TOPPERS/OSEK OS
SG 取扱説明書

Ver.3.00

2006/05/30

株式会社ヴィッツ　開発第３部

TOPPERS/OSEK Kernel
　　Toyohashi Open Platform for Embedded Real-Time Systems/OSEK Kernel

＜目次＞

1. 概要

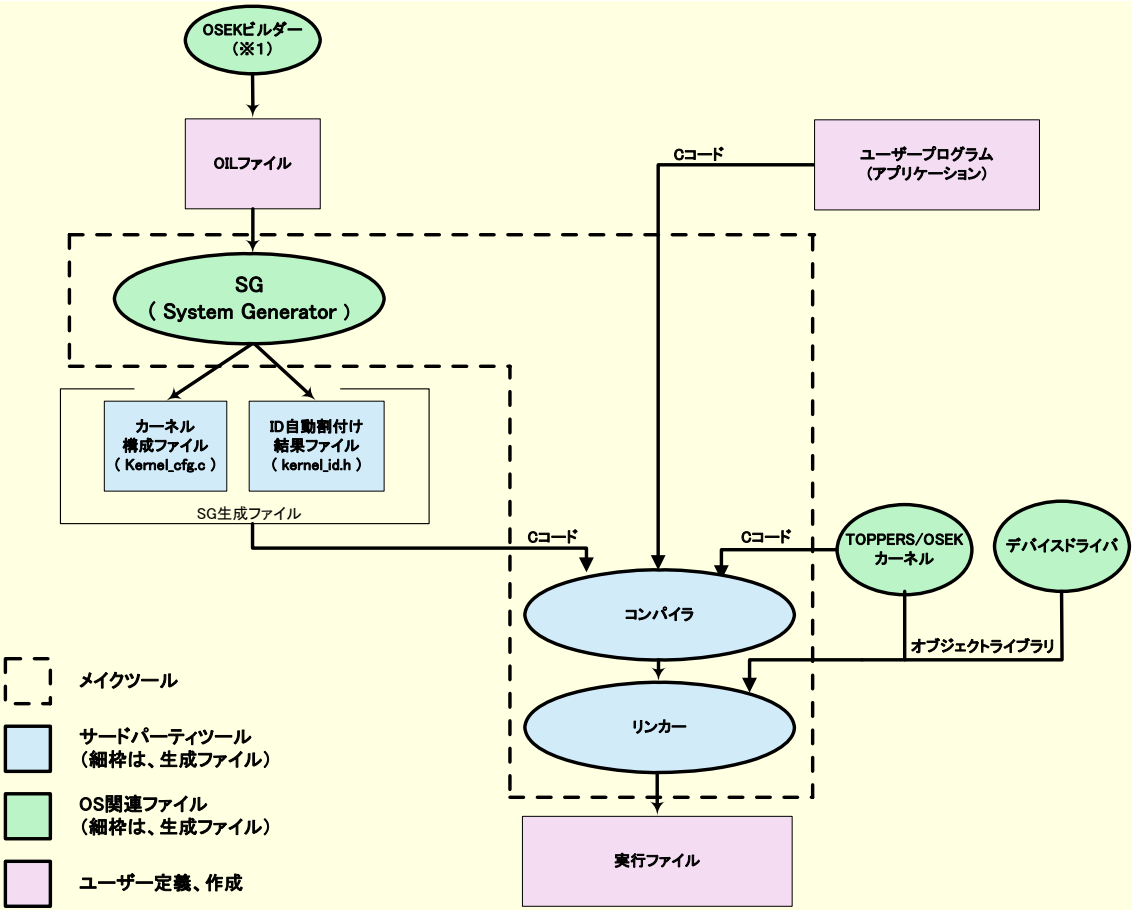
本 SG 取扱説明書では、OSEK／VDX 仕様に準拠した TOPPERS/OSEK カーネル付属の OSEK System Generator（以下 SG）の使用方法を記述します。

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、カーネルが必要とする 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下記図に示します。



※1　OSEK ビルダーは、OIL ファイルを生成するための上位ツールを想定しています。

SG 生成ファイル、ユーザプログラム、TOPPERS/OSEK カーネルをコンパイルし、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（カーネル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

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。

次に使用例を記載します（OIL ファイルは sample.oil とする）。

■例 1　：　sg.exe sample.oil –os=BCC1 –cpu=h8s –lj –I./../impl_oil
指定内容：
コンフォーマンスクラス BCC1、ターゲット CPU は H8S、出力メッセージを日本語表示、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。

■例 2　：　sg.exe sample.oil –os=ECC2 –cpu=m16c –debug –cfg=test.c
指定内容：
コンフォーマンスクラス ECC2、ターゲット CPU は M16C、デバッグメッセージの出力、カーネル構成ファイル名を”test.c”に変更。

■例 3　：　sg.exe sample.oil –os=BCC2 –template=m32ctemp -withouttime
指定内容：
コンフォーマンスクラス BCC2、テンプレートファイル”m32ctemp”の読込指定、出力ファイルに生成日時の出力を拒否。

上記例を参考に SG を実行すると、カーネル構成ファイル及び ID 自動割付ファイルを出力します。

SGコマンドラインオプション一覧

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>		OS 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=h8s：H8S 指定（H8S 依存処理の実行）
-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変更（デフォルトは構成ファイル内に出力）
-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>		OSEK/OS 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
-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。
-template		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ら-cpu にて指定は無効となります。
-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と想定しています。

3. テンプレート

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。

3.1. SG 使用情報

最大/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_MIN_PRIORITY=7@@
        @@ISR_MIN_ENTRY=0@@
        @@ISR_MAX_ENTRY=71@@
        @@ISR_ENTRY_INTERVAL=1@@
```

3.2. 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、割り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@@
```

⇒出力マクロ

- ISR1 の場合

  ```
  ISR1_ENTERNAL(ISR 名);
  ```

- ISR2 の場合

  ```
  ISR2_ENTERNAL(ISR 名);
  ```

3.3. ベクタエントリ

ベクタエントリ数分の「@@INT_ENTRY｛エントリ番号｝@@　/* コメント */」といった記述がある。
・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          */
```

⇒出力マクロ

- エントリ番号に対応する割り込みが OIL ファイルで指定されていない場合

  ```
  UNUSED_INT_SYMBOL()
  ```

・ エントリ番号に対応する割り込みが ISR1 の場合
ISR1_SYMBOL(ISR 名)
・ エントリ番号に対応する割り込みが ISR2 の場合
ISR2_SYMBOL(ISR 名)

3.4. 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@@

⇒出力マクロ

NULL_ERRORHOOK_SYMBOL
NULL_STARTUPHOOK_SYMBOL
NULL_SHUTDOWNHOOK_SYMBOL
NULL_PRETASKHOOK_SYMBOL
NULL_POSTTASKHOOK_SYMBOL

3.5. 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

3.6. 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@@

3.7. 割り込み入り口処理

割り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、割り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@@
```

⇒出力マクロ

・ISR1 の場合

```
ISR1_ENTRY(ISR 名);
```

・ISR2 の場合

```
ISR2_ENTRY(ISR 名,ID);
```

3.8.　割り込み要因情報

OIL ファイルより各割り込み要因情報を収集し、指定された割り込み優先度レベルに対して、設定するべき割り込み禁止/許可コードを出力することが可能です。例えば、割り込み優先度レベルを 3 に設定する場合、優先度が 0～3 に登録された割り込み要因を全て禁止、4～7 に登録された割り込み要因を全て許可にするコードを出力することが可能です。
テンプレート記述と対応する出力マクロの例を以下に示します。

・　テンプレート記述例

```
@@ENAINT_IPL3@@
```

⇒出力マクロ定義

```
enaint(INTREG2)
enaint(INTREG14)
```

IPL4～7 に登録された割り込み許可を列挙を行います。

・　テンプレート記述例

```
@@DISINT_IPL3@@
```

⇒出力マクロ

```
disint(INTREG6);
disint(INTREG30);
disint(INTREG44);
```

IPL0～3 に登録された割り込み禁止の列挙を行います。

※ 本項目の機能は現在対応中です。

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 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/OSEK カーネルで使用する 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	OSEK　OS 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。
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。
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。
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。
ISR	割り込みサービスルーチンを使用す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。

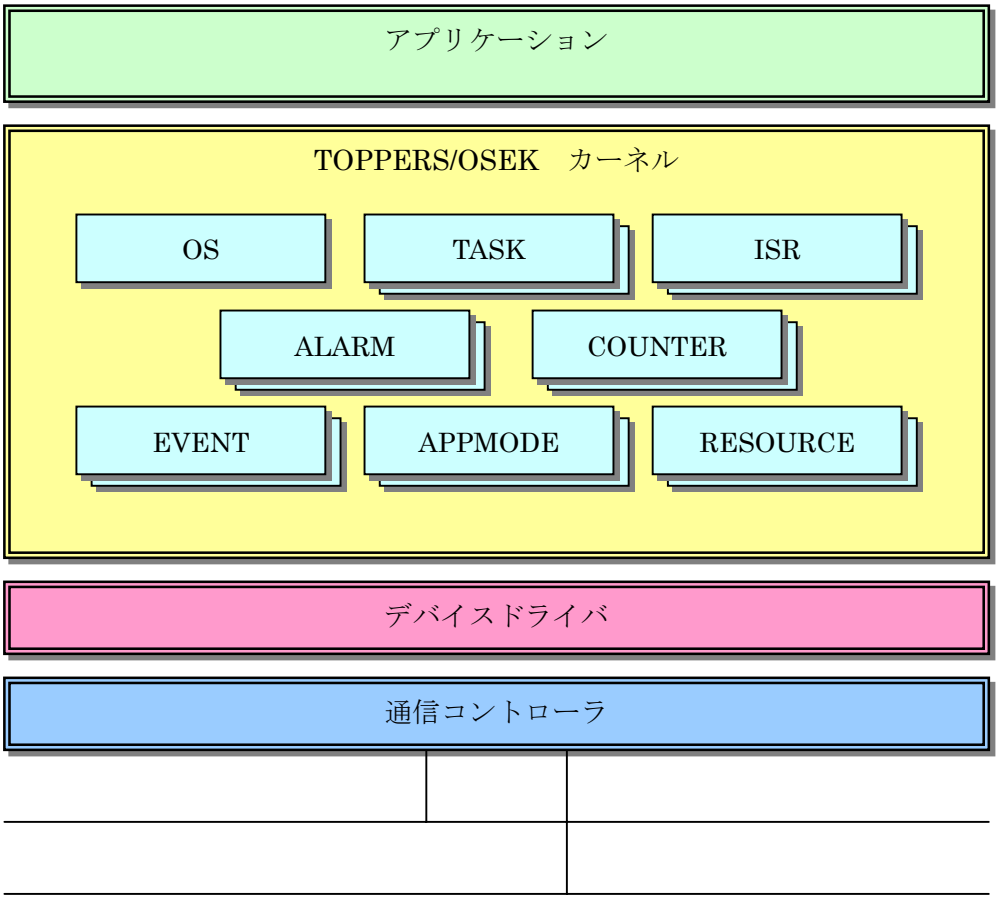
型	説明
UINT32	32 ビット長符号無し整数
INT32	32 ビット長符号付き整数
UINT64	64 ビット長符号無し整数
INT64	64 ビット長符号付き整数
FLOAT	浮動小数
ENUM	離散的な集合
BOOLEAN	TRUE,FALSE の指定
STRING	文字列（OIL 標準）
SYMBOLNAME	文字列（本 SG 独自拡張、SG にてシンボル名として有効かのチ

	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。

4.2.1. 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。



4.3. 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";
```

4.4. 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。また、５出力処理に各オブジェクトの属性について説明が記載されていますので、そちらを参照して下さい。

4.4.1. OS

OS {			補足説明
ENUM	[STANDARD, EXTENDED]	STATUS = EXTENDED;	未サポート(EXTENDED 固定)
BOOLEAN	STARTUPHOOK	= FALSE;	
BOOLEAN	ERRORHOOK	= FALSE;	
BOOLEAN	SHUTDOWNHOOK	= FALSE;	
BOOLEAN	PRETASKHOOK	= FALSE;	
BOOLEAN	POSTTASKHOOK	= FALSE;	
BOOLEAN	USEGETSERVICEID	= TRUE;	未サポート(TRUE 固定)
BOOLEAN	USEPARAMETERACCESS	= TRUE;	
BOOLEAN	USERESSCHEDULER	= FALSE;	
};			

■サンプル

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

```
};
```

4.4.2. APPMODE

APPMODE {	補足説明
	属性無し
};	

■サンプル

```
APPMODE sample_appmode1{};
APPMODE sample_appmode2{}
APPMODE    DEFAULTAPPMODE{}
```

4.4.3. TASK

TASK {			補足説明
	BOOLEAN [		
		TRUE {	
		APPMODE_TYPE	APPMODE[];
		},	
		FALSE	
	] AUTOSTART	= NO_DEFAULT;	
	UINT32	[0..15] PRIORITY	= NO_DEFAULT;
	UINT32	ACTIVATION	= NO_DEFAULT;
	ENUM	[NON, FULL] SCHEDULE	= NO_DEFAULT;
	EVENT_TYPE	EVENT[];	
	RESOUCE_TYPE	RESOURCE[];	
	MESSAGE_TYPE	MESSAGE[]:	未サポート
	UINT32	STACKSIZE	= 1024
};			

■サンプル

```
TASK sample_task1 {
        AUTOSTART       = FALSE;
        PRIORITY        = 10;
        ACTIVATION      = 3;
        SCHEDULE        = FULL;
```

```
        EVENT           = sample_event1;
        EVENT           = sample_event2;
        RESOURCE        = sample_resource1;
        STACKSIZE       = 1024
};
```

```
TASK sample_task2 {
        AUTOSTART       = TRUE {
            APPMODE     = sample_appmode1;
        };
        PRIORITY        = 5;
        ACTIVATION      = 1;
        SCHEDULE        = NONE;
        EVENT           = sample_event1;
        RESOURCE        = sample_resource2;
        STACKSIZE       = 256
};
```

4.4.4. ISR

ISR {				補足説明
	UINT32 [1, 2] CATEGORY		= NO_DEFALT;	
	UINT32	PRIORITY	= NO_DEFALT;	TOPPERS/OSEK 拡張
	UINT32	ENTRY	= NO_DEFALT;	TOPPERS/OSEK 拡張
	RESOURCE_TYPE	RESOURCE[];		
};				

■サンプル

```
ISR sample_isr1 {
        CATEGORY        = 1;
        PRIORITY        = 8;
        ENTRY           = 22;
};
```

```
ISR sample_isr2 {
```

```
        CATEGORY            = 2;
        PRIORITY            = 7;
        ENTRY               = 21;
        RESOURCE            = sample_resource2;
};
```

4.4.5. COUNTER

COUNTER {				補足説明
	UINT32	MINCYCLE	= NO_DEFALT;	
	UINT32	MAXALLOWEDVALUE	= NO_DEFALT;	
	UINT32	TICKSPERBASE	= NO_DEFALT;	
};				

■サンプル

```
COUNTER sample_counter1 {
        MINCYCLE            = 1;
        MAXALLOWEDVALUE     = 1000;
        TICKSPERBASE        = 1;
};
```

4.4.6. ALARM

ALARM {				補足説明
	COUNTER_TYPE	COUNTER;		
	EMUN [			
	ACTIVATETASK {			
		TASK_TYPE	TASK;	
	},			
	SETEVENT {			
		TASK_TYPE	TASK;	
		EVENT_TYPE	EVENT	
	},			
	ALARMCALLBACK {			
		STRING	ALARMCALLBACKNAME;	

```
        }
    ] ACTION        = NO_DEFAULT;
    BOOLEAN [
        TRUE {
            UINT32          ALARMTIME;
            UINT32          CYCLETIME;
            APPMODE_TYPE    APPMODE[];
        },
        FALSE
    ] AUTOSTART;
};
```

■サンプル

```
ALARM sample_alarm1 {
        COUNTER         = 1000;
        ACTION          = ACTIVATETASK {
                TASK    = sample_task1;
        };
        AUTOSTART       = FALSE;
};
```

```
ALARM sample_alarm2 {
        COUNTER         = 1000;
        ACTION          = SETEVENT {
                TASK    = sample_task2;
                EVENT   = sample_event1;
        };
        AUTOSTART       = TRUE {
                ALARMTIME   = 500;
                CYCLETIME   = 1000;
                APPMODE     = Appmode2;
        };
};
```

4.4.7. EVENT

EVENT {				補足説明
	UINT32 WITH_AUTO	MASK	= AUTO;	
};				

■サンプル

```
EVENT sample_event1 {
        MASK            = 7;
};
```

4.4.8. RESOURCE

RESOURCE {			補足説明
	ENUM [		
		STANDARD,	
		LINKED {	
		RESOURCE_TYPE　LINKEDRESOURCE	
		},	
		INTERNAL	
	] RESOURCEPROPERTY	= NO_DEFAULT;	
};			

■サンプル

```
RESOURCE sample_resource1 {
        RESOURCEPROPERTY                = STANDARD;
};
```

```
RESOURCE sample_resource2 {
        RESOURCEPROPERTY                = LINKED {
                LINKEDRESOURCE          = sample_resource1;
        };
};
```

4.5. 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。

5. 構文解析処理とエラー検出処理

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（OILVERSION）、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。

5.1. 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 OILVERSION の指定が無かった場合は、エラーとし処理を中断します。

5.2. 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。

5.3. 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。
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。以下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。

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。
`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。

6. 出力処理

TOPPERS/OSEK カーネルのコンフィギュレータは、システムコンフィギュレーションファイルを処理して、カーネル構成ファイル（kernel_cfg.c）と ID 自動割付け結果ファイル（kernel_id.h）を生成します。カーネル構成ファイルは、OIL 記述をもとにコンフィギュレータがカーネル必要な各種配列・値を C 言語ベースで出力したファイルです。ID 自動割付け結果ファイルは、コンフィギュレータが ID 番号を割り付けたオブジェクトの名前を、割り付けた ID 番号に定義するファイルです。

6.1. OS 管理情報（OS）

OIL ファイルには、アプリケーションに必要な OS の性質を記述することが必要です。
しかし本システムジェネレータでは Resource オブジェクトで使用されるだけであり、OS の管理情報に関してはコードの出力は行われません。

6.2. アプリケーションモード管理情報（APPMODE）

アプリケーションモードには、1 から始まるユニークなアプリケーション ID を割り付けます。
割り付けた値は、kernel_id.h に次のように出力します。

```
#define appmode1        (1U << 0)
#define appmode2        (1U << 1)
```

アプリケーションモードを使用しない場合、アプリケーションモードオブジェクト名に「OSDEFAULTAPPMODE」を設定します。前述のように定義されたオブジェクトはアプリケーション ID を kernel_id.h に出力しません。

6.3. タスク管理情報（TASK）

6.3.1. インクルードファイル

Kernel_cfg.c に次のように出力します。

```
#include "task.h"
```

6.3.2. タスク ID

タスクには、0 から始まるユニークなタスク ID を割り付けます。タスク ID の割り付けに際しては、拡張タスクに対して基本タスクよりも小さい値を割り付けます。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。
割り付けた値は、kernel_cfg.c に次のように出力します。

```
const TaskType Task1 = 0;
const TaskType Task2 = 1;
```

6.3.3. タスク関数定義

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。

kernel_cfg.c に次のように出力します。

```
extern void TASKNAME( task1 )( void );
extern void TASKNAME( task2 )( void );
```

6.3.4. タスク数

タスクの数および拡張タスクの数を、以下のマクロ及び定数で、kernel_cfg.c に出力します。

```
#define TNUM_TASK              xx        /* タスクの数 */
#define TNUM_EXTTASK           xx        /* 拡張タスクの数 */
const UINT8    tnum_task = TNUM_TASK;
const UINT8    tnum_exttask = TNUM_EXTTASK;
```

6.3.5. TASK オブジェクト属性

TASK オブジェクトの各属性を以下のように定めます。

PRIORITY	タスクの優先度（相対値）。 最低優先度は 0、値の大きい方が高優先度です。
SCHEDULE	ノンプリエンプティブ（NON）もしくはフルプリエンプティブ(FULL)を設定します。
ACTIVATION	多重起動要求の最大値。1 は多重起動なしを意味します。尚、拡張タスクの場合は 1 以外を指定することはできません。
AUTOSTART	システム初期化時に起動する場合 TRUE とし、そのとき APPMODE サブ属性にどのアプリケーションモードで起動するかを記述します。
RESOURCE	獲得するリソースのリストです。
EVENT	拡張タスクが待てるイベントのリストです。
STACKSIZE	スタックのサイズです。

6.3.6. 初期値優先度

各タスクの初期優先度（TPRI_MINTASK + PRIORITY 属性に指定した値）を格納します。

Kernel_cfg.c に次のように出力します。

```
const Priority    tinib_inipri[TNUM_TASK] = { ..... };
```

6.3.7. 実行開始直後の優先度

各タスクの実行開始直後の優先度を格納します。実行開始直後の優先度は、内部リソースが付加されていない場合には初期優先度、内部リソースが付加されている場合にはその上限優先度とします。

Kernel_cfg.c に次のように出力します。

```
const Priority   tinib_exepri[TNUM_TASK] = { ..... };
```

6.3.8.　多重起動要求キューイング数の最大値

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）を格納します。
Kernel_cfg.c に次のように出力します。

```
const UINT8    tinib_maxact[TNUM_TASK] = { ..... };
```

6.3.9.　起動するアプリケーションモード

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格納します。アプリケーションモードの値は、AUTOSTART 属性が FALSE の時は 0U 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。
Kernel_cfg.c に次のように出力します。

```
const AppModeType tinib_autoact[TNUM_TASK] = { ..... };
```

6.3.10. 起動番地

各タスクの起動番地を格納します。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。
Kernel_cfg.c に次のように出力します。

```
const FP         tinib_task[TNUM_TASK] = { ..... };
```

6.3.11. スタック領域の先頭番地

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）を格納します。スタック領域の定義もコンフィギュレータが生成します。
kernel_cfg.c に次のように出力します。

```
const VP         tinib_stk[TNUM_TASK] = { ..... };
```

6.3.12. スタック領域のサイズ

各タスクのスタック領域のサイズ（実際に確保したスタック領域のサイズ）を格納します。STACKSIZE 属性に指定した値を（必要なら）大きい方に丸めた値とするのが基本ですが、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あわせることになります。
kernel_cfg.c に次のように出力します。

```
const UINT16  tinib_stksz[TNUM_TASK] = { ..... };
```

6.3.13. 制御ブロックの出力

カーネルが使用するタスクの制御ブロックの実体として、kernel_cfg.c に次のように出力します。

```
TaskType         tcb_next[TNUM_TASK];
UINT8            tcb_tstat[TNUM_TASK];
```

```
Priority        tcb_curpri[TNUM_TASK];
UINT8           tcb_actcnt[TNUM_TASK];
EventMaskType   tcb_curevt[TNUM_EXTTASK];
EventMaskType   tcb_waievt[TNUM_EXTTASK];
ResourceType  tcb_lastres[TNUM_TASK];
DEFINE_CTXB(TNUM_TASK);
```

6.4. 割込み管理情報（ISR）

6.4.1. インクルードファイル

Kernel_cfg.c に次のように出力します。

```
#include "interrupt.h"
```

6.4.2. ISR ID

カテゴリ 2 の ISR には、0 から始まるユニークな ISR ID を割り付けます。
割り付けた値は、kernel_cfg.c に次のように出力します。

```
const IsrType   timer_a0 = 0;
const IsrType   uart1_rx = 1;
```

6.4.3. カテゴリ 2 ISR 数

カテゴリ 2 の ISR の数を、以下のマクロ及び定数で、kernel_cfg.c に出力します。

```
#define TNUM_ISR2    x          /* ISR（カテゴリ 2）の数 */
const UINT8   tnum_isr2 = TNUM_ISR2;
```

6.4.4. ISR エントリ定義

カテゴリ 2 のについては ISR オブジェクト名称と ISRID を、カテゴリ 1 については ISR オブジェクト名称のみを kernel_cfg.c に出力します。

```
ISR1 ENTRY(debug_sio_tx);
ISR2_ENTRY(sys_timer,(0));
```

6.4.5. ISR オブジェクト

ISR オブジェクトの各属性を以下のように定めます。

CATEGORY	ISR のカテゴリです。UINT32 型ですが、1 と 2 のみ有効です。
RESOURCE	獲得するリソースのリストです。
MESSAGE	アクセスするメッセージのリストです。
PRIORITY	ISR の割込み優先レベルです。
ENTRY	ISR を付加する割込み番号です。意味はターゲット依存だが、M32C ではベクトル番号とします。

6.4.6.　割込み優先レベル

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、コンフィギュレータがエラーを報告します。

Kernel_cfg.c に次のように出力します。

```
#define IPL_MAXISR2     x                   /* ISR2 の最大優先度  */
const IPL         ipl_maxisr2 = IPL_MAXISR2;
```

6.4.7.　割込み優先度

カテゴリ 2 の各 ISR の割込み優先度（TPRI_MINISR + IPL 属性に指定した値）を格納します。

Kernel_cfg.c に次のように出力します。

```
const Priority   isrinib_intpri[TNUM_ISR2];
```

6.4.8.　制御ブロック

カーネルが使用する ISR の制御ブロックの実体として、kernel_cfg.c に次のように出力します。

```
ResourceType   isrcb_lastres[TNUM_ISR2];
```

6.5.　リソース管理情報（RESOURCE）

6.5.1.　インクルードファイル

Kernel_cfg.c に次のように出力します。

```
#include "resource.h"
```

6.5.2.　リソース ID

リソースには、0 から始まるユニークなリソース ID を割り付けます。

割り付けた値は、kernel_cfg.c に次のように出力します。

```
const ResourceType Resource1 = 0;
const ResourceType Resource2 = 1;
```

6.5.3.　リソース数

リソースの数を、以下のマクロ及び定数で、kernel_cfg.c に出力します。

```
#define TNUM_RESOURCE     5
const UINT8    tnum_resource = TNUM_RESOURCE;
```

6.5.4.　上限優先度

各リソースの上限優先度を格納します。リソースの上限優先度は、そのリソースをアクセスするタスク

の優先度およびカテゴリ 2 の ISR の割込み優先度の最大値とします。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。
Kernel_cfg.c に次のように出力します。

```
const Priority   resinib_ceilpri[TNUM_RESOURCE];
```

6.5.5. 制御ブロック

カーネルが使用するリソースの制御ブロックの実体として、kernel_cfg.c に次のように出力します。

```
Priority         rescb_prevpri[TNUM_RESOURCE];
ResourceType  rescb_prevres[TNUM_RESOURCE];
```

6.6. カウンタ管理情報（COUNTER）

6.6.1. カウンタ ID

0 から始まるユニークなカウンタ ID を割り付けます。
割り付けた値は、kernel_cfg.c に次のように出力します。

```
const CounterType Counter1 = 0;
```

6.6.2. カウンタ数

カウンタの数を、以下のマクロ及び定数で、kernel_cfg.c に出力します。

```
#define TNUM_COUNTER        xx                    /* カウンタの数 */
const UINT8    tnum_counter = TNUM_COUNTER;
```

6.6.3. COUNTER オブジェクト

COUNTER オブジェクトの各属性を以下のように定めます。

MAXALLOWEDVALUE	カウンタの最大値です。0 を指定してはなりません。
TICKSPERBASE	カウンタ毎の 1 単位に達するまでのティックです。MAXALLOWEDVALUE 属性に指定した値以下のみ有効です。
MINCYCLE	周期として指定できる最小値です。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有効です。

6.6.4. 初期化ブロック

各カウンタの MAXALLOWEDVALUE 属性に指定した値、MAXALLOWEDVALUE 属性に指定した値 x2+1 の値、TICKSPERBASE 属性に指定した値、MINCYCLE 属性に指定した値を、それぞれ格納します。
Kernel_cfg.c に次のように出力します。

```
const TickType  cntinib_maxval[TNUM_COUNTER];
```

```
const TickType  cntinib_maxval2[TNUM_COUNTER];
const TickType  cntinib_tickbase[TNUM_COUNTER];
const TickType  cntinib_mincyc[TNUM_COUNTER];
```

6.6.5.　制御ブロック

カーネルが使用するカウンタの制御ブロックの実体として、kernel_cfg.c に次のように出力します。

```
AlarmType       cntcb_almque[TNUM_COUNTER];
TickType        cntcb_curval[TNUM_COUNTER];
```

6.7.　アラーム管理情報（ALARM）

6.7.1.　インクルードファイル

Kernel_cfg.c に次のように出力します。

```
#include "alarm.h"
```

6.7.2.　アラーム ID

0 から始まるユニークなアラーム ID を割り付けます。

割り付けた値は、kernel_cfg.c に次のように出力します。

```
const AlarmType Alarm1 = 0;
```

6.7.3.　アラーム数

アラームの数を、以下のマクロ及び定数で、kernel_cfg.c に出力します。

```
#define TNUM_ALARM          xx                  /* アラームの数 */
const UINT8    tnum_alarm = TNUM_ALARM;
```

6.7.4.　ALARM オブジェクト

ALARM オブジェクトの各属性を以下のように定めます。

COUNTER	割り付けるカウンタです。
ACTION	アラームが expire した時に行う動作をいずれか 1 つ指定します。 ・ACTIVATETASK { TASK_TYPE TASK; } ・SETEVENT { TASK_TYPE TASK; EVENT_TYPE EVENT; } ・ALARMCALLBACK { STRING ALARMCALLBACKNAME; }
AUTOSTART	システム初期化時に起動する場合は、 TRUE とします。その場合、以下 3 種をサブ属性とします ALARMTIME … 最初に expire するカウンタ値。0 より大きく（誤解をさけるために 0 は許さないことにする）、アラームが付加されているカウンタの MAXALLOWEDVALUE 属性に指定した値以下とします。 CYCLETIME … 繰り返し expire する場合の周期。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とします。 APPMODE … どのアプリケーションモードで起動するかを記述。

6.7.5. カウンタ ID

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 を格納します。

Kernel_cfg.c に次のように出力します。

```
const CounterType alminib_cntid[TNUM_ALARM];
```

6.7.6. アラームコールバックの起動番地

各アラームのアラームコールバックの起動番地を格納する。アラームコールバックの起動番地は、ユーザがアラームコールバックを記述する場合には、アラームコールバックのエントリ関数の名称を「ALARMCALLBACKNAME(＜アラームコールバック名＞)」の形で列挙します。アラームの動作が ACTIVATETASK または SETEVENT の場合には、アラームコールバック関数をコンフィギュレータが生成し、その関数名を列挙します。

Kernel_cfg.c に次のように出力します。

```
const FP        alminib_cback[TNUM_ALARM];
```

6.7.7. アプリケーションモード

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格納します。アプリケーションモードの値は、AUTOSTART 属性が FALSE の時は 0U 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。

Kernel_cfg.c に次のように出力します。

```
const AppModeType alminib_autosta[TNUM_ALARM];
```

6.7.8. カウンタ値

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）を格納します。AUTOSTART 属性が FALSE の時は、不定値で構いません。

Kernel_cfg.c に次のように出力します。

```
const TickType alminib_almval[TNUM_ALARM];
```

6.7.9. 周期

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）を格納する。AUTOSTART 属性が FALSE の時は、不定値で構いません。

Kernel_cfg.c に次のように出力します。

```
const TickTypealminib_cycle[TNUM_ALARM];
```

6.7.10. 制御ブロック

カーネルが使用するアラームの制御ブロックの実体として、kernel_cfg.c に次のように出力します。

```
AlarmType       almcb_next[TNUM_ALARM];
AlarmType       almcb_prev[TNUM_ALARM];
TickType        almcb_almval[TNUM_ALARM];
TickType        almcb_cycle[TNUM_ALARM];
```

6.8.　イベント管理情報（EVENT）

6.8.1.　イベント ID

0 から始まるユニークなイベント ID を割り付けます。

割り付けた値は、kernel_cfg.c に次のように出力します。

```
const EventMaskType event1 = (1UL << 1);
const EventMaskType event2 = (1UL << 2);
```

6.9.　オブジェクトの初期化処理

使用するオブジェクトの初期化処理を呼び出す関数を生成します。

Kernel_cfg.c に次のように出力します（例）。

```
void object_initialize(void)
{
        task_initialize();
        interrupt_initialize();
        resource_initialize();
        alarm_initialize();
        hw_timer_initialize();
}
```

6.10. ターゲット依存情報

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、デフォルト構成ファイル（kernel_cfg.c）内に出力します。

出力情報には次の通りです。

- 外部シンボルの参照宣言
- 割込みベクタの登録
- 存在しないシンボルの 0 番地宣言（コンパイル対策）
- Hook ルーチンを使用しない場合

7. 補足資料

7.1. テンプレート資料

7.1.1. サンプルテンプレート

```
@@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@@
```

7.1.2. 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}@@	・対応する割り込みがない場合 <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
@@DISINT_IPL{割り込み優先度レベル}@@	disint(INTREG{エントリ番号});
@@ENAINT_IPL{割り込み優先度レベル}@@	enaint(INTREG{エントリ番号});
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

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	・割り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 公開用にバージョンアップ	ヴィッツ