JSP カーネル コンフィギュレータ仕様

（Release 1.4 対応，最終更新: 20-Dec-2003）

注意 : このマニュアルは TOPPERS/JSP 1.4 の doc サブディレクトリにある configurator.txt を word ファイルに変換したものである。内容は原則として同一であるが、疑問がある場合には必ず最新版の原文書を調べること。

word/pdf 文書の問題については、配布位置である sourceforge の掲示板で問い合わせいただきたい。また、原文書自身の問題あるいはそれに関する疑問は、情報共有のためにも TOPPERS プロジェクトのメールリストで問い合わせいただきたい。

なお、以下の点は原文書と異なる

- 著作権表示。この文書の著作権表示は armv4.doc のものをそのまま使っている。

このドキュメントでは，JSP カーネルのコンフィギュレータが生成すべきファイルの内容について解説する．

JSP 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）を生成する．また，静的 API のパラメータチェックに用いるファイル（kernel_chk.c）を生成する．

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ある．

以下では，カーネル構成ファイルの内容について解説する．

(1) 固定生成部分

(1-1) 標準インクルードファイル

kernel_cfg.c 用のインクルードファイルをインクルードするプリプロセッサディレクティブ（#include）を生成する．具体的には，次の行を生成する．

```
#include "kernel_cfg.h"
```

また，必要であれば，ID 自動割付け結果ファイルをインクルードするプリプロセッサディレクティブ（#include）を生成する．具体的には，次の行を生成する．

```
#include "kernel_id.h"
```

(1-2) マクロの置換順序の制御

マクロの置換順序を変えるために，以下のマクロ定義行を生成する．

```
#define CFG_INTHDR_ENTRY(inthdr) INTHDR_ENTRY(inthdr)
#define CFG_EXCHDR_ENTRY(exchdr) EXCHDR_ENTRY(exchdr)
#define CFG_INT_ENTRY(inthdr) INT_ENTRY(inthdr)
#define CFG_EXC_ENTRY(exchdr) EXC_ENTRY(exchdr)
```

なお，これらのマクロは，元々は過去のバージョンとの互換性を保つために入れたものである．ただし，過去のバージョンとの互換性が必要なくても，マクロの置換順序を変えるために必要である．

(2) 「INCLUDE」静的 API の処理

システムコンフィギュレーションファイルに含まれる「INCLUDE」静的 API に対応するプリプロセッサディレクティブ（#include）を生成する．例えば，

```
INCLUDE("\"sample1.h\"");
```

という静的 API に対して，

```
#include "sample1.h"
```

というディレクティブを生成する．生成するディレクティブの順序は，システムコンフィギュレーションファイル中での静的 API の順序に一致させる．

(3) 各カーネルオブジェクトに関する定義

システムコンフィギュレーションファイル中に，オブジェクトを生成する静的 API「CRE_XXX」が含まれる各カーネルオブジェクトに関して，オブジェクト生成のための定義を生成する．

コンフィギュレータは，同じ種類のオブジェクトを生成する静的 API を集め，それらを ID 番号の順に並べ替える．同じ ID 番号のオブジェクトを生成する静的 API が複数含まれている場合には，コンフィギュレータがエラーを報告する． JSP カーネルは，オブジェクトの ID 番号が連続していることを仮定して実装してある．ID 番号が連続していない場合には，コンフィギュレータがエラーを報告する．

また，コンフィギュレータは，オブジェクトの ID 番号の代わりに識別子が記述されている場合に，そのオブジェクトに ID 番号を割り付ける．ID 番号は，他のオブジェクトの ID 番号と重複がなく，ID 番号ができる限り連続するように割り付ける．それでも ID 番号が連続にならない場合には，コンフィギュレータがエラーを報告する．

各カーネルオブジェクトに関する定義の標準的な構成は，次の通りである．オブジェクトによって例外がある場合には，オブジェクト毎の項で説明する．

(a) オブジェクトの数

オブジェクトの数をマクロ定義するプリプロセッサディレクティブ（#define）を生成する．具体的には，オブジェクトの省略記号を「XXX」とすると，次のような行を生成する．

```
#define TNUM_XXXID <オブジェクトの数>
```

(b) 最大のオブジェクト ID の変数の定義

最大のオブジェクト ID を持つ変数の定義を生成する．具体的には，オブジェクトの省略記号を「XXX／xxx」とすると，次のような行を生成する．

```
const ID tmax_xxxid = (TMIN_XXXID + TNUM_XXXID - 1);
```

(c) オブジェクトに必要なメモリ領域の定義

オブジェクトによっては，オブジェクトに必要なメモリ領域の定義を生成する．具体的には，オブジェクト毎の項で説明する．

(d) オブジェクトの初期化ブロックの定義

オブジェクトの初期化ブロックの定義を生成する．具体的には，オブジェクトの省略記号を「XXX／xxx」とすると，次のような行を生成する．

```
const XXXINIB xxxinib_table[TNUM_XXXID] = {
    <オブジェクト ID が 1 のオブジェクトの初期化情報>,
    <オブジェクト ID が 2 のオブジェクトの初期化情報>,
        ……
    <オブジェクト ID が tmax_xxxid のオブジェクトの初期化情報>
};
```

オブジェクトの初期化情報の形式は，オブジェクト毎に異なる．具体的には，オブジェクト毎の項で説明する．

(e) オブジェクトのコントロールブロックの定義

オブジェクトのコントロールブロックの定義を生成する．具体的には，オブジェクトの省略記号を「XXX／xxx」とすると，次のような行を生成する．

```
XXXCB xxxcb_table[TNUM_XXXID];
```

(3-1) タスクに関する定義

JSP カーネルは，タスクが一つもないケースに対応していないため，タスクに関する定義は必ず生成しなければならない．

タスクに関する定義のインクルードファイル名とオブジェクトの省略記号は次の通りである．ただし，タスク初期化ブロックのデータ型は TINIB，変数名は tinib_table，タスクコントロールブロックのデータ型は TCB，変数名は tcb_table である（いずれも「TSK／tsk」に代えて「T／t」を用いている）．

インクルードファイル名: task.h

オブジェクトの省略記号: TSK

タスク初期化ブロックには，「CRE_TSK」静的 API で指定される情報に加えて，「DEF_TEX」静的 API で指定される情報を含める．

以下では，システムコンフィギュレーションファイルに次の静的 API が含まれている時に生成すべき情報について述べる．

```
CRE_TSK(tskid, { tskatr, exinf, task, itskpri, stksz, stack });
DEF_TEX(tskid, { texatr, texrtn });
```

(3-1-1) タスクに必要なメモリ領域の定義

タスクに必要なメモリ領域として，タスクのスタック領域がある．生成する各タスク毎に，指定されたサイズのスタック領域を定義する．具体的には，上記の静的 API に対して，次の定義を生成する．

```
static __UNIT_STK <スタック領域の変数名>[__TCOUNT_UNIT_STK(stksz)];
```

ここで，<スタック領域の変数名> は，タスク毎に異なる識別子を生成して用いる．

(3-1-2) タスクの初期化情報

タスク初期化ブロック中に生成するタスクの初期化情報は，次の形式とする．

```
{ tskatr, (VP_INT)(exinf), (FP)(task), INT_PRIORITY(itskpri),
  __TROUND_UNIT_STK(stksz), <スタック領域の変数名>,
  texatr, (FP)(texrtn) }
```

ここで，CRE_TSK に対応する DEF_TEX がない場合には，texatr を TA_NULL，texrtn を NULL とする．

(3-1-3) タスク生成順序テーブルの定義

タスクに対しては，生成された順序（タスクを生成する静的 API が記述された順序）をテーブルに出力する必要がある．これは，タスクの生成された順序で，タスクの初期化（より具体的には，タスクの起動）を行う必要があるためである．

具体的には，次のような行を生成する．

```
const ID torder_table[TNUM_TSKID] = {
    <最初に生成されたタスクのタスク ID>,
    <2 番目に生成されたタスクのタスク ID>,
        ……
    <最後に生成されたタスクのタスク ID>
};
```

(3-1-4) エラー条件

タスクの初期化に関するエラー条件は次の通りである．

- DEF_TEX に対応する CRE_TSK がない場合（E_NOEXS）
- (tskatr & ~(TA_ACT)) が 0 でない場合（E_RSATR）
- (TMIN_TPRI <= itskpri && itskpri <= TMAX_TPRI) でない場合（E_PAR）
- stack が NULL でない場合（E_NOSPT）
- texatr が 0 でない場合（E_RSATR）

※ この他に，task や texrtn がプログラムの開始番地として正しいない場合や，stksz が小さすぎる場合にもエラーとすべきだが，エラー条件がターゲットに依存してしまうため，今後の課題とする．

(3-2) セマフォに関する定義

セマフォに関する定義のインクルードファイル名とオブジェクトの省略記号は次の通りである．

インクルードファイル名: semaphore.h

オブジェクトの省略記号: SEM

以下では，システムコンフィギュレーションファイルに次の静的 API が含まれている時に生成すべき情報について述べる．なお，セマフォに必要なメモリ領域はない．

```
CRE_SEM(semid, { sematr, isemcnt, maxsem });
```

(3-2-1) セマフォの初期化情報

セマフォ初期化ブロック中に生成するセマフォの初期化情報は，次の形式とする．

```
{ sematr, isemcnt, maxsem }
```

(3-2-2) エラー条件

セマフォの初期化に関するエラー条件は次の通りである．

- (sematr & ~(TA_TPRI)) が 0 でない場合（E_RSATR）

- (isemcnt > maxsem) **の場合（**E_PAR**）**
- (1 <= maxsem && maxsem <= TMAX_MAXSEM) **でない場合（**E_PAR**）**

(3-3) イベントフラグに関する定義

イベントフラグに関する定義のインクルードファイル名とオブジェクトの省略記号は次の通りである．

インクルードファイル名: eventflag.h

オブジェクトの省略記号: FLG

以下では，システムコンフィギュレーションファイルに次の静的 API **が含まれている時に生成すべき情報について述べる．なお，イベントフラグに必要なメモリ領域はない．**

```
CRE_FLG(flgid, { flgatr, iflgptn });
```

(3-3-1) イベントフラグの初期化情報

イベントフラグ初期化ブロック中に生成するイベントフラグの初期化情報は，次の形式とする．

```
{ flgatr, iflgptn }
```

(3-3-2) エラー条件

イベントフラグの初期化に関するエラー条件は次の通りである．

- (flgatr & ~(TA_TPRI|TA_CLR)) **が** 0 **でない場合（**E_RSATR**）**

(3-4) データキューに関する定義

データキューに関する定義のインクルードファイル名とオブジェクトの省略記号は次の通りである．

インクルードファイル名: dataqueue.h

オブジェクトの省略記号: DTQ

以下では，システムコンフィギュレーションファイルに次の静的 API **が含まれている時に生成すべき情報について述べる．**

```
CRE_DTQ(dtqid, { dtqatr, dtqcnt, dtq });
```

(3-4-1) データキューに必要なメモリ領域の定義

データキューに必要なメモリ領域として，データキュー領域がある．生成する各データキュー毎に，必要なサイズのデータキュー領域を定義する．具体的には，上記の静的 API **に対して，次の定義を生成する．**

```
#if (dtqcnt) > 0
static VP_INT <データキュー領域の変数名>[dtqcnt];
#else
#define <データキュー領域の変数名> NULL
#endif
```

ここで，<データキュー領域の変数名> は，データキュー毎に異なる識別子を生成して用いる．

(3-4-2) データキューの初期化情報

データキュー初期化ブロック中に生成するデータキューの初期化情報は，次の形式とする．

```
{ dtqatr, dtqcnt, <データキュー領域の変数名> }
```

(3-4-3) エラー条件

データキューの初期化に関するエラー条件は次の通りである．

- (dtqatr & ~(TA_TPRI)) が 0 でない場合（E_RSATR）
- dtq が NULL でない場合（E_NOSPT）

(3-5) メールボックスに関する定義

メールボックスに関する定義のインクルードファイル名とオブジェクトの省略記号は次の通りである．

インクルードファイル名: mailbox.h

オブジェクトの省略記号: MBX

以下では，システムコンフィギュレーションファイルに次の静的 API が含まれている時に生成すべき情報について述べる．なお，メールボックスに必要なメモリ領域はない．

```
CRE_MBX(mbxid, { mbxatr, maxmpri, mprihd });
```

(3-5-1) メールボックスの初期化情報

メールボックス初期化ブロック中に生成するメールボックスの初期化情報は，次の形式とする．

```
{ mbxatr, maxmpri }
```

(3-5-2) エラー条件

メールボックスの初期化に関するエラー条件は次の通りである．

- (mbxatr & ~(TA_TPRI|TA_MPRI)) が 0 でない場合（E_RSATR）
- (TMIN_MPRI <= maxmpri && maxmpri <= TMAX_MPRI) でない場合（E_PAR）
- mprihd が NULL でない場合（E_NOSPT）

(3-6) 固定長メモリプールに関する定義

固定長メモリプールに関する定義のインクルードファイル名とオブジェクトの省略記号は次の通りである．

インクルードファイル名: mempfix.h

オブジェクトの省略記号: MPF

以下では，システムコンフィギュレーションファイルに次の静的 API が含まれている時に生成すべき情報について述べる．

```
CRE_MPF(mpfid, { mpfatr, blkcnt, blksz, mpf });
```

(3-6-1) 固定長メモリプールに必要なメモリ領域の定義

固定長メモリプールに必要なメモリ領域として，固定長メモリプール領域がある．生成する各固定長メモリプール毎に，必要なサイズの固定長メモリプール領域を定義する．具体的には，上記の静的 API に対して，次の定義を生成する．

```
static __MPF_UNIT <固定長メモリプール領域の変数名>
                    [__TCOUNT_MPF_UNIT(blksz) * (blkcnt)];
```

ここで，<固定長メモリプール領域の変数名> は，固定長メモリプール毎に異なる識別子を生成して用いる．

(3-6-2) 固定長メモリプールの初期化情報

固定長メモリプール初期化ブロック中に生成する固定長メモリプールの初期化情報は，次の形式とする．

```
{ mpfatr, __TROUND_MPF_UNIT(blksz), <固定長メモリプール領域の変数名>,
    (VP)(((VB *) <固定長メモリプール領域の変数名>)
        + sizeof(<固定長メモリプール領域の変数名>)) }
```

(3-6-3) エラー条件

固定長メモリプールの初期化に関するエラー条件は次の通りである．

- (mpfatr & ~(TA_TPRI)) が 0 でない場合（E_RSATR）
- blkcnt が 0 の場合（E_PAR）
- blksz が 0 の場合（E_PAR）
- mpf が NULL でない場合（E_NOSPT）

(3-7) 周期ハンドラに関する定義

周期ハンドラに関する定義のインクルードファイル名とオブジェクトの省略記号は次の通りである．

インクルードファイル名: cyclic.h

オブジェクトの省略記号: CYC

以下では，システムコンフィギュレーションファイルに次の静的 API が含まれている時に生成すべき情報について述べる．なお，周期ハンドラに必要なメモリ領域はない．

```
CRE_CYC(cycid, { cycatr, exinf, cychdr, cyctim, cycphs });
```

(3-7-1) 周期ハンドラの初期化情報

周期ハンドラ初期化ブロック中に生成する周期ハンドラの初期化情報は，次の形式とする．

```
{ cycatr, exinf, (FP)(cychdr), cyctim, cycphs }
```

(3-7-2) エラー条件

周期ハンドラの初期化に関するエラー条件は次の通りである．

- (cycatr & ~(TA_STA)) が 0 でない場合（E_RSATR）
- (cyctim > TMAX_RELTIM) の場合（E_PAR）

- (cycphs > TMAX_RELTIM) **の場合（**E_PAR**）**

※ この他に，cychdr **がプログラムの開始番地として正しいない場合にもエラーとすべきだが，エラー条件がターゲットに依存してしまうため，今後の課題とする．**

(4) 割込みハンドラに関する定義

システムコンフィギュレーションファイル中に，割込みハンドラを定義する静的 API「DEF_INH」が含まれている場合に，割込みハンドラに関する定義を生成する．具体的には次の通り．

(4-1) 定義する割込みハンドラの数

定義する割込みハンドラの数をマクロ定義するプリプロセッサディレクティブ（#define）を生成する．また，その値を持つ変数の定義を生成する．具体的には，次のような行を生成する．

```
#define TNUM_INHNO <定義する割込みハンドラの数>
const UINT      tnum_inhno = TNUM_INHNO;
```

(4-2) 割込みハンドラの出入口処理

定義する各割込みハンドラ毎に，割込みハンドラの出入口処理ルーチンを生成する．具体的には，

```
DEF_INH(inhno, { inhatr, inthdr });
```

という静的 API に対して，

```
CFG_INTHDR_ENTRY(inthdr);
```

という行を生成する．

(4-3) 割込みハンドラ初期化ブロックの定義

割込みハンドラ初期化ブロックを生成する．具体的には，次のような行を生成する．

```
const INHINIB inhinib_table[TNUM_INHNO] = {
        <割込みハンドラ 1 の初期化情報>,
        <割込みハンドラ 2 の初期化情報>,
              ……
        <割込みハンドラ TNUM_INHNO の初期化情報>
};
```

この中の割込みハンドラの初期化情報は，次の形式とする．

```
        { inhno, inhatr, (FP)CFG_INT_ENTRY(inthdr) }
```

(4-4) エラー条件

割込みハンドラに関するエラー条件は次の通りである．

- inhatr が 0 でない場合（E_RSATR）

※ この他に，inthdr がプログラムの開始番地として正しいない場合や，inhno が割込みハンドラ番号として正しくない場合にもエラーとすべきだが，エラー条件がターゲットに依存してしまうため，今後の課題とする．

(5) CPU 例外ハンドラに関する定義

システムコンフィギュレーションファイル中に，CPU 例外ハンドラを定義する静的 API「DEF_EXC」が含まれている場合に，CPU 例外ハンドラに関する定義を生成する．具体的には次の通り．

(5-1) 定義する CPU 例外ハンドラの数

定義する CPU 例外ハンドラの数をマクロ定義するプリプロセッサディレクティブ（#define）を生成する．また，その値を持つ変数の定義を生成する．具体的には，次のような行を生成する．

```
#define TNUM_EXCNO <定義する CPU 例外ハンドラの数>
const UINT      tnum_excno = TNUM_EXCNO;
```

(5-2) CPU 例外ハンドラの出入口処理

定義する各 CPU 例外ハンドラ毎に，CPU 例外ハンドラの出入口処理ルーチンを生成する．具体的には，

```
DEF_EXC(excno, { excatr, exchdr });
```

という静的 API に対して，

```
CFG_EXCHDR_ENTRY(exchdr);
```

という行を生成する．

(5-3) CPU 例外ハンドラ初期化ブロックの定義

CPU 例外ハンドラ初期化ブロックを生成する．具体的には，次のような行を生成する．

```
const EXCINIB excinib_table[TNUM_EXCNO] = {
        <CPU 例外ハンドラ 1 の初期化情報>,
        <CPU 例外ハンドラ 2 の初期化情報>,
            ……
        <CPU 例外ハンドラ TNUM_EXCNO の初期化情報>
};
```

この中の CPU 例外ハンドラの初期化情報は，次の形式とする．

```
        { excno, excatr, (FP)CFG_EXC_ENTRY(exchdr) }
```

(5-4) エラー条件

CPU 例外ハンドラに関するエラー条件は次の通りである．

- excatr が 0 でない場合（E_RSATR）

※ この他に，excthdr がプログラムの開始番地として正しいない場合や，excno が CPU 例外ハンドラ番号として正しくない場合にもエラーとすべきだが，エラー条件がターゲットに依存してしまうため，今後の課題とする．

(6) タイムイベント管理に関する定義

タイムイベント管理に関連して，次の定義を生成する．

```
TMEVTN  tmevt_heap[TNUM_TSKID + TNUM_CYCID];
```

(7) 各モジュールの初期化関数の定義

各カーネルオブジェクトの管理，割込み管理，CPU 例外ハンドラ管理の各機能を初期化関数を順に呼び出す関数（object_initialize）を生成する．使用しない機能の初期化関数は，呼び出さない．

すべての機能を使った場合に生成される関数は次の通りである．

```
void
object_initialize()
{
        task_initialize();
        semaphore_initialize();
        eventflag_initialize();
        dataqueue_initialize();
        mailbox_initialize();
        mempfix_initialize();
        cyclic_initialize();
        interrupt_initialize();
        exception_initialize();
}
```

(8) 初期化ルーチンの実行関数の定義

「ATT_INI」静的 API で追加した初期化ルーチンを順に呼び出す関数を生成する．具体的には，

```
ATT_INI({ iniatr, exinf, inirtn });
```

という静的 API に対して，

```
inirtn((VP_INT)(exinf));
```

を呼び出す関数を，call_inirtn という名前で生成する．初期化ルーチンを呼び出す順序は，システムコンフィギュレーションファイル中での静的 API の順序に一致させる．

例えば，

```
ATT_INI({ TA_HLNG, 0, timer_initialize });
ATT_INI({ TA_HLNG, (INT) CONSOLE_PORTID, serial_initialize });
```

という二つの静的 API がこの順序で記述された時に生成する関数は次の通りである．

```
void
call_inirtn()
{
        timer_initialize((VP_INT)(0));
        serial_initialize((VP_INT)((INT) CONSOLE_PORTID));
}
```

(8-1) エラー条件

初期化ルーチンに関するエラー条件は次の通りである．

- iniatr が 0 でない場合（E_RSATR）

※ この他に，inirtn がプログラムの開始番地として正しいない場合にもエラーとすべきだが，エラー条件がターゲットに依存してしまうため，今後の課題とする．

以上