-TRACE は組み込みシステム用のソフトウェアロジックアナライザです。アプリケーションと組み合わせて使用することにより、システムのデバッグやテストに役立つさまざまなサービスを利用できます。中でも特に優れた機能として、量産用にビルドされたアプリケーションソフトウェアについて、ランタイムにシステム内で起こっている事象を正確に把握することができます。

本書では、RTA-OSEK と ERCOS^{EK} に固有の RTA-TRACE コンフィギュレーションオプションについて説明します。

1.1 本書の対象ユーザー

本書は、RTA-OSEK または ERCOS^{EK} 環境下で RTA-TRACE を使用してアプリケーションを検証しようとするソフトウェア技術者を対象としています。RTA-TRACE の API（C 関数）については『RTA-TRACE ユーザーズガイド』で説明しているので、そちらも併せてお読みください。

1.2 表記上の規約

重要：このように表記されている注記には、ユーザーが知っておく必要のある重要な情報が記載されています。内容をよく読み、記載されているすべての指示に必ず従ってください。

移植性：このように表記されている注記では、RTA-OSEK コンポーネントが実行されるプロセッサ上で実行できるコードを作成する場合に知っておく必要がある事柄について説明されています。

本書では、プログラムコード、ヘッダファイル名、C のデータ型名、C 関数および API 関数名はすべてクーリエ体（`courier`）で表記されています。オブジェクトの名前も、プログラマに公開され次第やはりクーリエ体で表記されます。たとえば、Task1 という名前のタスクは、`Task1` という名前のタスクハンドルとして表記されます。

GUI エレメントとのインタラクションについての記述では、エレメントのキャプションは**ボールド体（bold）**で表記されています。また、メニューなどの階層的なナビゲーションは矢印でレベルを区切り、たとえば、「メニューコマンド **Edit → Select All** を選択します。」、または「メニューから **Edit → Select All** を選択します。」のように表記されています。

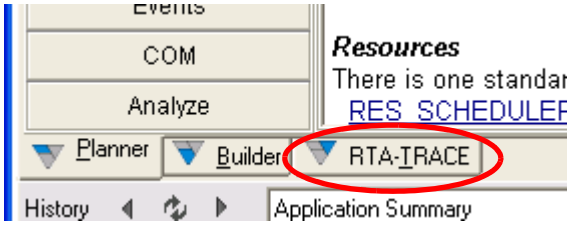
また PDF 文書において、索引、および他の部分を参照する個所（例：「第 3 章を参照してください」の部分）については、その参照先へのリンクが設けられているので、必要な参照個所を素早く見つけることができます。

2 RTA-OSEK のコンフィギュレーション

RTA-OSEK には RTA-TRACE 用オプションパラメータがあり、これらを RTA-OSEK GUI で設定することができます。GUI の操作方法は非常にわかりやすくできているので、ここでは、各パラメータの内容と簡単な設定方法のみ紹介します。

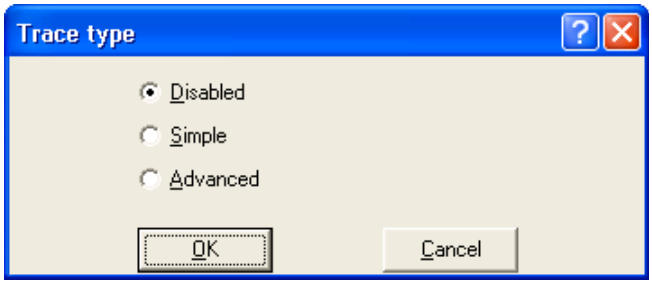
なお本章の内容は、RTA-OSEK GUI の操作方法をすでに熟知しているユーザーを対象としていますので、アプリケーションの作成やコンフィギュレーション設定についてはここでは説明されていません。

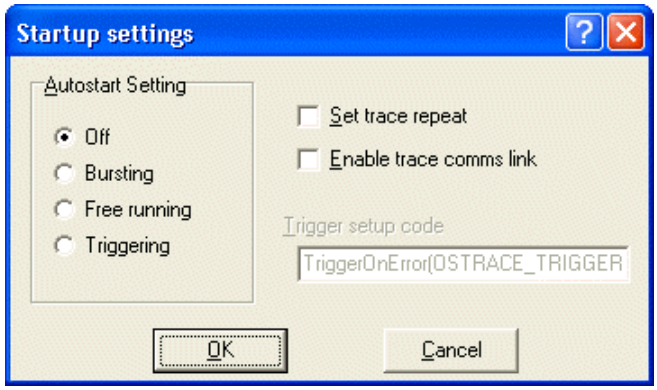
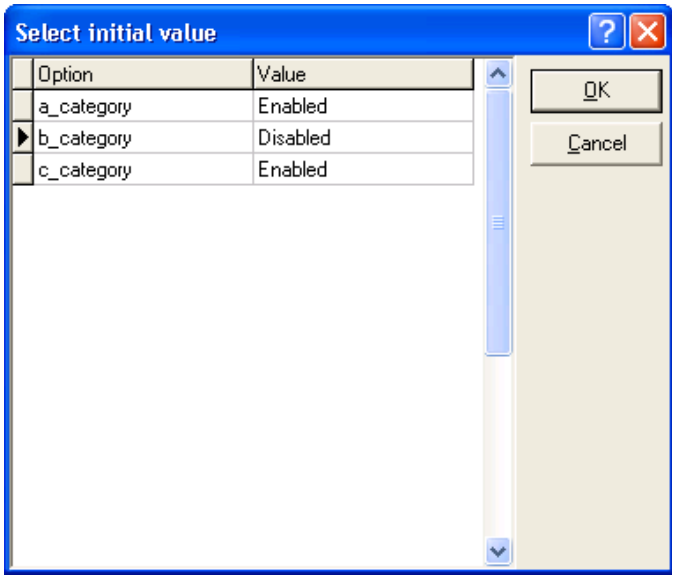
RTA-TRACE 用オプションは、RTA-OSEK GUI ウィンドウの左下にある **RTA-TRACE** タブからアクセスします。

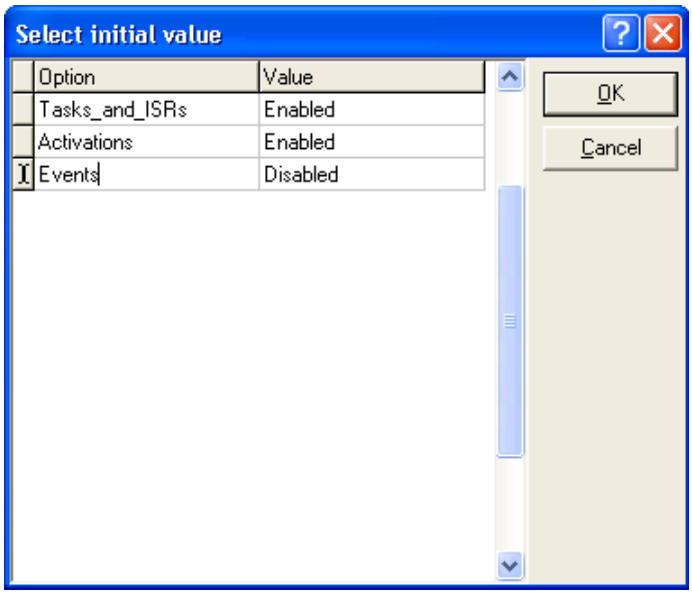
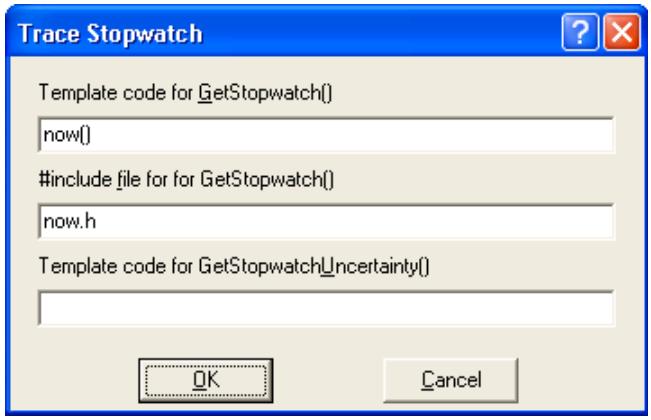


2.1 一般的なコンフィギュレーション（'Configuration' ペーン）

このペーンでは以下のオプションを設定できます。

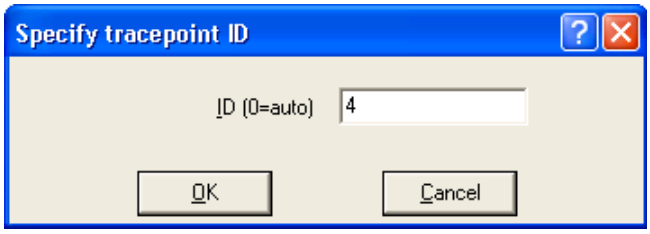
オプション	説明
Trace Type	トレース処理を無効／有効（シンプルまたはアドバンスド）にします。 アドバンスドトレースはシンプルトレースよりも詳細なトレースを行うため、トレースレコードの数が増加します。 
Compact IDs	コンパクトトレースフォーマットを有効にします。バッファスペースの削減のため、タスクトレースポイント ID は 4 ビット、トレースポイントおよびインターバル ID は 8 ビットになります。その他の ID（タスク、リソースなど）は 8 ビットです。 コンパクト ID が指定されない場合は、トレースポイント、タスクトレースポイント、インターバルの各 ID は 12 ビット、その他の ID は 16 ビットとなります。
Compact Time	*compact*（16 ビット）、*extended*（32 ビット）の時間フォーマットを選択します。 このオプションは一部のターゲットには存在しません。
Trace Stack	スタック使用量を記録するかどうかを指定します。
Target Triggering	ランタイムターゲットトリガを使用するかどうかを指定します。

オプション	説明
Buffer Size	ターゲット上に予約されるトレース情報用バッファのサイズを指定します。ここには、バイト数ではなくレコード数を入力します。実際のバッファサイズは、時間と ID のサイズに応じて変わります。 
Autostart	トレースを自動的に開始するかどうか、および開始時のトレースモードを指定します。 トリガを選択した場合は、トリガセットアップコード（`TriggerOn...`）を入力します。トリガ API についての詳細な情報は、『RTA-TRACE ユーザーズガイド』を参照してください。 
Initial Categories	ランタイムカテゴリ（『RTA-TRACE ユーザーズガイド』を参照してください）が定義されている場合、このダイアログボックスで、トレース開始時にどのユーザー定義のランタイムカテゴリが有効になるようにするかを指定できます。以下の図では、3 つのランタイムカテゴリがユーザー定義されていて、そのうちの 1 つが初期状態においては無効になるように設定されています。 

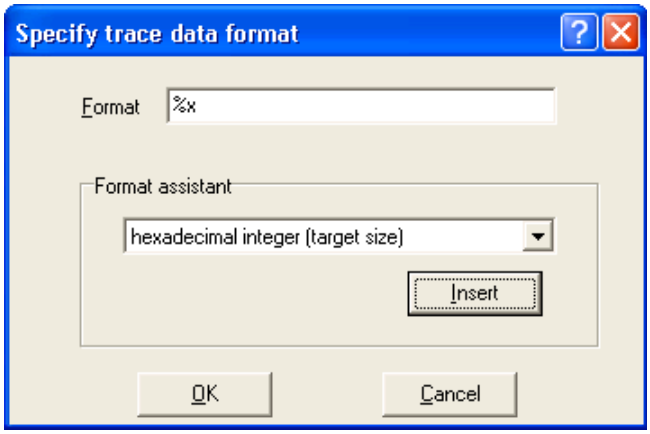
オプション	説明
Initial Classes	トレース開始時にどのレコードクラスが有効になるようにするかを指定します。下の図では、タスクと ISR、起動、イベントトレースの 3 つをランタイムに有効／無効にすることができ、ここでは初期状態においてイベントトレースが無効となるように指定されています。 
Stopwatch	このダイアログボックスで、`GetStopwatch()` を実装する関数を指定します。以下の例では、ユーザー定義の `now()` という関数が指定されていて、この関数に対応するヘッダファイルは `now.h` となっています。 

2.2 トレースポイント（'Tracepoints' ペーン）

このペーンでトレースポイントを設定します。作成された新しいトレースポイントには自動的に ID が割り当てられますが、**ID** ボタンをクリックするとこれを任意に変更できます。



トレースポイントにデータが定義されている場合、フォーマット文字列（第 4 章を参照してください）を使用してデータの出力形式を設定することができます。



2.3 タスクトレースポイント（'Task Tracepoints' ペーン）

このペーンでは、タスクトレースポイントを設定できます。作成された新しいタスクトレースポイントには自動的に ID が割り当てられますが、**ID** ボタンをクリックするとこれを任意に変更できます。

トレースポイントと同様に、フォーマット文字列を使用できます。

2.4 インターバル（'Intervals' ペーン）

このペーンでインターバルを定義できます。作成された新しいインターバルには自動的に ID が割り当てられますが、**ID** ボタンをクリックするとこれを任意に変更できます。

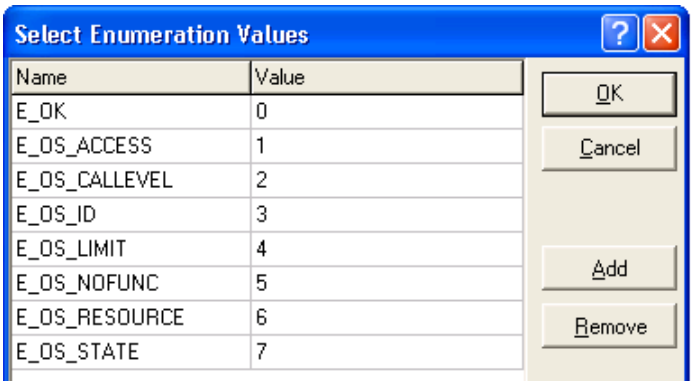
トレースポイントと同様に、フォーマット文字列を使用できます。

2.5 カテゴリ（'Categories' ペーン）

このペーンでトレースカテゴリとそのマスク値を定義できます。カテゴリについての詳細は、『RTA-TRACE ユーザーズガイド』に説明されています。カテゴリは、**filter** ペーンで、ランタイムにおいて「常に有効」、「常に無効」、または「有効／無効」のいずれかに設定できます。

2.6 列挙（'Enumerations' ペーン）

このペーンで列挙型 ID とその列挙子を定義できます。以下の例には、OSEK のエラーコードを表わす列挙子が表示されています。



2.7 フィルタ （'Filter' ペーン）

このペーンでイベントクラスとカテゴリのフィルタ設定を行えます。ランタイムにおけるイベントを、「常に有効」、「常に無効」、または「有効／無効」のいずれかに設定できます。ランタイムクラスの初期状態は、Configuration ペーン（2.1 項を参照してください）の **Initial Classes** オプションで有効／無効を切替えることができます。

2.8 osTraceStopwatch

RTA-TRACE には、現在のシステムタイムを返す関数が必要です。この関数のプロトタイプは以下のとおりでなければなりません。

```
OS_NONREENTRANT(StopwatchTickType) osTraceStopwatch(void);
```

GetStopwatch を定義すると、osTraceStopwatch と同じタイマハードウェアが使用されます。ターゲットの種類によっては 1 命令でタイマを読み取ることのできないものがあり、そのようなターゲットの場合、以下のように OS_ATOMIC() マクロを使用することにより、タイマ読み込み時に GetStopwatch による割込みが絶対に発生しないようにする必要があります。

```
OS_NONREENTRANT(StopwatchTickType)
osTraceStopwatch(void)
{
      /* GET_TIMER_VAL() is a user-defined
       * macro that reads the appropriate
       * timer hardware */

      return GET_TIMER_VAL();
}

…

OS_NONREENTRANT(StopwatchTickType)
GetStopwatch(void)
{
      StopwatchTickType temp;

      /* GET_TIMER_VAL() is a user-defined
       * macro that reads the appropriate
       * timer hardware */

      OS_ATOMIC(temp = GET_TIMER_VAL());
      return temp;
}
```

注記

ストップウォッチダイアログボックス（2.1 項を参照してください）で GetStopwatch() が定義されている場合には、osTraceStopwatch() が自動的に定義されます。

3 ERCOS^{EK} コンフィギュレーションファイル

ERCOS^{EK} 環境下で RTA-TRACE の設定を行うには、`RTAtrace.cfg` というファイル（`project_settings.mk` と同じディレクトリにあります）を使用します。このファイルには、トレースサブシステムを制御する指示語が含まれており、指示語は 1 行につき 1 つを記述できます。コメント行の冒頭には '#' を付けます。つまり各行の最初の '#' より後ろはコメントと見なされます。

どの指示語も任意に使用でき、必須のものはありません。

3.1 ターゲットのコンフィギュレーション

3.1.1 BUFFER_SIZE

用法 `BUFFER_SIZE = size`

デフォルト `200`

説明 ターゲット上に確保されるトレースバッファのサイズを調整します。ここで設定する数値はバイト数ではなくレコード数なので、実際のバッファのバイト数はシステムタイムと識別子のサイズ設定により決まります（3.1.2 項および 3.1.3 項を参照してください）。

有効な値 `32` ～ `65535`

3.1.2 TIME_SIZE

用法 `TIME_SIZE = size`

デフォルト `32`

説明 タイムスタンプの記録に必要なビット数を定義します。

有効な値 `16` または `32`

3.1.3 COMPACT

用法 `COMPACT = TRUE or FALSE`

デフォルト `TRUE`

説明 `COMPACT` トレースフォーマットを使うと、タスクトレースポイント `ID` は `4` ビット、トレースポイント `ID` とインターバル `ID` はそれぞれ `8` ビットで表されるようになり、バッファスペースを節約できます。その他の `ID`（タスク、リソースなど）は `8` ビットです。
上記のビット数では収まりきらない大きな値を `ID` に使用する必要がある場合には、`COMPACT=FALSE` と設定します。これにより、トレースポイント、タスクトレースポイント、およびインターバルの `ID` はそれぞれ `12` ビットで表され、その他の `ID` は `16` ビットで表されるようになります。

有効な値 `TRUE` または `FALSE`

3.2 トレースのコンフィギュレーション

ここで紹介する指示語はトレース用のオブジェクト、またはオブジェクトのクラスを選択するものです。これらを利用することにより、余計なデータをトレース対象から除外し、分析の目的に合った詳細レベルのトレースデータが得られるようにしてください。

以降に示す各指示語の表には、指示語の値が次のような規則に従って記述されています。

true-false	TRUE または FALSE
true-false-runtime	TRUE または FALSE または RUNTIME RUNTIME を設定すると、クラスの有効と無効の切り替えを、コード内で EnableTraceClass() と DisableTraceClass() という API 関数を使用して行えるようになります。
Maskbit	2 の整数乗、つまり 32 ビットバイナリで表現された値の中の 1 ビットだけがセットされた値です。
Identifier	ある特定のオブジェクト（タスクなど）を識別するための C の識別子です。

注記

RUNTIME としてマークされたクラスの初期状態は、「無効」になります。

3.2.1 TASKS_AND_ISRS

用法： TASKS_AND_ISRS = *true-false-runtime*

デフォルト： TRUE

説明： 3.2.2 項の EXCLUDE_TASK_OR_ISR の対象となっているタスクと ISR の、実行状態に入った時と実行状態から抜けた時のロギングを有効にします。

3.2.2 EXCLUDE_TASK_OR_ISR

用法： EXCLUDE_TASK_OR_ISR = *identifier*

デフォルト： この指示語にはデフォルト値はありません。

説明： この指示語は *name* というタスクまたは ISR の開始、終了、および処理情報のロギングを行わないようにします。
ただし、この指示語により除外されたオブジェクトについても、起動イベントだけはロギングされます。

3.2.3 PROCESSES

用法： PROCESSES = *true-false-runtime*

デフォルト： FALSE

説明： ERCOS^{EK} プロセスオブジェクトのロギングを有効にします。プロセスについて完全なロギングを行うためには、SWITCHING_OVERHEADS（3.2.15 項を参照してください）も有効にしておく必要があります。これが有効になっていないと、プロセスの開始しかロギングされません。

3.2.4 STARTUP_AND_SHUTDOWN

用法： STARTUP_AND_SHUTDOWN = *true-false-runtime*

デフォルト： FALSE

説明： ERCOS^{EK} の StartOS() と ShutdownOS() のロギングを有効にします。

3.2.5 ACTIVATIONS

用法 : `ACTIVATIONS = true-false-runtime`

デフォルト : `FALSE`

説明 : タスクを起動しようとする処理を、その処理の成功または不成功に関わらずロギングします。

3.2.6 RESOURCES

用法 : `RESOURCES = true-false-runtime`

デフォルト : `FALSE`

説明 : リソースのロックとアンロックのロギングを有効にします。

3.2.7 INTERRUPT_LOCKS

用法 : `INTERRUPT_LOCKS = true-false-runtime`

デフォルト : `FALSE`

説明 : `OSEK API` を用いて割込みをイネーブル／ディセーブルにしようとする処理のロギングを有効にします。

3.2.8 ERRORS

用法 : `ERRORS = true-false-runtime`

デフォルト : `TRUE`

説明 : オペレーティングシステムのエラー状態のロギングを有効にします。

3.2.9 EXPLICIT_STATE_MESSAGES

用法 : `EXPLICIT_STATE_MESSAGES = true-false-runtime`

デフォルト : `FALSE`

説明 : `ERCOS`^{EK} の明示的なステートメッセージのロギングを有効にします。次の `IMPLICIT_STATE_MESSAGES` を参照してください。

3.2.10 IMPLICIT_STATE_MESSAGES

用法 : `IMPLICIT_STATE_MESSAGES = true-false`

デフォルト : `FALSE`

説明 : `ERCOS`^{EK} の暗黙的なステートメッセージのロギングを有効にします。`ERCOS`^{EK} メッセージングシステムの詳細については『`ESCAPE` リファレンスガイド』を参照してください。

3.2.11 OSEK_MESSAGES

用法： `OSEK_MESSAGES = true-false-runtime`

デフォルト： `FALSE`

説明： `OSEK COM MESSAGE` オブジェクトのロギングを有効にします。

注記

ERCOS^{EK} 4.3 以降に適用されます。

3.2.12 MESSAGE_DATA

用法： `MESSAGE_DATA = true-false-runtime`

デフォルト： `FALSE`

説明： ERCOS^{EK} ステートメッセージの内容をレポートします。

3.2.13 ALARMS

用法： `ALARMS = true-false-runtime`

デフォルト： `FALSE`

説明： `OSEK` アラームのロギングを有効にします。

3.2.14 TIMETABLES

用法： `TIMETABLES = true-false-runtime`

デフォルト： `FALSE`

説明： ERCOS^{EK} タイムテーブルのロギングを有効にします。

3.2.15 SWITCHING_OVERHEADS

用法： `SWITCHING_OVERHEADS = true-false-runtime`

デフォルト： `FALSE`

説明： タスクとプロセスのロギングレベルをより詳細にし、スイッチングオーバーヘッドもロギングされるようにします。スイッチングオーバーヘッドとは、`SystemISR`（これはタイマです。ターゲットによっては複数のシステム `ISR` がある場合もあります）、プリエンプティブスケジューリングのオーバーヘッド、およびプロセス間通信にかかる時間などです。

注記

`EXCLUDE_TASK_OR_ISR` をこの指示語と一緒に使用すると、オーバーヘッドの一部が正しく記録されなくなる場合があります。たとえば、`EXCLUDE_TASK_OR_ISR` で除外されているタスクの終了時オーバーヘッドは記録されません。

3.2.16 TRACEPOINTS

用法： `TRACEPOINTS = true-false-runtime`

デフォルト： `TRUE`

説明： API 関数 `LogTracePoint…()` のコールにより行われるトレースポイントのロギングを有効にします。

3.2.17 TASK_TRACEPOINTS

用法： `TASK_TRACEPOINTS = true-false-runtime`

デフォルト： `TRUE`

説明： API 関数 `LogTaskTracePoint…()` のコールにより行われるタスクトレースポイントのロギングを有効にします。

3.2.18 INTERVALS

用法： `INTERVALS = true-false-runtime`

デフォルト： `TRUE`

説明： API 関数 `LogInterval…()` のコールにより行われる経過時間のインターバルのロギングを有効にします。

3.2.19 STACK

用法： `STACK = true-false`

デフォルト： `FALSE`

説明： アプリケーション内のスタック使用についてのロギングを有効にします。

3.2.20 CATEGORY

用法： 以下のいずれか：

```
CATEGORY identifier = true-false
CATEGORY identifier = RUNTIME MASK AUTO
CATEGORY identifier = RUNTIME MASK maskbit
```

説明： カテゴリを定義します。カテゴリ `ID` はアプリケーション内で視覚化されます。カテゴリ `ID` を `Log…()` コールとともに使用して、特定グループのユーザートレースポイントを有効または無効にします。
`RUNTIME` という値が設定されたカテゴリのステータスは、`4` バイトのビットマップで保持されます。`RUNTIME` カテゴリは最大 `31` 個定義でき、コンフィギュレーション設定時にはカテゴリは何個でも定義できます。
あるカテゴリを `'RUNTIME MASK AUTO'` と指定すると、システムはそのカテゴリのステータス専用のビットをアロケートします。
カテゴリがプリコンパイル済みライブラリ内で参照されている場合には、コンパイルによりカテゴリ値がライブラリに組み込まれることになるので、値をマスクビットで明示的に設定する必要があります。

3.3 トレース表示の調整

ここで説明する指示語はターゲットコードには影響しません。これらはビジュアライザに表示される前のトレースデータの解釈に影響を与えます。

以降に示す各指示語の説明には、指示語の値が次のような規則に従って記述されています。

id	トレースオブジェクトを識別するために API に（たとえば LogTracePoint() を使用して）渡される番号です。
identifier	ある特定のオブジェクト（タスクなど）を識別する C 識別子です。
name	オブジェクトタイプについて、ID と共にビジュアライザに表示される名前です。これもターゲットコードには影響しませんが、C の識別子のシンタックスに従っている必要があります。
string	二重引用符（"）に囲まれた文字列です。
format-string	トレースデータの解釈と表現について定義する文字列です。詳しくは第 4 章を参照してください
index	列挙型クラス内の名前に対応する値です。

3.3.1 MESSAGE

用法： `MESSAGE = identifier [ AS format-string ]`

説明： メッセージの識別子 *identifier* のビジュアライザへの表示形式を指定します。

3.3.2 TRACEPOINT

用法： `TRACEPOINT = id : name [AS format-string]`

説明： トレースポイント *id* に名前 *name* を割り当てます。
トレースポイントにデータが関連付けられている場合、そのデータは *format-string* に従って表示されます。

3.3.3 TASK_TRACEPOINT

用法： `TASK_TRACEPOINT = id [, identifier ]: name [AS format-string]`

説明： タスクトレースポイント *id* に名前 *name* を割り当てます。必要に応じて、タスク *identifier* のタスクトレースポイントだけに名前を割り当てるように制限できます。
タスクトレースポイントにデータが関連付けられている場合、そのデータは *format-string* に従って表示されます。

3.3.4 INTERVAL

用法： `INTERVAL = id : name [AS format-string ]`

説明： *id* により識別されるインターバルに名前 *name* を割り当てます。
インターバルにデータが関連付けられている場合、そのデータは *format-string* に従って表示されます。

3.3.5 COUNTER

用法 : `COUNTER = identifier [ AS format-string ]`

説明 : ビジュアライザ上でのカウンタの表示形式を指定します。

3.3.6 ENUM

用法 : `ENUM = id : index [CALLED string]`

説明 : 列挙型のデータのビジュアライザへの表示形式を指定します。

例 ここでは C の `enum` と、それが RTA-TRACE の指示語に反映されたものの例を紹介します。

```
/* C fragment */

enum e_Rainbow {
    E_RED,
    E_ORANGE,
    E_YELLOW,
    E_GREEN,
    E_BLUE,
    E_INDIGO,
    E_VIOLET
};

/* End C fragment */
```

```
# RTAtrace.cfg fragment

enum = 1 : 0 as "Red";
enum = 1 : 1 as "Orange";
enum = 1 : 2 as "Yellow";
enum = 1 : 3 as "Green";
enum = 1 : 4 as "Blue";
enum = 1 : 5 as "Indigo";
enum = 1 : 6 as "Violet";

# End RTAtrace.cfg fragment
```

4 フォーマット文字列（'Format Strings'）

フォーマット文字列で、各トレースアイテムのデータの出力形式を指定することができます。単純な数値データは 1 つのフォーマット指定子（'format specifier'）で出力し、複雑なデータ（C 言語の構造体など）の場合は、データポインタをデータブロックの前後に移動させながらさまざまなフォーマット指定子を使用してデータを出力します。

フォーマット文字列が指定されていないと、データは以下のように出力されます。

- データサイズがターゲットの `int` 型を超えない場合は、データは "%d" と指定されたものとみなされて出力されます。
- 上記以外の場合は、以下のように HEX コードでダンプされます。

```
0000 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
0010 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
```

- 最大 256 バイトまで出力されます。

注記：フォーマット記述子が定義されていると、ターゲットのエンディアンが考慮されますが、HEX コードのダンプ出力の場合は、ターゲットのメモリが順に 1 バイトずつ出力されます。このため、`%x` というフォーマット記述子を使用した場合、HEX ダンプの内容とは違う出力内容になる場合があります。

4.1 フォーマット規則

フォーマット文字列には、C 関数の `printf()` の 1 番目の引数とほぼ同じ規則が適用されます。

- フォーマット文字列は、二重引用符（"）で囲みます。
- フォーマット文字列には 2 種類のタイプのオブジェクトを含めることができます。1 つは出力ストリームにそのままコピーされる通常の文字で、もう 1 つは、イベントとともに供給されるデータを変換して出力するためのフォーマットエレメントです。
- フォーマットエレメントは、`%` 文字と、桁数を表わす数字、そして 1 つの文字で構成されます。ただし `%E` のみは例外です。以下の項を参照してください。
- フォーマットエレメントは、以下の表の規則に従って変換され、その結果が出力文字列に加えられます。
- 特殊なフォーマットエレメント `%%` は、`%` と出力されます。
- 通常の文字や変換方法に加え、「バックスラッシュ（= 円記号）- エスケープシーケンス」を用いて特殊な文字を出力することができます。たとえば、文字の二重引用符（"）を出力するには `\"`（または `¥"`）と表わし、`\`（または `¥`）文字を出力する場合は `\\`（または `¥¥`）と表わします。
- 整数フォーマット指定子用のオプションのサイズパラメータは、フィールドの幅をバイト数で表わすものです。有効な値は、1、2、4、8 です。

注記： `printf()` とは異なり、フィールドを出力する際、フィールドのポインタは現在の位置から自動的に移動しません。これは、1 つのフィールドを複数のフォーマットで出力する場合を考慮しているためです。

フォーマットエレメント	説明
`%offset@`	データポインタを *offset* バイト分だけ移動します。構造体の中の任意のフィールドの値を出力する際に使用します。
`%[size]d`	現在のアイテムを符号付整数として解釈し、符号付き 10 進数で出力します。
`%[size]u`	現在のアイテムを符号なし整数として解釈し、符号なし 10 進数で出力します。
`%[size]x`	現在のアイテムを符号なし整数として解釈し、符号なし 16 進数で出力します。
`%[size]b`	現在のアイテムを符号なし整数として解釈し、符号なし 2 進数で出力します。

フォーマットエレメント	説明
%enum[:size]E	現在のアイテムを、*enum* という ID を持つ列挙型クラス用のインデックスとして解釈し、列挙型クラス内のテキストのうち、そのインデックスの値に対応するものを出力します。 列挙型クラスは、ENUM 命令で定義されている必要があります。ただしスタートアップとエラーコードの列挙型クラス 98 および 99 に限り、すでに暗黙的に定義されています。
%F	現在の値を IEEE の倍精度の浮動小数点として解釈し、double 型（必要に応じて指数形式）として出力します。
%?	HEX ダンプ形式で出力します。
%%	% という文字を出力します。

4.2 フォーマットの例

出力内容	フォーマット文字列	表記例	注意事項
1 つの整数値を 10 進数と 16 進数で出力	"%d 0x%x"	10 0xA	%x というフォーマット指定子のみでは "0x" という文字は出力されないので、直接それらの文字を文字列として表記する必要があります。
1 つの符号なしのバイト値を % という文字とともに出力	"%1u%%"	73%	1 バイトのサイズ指定子を使用し、% という文字は %% と表記します。

32 ビットプロセッサで以下の内容を出力
struct {
int x;
int y;
};
	"(%d, %4@%d)"	(20, -15)	%offset@ を使用して構造体内部のバイトオフセットを指定します。

enum 型の e_Rainbow（3.3.6 項を参照してください）内の値を出力	"%1E"	Yellow	1 という数字は、ENUM 命令で定義された列挙型クラスの ID を示し、フィールドの幅を示すものではありません。

5 お問い合わせ先

製品サポートに関しては、各 ETAS 支社までお問い合わせください。

ドイツ（ETAS 本社）

ETAS GmbH

Borsigstr. 14	Phone:	+49 (711) 8 96 61-0
70469 Stuttgart	Fax:	+49 (711) 8 96 61-105
Germany	E-mail:	sales@etas.de
	WWW:	www.etasgroup.com

日本

イータス株式会社

〒 220-6217	Phone:	(045) 222-0900
神奈川県横浜市西区	Fax:	(045) 222-0956
みなとみらい 2-3-5 クイーンズタワー C 17F	E-mail	sales@etas.co.jp
	WWW:	www.etasgroup.com

韓国

ETAS Korea Co., Ltd.

3F Samseung Bldg	Phone:	+82 (2) 5747 016
61-1, Yangjae-dong	Fax:	+82 (2) 5747 120
Seocho-gu, Seoul	E-mail:	sungik.hong@etas.co.kr
Republic of Korea		

イギリス

ETAS Engineering Tools Application and Services Ltd.

Studio 3, Waterside Court	Phone:	+44 (0) 1283 - 546512
3rd Avenue, Centrum 100	Fax:	+44 (0) 1283 - 548767
Burton-upon-Trent	E-mail	sales@etas-uk.net
Staffordshire DE14 2WQ	WWW:	www.etasgroup.com
UK		

フランス

ETAS S.A.S

1, place des Etats-Unis	Phone:	+33 (1) 56 70 00 50
SILIC 310	Fax:	+33 (1) 56 70 00 51
94588 Rungis Cedex	E-mail	sales@etas.fr
France	WWW:	www.etasgroup.com

北米

ETAS Inc.

3021 Miller Road	Phone:	+1 (888) ETAS INC
Ann Arbor, MI 48103	Fax:	+1 (734) 997-9449
USA	E-mail	sales@etasinc.com
	WWW:	www.etasgroup.com

南米

UNIT

Rua Adolfo Maraccini, 399
c/o Sergio Augusto Alves
Campinas SP 13086 - 010
Brazil

Mobile: +55 19 9772 0793
Telefax: +55 (19) 3256 1939
E-mail: unit@mpc.com.br

索引

A
ACTIVATIONS 14
ALARMS 15

B
BUFFER_SIZE 12

C
CATEGORY 16
COMPACT 12
COUNTER 18

E
ENUM 18
ERRORS 14
EXCLUDE_TASK_OR_ISR 13, 15
EXPLICIT_STATE_MESSAGES 14

I
IMPLICIT_STATE_MESSAGES 14
INTERRUPT_LOCKS 14
INTERVAL 17
INTERVALS 16

M
MESSAGE 15, 17

MESSAGE_DATA 15

O
OSEK_MESSAGES 15

P
PROCESSES 13

R
RESOURCES 14

S
STACK 16
STARTUP_AND_SHUTDOWN 13
SWITCHING_OVERHEADS 13, 15

T
TASKS_AND_ISRS 13
TASK_TRACEPOINT 17
TIME_SIZE 12
TIMETABLES 15
TRACEPOINT 17
TRACEPOINTS 16