TOPPERS Automotive Kernel
SG 取扱説明書

Ver.5.00

2008/10/20

株式会社ヴィッツ　組込みソフトウェア開発部

承認	審査	作成

＜目次＞

1. 概要

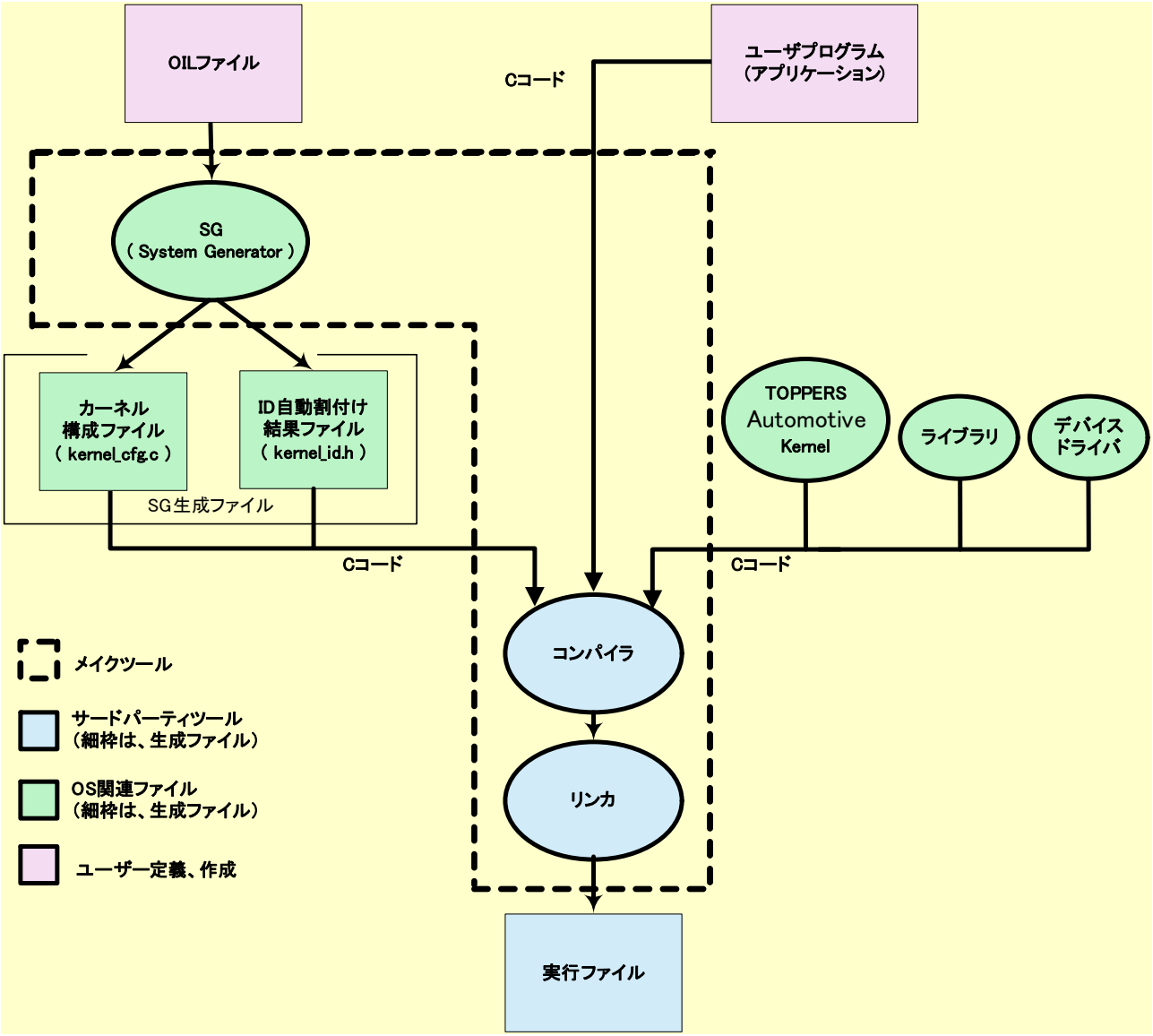
本 SG 取扱説明書では、OSEK/VDX 仕様に準拠した TOPPERS Automotive Kernel 付属の OSEK System Generator（以下 SG）の使用方法を記述します。

1.1. はじめに

本 SG では、OIL（OSEK Implemantaion Language）と呼ばれる専門の記述言語によって書かれたソーステキストを入力し、TOPPERS Automotive Kernel が必要とする C 言語のソースファイル及びその関連ファイルを出力するツールです。使用する OIL ファイルは「OSEK/VDX Implementation Language (OIL) Ver2.5」の仕様に基づいている必要があります。

1.2. 処理フロー

SG を用いて生成ファイルが出力される処理を以下に示します。



SG 生成ファイル、ユーザプログラム、TOPPERS Automotive Kernel をコンパイルし、ライブラリとデバイスドライバをリンクして最終的にオブジェクトファイルを生成する流れになっています。

OILファイルには、SGが必要する情報（TOPPERS Automotive Kernelの一部を生成するといった動作定義）が記載されています。OILファイルについては、4. OILファイルを参照して下さい。

1.3. 関連文書

・OSEK/VDX System Generation OIL:OSEK Implementation Language Version 2.5

・OSEK/VDX Operating System Version 2.2.1

2. 使用方法

2.1. SG実行方法

SG の実行は、コマンドプロンプト上から行います。
基本的な操作としてコマンドラインにより、SG 実行ファイル名、OIL ファイル名、及びコマンドラインオプションを指定します。

以下に、本 SG の使用例を記載します（OIL ファイル名は sample.oil とします)。

■例 1

sg.exe sample.oil –os=ECC2 –lj –I ./../impl_oil –template=m32c.sgt

指定オプション概要

–os=ECC2	TOPPERS Automotive Kernel のコンフォーマンスクラス ECC2 を使用
–lj	出力メッセージを日本語で表示
–I ./../impl_oil	OIL ファイルのインクルードパスを一階層上の impl_oil フォルダに指定
–template=m32c.sgt	テンプレートファイル”m32c.sgt”の読み込み

■例 2

sg.exe sample.oil –os=ECC2 –lj –I ./../impl_oil –cpu=m16c –debug –cfg=test.c –withouttime

指定オプション概要（例 1 と重複は省略）

–cpu=m16c	ターゲット CPU は M16C を使用
–debug	デバッグメッセージの出力
–cfg=test.c	カーネル構成ファイル名を”test.c”に変更
–withouttime	出力ファイルに生成日時の出力を行わない

上記例を参考に SG を実行すると、カーネル構成ファイル（kernel_cfg.c）及び ID 自動割付ファイル(kernel_id.h)を出力します。

2.2. SGコマンド一覧

SG を実行する際に、指定可能なコマンドラインオプションの一覧を以下に示します。

オプション	別指定方法	内容
-h	--help	コマンドラインヘルプを表示します。
-debug		デバッグ用メッセージやツリーダンプを出力します。
-lj	--japanese	出力メッセージを日本語表示します。
-le	--english	出力メッセージを英語表示します。
-I <directory>		OIL ファイルのインクルードパスを指定します。
-cfg=<file>		カーネル構成ファイルのファイル名を<file>に変更します。 （デフォルト設定：kernel_cfg.c）
-id=<file>		TOPPERS Automotive Kernel の ID 自動割付ファイルのファイル名を<file>に変更します。(デフォルト設定：kernel_id.h)
-cpu=<cpu>		<cpu>に依存した処理を要求します。省略した場合、依存処理の出力を行いません。 例：-cpu=m32c：M32C 指定（M32C 依存処理の実行）
-template=<file>		読み込むテンプレートファイルを指定します。本オプションが指定された場合、-cpu の指定は無効となります。
-tool=<tool>		<tool>に依存した処理を要求します。省略した場合、依存処理の出力を行いません。
-ovec=<file>		-cpu オプションを指定した場合の依存ファイルの出力先を変更します（デフォルトは構成ファイル内に出力）
-odep=<file>		テンプレートファイル内で指定された箇所を、<file>で指定されたファイルへ出力します。アセンブラファイルを想定しています。
-sjis		OIL ファイル内のシフト JIS を許可します。
-n <directory>		ファイルの出力先として<directory>を指定します。
-os=<class>		TOPPERS Automotive Kernel のコンフォーマンスクラスを指定します。 =BCC1：BCC1 指定 =BCC2：BCC2 指定 =ECC1：ECC1 指定 =ECC2：ECC2 指定
-osinc=<file>		カーネル構成ファイルにて<file>をインクルードします。

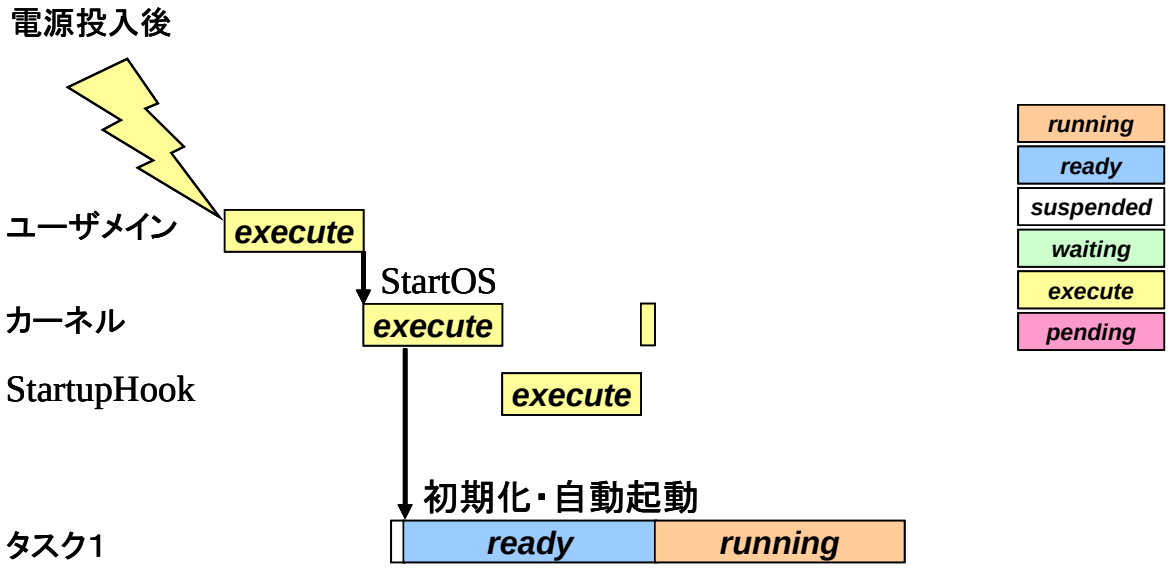
-withoutinfo		出力ファイルのヘッダに生成日時を出力しないようにします。

3. クイックリファレンス

TOPPERS Automotive Kernel の基本的な動作イメージ及び、OIL の記述方法を示します。尚、OIL 記述に関して、該当する機能の実行に不要な箇所は記述していません。

3.1. OS起動前タイミングで処理を行う

OS を起動する前のタイミングでユーザ処理を実行することができます。デバイスドライバの初期化等に利用可能です。動作イメージを以下に示します。



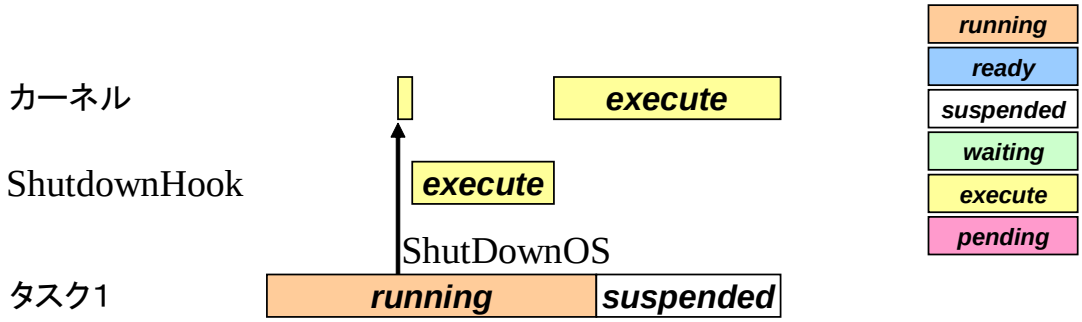
このような動作を実現したい場合は、以下のように OIL ファイルを設定して下さい。

```
OS   OS オブジェクト名(任意) {
       STATUS =
       STARTUPHOOK = TRUE;
       ERRORHOOK =
       SHUTDOWNHOOK =
       PRETASKHOOK =
       POSTTASKHOOK =
       USEGETSERVICEID =
       USEPARAMETERACCESS =
       USERESSCHEDULER =
};
```

OS 起動前タイミングにて、ユーザ処理を実施したい場合は「TRUE」を指定し、実施しない場合は「FALSE」を指定します。

3.2. OS終了時タイミングで処理を行う

OS を終了するタイミングでユーザ処理を行うことができます。デバイスドライバの終了処理等に利用可能です。動作イメージを以下に示します。



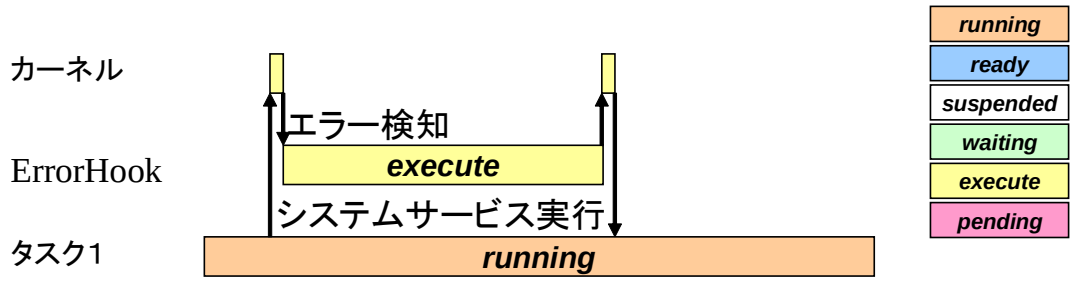
このような動作を実現したい場合には、以下のように OIL ファイルを記述して下さい。

```
OS    OS オブジェクト名(任意) {
        STATUS =
        STARTUPHOOK =
        ERRORHOOK =
        SHUTDOWNHOOK = TRUE;
        PRETASKHOOK =
        POSTTASKHOOK =
        USEGETSERVICEID =
        USEPARAMETERACCESS =
        USERESSCHEDULER =
};
```

OS 終了時タイミングにて、ユーザ処理を実施したい場合は「TRUE」を指定し、実施しない場合は「FALSE」を指定します。

3.3. エラーハンドリング処理を行う

TOPPERS Automotive Kernel においてエラーハンドリング処理を行うことができます。システムサービス実行時のエラーハンドリングに利用可能です。動作イメージを以下に示します。



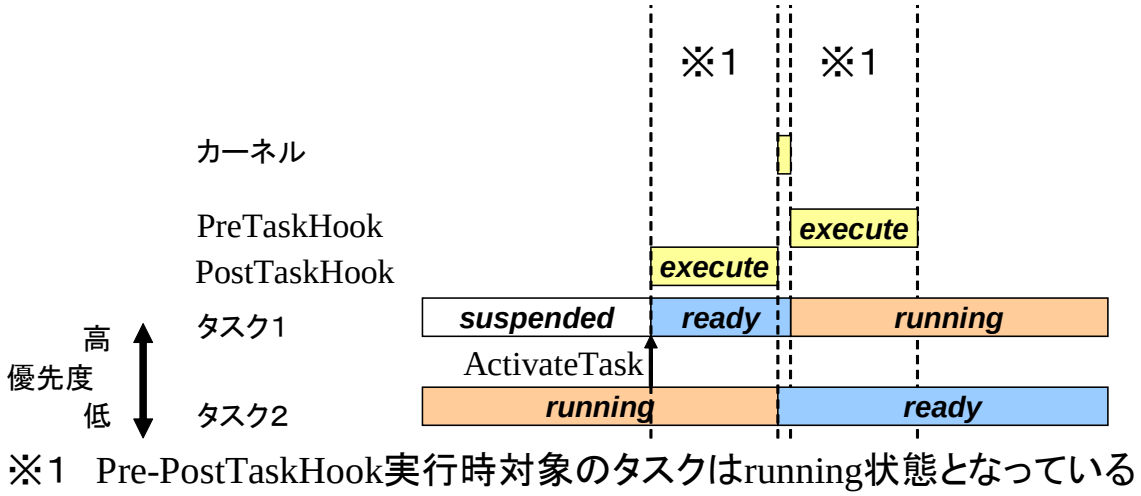
このような動作を実現したい場合は、以下のように OIL ファイルを設定して下さい。

```
OS    OS オブジェクト名(任意) {
      STATUS =
      STARTUPHOOK =
      ERRORHOOK = TRUE ;
      SHUTDOWNHOOK =
      PRETASKHOOK =
      POSTTASKHOOK =
      USEGETSERVICEID =
      USEPARAMETERACCESS =
      USERESSCHEDULER =
};
```

OS 終了時タイミングにて、ユーザ処理を
実施したい場合は「TRUE」を指定し、
実施しない場合は「FALSE」を指定します。

3.4. タスク実行前/後タイミングで処理を行う

TOPPERS Automotive Kernel において、タスク実行前及び実行後で処理を行うことができます。タスクの実行時間計測等に実行可能です。動作イメージを以下に示します。



※1　Pre-PostTaskHook実行時対象のタスクはrunning状態となっている

このような動作を実現したい場合は、以下のように OIL ファイルを設定して下さい。

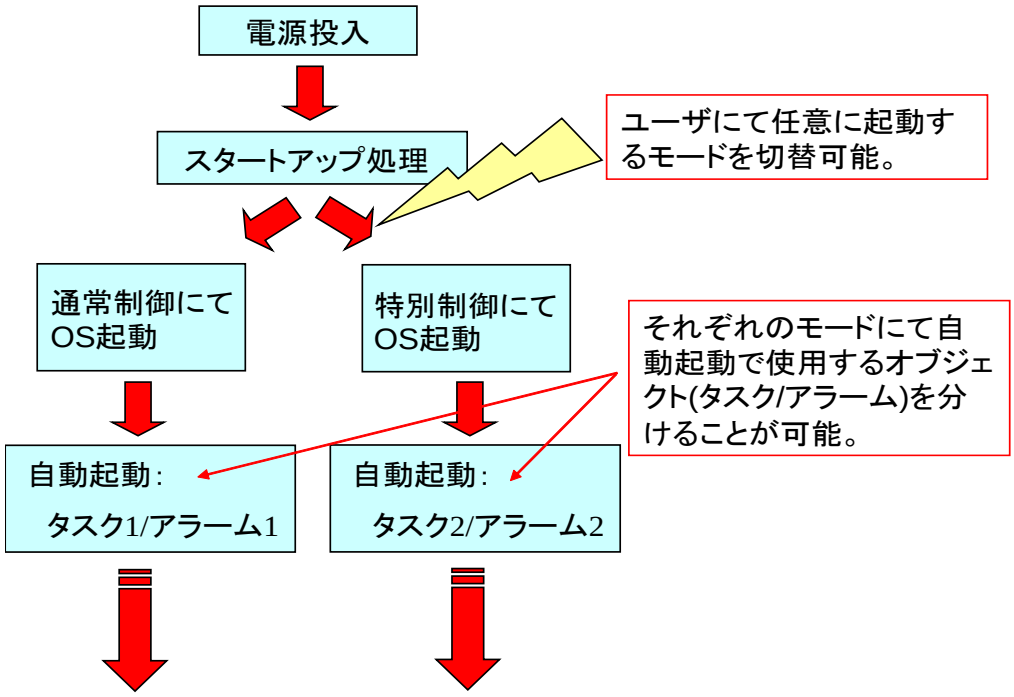
```
OS    OS オブジェクト名(任意) {
        STATUS =
        STARTUPHOOK =
        ERRORHOOK =
        SHUTDOWNHOOK =
        PRETASKHOOK = TRUE ;
        POSTTASKHOOK = TRUE ;
        USEGETSERVICEID =
        USEPARAMETERACCESS =
        USERESSCHEDULER =
};
```

タスクの状態が「実行状態」になった後で
ユーザ処理を実施したい場合、「TRUE」を指定し、
実施しない場合は「FALSE」を指定します。

タスクの状態が「実行状態」でなくなる前に
ユーザ処理を実施したい場合、「TRUE」を指定し、
実施しない場合は「FALSE」を指定します。

3.5. OS起動モードの制御

TOPPERS Automotive Kernel を起動するモードの設定を行うことができます。また、アラームやタスクの起動モードとして使用することも出来ます。動作イメージを以下に示します。



このような動作を実現したい場合は、以下のように OIL ファイルを設定して下さい。

```
APPMODE アプリケーションモード名(任意) {};

TASK タスク名(任意) {
  AUTOSTART = TRUE {
    APPMODE = アプリケーションモード名(任意);
  };
  .....
};
ALARM アラーム名(任意) {
  AUTOSTART = TRUE {
    APPMODE = アプリケーションモード名(任意);
  };
  .....
};
```

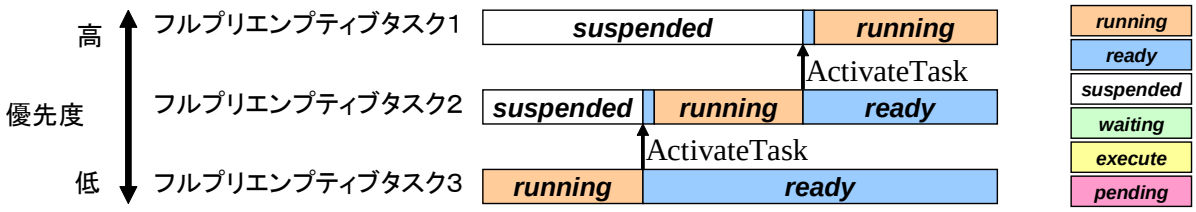
アプリケーションモードは
複数定義することが可能です。

自動起動を許可している場合、自動起動するアプリケーションモードを指定します。
複数指定することが可能です。

自動起動を許可している場合、自動起動するアプリケーションモードを指定します。
複数指定することが可能です。

3.6. 優先度ベースタスク制御（フルプリエンプティブ）

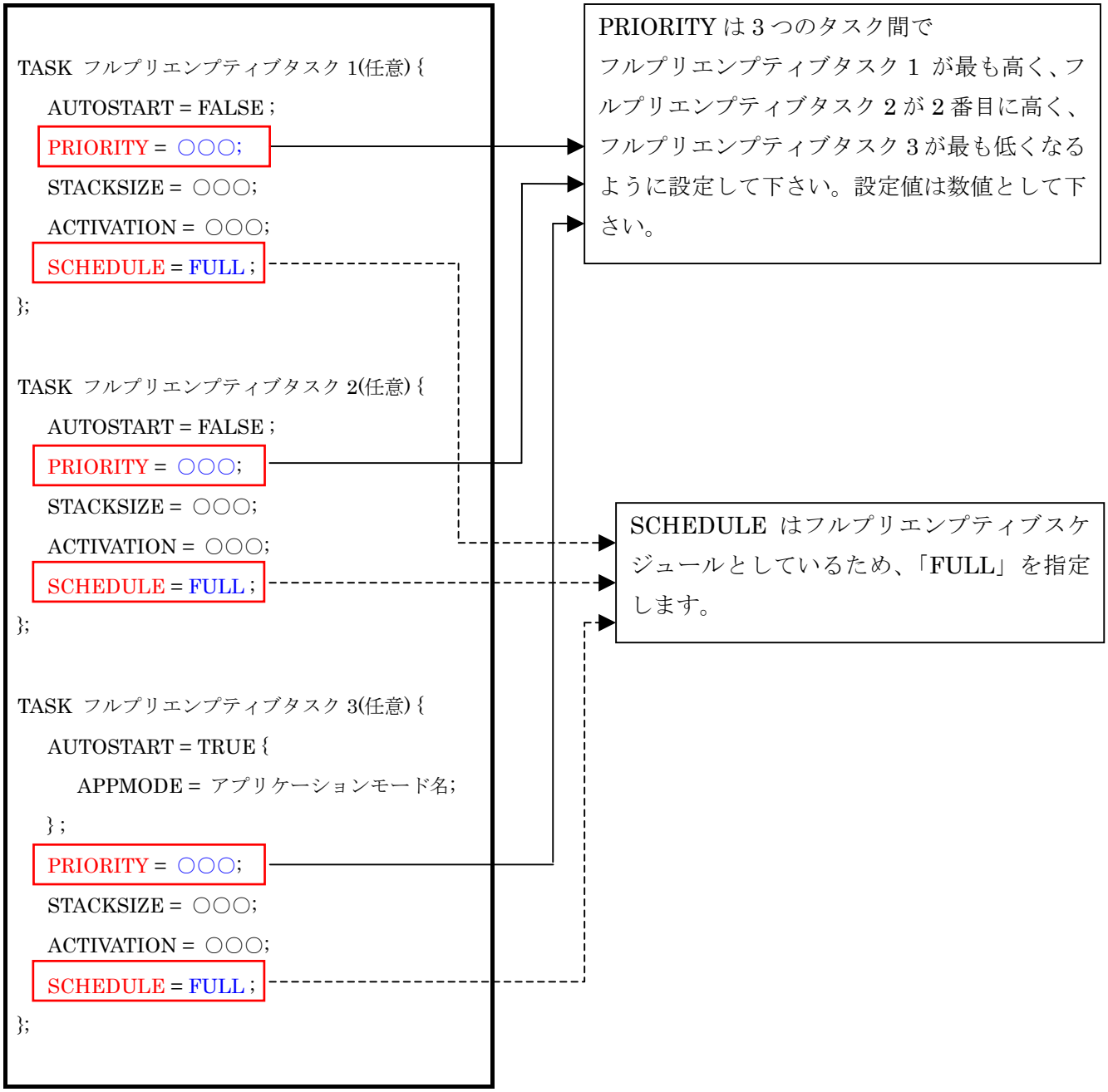
TOPPERS Automotive Kernel において、フルプリエンプティブな優先度ベースにてタスクの制御を行うことができます。動作イメージを以下に示します。



このような動作を実現したい場合は、次のように OIL ファイルを設定して下さい。

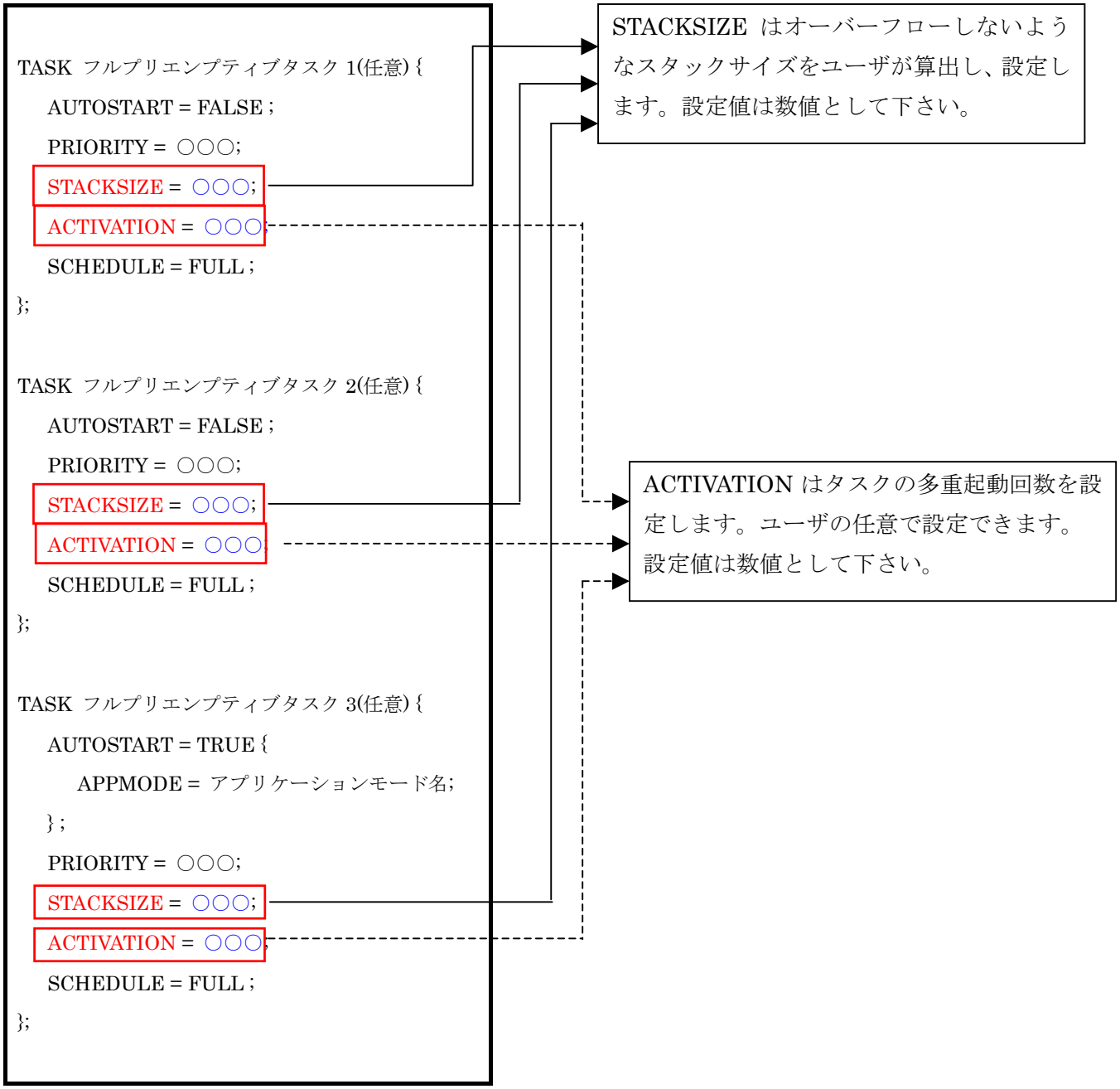
フルプリエンプティブスケジュール

OIL ファイル記述例



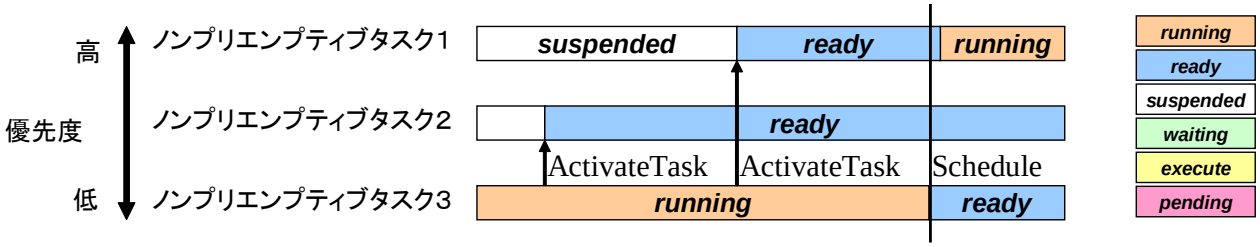
フルプリエンプティブスケジュール

OIL ファイル記述例続き



3.7. 優先度ベースタスク制御（ノンプリエンプティブ）

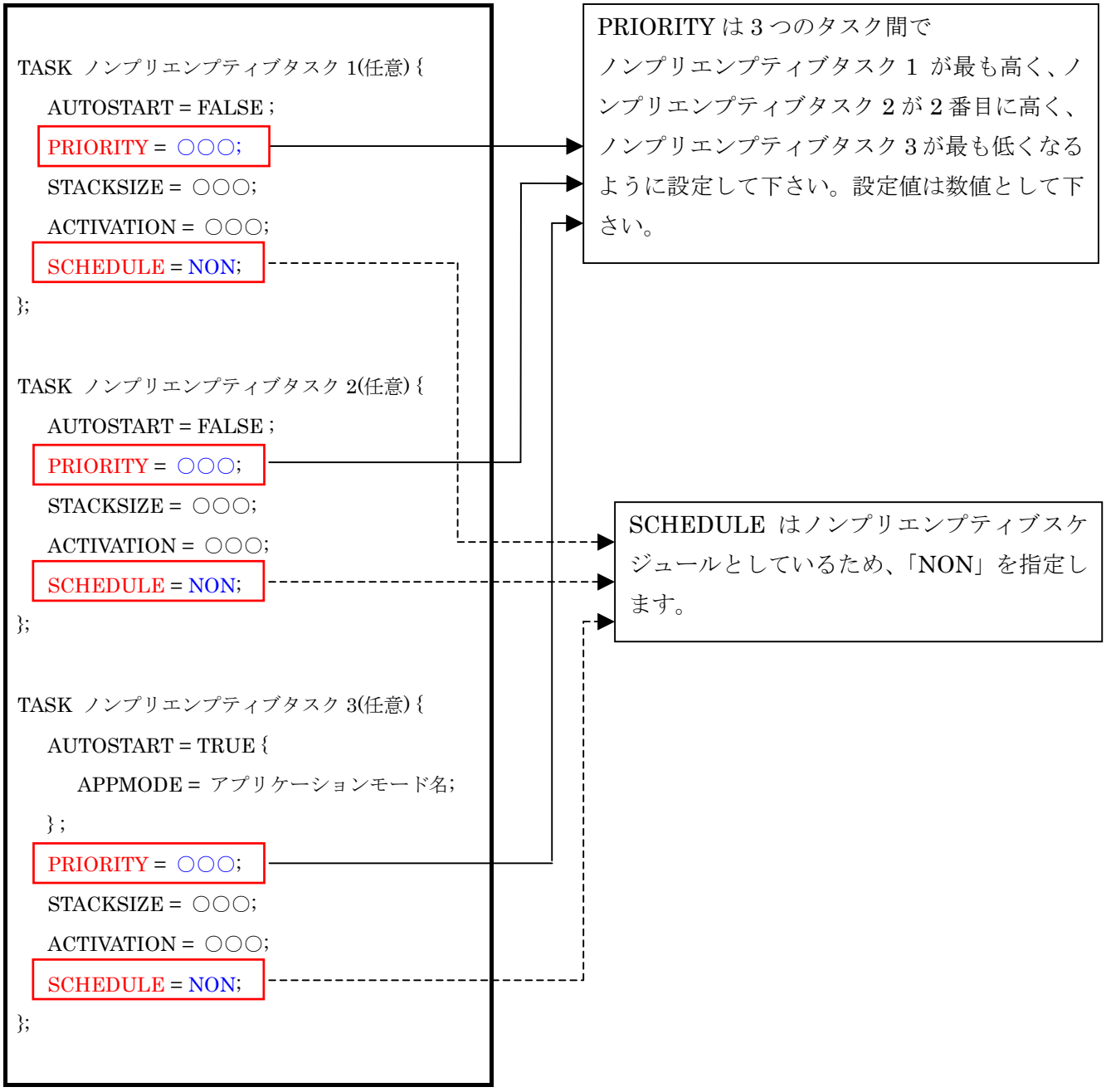
TOPPERS Automotive Kernel において、ノンプリエンプティブな優先度ベースにてタスクの制御を行うことができます。動作イメージを以下に示します。



このような動作を実現したい場合は、次のように OIL ファイルを設定して下さい。

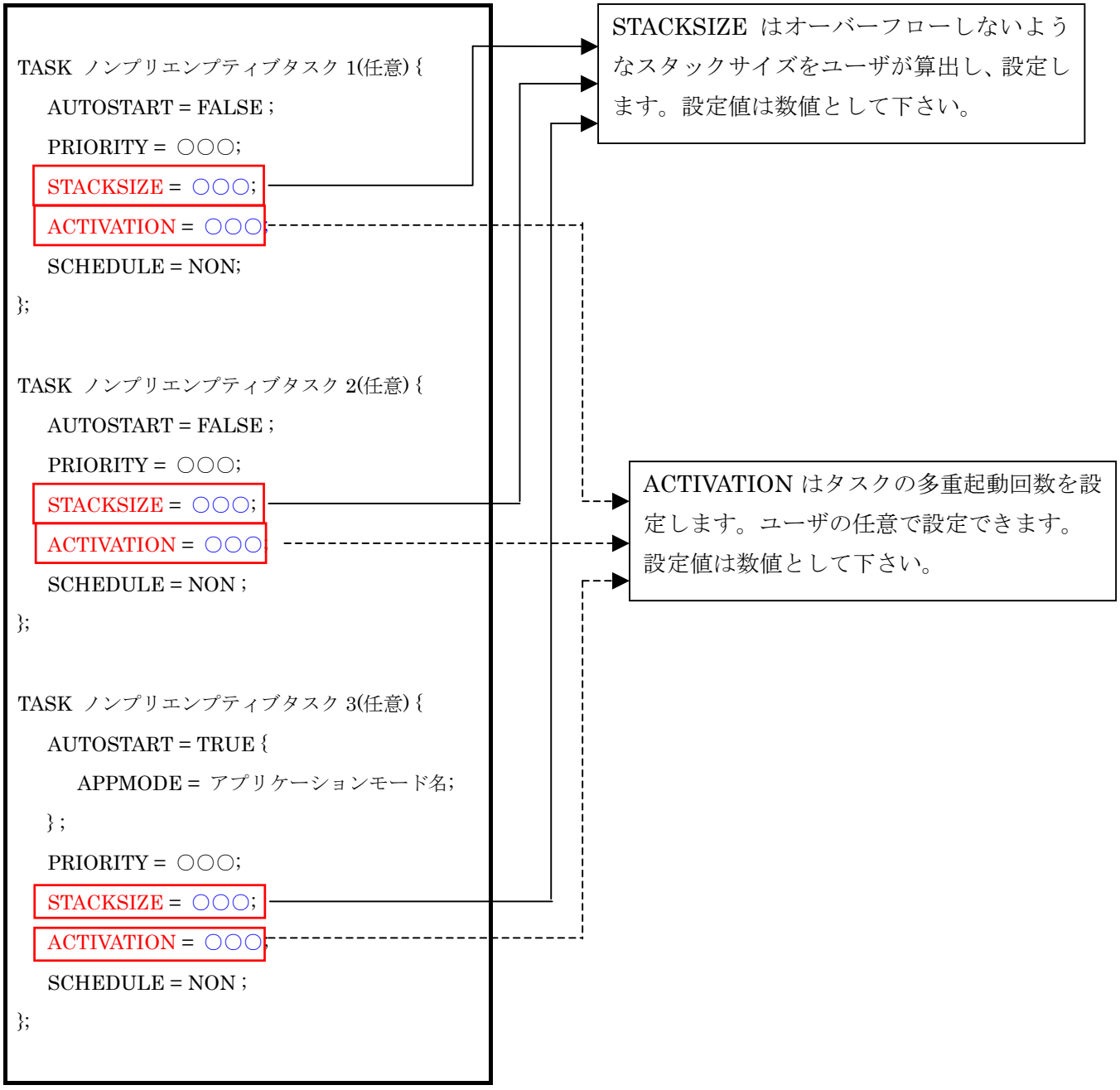
ノンプリエンプティブスケジュール

OIL ファイル記述例



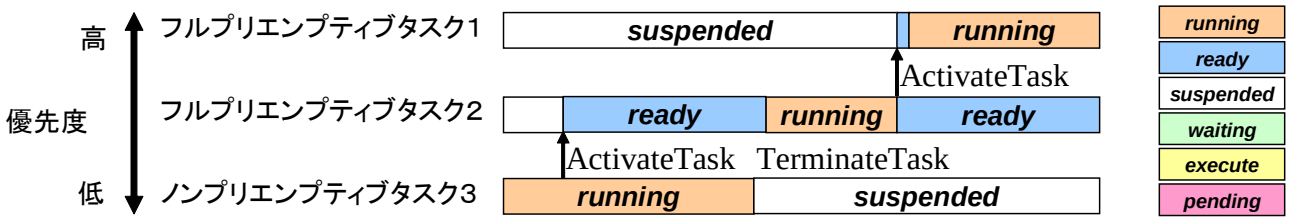
ノンプリエンプティブスケジュール

OIL ファイル記述例続き



3.8. 優先度ベースタスク制御（ミクスドプリエンプティブ）

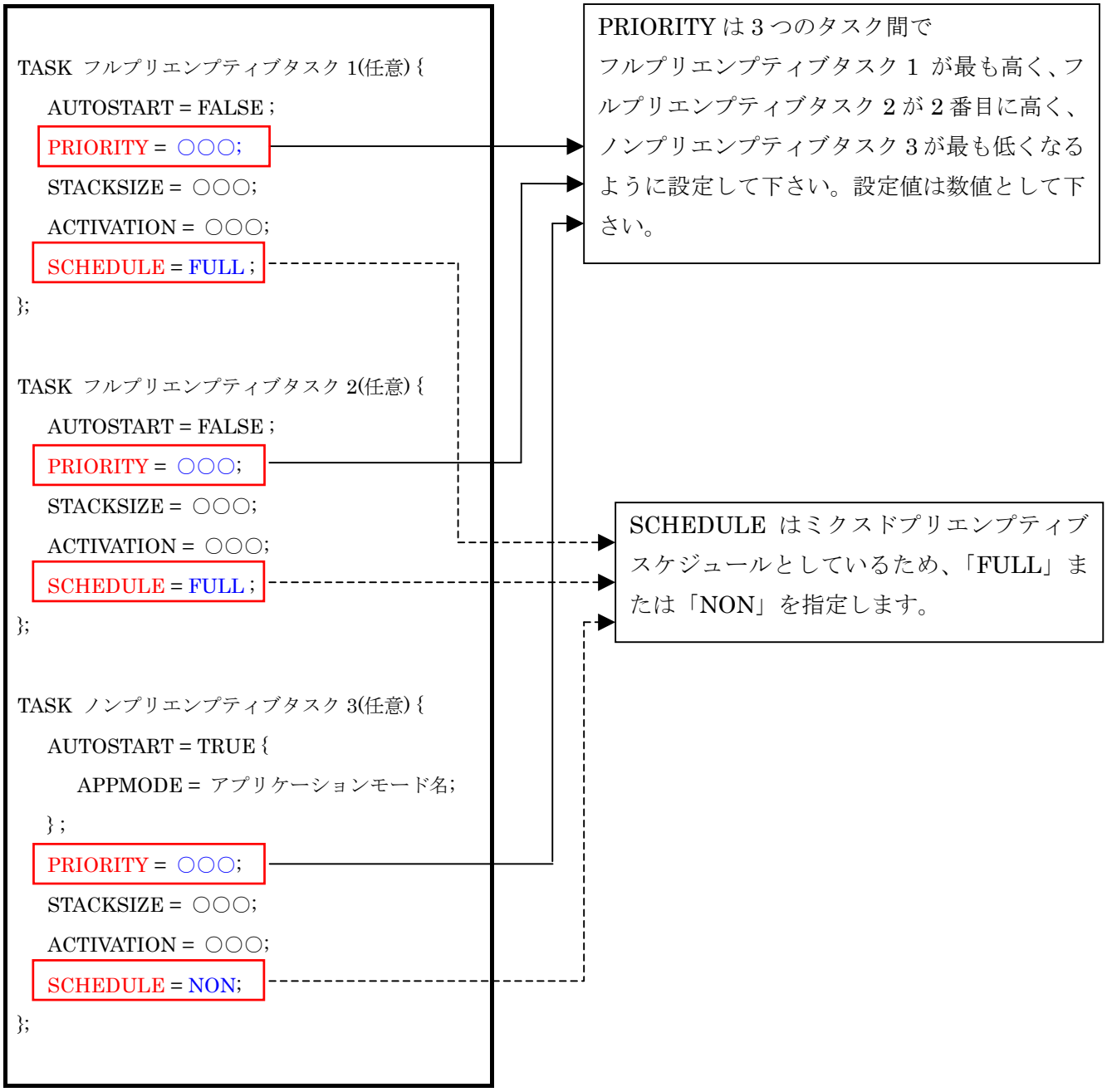
TOPPERS Automotive Kernel において、ミクスドプリエンプティブな優先度ベースにてタスクの制御を行うことができます。動作イメージを以下に示します。



このような動作を実現したい場合は、次のように OIL ファイルを設定して下さい。

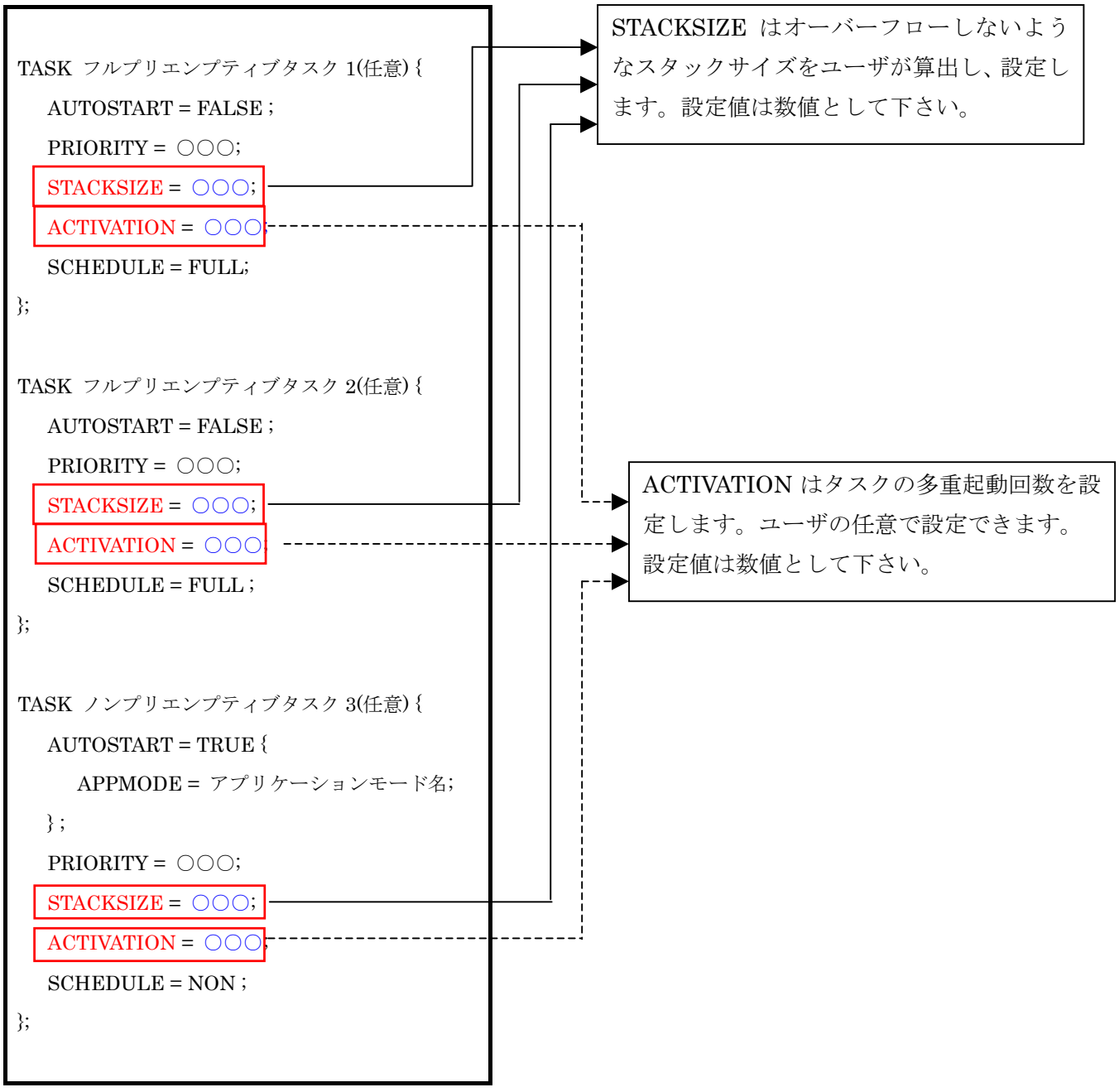
ミクスドプリエンプティブスケジュール

OIL ファイル記述例



ミクスドプリエンプティブスケジュール

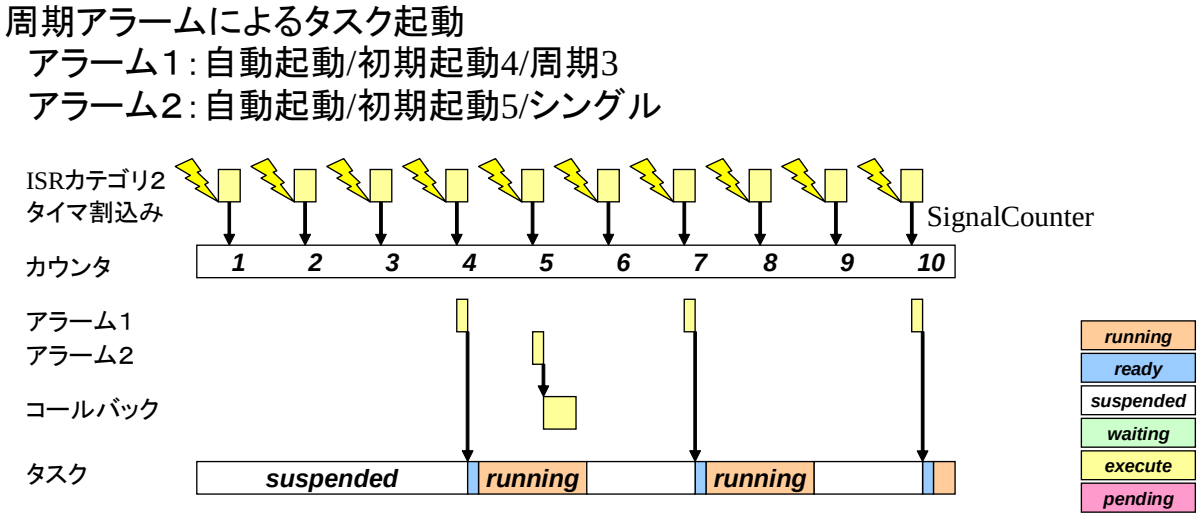
OIL ファイル記述例続き



3.9. 周期タスク制御の実現方法

TOPPERS Automotive Kernel ではアラームを使用することで周期的にタスクを制御することができます。

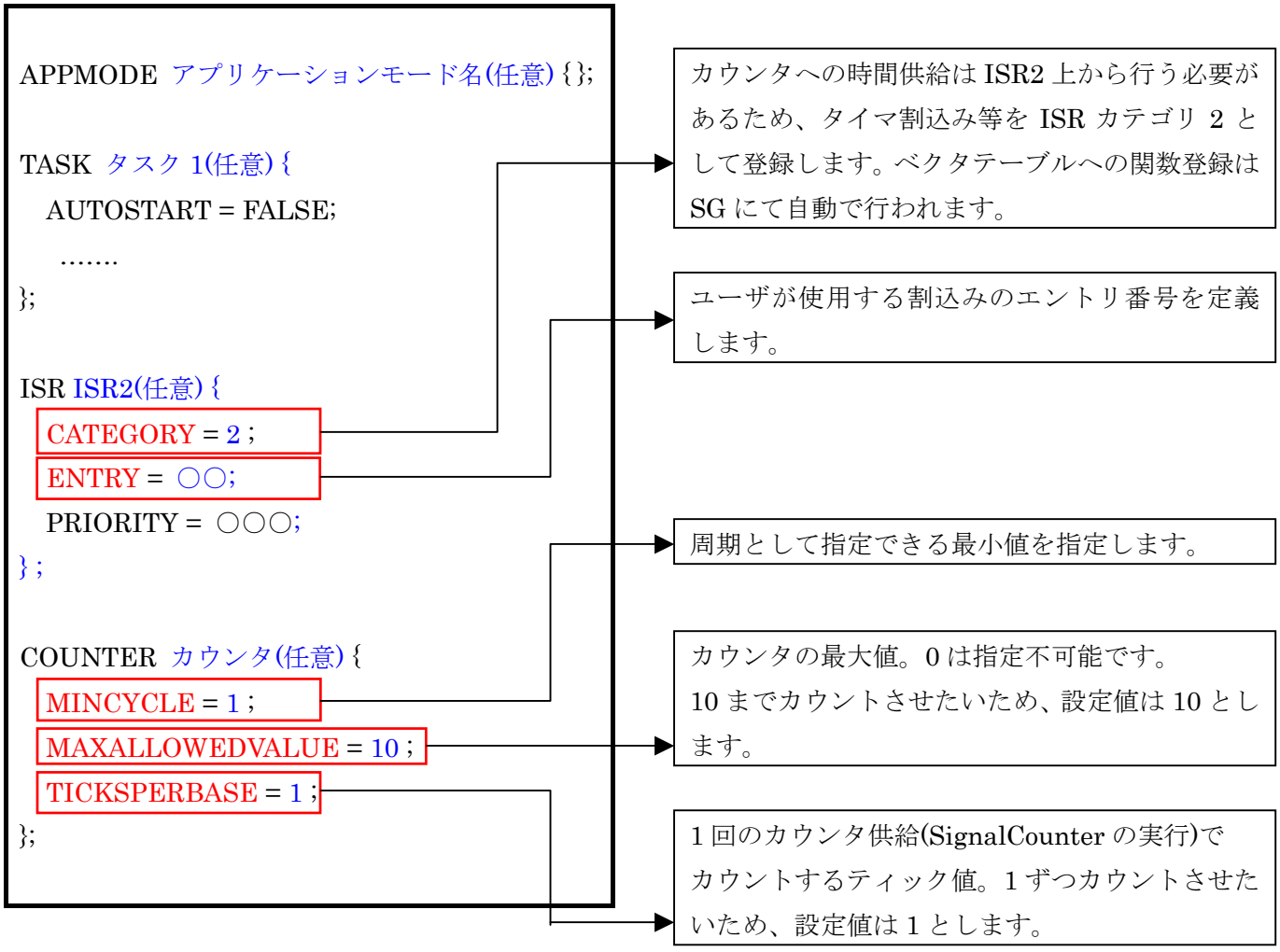
動作イメージを以下に示します。



このような動作を実現したい場合は、次のように OIL ファイルを設定して下さい。

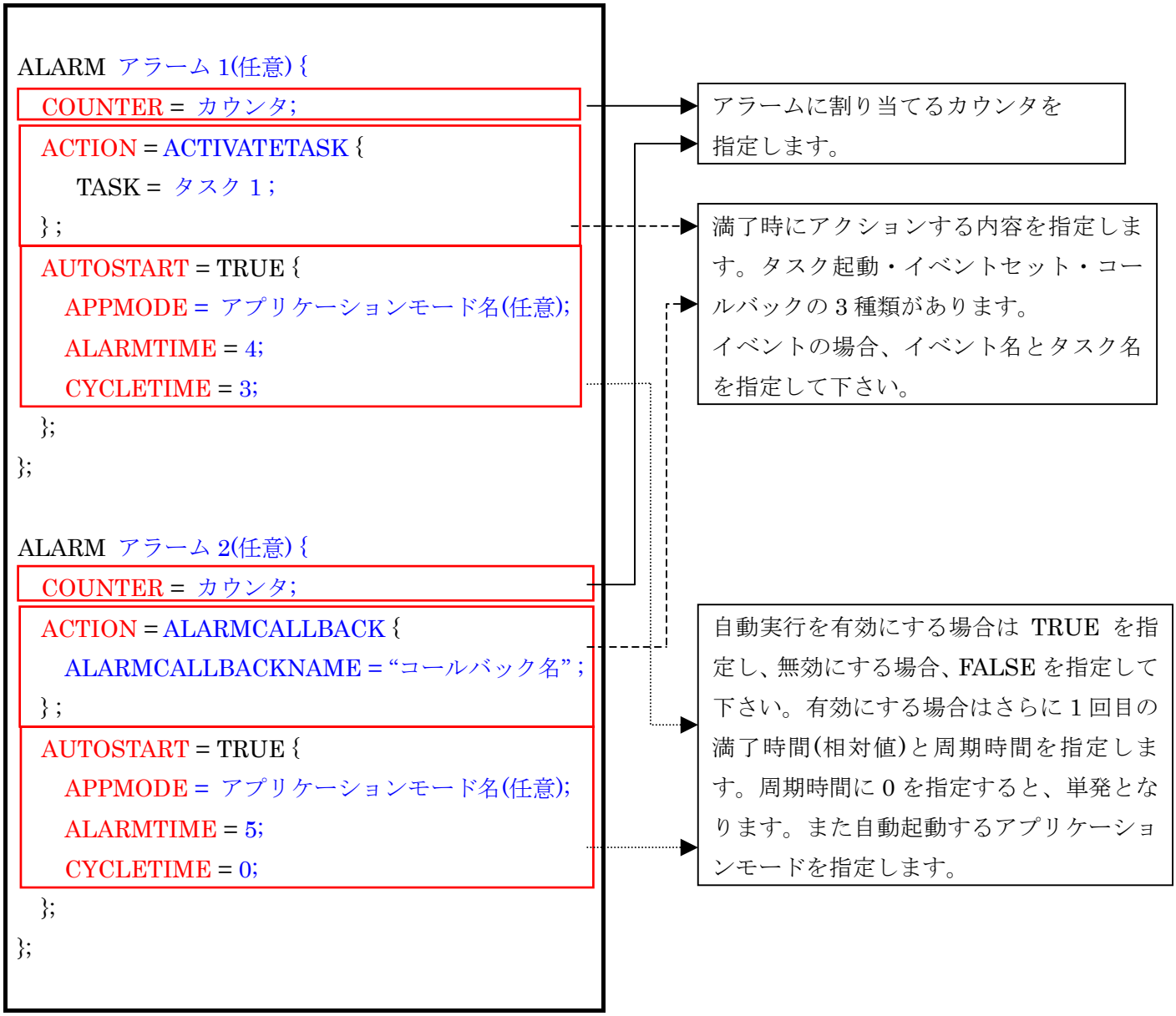
アラームを用いた周期タスク制御

OIL ファイル記述例



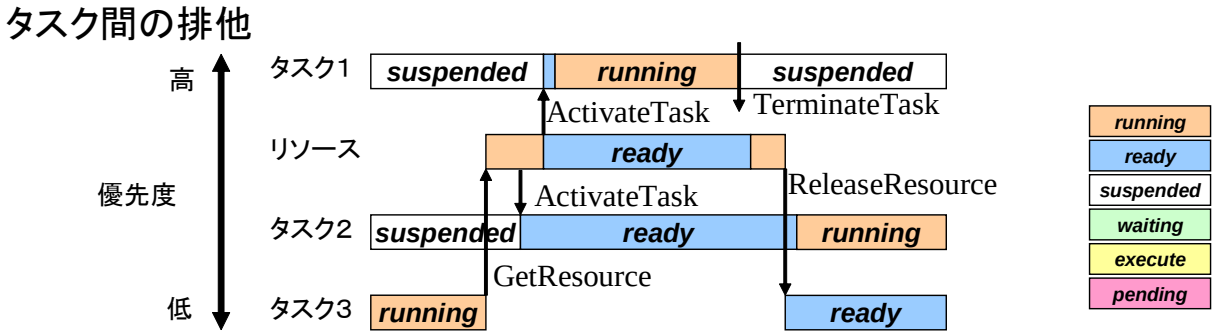
アラームを用いた周期タスク制御

OIL 記述例続き



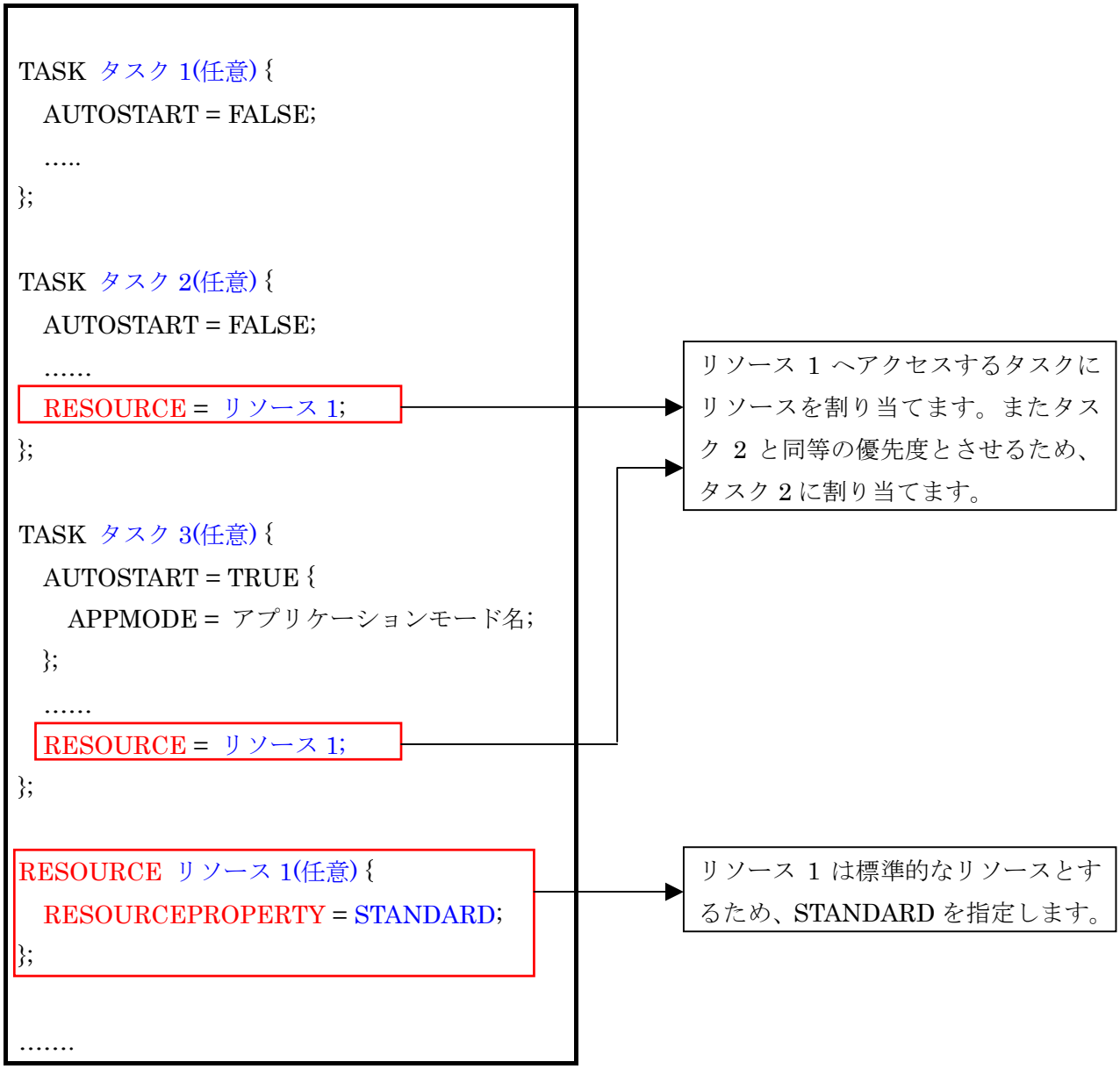
3.10. 排他制御（タスク間リソース使用）

TOPPERS Automotive Kernel ではリソースを使用することでタスク間の排他制御を行うことができます。タスク間のほかに、タスクーISR 間、ISR－ISR 間の排他が可能です。
以下に動作イメージを示します。



このような動作を実現したい場合は、次のように OIL ファイルを設定して下さい。

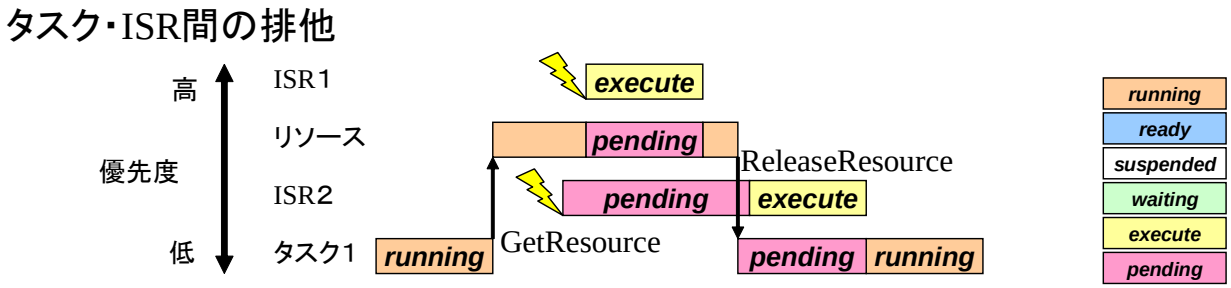
OIL ファイル記述例


TASK タスク 1(任意) {
AUTOSTART = FALSE;
.....
};
TASK タスク 2(任意) {
AUTOSTART = FALSE;
.....
RESOURCE = リソース 1;
};
TASK タスク 3(任意) {
AUTOSTART = TRUE {
APPMODE = アプリケーションモード名;
};
.....
RESOURCE = リソース 1;
};
RESOURCE リソース 1(任意) {
RESOURCEPROPERTY = STANDARD;
};
.....
リソース 1 へアクセスするタスクにリソースを割り当てます。またタスク 2 と同等の優先度とさせるため、タスク 2 に割り当てます。
リソース 1 は標準的なリソースとするため、STANDARD を指定します。

3.11. 排他制御（ISR間リソース使用）

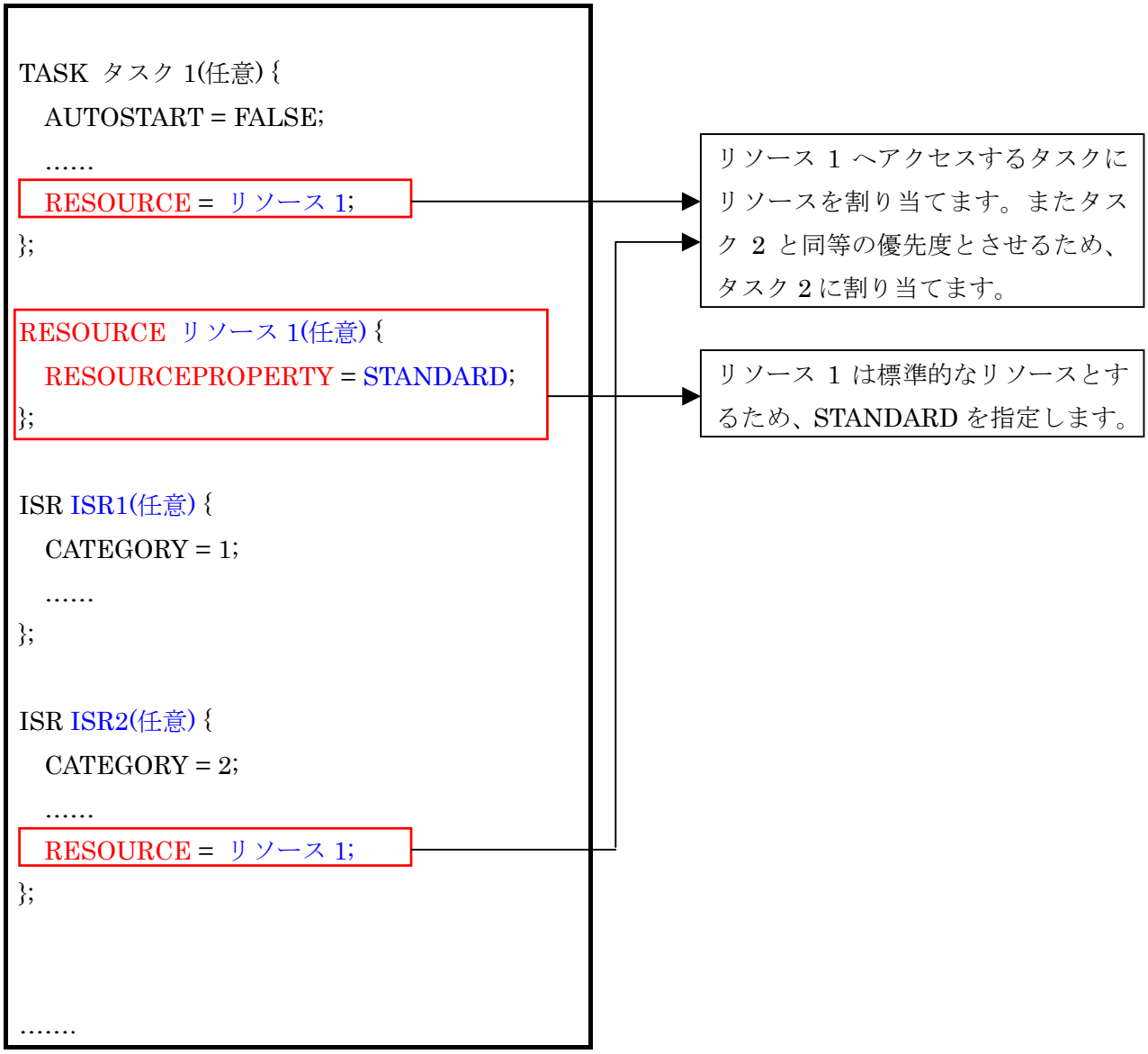
TOPPERS Automotive Kernel ではリソースを使用することでタスク・ISR 間の排他制御を行うことができます。タスク間のほかに、タスクータスク間、タスクーISR 間、ISRーISR 間の排他が可能です。

以下に動作イメージを示します。



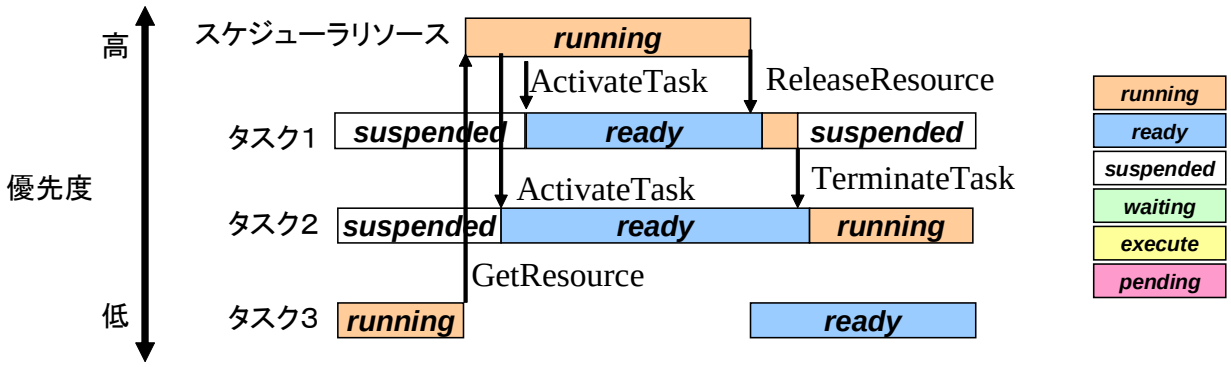
このような動作を実現したい場合は、次のように OIL ファイルを設定して下さい。

OIL ファイル記述例


TASK タスク 1(任意) {
AUTOSTART = FALSE;
……
RESOURCE = リソース 1;
};
RESOURCE リソース 1(任意) {
RESOURCEPROPERTY = STANDARD;
};
ISR ISR1(任意) {
CATEGORY = 1;
……
};
ISR ISR2(任意) {
CATEGORY = 2;
……
RESOURCE = リソース 1;
};
……
リソース 1 へアクセスするタスクにリソースを割り当てます。またタスク 2 と同等の優先度とさせるため、タスク 2 に割り当てます。
リソース 1 は標準的なリソースとするため、STANDARD を指定します。

3.12. 排他制御（スケジューラリソース使用）

TOPPERS Automotive Kernel ではスケジューラリソースを使用することでタスク間の排他制御を行うことができます。スケジューラリソースとはスケジューラレベルの優先度を持つリソースで、獲得するとスケジューラは動作できない仕組みとなっています。以下に動作イメージを示します。



このような動作を実現したい場合は、次のように OIL ファイルを設定して下さい。

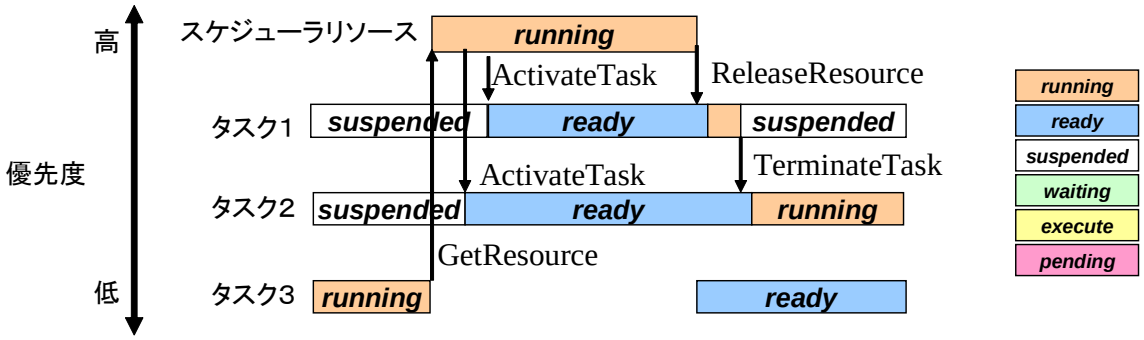
OIL ファイル記述例

```
OS OS オブジェクト名(任意) {
    USERESSCHEDULER = TRUE;
     .......
};

TASK  タスク 1(任意) {
    AUTOSTART = FALSE;
    .....
};

TASK  タスク 2(任意) {
    AUTOSTART = FALSE;
    ......
};

TASK  タスク 3(任意) {
    AUTOSTART = TRUE {
       APPMODE =  アプリケーションモード名;
    };
    ......
};
```

スケジューラリソースを使用する場合、「TRUE」を指定します。使用しない場合、「FALSE」を指定します。

3.13. 排他制御（内部リソース使用）

TOPPERS Automotive Kernel では内部リソースを使用することでタスク間の排他制御を行うことができます。内部リソースとはタスクが起動すると同時に自動取得されるリソースを指します。内部リソースを獲得すると、そのタスクが終了または自ら再スケジューリングを要求するまでは解放されない仕組みとなっています。以下に動作イメージを示します。



このような動作を実現したい場合は、次のように OIL ファイルを設定して下さい。

OIL ファイル記述例

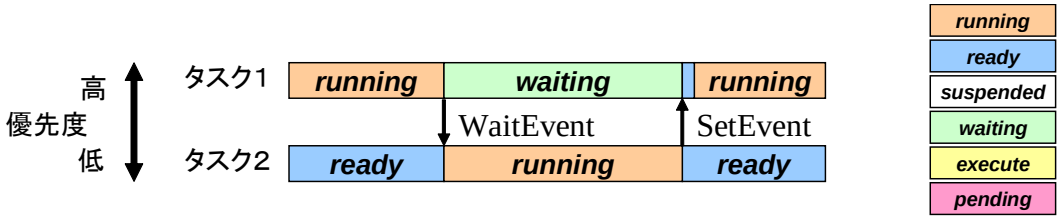
```
TASK タスク 1(任意) {
   AUTOSTART = FALSE;
   .....
};

TASK タスク 2(任意) {
   AUTOSTART = FALSE;
   ......
   RESOURCE = リソース 1;
};

TASK タスク 3(任意) {
   AUTOSTART = FALSE ;
   ......
   RESOURCE = リソース 1;
};

TASK タスク 4(任意) {
   AUTOSTART = TRUE {
      APPMODE = アプリケーションモード名;
   } ;
};

RESOURCE リソース 1(任意) {
   RESOURCEPROPERTY = INTERNAL ;
};
.....
```

リソース 1 へアクセスするタスクにリソースを割り当てます。また、タスク 2 と同等の優先度とさせるため、タスク 2 に割り当てます。

リソース 1 は内部リソースとするため、INTERNAL を指定します。

3.14. 排他制御（イベント使用）

TOPPERS Automotive Kernel ではイベントを使用することでタスク間の排他制御を行うことができます。イベント発生（同期待ち）をするためにイベントオブジェクトを使用します。以下に動作イメージを示します。



このような動作を実現したい場合は、以下のように OIL ファイルを設定して下さい。

OIL ファイル記述例

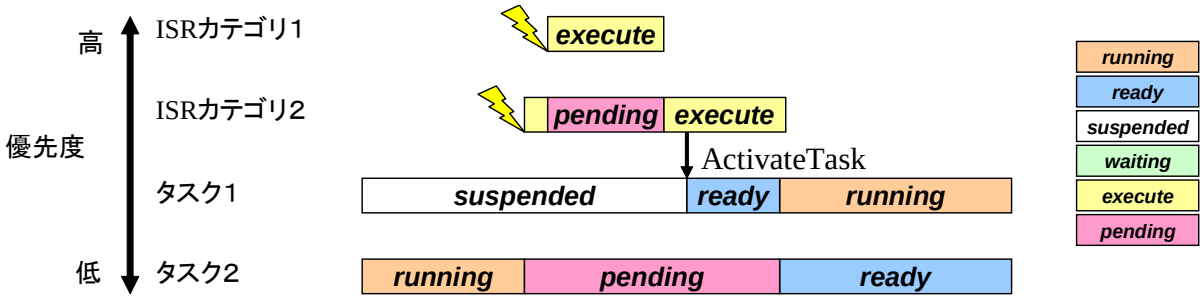
```
TASK タスク 1(任意) {
   AUTOSTART = TRUE {
      APPMODE = アプリケーションモード名;
   };
   EVENT = イベント名;
   ……
};

TASK タスク 2(任意) {
   AUTOSTART = FALSE;
   ……
};

EVENT イベント名(任意) {
  MASK = AUTO;
};
```

イベントを使用するタスクにイベントを割り当てます。

使用するイベントのマスク値を指定します。
マスク値を自動生成させる場合は AUTO を指定します。ユーザにて任意の値としたい場合はビットパターン値を指定します。

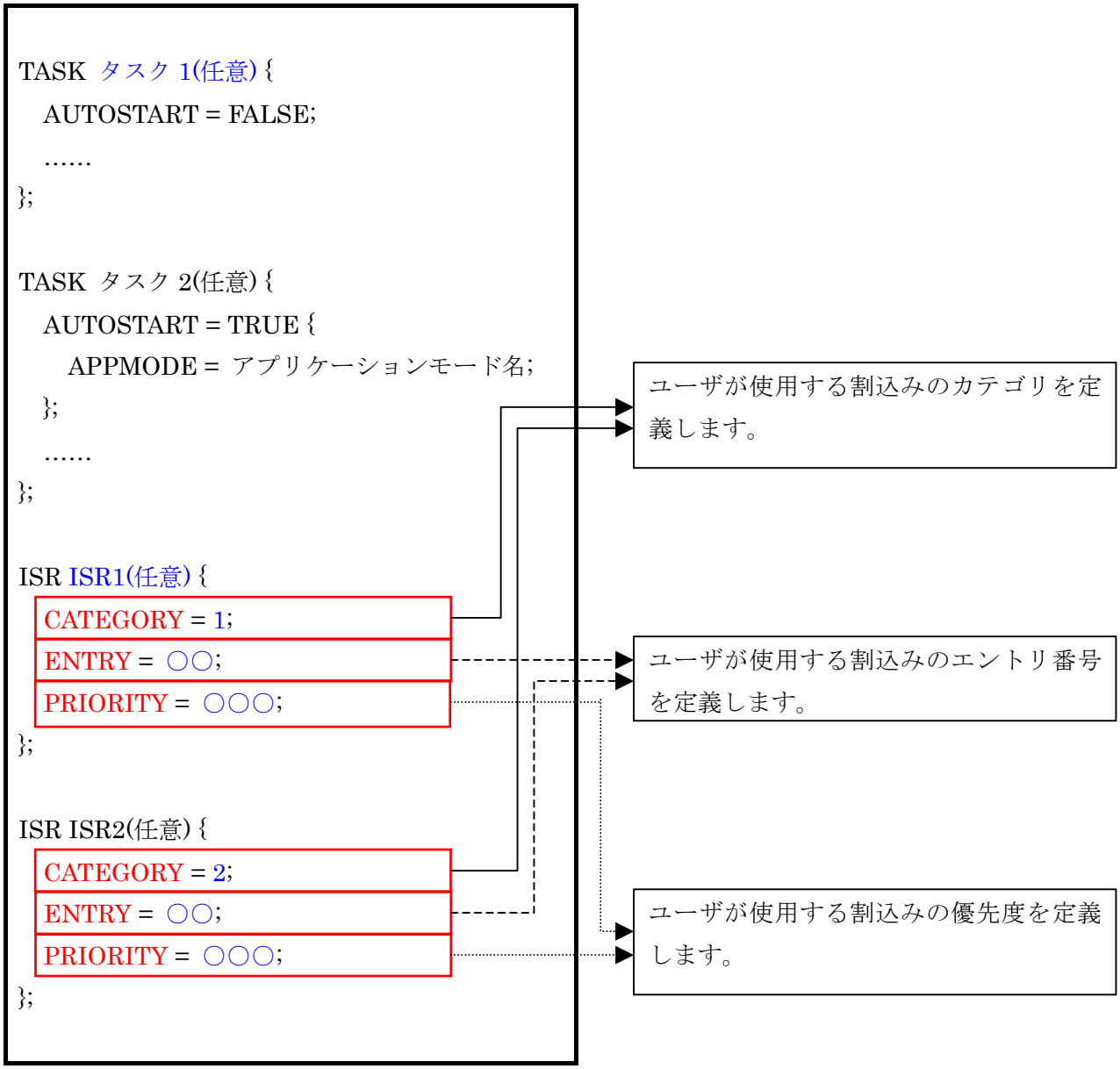
3.15. 割込み制御の実行方法

TOPPERS Automotive Kernel では割込みを使用することである要因によるタイミングで処理を実行することができます。動作イメージを以下に示します。



このような動作を実現したい場合は、以下のように OIL ファイルを設定して下さい。

OIL ファイル記述例


TASK タスク 1(任意) {
AUTOSTART = FALSE;
……
};
TASK タスク 2(任意) {
AUTOSTART = TRUE {
APPMODE = アプリケーションモード名;
};
……
};
ISR ISR1(任意) {
CATEGORY = 1;
ENTRY = ○○;
PRIORITY = ○○○;
};
ISR ISR2(任意) {
CATEGORY = 2;
ENTRY = ○○;
PRIORITY = ○○○;
};
ユーザが使用する割込みのカテゴリを定義します。
ユーザが使用する割込みのエントリ番号を定義します。
ユーザが使用する割込みの優先度を定義します。

4. OILファイル

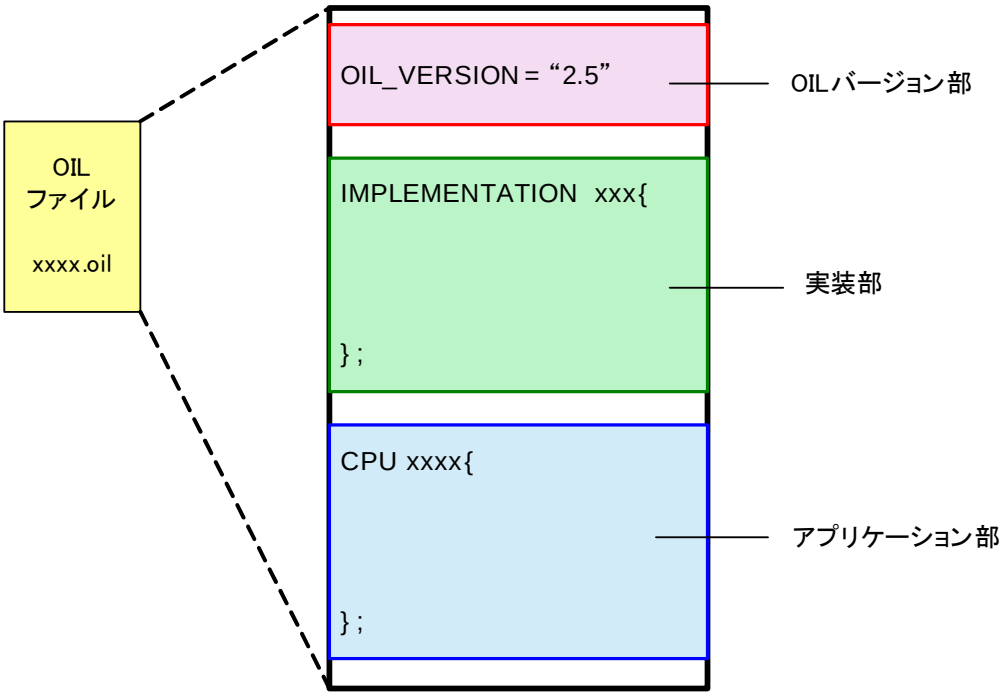
本 SG がサポートしている OIL ファイルは、「OSEK/VDX Implementation Language (OIL) Version 2.5」に準拠したものになります。

本項目では、サポートしている OIL オブジェクト概要についてのみ記載します。OIL オブジェクトについての詳細な仕様は「OSEK/VDX Implementation Language (OIL) Version 2.5」を参照して下さい。

4.1. OILファイル構造

OIL の記述は大きく分けて 3 部構成となっています。初めに準拠している OIL 仕様のバージョン情報を定義（OIL バージョン部）、次に標準的な実装仕様を示す定義（実装部）、最後に特定の CPU に配置されたアプリケーションの構造定義（アプリケーション部）です。

OIL ファイルの構造図を以下に示します。



OIL ファイルの文法ルールは BNF 表記を使用した文章で表されます。
全てのキーワード、属性、オブジェクト名、他の識別子は、大文字と小文字を識別します。
BNF 表記のコメントは、C++スタイルのコメントとしても書かれます。
また、インクルードファイルを参照可能な構成になっています。

4.2. OILバージョン部

本 SG では、OIL バージョン 2.5 に対応しています。
そのため使用する OIL ファイルの先頭に必ず次の一文を記載する必要があります。

```
OIL_VERSION = "2.5";
```

4.3. 実装部

実装部では、OIL オブジェクトに対して OIL 仕様に準拠している全ての属性と本 SG 独自拡張の属性を定義します。

実装部は、

```
IMPLEMENTATION　実装部名称　{　…　}
```

によって記述されます。

以下に、本 SG がサポートする OIL 仕様準拠であるオブジェクトと属性について記載します。下記位階の実装部記述でも解釈は可能ですが、エラー処理が実行されない場合があります。
標準属性の詳細に関しては、「OSEK/VDX Implementation Language (OIL) Version 2.5」を参照して下さい。

4.4. アプリケーション部

アプリケーション部では、ユーザアプリケーションの実装に合わせたオブジェクトの状態を記述します。

SG ではアプリケーション部の記述が次の場合エラーにし、処理を停止します。

- 実装部に存在しないオブジェクトや属性を指定した。
- 実装部に記述した属性の型に合わない初期値を指定した。
- 実装部に記述した属性の範囲外の初期値を指定した。
- 実装部に NO_DEFAULT と指定があるのに、初期値を指定しなかった。
- 参照型属性に、存在しないオブジェクトを指定した。

また、SG にて値の整合性のチェックや、出力時に関連のあるオブジェクトを参照し合いますが、その処理にて不整合な設定が発覚した場合もエラーとしています。

4.5. OIL記述について

OIL 記述は、OIL オブジェクトの集合で構成されており、OIL オブジェクトは OIL より明示的に定義されています。以下に、TOPPERS Automotive Kernel で使用する OIL オブジェクトを記載します。

オブジェクト名	説明
CPU	CPU は全てのオブジェクトのコンテナとして使用されます。厳密には、オブジェクトではなくコンテナです。
OS	TOPPERS Automotive Kernel のプロパティを定義するのに使用されるオブジェクト。CPU 内に 1 つの OS オブジェクトを定義しなければなりません。
APPMODE	アプリケーションの異なるモードの操作を定義するのに使用されるオブジェクトです。CPU 内に 1 つの APPMODE オブジェクトを定義しなければなりません。
TASK	OSEK タスクの定義をするために使用されるオブジェクトです。
ISR	割込みサービスルーチンを使用するオブジェクトです。
COUNTER	ALARM 機構の仕組みとして使用されるオブジェクトです。
ALARM	COUNTER をベースに、タスクの起動、イベントの設定、アラームコールバックルーチンの起動の何れかを行うのに使用されるオブジェクトです。
EVENT	タスクからのイベント発生の際に使用されるオブジェクトです。
RESOURCE	タスクや ISR によるリソースアクセスを調整するのに使用されるオブジェクトです。

また、本 SG がサポートする OIL で使用できる型は次の通りです。

型	説明
UINT32	32 ビット長符号無し整数
INT32	32 ビット長符号付き整数
UINT64	64 ビット長符号無し整数
INT64	64 ビット長符号付き整数
FLOAT	浮動小数
ENUM	離散的な集合
BOOLEAN	TRUE,FALSE の指定
STRING	文字列（OIL 標準）
SYMBOLNAME	文字列（本 SG 独自拡張、SG にてシンボル名として有効かのチェック有り）
DATATYPE	文字列（本 SG 独自拡張、SG にて型名として有効かのチェック有り）

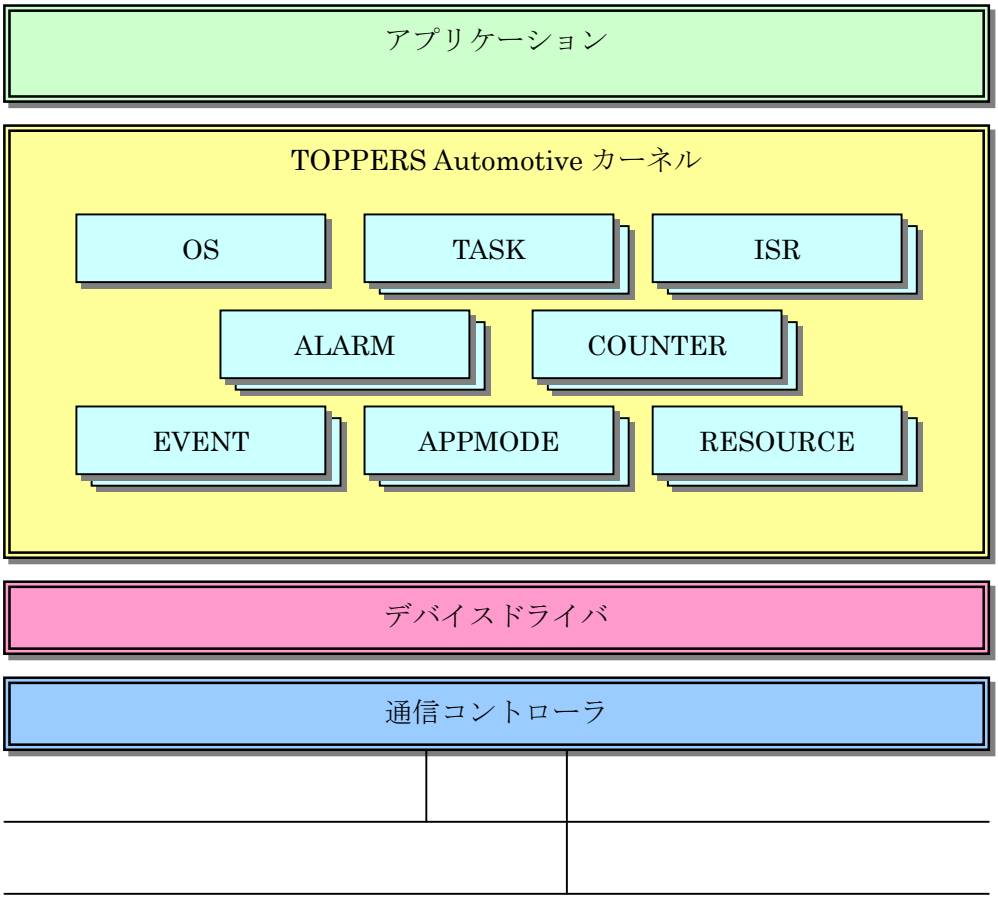
上記以外の型を指定した場合、OIL ファイルを認識することが出来ず出力ファイル生成エラーとなります。

各整数型では入力可能な範囲を以下のように定義します。
型名には UINT32,INT32,UINT64,INT64,FLOAT,ENUM の何れかが使用できます。

記述	説明
型名 [n..n+m]	指定された型名で、設定値は、n～n+m の値を許可します。 例： UINT32[0..20] VALUE UINT32 型で設定可能な範囲は 0～20
型名 [n,o,p,…..q]	指定された型名で、離散的な集合値を許可します。 例： UINT32[1,3,5,7] VALUE UINT32 型で設定可能な値は、1、3、5、7 の何れか

4.6. オブジェクト構成

オブジェクト構成図を下記に記載します。



次章より各オブジェクトの属性について説明を記載します。

4.6.1. OS

OS オブジェクトは、TOPPERS Automotive Kernel のプロパティを定義するのに使用されるオブジェクトです。CPU 内に必ず 1 つの OS オブジェクトを定義する必要があります。
OS オブジェクトの記述フォーマット及び各属性について記載します。

OS {			初期値
	ENUM	[STANDARD, EXTENDED] STATUS	= EXTENDED;
	BOOLEAN	STARTUPHOOK	= FALSE;
	BOOLEAN	ERRORHOOK	= FALSE;
	BOOLEAN	SHUTDOWNHOOK	= FALSE;
	BOOLEAN	PRETASKHOOK	= FALSE;
	BOOLEAN	POSTTASKHOOK	= FALSE;
	BOOLEAN	USEGETSERVICEID	= FALSE;
	BOOLEAN	USEPARAMETERACCESS	= FALSE;
	BOOLEAN	USERESSCHEDULER	= FALSE;
};			

4.6.1.1. STATUS

未サポートになります。STANDARD または EXTENDED の 2 種類が設定できますが、どちらを設定しても EXTENDED 固定となります。

4.6.1.2. STARTUPHOOK

スタートアップルーチンの設定を行います。スタートアップルーチンを使用する場合は TRUE を、使用しない場合は FALSE を設定します。

4.6.1.3. ERRORHOOK

エラーフックルーチンの設定を行います。エラーフックルーチンを使用する場合は TRUE を、使用しない場合は FALSE を設定します。

4.6.1.4. SHUTDOWNHOOK

シャットダウンフックルーチンの設定を行います。シャットダウンフックルーチンを使用する場合は TRUE を、使用しない場合は FALSE を設定します。

4.6.1.5. PRETASKHOOK

プレタスクフックルーチンの設定を行います。プレタスクフックルーチンを使用する場合は TRUE を、使用しない場合は FALSE を設定します。

4.6.1.6. POSTTASKHOOK

ポストタスクフックルーチンの設定を行います。ポストタスクフックルーチンを使用する場合は TRUE を、使用しない場合は FALSE を設定します。

4.6.1.7. USEGETSERVICEID

未サポートになります。TRUE または FALSE のどちらを指定しても TRUE 固定となります。

4.6.1.8. USEPARAMETERACCESS

未サポートになります。TRUE または FALSE のどちらを指定しても TRUE 固定となります。

4.6.1.9. USERESSCHEDULER

スケジューラリソースの設定を行います。スケジューラリソースを使用する場合は TRUE、使用しない場合は FALSE を設定します。

4.6.1.10. EXAMPLE

```
OS    os {
        STATUS                  = EXTENDED;
        STARTUPHOOK             = TRUE;
        ERRORHOOK               = FALSE;
        SHUTDOWNHOOK            = TRUE;
        PRETASKHOOK             = FALSE;
        POSTTASKHOOK            = FALSE;
        USEGETSERVICEID         = TRUE;
        USEPARAMETERACCESS      = TRUE;
        USERESSCHEDULER         = TRUE;
};
```

4.6.2. APPMODE

APPMODE オブジェクトは、アプリケーションの異なるモードの操作を定義するのに使用されるオブジェクトです。CPU 内に 1 つ以上の APPMODE オブジェクトを定義しなければなりません。

ユーザ定義のアプリケーションモードを使用せず、OS にて用意されたアプリケーションモードを使用することも可能です。その場合は、「OSDEFAULTAPPMODE」を定義し、他オブジェクトで使用することができます。

APPMODE オブジェクトの記述フォーマット及び各属性について記載します。

APPMODE {	説明
	サブ属性の設定は特にありません
};	

4.6.2.1. EXAMPLE

■APPMODE を定義しない場合（OSDEFAULTAPPMODE を使用する場合）

```
APPMODE   OSDEFAULTAPPMODE   {

};
```

■APPMODE を定義する場合

```
APPMODE   AppMode1   {

};
```

4.6.3. TASK

TASK オブジェクトは OSEK タスクの定義をするために使用されるオブジェクトです。EVENT 属性を付加した場合は拡張タスク*1 となり、付加しない場合は基本タスクとなります。
TASK オブジェクトの記述フォーマット及び各属性について記載します。

					初期値
TASK {					
	BOOLEAN [				
		TRUE {			
			APPMODE_TYPE	APPMODE[];	
		},			
		FALSE			
	] AUTOSTART				= FALSE;
	UINT32		[0..15] PRIORITY*2		= NO_DEFAULT;
	UINT32		[1..256] ACTIVATION*2		= NO_DEFAULT;
	ENUM		[NON, FULL] SCHEDULE		= NO_DEFAULT;
	EVENT_TYPE		EVENT[];*1		
	RESOURCE_TYPE		RESOURCE[];		
	UINT32		STACKSIZE		= 1024;
};					

※1　設定に関しては 4.6.3.5. EVENTを参照して下さい
※2　値の範囲はコンフォーマンスクラスによって異なります。4.6.3.2. PRIORITY、4.6.3.3. ACTIVATIONを参照して下さい。

4.6.3.1. AUTOSTART

タスクをシステム初期化時に起動する場合の設定を行います。タスクを自動起動させる場合は TRUE、自動機能させない場合は FALSE を設定します。また、TRUE の場合は、APPMODE サブ属性にて、どのアプリケーションモードで起動するか設定します。

4.6.3.2. PRIORITY

タスクの優先度（相対値）の設定を行います。最低優先度は 0、値が大きいほど高優先度となります。また、コンフォーマンスクラスにより、最高優先度が決められています。BCC1、BCC2 は 7、ECC1、ECC2 は 15 となります。
NO_DEFAULT 指定は初期値が設定されていないことを示します。そのため、必ず初期値を設定してください。

4.6.3.3. ACTIVATION

タスク多重起動の最大値を設定します。1 は多重起動なしを意味します。拡張タスクの場合は 1 以外を設定することはできません。また、コンフォーマンスクラスにより、タスク起動の最大値が決められています。BCC1、BCC2 は 1、ECC1、ECC2 は 256 となります。

本属性は NO_DEFAULT 指定になります。必ず初期値を設定してください。

4.6.3.4. SCHEDULE

スケジューリングの設定を行います。ノンプリエンプティブにするなら NON、フルプリエンプティブ設定にするなら FULL と設定します。

本属性は NO_DEFAULT 指定になります。必ず初期値を設定してください。

4.6.3.5. EVENT

拡張タスクが持つイベントの設定を行います。本設定を行うと、拡張タスクとなります。また、設定しない場合は、基本タスクとなります。ただし、本設定はコンフォーマンスクラスが ECC1、ECC2 のときのみ可能です。

4.6.3.6. RESOURCE

使用リソースの設定を行います。設定しないことも可能です。

4.6.3.7. STACKSIZE

スタックサイズの設定を行います。初期値は 1024 になります。

4.6.3.8. EXAMPLE

■ 基本タスクの定義方法

```
TASK    Task1 {
        AUTOSTART = TRUE {
            APPMODE         = AppMode1;
        };
        PRIORITY        = 8;
        ACTIVATION      = 5;
        SCHEDULE        = NON;
        RESOURCE        = resource1;
        STACKSIZE       = 1024;
};
```

■ 拡張タスクの定義方法

```
TASK    Task2 {
        AUTOSTART       = FALSE;
        PRIORITY        = 7;
        ACTIVATION      = 1;
        SCHEDULE        = FULL;
        EVENT           = event1;
        STACKSIZE       = 512;
};
```

4.6.4. ISR

ISR オブジェクトは、割込みサービスルーチンを使用するオブジェクトです。

ISR オブジェクトの記述フォーマット及び各属性について記載します。

ISR			初期値
	UINT32	[1,2] CATEGORY	= NO_ DEFAULT;
	UINT32	PRIORITY	= NO_ DEFAULT;
	UINT32	ENTRY	= NO_ DEFAULT;
	RESOURCE_TYPE	RESOURCE[];	
};			

4.6.4.1. CATEGORY

ISR のカテゴリを設定します。カテゴリ 1 の割込みを使用する場合は 1 、カテゴリ 2 の割込みを使用する場合は 2 を設定します。

本属性は NO_DEFAULT 指定になります。必ず初期値を設定して下さい。

4.6.4.2. PRIORITY

ISR の割込み優先レベルを設定します。TOPPERS Automotive 拡張になります。

カテゴリ 1 とカテゴリ 2 の ISR を両方定義する際には、カテゴリ 1 の割込み優先度レベルは、カテゴリ 2 の ISR の割込み優先レベルの最高値より高い値を設定して下さい。

本属性は NO_DEFAULT 指定になります。必ず初期値を設定して下さい。

4.6.4.3. ENTRY

ISR を付加する割込み番号を設定します。TOPPERS Automotive 拡張になります。

本属性は NO_DEFAULT 指定になります。必ず初期値を設定して下さい。

4.6.4.4. RESOURCE

ISR が獲得するリソースを指定します。使用しないことも可能です。

4.6.4.5. EXAMPLE

■ISR カテゴリ 1 を定義する場合

ISR isr1		
	CATEGORY	= 1;
	PRIORITY	= 7;
	ENTRY	= 18;
};		

■ISR カテゴリ 2 を定義する場合

ISR isr2		
	CATEGORY	= 2;
	PRIORITY	= 5;
	ENTRY	= 18;
	RESOURCE	= resource1;
};		

4.6.5. COUNTER

COUNTER オブジェクトは、ALARM 機構の仕組みとして使用されるオブジェクトです。
COUNTER オブジェクトの記述フォーマット及び各属性について記載します。

COUNTER {			初期値
	UINT32	MINCYCLE	= NO_DEFALT;
	UINT32	[1..2147483647] MAXALLOWEDVALUE	= NO_DEFALT;
	UINT32	TICKSPERBASE	= NO_DEFALT;
};			

4.6.5.1. MINCYCLE

周期として指定できる最小値を設定します。
TICKSPERBASE 属性に指定した値以上で、MAXALLOWEDVALUE 属性に指定した値以下のみが有効になります。
本属性は NO_DEFAULT 指定になります。必ず初期値を設定して下さい。

4.6.5.2. MAXALLOWEDVALUE

カウンタの最大値を設定します。設定可能範囲は 1～2147483647（1～0x7FFFFFFF）です。
本属性は NO_DEFAULT 指定になります。必ず初期値を設定して下さい。

4.6.5.3. TICKSPERBASE

カウンタごとの 1 単位に達するまでのティックを設定します。ただし、MAXALLOWEDVALUE 以下の値のみが設定可能です。
本属性は NO_DEFAULT 指定になります。必ず初期値を設定して下さい。

4.6.5.4. EXAMPLE

COUNTER　counter {		
	MINCYCLE	= 1;
	MAXALLOWEDVALUE	= 999;
	TICKSPERBASE	= 1;
};		

4.6.6. ALARM

ALARM オブジェクトは、COUNTER をベースに、タスクの起動、イベントの設定、アラームコールバックルーチンの起動の何れかを行うのに使用されるオブジェクトです。

ALARM オブジェクトの記述フォーマット及び各属性について記載します。

ALARM {				初期値
	COUNTER_TYPE		COUNTER;	
	ENUM [			
		ACTIVATETASK {		
			TASK_TYPE TASK;	
		},		
		SETEVENT {		
			TASK_TYPE TASK;	
			EVENT_TYPE EVENT;	
		},		
		ALARMCALLBACK {		
			STRING ALARMCALLBACKNAME	= NO_DEFAULT;
		}		
	] ACTION			= NO_DEFAULT;
	BOOLEAN [			
		TRUE {		
			UINT32 ALARMTIME	= NO_DEFAULT;
			UINT32 CYCLETIME	= NO_DEFAULT;
			APPMODE_TYPE APPMODE[];	
		},		
		FALSE		
	] AUTOSTART;			= FALSE;
};				

4.6.6.1. COUNTER

ALARM に割り付けるカウンタを設定します。

4.6.6.2. ACTION

アラームが満了した際の動作を設定します。
「ACTIVATETASK」「SETEVENT」「ALARMCALLBACK」のうちどれか１つを設定します。
以下に各属性を設定した場合の動作を示します。
本属性は NO_DEFAULT 指定になります。必ず初期値を設定してください。

設定値	動作
ACTIVATETASK	アラーム満了時にタスクを起動します。サブ属性の TASK にて起動するタスク名を指定して下さい。
SETEVENT	アラーム満了時にイベントをセットします。サブ属性にて起動するタスクとセットするイベント名を指定してください。
ALARMCALLBACK	アラーム満了時にコールバックを起動します。サブ属性にて、起動するコールバック名を指定して下さい。

4.6.6.3. AUTOSTART

アラームをシステム初期化時に起動するか否かを設定します。初期化時に起動する場合は TRUE、起動しない場合は FALSE を設定して下さい。また、TRUE の場合は以下の属性を設定して下さい。
本属性の初期値は FALSE となります。

- **ALARMTIME**

  初回満了時のカウンタ値を設定します。0 は設定不可となります。また、COUNTER オブジェクトの MAXALLOWEDVALUE 属性に指定した値以下を設定して下さい。

- **CYCLETIME**

  繰り返し expire するカウンタ値を設定します。アラームが付加されているカウンタの MINCYCLE 属性に指定した値以上で、MAXALLOWEDVALUE 属性に指定した値以下を設定して下さい。

- **APPMODE**

  どのアプリケーションで起動するかを設定して下さい。

4.6.6.4. EXAMPLE

■アラーム満了時にタスクを起動する場合

```
ALARM    acttask_alarm {
            COUNTER                 = counter1;
            ACTION = ACTIVATETASK {
                TASK                = Task1;
            };
            AUTOSTART = TRUE {
                ALARMTIME           = 10;
                CYCLETIME           = 100;
                APPMODE             = AppMode1;
            };
};
```

■アラーム満了時にイベントをセットする場合

```
ALARM    setevt_alarm {
            COUNTER                 = counter1;
            ACTION = SETEVENT{
                TASK                = Task1;
                EVENT               = event1;
            };
            AUTOSTART               = FALSE;
};
```

■アラーム満了時にコールバックを起動する場合

```
ALARM    setevt_alarm {
                        COUNTER                 = counter1;
                        ACTION = ALARMCALLBACK {
                            ALARMCALLBACKNAME   = alarm_cb;
                        };
                        AUTOSTART = TRUE {
                            ALARMTIME           = 10;
                            CYCLETIME           = 100;
                            APPMODE             = AppMode1;
                        };
};
```

4.6.7. EVENT

EVENT オブジェクトはタスクからのイベント発生の際に使用されるオブジェクトです。
EVENT オブジェクトの記述フォーマット及び各属性について記載します。

EVENT {			初期値
	UINT32 WITH_AUTO	MASK	= AUTO;
};			

4.6.7.1. MASK

イベントのマスク値を設定します。SG にて自動割付する場合には AUTO を設定して下さい。具体的な数値を指定することも可能です。

4.6.7.2. EXAMPLE

■マスク値を手動で割付する場合

EVENT event1 {	
MASK	= 0x01;
};	

■マスク値を自動割付する場合

EVENT event2 {	
MASK	= AUTO;
};	

4.6.8. RESOURCE

RESOURCE オブジェクトは、タスクや ISR によるリソースアクセスを調整するのに使用されるオブジェクトです。

RESOURCE オブジェクトの記述フォーマット及び各属性について記載します。

RESOURCE {			初期値
	ENUM [		
		STANDARD,	
		LINKED {	
		RESOURCE_TYPE LINKEDRESOURCE;	
		},	
		INTERNAL	
	] RESOURCEPROPERTY		= NO_DEFAULT;
};			

4.6.8.1. RESOURCEPROPERTY

リソースの属性を指定します。

「STANDARD」「LINKED」「INTERNAL」のうちどれか１つを設定します。また、本属性は NO_DEFAULT 指定のため、必ず初期値を設定して下さい。

設定値	動作
STANDARD	標準リソースを使用します。
LINKED	リンクリソースを使用します。この場合、サブ属性にリンクするリソース名を指定して下さい。
INTERNAL	内部リソースを使用します。

4.6.8.2. EXAMPLE

■ 標準リソースを使用する場合

```
RESOURCE    std_resource {
    RESOURCEPROPERTY    = STANDARD;
};
```

■ リンクリソースを使用する場合

```
RESOURCE    link_resource {
    RESOURCEPROPERTY    = LINKED {
        LINKEDRESOURCE    = resource1;
    };
};
```

■ 内部リソースを使用する場合

```
RESOURCE    intr_resource {
    RESOURCEPROPERTY    = INTERNAL;
};
```

5. テンプレートファイル

SG では、ターゲット依存情報（ベクタテーブル情報など）が記載されたテンプレートファイルを読み込むことで、ターゲット依存情報をカーネル構成ファイルに出力することが可能です。
テンプレートファイルにはマクロ言語で任意の依存部情報を記載します。
マクロ言語については次章にて説明します。

5.1. SG使用情報

最高/最低プライオリティ、最大/最低エントリ数、エントリ間隔の指定を行います。

・テンプレート記述例

```
@@ISR_MIN_PRIORITY=1@@
@@ISR_MAX_PRIORITY=7@@
@@ISR_MIN_ENTRY=0@@
@@ISR_MAX_ENTRY=71@@
@@ISR_ENTRY_INTERVAL=1@@
```

5.2. ベクタテーブル登録シンボル外部参照

ベクタテーブル登録シンボル外部参照に関連する情報はテンプレート情報を必要としないため、下記のサンプルに記述されるように FOR_EACH の記述をして下さい。以下の記述があった場合に、割込み入り口処理生成マクロを OIL で指定した ISR の数だけ出力します。

・テンプレート記述例

```
@@FOR_EACH EXTERNAL_SYMBOL_FOR_ISR@@
```

5.3. ベクタエントリ

ベクタエントリの個数分、「@@INT_ENTRY｛エントリ番号｝@@　/* コメント */」という記述があります。

・テンプレート記述例

```
@@INT_ENTRY0@@; /*   0, +0x00: BRK 命令          */
@@INT_ENTRY1@@; /*   1, +0x04: 予約領域          */
@@INT_ENTRY2@@; /*   2, +0x08: 予約領域          */
```

5.4. フックルーチン

ターゲット依存部ヘッダファイル（巻末参照）に記載するフックルーチンの名を記述する。

・テンプレート記述例

```
@@NULL_ERRORHOOK_SYMBOL@@
@@NULL_STARTUPHOOK_SYMBOL@@
@@NULL_SHUTDOWNHOOK_SYMBOL@@
@@NULL_PRETASKHOOK_SYMBOL@@
@@NULL_POSTTASKHOOK_SYMBOL@@
```

5.5. テンプレートデータ読み込み

テンプレートファイル内でインクルードファイル指定が可能です。

```
＃include　xxxx
```

5.6. 指定ファイル出力範囲指定

テンプレートファイル内で下記に示す記述で範囲指定を行うことにより、–odep=<file>オプションで指定した<file>に出力することが可能です。

・テンプレート記述例

```
@@START_OUTPUT_DEPENDENT@@
…指定ファイル出力内容
@@END_OUTPUT_DEPENDENT@@
```

5.7. 割込み入り口処理

割込み入り口処理に関連する情報はテンプレート情報を必要としないため、下記のサンプルに記述されるように FOR_EACH の記述をしてください。以下の記述があった場合に、割込みシンボルマクロを OIL で指定した ISR の数だけ出力します。また、指定ファイル出力範囲指定内に本記述が含まれていた場合には-odep=<file>オプションで指定した<file>にアセンブラ形式で出力され、範囲外の場合には C 形式で出力されます。

・ テンプレート記述例

```
@@FOR_EACH ENTRY_FOR_ISR@@
```

6. OILファイルの解釈方法

SG では OIL ファイルを入力とし構文解析を行います。
OIL ファイルは、OIL バージョン部(OIL_VERSION)、実装部(IMPLEMENTATION)、アプリケーション部(CPU)で構成されています。OIL バージョン部、実装部、アプリケーション部の順に構文解析を行います。

6.1. OILバージョン部

OIL バージョン部の解析では、OIL のバージョンが”2.5”となっていることをチェックします。これ以外のバージョンや文字など不適切な指定、また OIL_VERSION の指定が無かった場合は、エラーとし処理を中断します。

6.2. 実装部

実装部の解析では、オブジェクト毎に属性情報の管理を行います。
まず、

```
オブジェクト種別 {
  .......
```

の記述を検出したら、それが OIL でサポートされているオブジェクト(OS、APPMODE、TASK、ISR、COUNTER、ALARM、EVENT、RESOURCE)か否かを判別し、適合した場合その属性の解析に入ります。

属性の解析では

- 属性名
- 型
- 範囲
- デフォルト値
- 説明文

を管理します。

6.3. アプリケーション部

アプリケーション部の解析では、生成するオブジェクトの情報をチェックします。

```
オブジェクト種別　　オブジェクト名 {
```

を解析したら、以降そのオブジェクトの属性について解析します。
以下にアプリケーション部の具体例を示します。

例

```
TASK　　task1 {
```

存在する属性か否か、又は属性の値が範囲内か否かを実装部の解析結果と照らし合わせてチェックします。それが OK であれば、その属性の値を確定します。

ただし、値が AUTO であれば、実装部に WITH_AUTO の記述があることをチェックした後、SG 内で自動的に値を割り付けます。値の割り付けは属性によって異なります。尚、本 SG では、EVENT オブジェクトの MASK 以外は WITH_AUTO をサポートしていないため、自動割付はされません。

SGを実行すると、構文解釈処理により、OILファイルの記述内容によっては、エラーが出力される場合があります。この場合、SGはC言語コードファイルの生成処理を中断し、中断した原因を示すエラーメッセージが表示されます。9.エラーメッセージ にエラーメッセージの一例と表示された場合の対処方法を記載します。

7. 出力ファイル

本 SG は、システムコンフィギュレーションファイル（OIL ファイル）を処理して、各種パラメータ設定ファイルを生成します。各ファイルは、OIL 記述をもとにコンフィギュレータが TOPPERS Automotive Kernel で必要な各種配列・値を C 言語ベースで出力したファイルです。

本 SG が出力するファイルを以下に示します。

ファイル名	説明
kernel_cfg.c	カーネル構成ファイルと呼びます。 OIL 記述を元に TOPPERS Automotive Kernel で必要な各種データブロックの宣言と初期化設定を行います。
kernel_id.h	ID 自動割り付け結果ファイルと呼びます。 TOPPERS Automotive Kernel で使用するオブジェクト ID 値の定義やフックルーチンを使用する場合に定義を出力します。

8. 出力内容

本 SG を実行した際に生成されるファイルの内容について示します。青字で示されている部分は、ユーザが OIL ファイルに記述した情報に依存する箇所（オブジェクトの名称、属性設定値など）を示します。

8.1. オペレーティングシステム（OS）

8.1.1. インクルードファイル

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_cfg.c
OS	－	－	－	
概要				
TOPPERS Automotive Kernel において、使用するインクルードファイルを出力します。				
出力内容				
#include " osek_kernel.h" #include “kernel_id.h”				

8.1.2. スタートアップルーチン

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_id.h
OS	BOOLEAN	STARTUPHOOK	TRUE	
概要				
スタートアップルーチンの定義を行います。 TOPPERS Automotive Kernel の起動前のタイミングにて、ユーザ処理を実行させたい場合、初期化ルーチンとして使用可能です。ただし、STARTUPHOOK が TRUE の場合のみ出力します。				
出力内容				
#define USE_ STARTUPHOOK				

8.1.3. シャットダウンフックルーチン

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_id.h
OS	BOOLEAN	SHUTDOWNHOOK	TRUE	
概要				
シャットダウンフックルーチンの定義を行います。 TOPPERS Automotive Kernel 終了時のタイミングにて、ユーザ処理を実行させたい場合、終了処理ルーチンとして使用可能です。ただし、SHUTDOWNHOOK が TRUE の場合のみ出力します。				
出力内容				
#define USE_ SHUTDOWNHOOK				

8.1.4. エラーフックルーチン

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_id.h
OS	BOOLEAN	ERRORHOOK	TRUE	
概要				
エラーフックルーチンの定義を行います。 システムサービスが実行エラーとなった場合のタイミングにて、ユーザ処理を実行したい場合、使用可能です。ただし、ERRORHOOK が TRUE の場合のみ出力します。				
出力内容				
#define USE_ ERRORHOOK				

8.1.5. プレタスクフックルーチン

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_id.h
OS	BOOLEAN	PRETASKHOOK	TRUE	
概要				
プレタスクフックルーチンの定義を行います。 タスクの状態が「実行状態」になった後で、ユーザ処理を実行したい場合に使用可能です。 ただし、PRETASKHOOK が TRUE の場合のみ出力します。				
出力内容				
#define USE_ PRETASKHOOK				

8.1.6. ポストタスクフックルーチン

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_id.h
OS	BOOLEAN	POSTTASKHOOK	TRUE	

概要
プレタスクフックルーチンの定義を行います。 TOPPERS Automotive Kernel において、タスクの状態が「実行状態」でなくなる前に、ユーザ処理を実行したい場合に使用可能です。 ただし、POSTTASKHOOK が TRUE の場合のみ出力します。

出力内容
#define USE_ POSTTASKHOOK

8.1.7. スケジューラリソース

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
OS	BOOLEAN	USERESSCHEDULER	TRUE / FALSE	

概要
スケジューラリソースの定義を行います。 リソースの上限優先度を保持するデータブロック”resinib_ceilpri[]”の要素 0 番目に出力します。 ただし、USERESSCHEDULER が TRUE の場合のみ出力します。FALSE の場合は 0 を出力します。

出力内容

const Priority resinib_ceilpri[TNUM_RESOURCE] = {
　TPRI_SCHEDULER, …
};

8.2. アプリケーションモード（APPMODE）

8.2.1. アプリケーションモード

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_id.h
APPMODE	－	－	－	
概要				
アプリケーションモードの定義を行います。 ただし、OSDEFAULTAPPMODE に関しては OS 内部にて用意されているため出力されません。				
出力内容				
#define アプリケーションモード名（1UL << アプリケーションモードの ID）				

8.3. タスク管理機能（TASK）

8.3.1. インクルードファイル

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
TASK オブジェクトで使用するインクルードファイルを出力します。				
出力内容				
#include "task.h"				

8.3.2. タスクID

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
TASK オブジェクトの ID を出力します。 ID は 0 から順にユニークに割り付けられます。ただし、拡張タスクを定義した場合、拡張タスクには基本タスクより小さな ID 値を割り付けます。言い換えると、まず拡張タスクに 0 から順にタスク ID を割り付け、その後に基本タスクにタスク ID を割り付けます。				
出力内容				
const TaskType TASK オブジェクト名 = タスク ID ;				

8.3.3. タスク関数定義

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
アプリケーションコードで使用するタスクの関数定義を行います。				
出力内容				
extern void TASKNAME(タスク名)(void);				

8.3.4. タスク数

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
タスク数と拡張タスクの数を出力します。				
出力内容				

#define TNUM_TASK 全タスク数
#define TNUM_EXTTASK 拡張タスク数
const UINT8 tnum_task = TNUM_TASK;
const UINT8 tnum_exttask = TNUM_EXTTASK;				

8.3.5. タスクの初期優先度

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	UINT32	PRIORITY	－	
概要				
各タスクの初期優先度（TPRI_MINTASK + PRIORITY 属性に指定した値）を出力します。				
出力内容				

const Priority tinib_inipri[TNUM_TASK] = {
TPRI_MINTASK + TASK オブジェクトの PRIORITY 属性値, …
};				

8.3.6. 実行開始直後の優先度

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	UINT32	PRIORITY	－	
	ENUM	SCHEDULE	NON/FULL	

概要

各タスクの実行開始直後の優先度を出力します。
SCHEDULE 属性での設定により、出力マクロ名が異なります。
NON の場合：TPRI_MAXTASK
FULL の場合：TPRI_MINTASK + PRIORITY 属性値
また、内部リソースが付加されている場合は、それらのタスクの上限優先度を出力します。

出力内容

```
const Priority tinib_exepri [ TNUM_TASK ] = {
   TPRI_MAXTASK ,                                          /* NON の場合  */
   TPRI_MINTASK + TASK オブジェクトの PRIORITY 属性値,     /* FULL の場合  */
   … ,
};
```

8.3.7. 多重起動要求キューイング数の最大値

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	UINT32	ACTIVATION	－	

概要

各タスクの多重起動要求キューイング数の最大値（ACTIVATION 属性に指定した値－1）を出力します。ただし、コンフォーマンスクラスが BCC2,ECC2 のときのみ出力します。

出力内容

```
const UINT8 tinib_maxact[ TNUM_TASK ] = {
   (ACTIVATION 属性に指定した値) － 1 , …
};
```

8.3.8. 起動するアプリケーションモード

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	BOOLEAN	AUTOSTART	TRUE	
	APPMODE_TYPE	APPMODE[]	－	

概要
各タスクをシステム初期化時に起動するアプリケーションモードを出力します。 アプリケーションモードの値は、AUTOSTART 属性が FALSE の時は 0 に、TRUE の場合は APPMODE サブ属性に指定されたアプリケーションモードの ID 値の論理和とします。

出力内容

const AppMode tinib_autoact [TNUM_TASK] = {
　指定したアプリケーションモードの値, …
};

8.3.9. タスクの起動番地

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	

概要
各タスクの起動番地を出力します。 タスクの起動番地は、タスクのエントリ関数の名称を「TASKNAME(タスク名)」の形で列挙します。

出力内容

const FP tinib_task [TNUM_TASK] = {
　TASKNAME(タスク名) , …
};

8.3.10. スタック領域の定義

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	UINT32	STACKSIZE	－	
概要				
各タスクのスタック領域の定義を出力します。				
出力内容				
static __STK_UNIT _stack_タスク名[__TCOUNT_STK_UNIT(設定した STACKSIZE 値)];				

8.3.11. スタック領域の先頭番地

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
各タスクのスタック領域の先頭番地（最も小さいアドレス）を出力します。				
出力内容				

const VP tinib_stk [TNUM_TASK] = {
(__STK_UNIT)_stack_タスク名, ...
};				

8.3.12. スタック領域のサイズ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	UINT32	STACKSIZE	－	
概要				
各タスクのスタック領域のサイズを出力します。 複数のタスクでスタック領域を共有する場合には、最も大きいスタック領域を必要とするタスクに合わせて出力します。				
出力内容				

const UINT16 tinib_stksz [TNUM_TASK] = {
STACKSIZE 属性値に設定した値, ...
};				

8.3.13. タスクコンテキストブロック

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
タスクコンテキストブロックを出力します。				
出力内容				
DEFINE_CTXB(TNUM_TASK);				

8.3.14. タスク状態保持バッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
タスク状態保持バッファを出力します。				
出力内容				
UINT8 tcb_tstat [TNUM_TASK];				

8.3.15. 現在優先度保持バッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
現在の優先度を保持するバッファを出力します。				
出力内容				
Priority tcb_curpri[TNUM_TASK];				

8.3.16. タスクキュー内で次に呼ばれるタスク

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
タスクキュー内で次に呼ばれるタスクを保持するバッファを出力します。 ただし、コンフォーマンスクラスが BCC2、ECC1、ECC2 のときのみ出力します。				
出力内容				
TaskType tcb_next[TNUM_TASK];				

8.3.17. タスクの起動要求数保持バッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
タスクの起動要求数を保持するバッファを出力します。 ただし、コンフォーマンスクラスが BCC2、ECC1、ECC2 のときのみ出力します。				
出力内容				
UINT8 tcb_actcnt[TNUM_TASK];				

8.3.18. 現在のイベント保持バッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
現在のイベント保持バッファを出力します。 ただし、コンフォーマンスクラスが ECC1、ECC2 のときのみ出力します。				
出力内容				
EventMaskType tcb_curevt[TNUM_EXTTASK];				

8.3.19. 待ちイベント保持バッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
待ちイベント保持バッファを出力します。 ただし、コンフォーマンスクラスが ECC1、ECC2 のときのみ出力します。				
出力内容				
EventMaskType tcb_waitevt[TNUM_EXTTASK];				

8.3.20. 最後に取得したリソースID

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
TASK	－	－	－	
概要				
最後に取得したリソース ID を保持するバッファを出力します。 ただし、コンフォーマンスクラスが BCC2、ECC1、ECC2 のときのみ出力します。				
出力内容				
ResourceType tcb_lastres[TNUM_TASK];				

8.4. 割込み管理機能（ISR）

8.4.1. インクルードファイル

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ISR	－	－	－	
概要				
ISR オブジェクトで使用するインクルードファイルを記述します。				
出力内容				
#include " interrupt.h"				

8.4.2. ISRカテゴリ 2 のISR ID

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ISR	UINT32	CATEGORY	2	
概要				
カテゴリ 2 の ISR オブジェクトの ID を出力します。ID には 0 から始まるユニークな ID を割り付けます。				
出力内容				
const IsrType ISR カテゴリ 2 のオブジェクト名 = ISRID;				

8.4.3. ISRカテゴリ 2 の定義数

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ISR	UINT32	CATEGORY	2	
概要				
カテゴリ 2 の ISR の数を出力します。				
出力内容				
#define TNUM_ISR2　カテゴリ 2 の ISR 数 const UINT8 tnum_isr2 = TNUM_ISR2;				

8.4.4. ISRカテゴリ 1 のエントリ定義

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ISR	UINT32	CATEGORY	1	
概要				
ISR カテゴリ 1 のエントリ定義を出力します。 カテゴリ 1 では、ISR オブジェクト名称のみを出力します。				
出力内容				
ISR1 ENTRY(カテゴリ 1 の ISR オブジェクト名);				

8.4.5. ISRカテゴリ 2 のエントリ定義

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ISR	UINT32	CATEGORY	2	
概要				
ISR カテゴリ 2 のエントリ定義を出力します。 カテゴリ 2 では、ISR オブジェクト名称と ISRID を出力します。				
出力内容				
ISR2_ENTRY(カテゴリ 2 の ISR オブジェクト名 ,(ISRID));				

8.4.6. ISRカテゴリ 2 のISR割込み優先レベル

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ISR	UINT32	CATEGORY	2	
	UINT32	PRIORITY	－	
概要				
カテゴリ 2 の ISR の割込み優先レベル（PRIORITY 属性に指定した値）の最大値を求め、以下のマクロ及び定数に出力します。 カテゴリ 1 の ISR の割込み優先レベルは、この値よりも大きくなければなりません。そうでない場合は、SG がエラーを報告します。				
出力内容				
#define IPL_MAXISR2　　ISR カテゴリ 2 の ISR 割込み優先レベル const IPL ipl_maxisr2 = IPL_MAXISR2;				

8.4.7. ISRカテゴリ 2 の割込み優先度

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ISR	UINT32	CATEGORY	2	
	UINT32	PRIORITY	－	
概要				
カテゴリ 2 の各 ISR の割込み優先度（TPRI_MINISR + PRIORITY 属性に指定した値）を出力します。ただし、コンフォーマンスクラスが BCC2、ECC1、ECC2 のときのみ出力します。				
出力内容				

const Priority isrinib_intpri [TNUM_ISR2] = {
TPRI_MINISR + (PRIORITY 属性に設定された値), ……
};				

8.4.8. ISR獲得リソースバッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ISR	－	－	－	
概要				
ISR が獲得したリソースを保持するバッファを出力します。 ただし、コンフォーマンスクラスが、BCC2、ECC1、ECC2 のときのみ出力します。				
出力内容				
ResourceType　isrcb_lastres[TNUM_ISR2];				

8.5. リソース管理機能（RESOURCE）

8.5.1. インクルードファイル

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
RESOURCE	－	－	－	
概要				
RESOURCE オブジェクトで使用するインクルードファイルを出力します。				
出力内容				
#include "resource.h"				

8.5.2. リソースID

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
RESOURCE	－	－	－	
概要				
リソース ID を出力します。ID には 0 から始まるユニークな ID を割り付けます。 ただし、OS オブジェクトの USERESSCHEDULER 属性が TRUE の場合は、ID0 は予約 ID となるため、1 から割り付けます。FALSE の場合は、通常通り 0 から割り付けます。				
出力内容				
const ResourceType リソースオブジェクト名 = リソース ID;				

8.5.3. リソース数

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
RESOURCE	－	－	－	
概要				
リソースオブジェクトの数を出力します。				
出力内容				
#define TNUM_RESOURCE リソースオブジェクト数 const UINT8 tnum_resource = TNUM_RESOURCE;				

8.5.4. リソースの上限優先度

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
RESOURCE	－	－	－	

概要

各リソースの上限優先度を出力します。
OS オブジェクトの USERESSCHEDULER 属性が TRUE の場合には、要素の 0 番目に”TPRI_SCHEDULER” を出力します。
リソースの上限優先度は、

- そのリソースにアクセスするタスクの中での最大優先度
- または、そのリソースにアクセスするカテゴリ 2 の ISR の中での最大優先度

のことを指します。
ただし、タスクと ISR 両方からアクセスする場合は ISR の最大優先度を取ります。
各リソースをアクセスするタスク／カテゴリ 2 の ISR は、タスク／ISR オブジェクトの RESOURCE 属性で知ることができます。
タスクの優先度をとる場合は、「TPRI_MINTASK + アクセスするタスクの最大優先度」、
ISR の優先度をとる場合は、「TPRI_MINISR + アクセスする ISR カテゴリ 2 の最大優先度」を出力します。ただし、コンフォーマンスクラスが BCC2、ECC1、ECC2 の場合のみ出力します。

出力内容

```
const Priority   resinib_ceilpri[TNUM_RESOURCE] = {
    TPRI_SCHEDULER,
    TPRI_MINTASK + アクセスするタスクの最大優先度 ,
    TPRI_MINISR + アクセスする ISR カテゴリ 2 の最大優先度 ,
    ……
};
```

8.5.5. リソース獲得前の優先度保持バッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
RESOURCE	－	－	－	
概要				
リソースを獲得する前の優先度を保持するバッファを出力します。				
出力内容				
Priority　rescb_prevpri[TNUM_RESOURCE];				

8.5.6. 前回獲得リソース保持バッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
RESOURCE	－	－	－	
概要				
前回獲得したリソースを保持するバッファを出力します。				
出力内容				
ResourceType　rescb_prevres[TNUM_RESOURCE];				

8.6. カウンタ管理機能（COUNTER）

8.6.1. カウンタID

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
COUNTER	－	－	－	
概要				
COUNTER オブジェクトの ID を出力します。ID には 0 から始まるユニークな ID を割り付けます。				
出力内容				
const CounterType COUNTER オブジェクト名 = カウンタ ID;				

8.6.2. カウンタ数

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
COUNTER	－	－	－	
概要				
COUNTER オブジェクト数を出力します。				
出力内容				
#define TNUM_COUNTER　カウンタ数 const UINT8 tnum_counter = TNUM_COUNTER;				

8.6.3. カウンタの最大値

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
COUNTER	UIN32	MAXALLOWEDVALUE	－	
概要				
OIL で指定したカウンタの最大値を保持するバッファを出力します。 COUNTER オブジェクトの MAXALLOWEDVALUE 属性に設定された値を出力します。				
出力内容				

const TickType　cntinib_maxval[TNUM_COUNTER] = {
MAXALLOWEDVALUE 属性値, ……
};				

8.6.4. 制御用カウンタの最大値

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
COUNTER	UIN32	MAXALLOWEDVALUE	－	
概要				
制御用のカウンタの最大値を出力します。 COUNTER オブジェクトの MAXALLOWEDVALUE 属性に設定された値を 2 倍し、1 をプラスした値を出力します。				
出力内容				

const TickType　cntinib_maxval2[TNUM_COUNTER] = {
（MAXALLOWEDVALUE 属性値 *2）+ 1, ……
};				

8.6.5. カウンタ毎の 1 単位に達するまでのティック

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
COUNTER	UIN32	TICKSPERBASE	－	
概要				
制御用のカウンタの最大値を出力します。 COUNTER オブジェクトの TICKSPERBASE 属性に設定された値を出力します。				
出力内容				
const TickType　cntinib_tickbase[TNUM_COUNTER]; TICKSPERBASE 属性値, …… };				

8.6.6. カウンタ周期の最小値

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
COUNTER	UIN32	MINCYCLE	－	
概要				
カウンタ周期の最小値を出力します。 COUNTER オブジェクトの MINCYCLE 属性に設定された値を出力します。				
出力内容				
const TickType　cntinib_mincyc[TNUM_COUNTER]; 　MINCYCLE 属性値, …… };				

8.6.7. アラームのキュー

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
COUNTER	－	－	－	
概要				
アラームのキューを出力します。				
出力内容				
AlarmType　cntcb_almque[TNUM_COUNTER];				

8.6.8. カウンタの現在ティック保持バッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
COUNTER	－	－	－	
概要				
カウンタの現在ティックを保持するバッファを出力します。				
出力内容				
TickType　cntcb_curval[TNUM_COUNTER];				

8.7. アラーム管理機能（ALARM）

8.7.1. インクルードファイル

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	－	－	－	
概要				
ALARM オブジェクトで使用するインクルードファイルを出力します。				
出力内容				
#include "alarm.h"				

8.7.2. アラームID

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	－	－	－	
概要				
ALARM オブジェクトの ID を出力します。ID には 0 から始まるユニークな ID を割り付けます。				
出力内容				
const AlarmType ALARM オブジェクト名 = アラーム ID;				

8.7.3. アラーム数

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	－	－	－	
概要				
ALARM オブジェクト数を出力します。				
出力内容				
#define TNUM_ ALARM　アラーム数 const UINT8 tnum_alarm = TNUM_ ALARM;				

8.7.4. アラームに付加されたカウンタID

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	COUNTER_TYPE	COUNTER	－	

概要

各アラームが付加されているカウンタ（COUNTER 属性に指定したカウンタ）のカウンタ ID を出力します。

出力内容

```
const CounterType alminib_cntid[TNUM_ALARM] = {
    COUNTER 属性に指定したカウンタの ID, ……
};
```

8.7.5. アラームコールバックの起動番地

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	ENUM	ACTION	ACTIVATETASK / SETEVENT / ALARMCALLBACK	
	SYMBOLNAME	ALARMCALLBACKNAME		

概要

各アラームのアラームコールバックの起動番地を出力します。
アラームコールバックの起動番地は、ACTION 属性で選択した属性により出力が異なります。
・ALARMCALLBACK の場合
　　ALARMCALLBACKNAME(ALARMCALLBACKNAME に設定した名称)
・ACTIVATETASK / SETEVENT の場合
　　_activate(または_setevent)_alarm_アラームオブジェクト名
アラームコールバックのエントリ関数の名称及び、アラームコールバック関数名を列挙して出力します。

出力内容

```
const FP alminib_cback[TNUM_ALARM] = {
  _activate_alarm_アラームオブジェクト名,
  _setevent_alarm_アラームオブジェクト名,
  ALARMCALLBACKNAME( ALARMCALLBACKNAME 属性で設定した名称 ),
  ……..
};
```

8.7.6. アプリケーションモード

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	BOOLEAN	AUTOSTART	TRUE	
	APPMODE_TYPE	APPMODE[]	－	
概要				
各アラームをシステム初期化時に起動するアプリケーションモードを出力します。 アプリケーションモードの値は、AUTOSTART 属性が FALSE の時は 0 に、TRUE の場合は APPMODE サブ属性に指定されたアプリケーションモードの値の論理和とします。				
出力内容				
const AppModeType alminib_autosta[TNUM_ALARM]; 　選択したアプリケーションモードの ID 値, …… };				

8.7.7. アラームが最初に満了する場合のカウンタ値

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	BOOLEAN	AUTOSTART	TRUE	
	UINT32	ALARMTIME	－	
概要				
各アラームをシステム初期化時に起動する場合の最初に expire するカウンタ値（ALARMTIME サブ属性に指定した値）を出力します。AUTOSTART 属性が FALSE の時は、0 を出力します。				
出力内容				
const TickType alminib_almval[TNUM_ALARM]; 　ALARMTIME に設定した値, …… };				

8.7.8. アラームの周期

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	BOOLEAN	AUTOSTART	TRUE	
	UINT32	CYCLETIME	－	
概要				
各アラームをシステム初期化時に起動する場合の expire する周期（CYCLETIME サブ属性に指定した値）を出力します。AUTOSTART 属性が FALSE の時は、0 を出力します。				
出力内容				
const TickType alminib_cycle[TNUM_ALARM]; CYCLETIME に設定した値, …… };				

8.7.9. アラームキュー（次のID情報）

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	－	－	－	
概要				
次の ID 情報を保持するアラームキューを出力します。				
出力内容				
AlarmType almcb_next[TNUM_ALARM];				

8.7.10. アラームキュー（前のID情報）

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	－	－	－	
概要				
前の ID 情報を保持するアラームキューを出力します。				
出力内容				
AlarmType almcb_prev[TNUM_ALARM];				

8.7.11. アラーム毎のexpire値保持バッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	－	－	－	
概要				
アラーム毎の expire 値を保持するバッファを出力します。				
出力内容				
TickType　almcb_almval[TNUM_ALARM];				

8.7.12. アラームの周期値保持バッファ

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_ cfg.c
ALARM	－	－	－	
概要				
アラーム毎の周期 expire 値を保持するバッファを出力します。				
出力内容				
TickType　　almcb_cycle[TNUM_ALARM];				

8.8. イベント管理機能（EVENT）

8.8.1. イベントID

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_cfg.c
EVENT	－	－	－	
概要				
EVENT オブジェクトの MASK 値を出力します。 MASK 属性値に AUTO が指定された場合は、0 から順にユニークな値を割り付けます。また、MASK 属性値に値が設定された場合はその値を出力します。AUTO 指定により 0 から順に ID を割り付けている場合に、				
出力内容				
const EventMaskType EVENT オブジェクト名 = (1UL << イベントマスク値) ;				

8.9. オブジェクトの初期化処理

OIL				出力ファイル
オブジェクト	型	属性	設定値	
OS	－	－	－	kernel_cfg.c
TASK	－	－	－	
ISR	－	－	－	
RESOURCE	－	－	－	
COUNTER	－	－	－	

概要

使用するオブジェクトの初期化処理を呼び出す関数を出力します。
オブジェクト（TASK、ISR、RESOURCE）の定義が 1 つもない場合はそのオブジェクトに関する初期化関数は出力されません。ただし、RESOURCE オブジェクトが 1 つも定義されていない場合でも、OS オブジェクトにて USERESSHEDULER が TRUE に設定されている場合は、RESOURCE の初期化処理を出力します。また、ALARM の初期化関数は、COUNTER オブジェクトが 1 つ以上定義されている場合に出力します。

出力内容

```
void object_initialize(void)
{
   task_initialize();          /* TASK 初期化処理 */
   interrupt_initialize();     /* ISR 初期化処理 */
   resource_initialize();      /* RESOURCE 初期化処理 */
   alarm_initialize();         /* ALARM 初期化処理 */
}
```

8.10. ターゲット依存情報

5. テンプレートファイルにて指定されたターゲット依存情報を出力します。
出力先は –ovec オプションにて指定しますが、デフォルトはカーネル構成ファイル（kernel_cfg.c）内に出力します。

8.10.1. ベクタテーブル登録シンボル外部参照

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_cfg.c
ISR	UINT32	CATEGORY	1/2	
テンプレート記述				
@@FOR_EACH EXTERNAL_SYMBOL_FOR_ISR@@				
概要				
上記記述があった場合、割込み入り口処理生成マクロを OIL で指定した ISR の数だけ出力します。				
出力内容				
ISR1_ENTERNAL(ISR1 名);　/* ISR カテゴリ 1 の場合 */ ISR2_ENTERNAL(ISR2 名);　/* ISR カテゴリ 2 の場合 */				

8.10.2. ベクタテーブル登録シンボル外部参照

OIL				出力ファイル
オブジェクト	型	属性	設定値	kernel_cfg.c
ISR	UINT32	CATEGORY	1/2	
	UINT32	ENTRY	－	
テンプレート記述				
@@INT_ENTRY｛エントリ番号｝@@　/* コメント */				
概要				
ベクタエントリの個数分、割込み用シンボルを出力します。				
出力内容				
UNUSED_INT_SYMBOL();　/*エントリ番号に対応する割込みが OIL ファイルで指定されていない場合 */ ISR1_SYMBOL(ISR1 名);　/* エントリ番号に対応する割込みが ISR1 の場合 */ ISR2_SYMBOL(ISR2 名);　/* エントリ番号に対応する割込みが ISR2 の場合 */				

8.10.3. フックルーチン

<table>
<tr><th colspan="4">OIL</th><th>出力ファイル</th></tr>
<tr><td>オブジェクト</td><td>型</td><td>属性</td><td>設定値</td><td rowspan="12">kernel_cfg.c</td></tr>
<tr><td rowspan="5">OS</td><td>BOOLEAN</td><td>STARTUPHOOK</td><td>FALSE</td></tr>
<tr><td>BOOLEAN</td><td>ERRORHOOK</td><td>FALSE</td></tr>
<tr><td>BOOLEAN</td><td>SHUTDOWNHOOK</td><td>FALSE</td></tr>
<tr><td>BOOLEAN</td><td>PRETASKHOOK</td><td>FALSE</td></tr>
<tr><td>BOOLEAN</td><td>POSTTASKHOOK</td><td>FALSE</td></tr>
<tr><td colspan="4">テンプレート記述</td></tr>
<tr><td colspan="4">@@NULL_STARTUPHOOK_SYMBOL@@</td></tr>
<tr><td colspan="4">@@NULL_ ERRORHOOK _SYMBOL@@</td></tr>
<tr><td colspan="4">@@NULL_ SHUTDOWNHOOK _SYMBOL@@</td></tr>
<tr><td colspan="4">@@NULL_ PRETASKHOOK _SYMBOL@@</td></tr>
<tr><td colspan="4">@@NULL_ POSTTASKHOOK _SYMBOL@@</td></tr>
<tr><td colspan="5">概要</td></tr>
<tr><td colspan="5">テンプレート指定があり、かつ OS オブジェクトの各フックが FALSE 指定の場合は、カーネル構成ファイルに以下のシンボルを出力する。</td></tr>
<tr><td colspan="5">出力内容</td></tr>
<tr><td colspan="5">NULL_STARTUPHOOK_SYMBOL /* スタートアップフックルーチン */
NULL_ERRORHOOK _SYMBOL /* エラーフックルーチン */
NULL_SHUTDOWNHOOK _SYMBOL /* シャットダウンフックルーチン */
NULL_PRETASKHOOK _SYMBOL /* プレタスクフックルーチン */
NULL_POSTTASKHOOK _SYMBOL /* ポストタスクフックルーチン */</td></tr>
</table>

8.10.4. 割込み入り口処理

<table>
<tr><th colspan="4">OIL</th><th>出力ファイル</th></tr>
<tr><td>オブジェクト</td><td>型</td><td>属性</td><td>設定値</td><td rowspan="4">kernel_cfg.c</td></tr>
<tr><td rowspan="2">ISR</td><td>UINT32</td><td>CATEGORY</td><td>1/2</td></tr>
<tr><td>UINT32</td><td>ENTRY</td><td>－</td></tr>
<tr><td colspan="4">テンプレート記述</td></tr>
<tr><td colspan="4">@@FOR_EACH ENTRY_FOR_ISR@@</td><td></td></tr>
<tr><td colspan="5">概要</td></tr>
<tr><td colspan="5">上記の記述があった場合、割込みシンボルマクロを OIL で指定した ISR の数だけ出力します。</td></tr>
<tr><td colspan="5">出力内容</td></tr>
<tr><td colspan="5">ISR1_ENTRY(ISR1 名);　　/* ISR1 の場合 */
ISR2_ENTRY(ISR2 名, ID);　/* ISR2 の場合 */</td></tr>
</table>

9. エラーメッセージ

SG を実行すると、構文解釈処理により、OIL ファイルの記述内容によっては、エラーが出力される場合があります。この場合、SG は C 言語コードファイルの生成処理を中断し、中断した原因を示すエラーメッセージを表示します。

以下にエラーメッセージのとそのメッセージが表示された場合の対処方法を記載します。

エラーメッセージ	説明	対応
OIL_VERSION exposes 2.5 as version	OIL_VERSION はバージョンとして 2.5 を期待しています。	OIL_VERSION は 2.5 を指定してください。
`XXX' is out of range	`XXX'が範囲を超えています。	OIL 仕様書を確認して有効属性値を設定して下さい。
undefined attribute `XXX'	未定義の属性`XXX'によりエラーが発生しています。	属性名が異なっていないか確認して下さい。
invalid attribute value `XXX'	属性値`XXX'が不正です。	OIL 仕様書を確認して有効属性値を設定して下さい。
The template pattern of `XXX' doesen't exit	XXX'というテンプレートパターンは存在しません。	テンプレート文が異なっていないかテンプレートファイルを確認して下さい。
too many XXX objects	１つしか定義できないオブジェクトが複数定義されている可能性があります。または同名のオブジェクトが存在しています。	オブジェクトの定義数を確認してください。または、同名のオブジェクトがないか確認してください。

10. 補足資料

10.1. テンプレート資料

```
@@ISR_MIN_PRIORITY=1@@
@@ISR_MAX_PRIORITY=7@@
@@ISR_MIN_ENTRY=0@@
@@ISR_MAX_ENTRY=71@@
@@ISR_ENTRY_INTERVAL=1@@

/* 割込み入り口処理          */
@@FOR_EACH ENTRY_FOR_ISR@@

/* ベクタテーブル登録シンボル外部参照      */
@@FOR_EACH EXTERNAL_SYMBOL_FOR_ISR@@
UNUSED_INT_EXTERNAL();      /* 未定義の割込み */
asm("    .glb       _start"); /* リセット */

/* 割込み可変ベクタテーブル          */
asm("    .section vvector");

@@INT_ENTRY0@@, /*   0, +0x00: BRK 命令          */
@@INT_ENTRY1@@, /*   1, +0x04: 予約領域          */
 （中略）
/* 割込み固定ベクタテーブル          */
          .section fvector   ;
 （中略）
@@INT_ENTRY71@@; /* 71, 0xFFFFF8: NMI          */
          .lword    _start    ; /* 72, 0xFFFFFC: リセット          */

/* フックルーチン            */
@@NULL_ERRORHOOK_SYMBOL@@
@@NULL_STARTUPHOOK_SYMBOL@@
@@NULL_SHUTDOWNHOOK_SYMBOL@@
@@NULL_PRETASKHOOK_SYMBOL@@
@@NULL_POSTTASKHOOK_SYMBOL@@
```

10.1.1. サンプルテンプレート

10.1.2. テンプレート記述と出力マクロ定義の対応

テンプレート記述	出力マクロ定義
ISR	
@@FOR_EACH ENTRY_FOR_ISR@@	OIL ファイルで定義した ISR の数だけ出力される ・ISR1 の場合 <C 出力> ISR1_ENTRY(ISR 名); <アセンブラ出力> ISR1_ENTRY ISR 名 ・ISR2 の場合 <C 出力> ISR2_ENTRY(ISR 名 (ID)) <アセンブラ出力> ISR2_ENTRY ISR 名,ID
@@FOR_EACH EXTERNAL_SYMBOL_FOR_ISR@@	OIL ファイルで定義した ISR の数だけ出力される ・ISR1 の場合 <C 出力> ISR1_ENTRY(ISR 名); <アセンブラ出力> ISR1_ENTRY ISR 名 ・ISR2 の場合 <C 出力> ISR2_ENTRY(ISR 名); <アセンブラ出力> ISR2_ENTRY ISR 名
@@INT_ENTRY{エントリ番号}@@	・対応する割込みがない場合 <C 出力> UNUSED_INT_SYMBOL() <アセンブラ出力> UNUSED_INT_SYMBOL

	・ISR1 の場合 <C 出力> ISR1_SYMBOL(ISR 名) <アセンブラ> ISR1_SYMBOL ISR 名 ・ISR2 の場合 <C 出力> ISR2_SYMBOL(ISR 名) <アセンブラ> ISR2_SYMBOL ISR 名
フックルーチン	
@@NULL_ERRORHOOK@@	NULL_ERRORHOOK
@@NULL_STARTUPHOOK@@	NULL_STARTUPHOOK
@@NULL_SHUTDOWNHOOK@@	NULL_SHUTDOWNHOOK
@@NULL_PRETASKHOOK@@	NULL_PRETASKHOOK
@@NULL_PRETASKHOOK@@	NULL_PRETASKHOOK

11. 変更履歴

Version	Date	Detail	Editor
1.00	2004/07/07	・ 新規作成	ヴィッツ
2.00	2006/03/14	・ CPU 依存部テンプレート対応追記 ・ 誤記修正 ・ 出力例の削除 ・ OIL 記述の修正	ヴィッツ
2.01	2006/03/24	・割込み入り口処理追記 ・テンプレート資料修正	ヴィッツ
2.02	2006/03/30	・ OIL　TASK,ALARM 記述修正	ヴィッツ
2.03	2006/05/13	・ テンプレート機能拡張及び記述修正 ・ アプリケーションモード管理情報修正 ・ SG 実行オプションの修正	ヴィッツ
3.00	2006/05/30	・ TOPPERS/OSEK(旧称)公開用にバージョンアップ	ヴィッツ
4.00	2007/09/20	・ フォーマット修正	ヴィッツ
4.01	2007/09/27	・ 部署名及びロゴマーク修正	ヴィッツ
4.02	2007/11/05	・ 誤記修正	ヴィッツ
4.03	2008/03/14	・ オブジェクトの初期化処理修正	ヴィッツ
5.00	2008/10/20	・ カーネル名称変更	ヴィッツ