TOPPERS 新世代カーネル用コンフィギュレータ仕様

作成: 2007 年 12 月 7 日
改定 11: 2012 年 10 月 24 日

株式会社きじねこ
高木信尚

目次

改定履歴

2007 年 12 月 7 日	新規作成
2008 年 4 月 1 日	誤記修正 2.1 いくつかのオプションを追加 6. 静的 API テーブルを追記 7. 値取得シンボルテーブルを追記 9. パラメータ計算用 C 言語ファイルを追加
2008 年 4 月 18 日	9.1 パラメータ計算用 C 言語ファイル内で使用するマクロの仕様変更 9.3 パラメータ計算用 C 言語ファイルからインクルードするヘッダファイルを追記
2008 年 4 月 30 日	7.1.3 符号付きフラグにおいて、設定値 1, 0 を s（または signed）, u に変更（元の記述は誤記）。 節番号 6.1.1～6.1.3 を 7.1.1～7.1.3 に修正（元の記述は誤記）。 6. 静的 API テーブルにおいて、複数指定できることを明記。 7. 値取得シンボルテーブルにおいて、複数指定できることを明記。
2008 年 12 月 16 日	--id-input-file をパス 1 でも使用できるように仕様変更 6.1.5 従属フラグの仕様変更 4.2.1 KERNEL_DOMAIN を追加 文字列定数式パラメータを追加
2009 年 1 月 26 日	3. コンフィギュレータの処理モデルにパス 4 を追加
2009 年 5 月 9 日	7.1.2 式に、'#'を用いた場合の振る舞いを追記
2010 年 2 月 23 日	2.1 起動オプションに―with-software-component を追加 2.1 起動オプションに―kernel オプションで'fmp'と'hrp2'の両方を指定できる記述を追加 5.3 ドメイン ID 番号の最大数を追記 6.1.3 パラメータリストに ... に関する記述を追加 7.1.2.式に、#@で始まる場合の記述を追加

2010 年 7 月 23 日	7.1 値取得シンボルテーブルの書式を仕様変更
2011 年 3 月 8 日	7.1.2 式に'@'で始まる形式を追加
2012 年 2 月 15 日	6.1.3 末尾に…を付加した場合の仕様を変更
2012 年 10 月 24 日	4.3.3 DOMAIN や CLASS の囲み全体を条件付き取り込み指令で囲んだ場合の振る舞いを明記

1. 概要

1.1 本仕様書について

本仕様書は、TOPPERS/ASP カーネルを基礎とする「TOPPERS 新世代カーネル」用のコンフィギュレータの仕様を記述したものである。TOPPERS 新世代カーネルそのものについては、現時点では TOPPERS/ASP カーネルに付属のドキュメントを参照のこと。

マクロプロセッサおよびテンプレートファイルに関しては、「TOPPERS 新世代カーネル用コンフィギュレータ内蔵マクロプロセッサ仕様」を参照のこと。

1.2 TOPPERS 新世代カーネル用コンフィギュレータの概要

TOPPERS 新世代カーネル用コンフィギュレータ（以下、単にコンフィギュレータと表記する）は、カーネルやソフトウェア部品の構成や初期状態を定義したシステムコンフィギュレーションファイルを解釈し、システムを構築する上で必要なファイル類を生成するためのツールである。

コンフィギュレータは、コンパイラやアセンブラを初めとする開発ツールと同様、PC 等の開発用コンピュータ上で動作するコマンドラインプログラムである。したがって、コンフィギュレータそのものが最終的な製品に組み込まれることは、原則としてない。

コンフィギュレータはコマンドラインツールであるため、コンパイラを初めとした他のコマンドラインツールと組み合わせて使用することを前提としている。しかし、設定等を行うことで、多くの統合開発環境から呼び出すこともできる。統合開発環境からの呼び出し方法については本書では言及しない。使用する統合開発環境に応じて、ユーザーが適切に設定を行う必要がある。

2. コンフィギュレータの起動

コンフィギュレータを起動するには、コマンドラインシェル（Bash、コマンドプロンプトなど）から、「cfg」というコマンドを呼び出すことで行う。

2.1 起動オプション

コンフィギュレータを実行する際には、適切なオプションを指定する必要がある。オプションはコマンドライン引数として指定する。

オプションは通常二つのハイフン‘--‘ から始まる。オプションによっては一つのハイフンから始まる省略形を持つものもある。また、オプションの中には引数を取るものもある。オプションに引数を指定するには、オプションに続けて、一つ以上の空白の後に引数を記述するか、等号‘=’ の後に引数を記述する。

コンフィギュレータに指定することができるオプションを以下に示す。

--help

オプションの一覧を表示する

--version または -v

コンフィギュレータのバージョン情報を表示する

--kernel または -k =カーネル名

システムを構成すべきカーネル名を小文字で指定する。具体的には、TOPPERS/ASP カーネルに対しては‘asp’ を、TOPPERS/FMP カーネルに対しては‘fmp’ を指定する。何も指定しない場合は、’asp’が指定されたものとして振舞う。

‘fmp+hrp2’または’hrp2+fmp’を指定することで、クラスとドメインの両方を使うことができる。

--pass または -p =パス番号

処理モデルにおけるパス番号を指定する。処理モデルおよびパス番号に関しては、「3. コンフィギュレータの処理モデル」を参照のこと。

--help，—version，および--print-dependencies オプションを指定する場合を除き、このオプションは必ず指定する必要がある。

--include-path または -I =パス名

インクルードファイルの検索パスを指定する。

このオプションは、システムコンフィギュレーションファイルにおける INCLUDE ディレクティブ、およびテンプレートファイルにおける$INCLUDE$命令に影響する。

--template-file または -T =ファイル名

マクロプロセッサに与えるテンプレートファイルを指定する。

このオプションはパス 2 およびパス 3 でのみ使用し、他のパスでは無視される。

--input-charset =文字コード

システムコンフィギュレーションファイルおよびテンプレートファイルに使用し

ている文字コードを指定する。
このオプションは将来に対する予約であり、現時点では機能しない。

--api-table　　=ファイル名
静的 API テーブルを指定する。詳細については、「6. 静的 API テーブル」を参照のこと。

--cfg1-def-table　　=ファイル名
値取得シンボルテーブルを指定する。詳細については、「7. 値取得シンボルテーブル」を参照のこと。

--cfg1_out　　=ファイル名
パス 1 で生成するパラメータ計算用 C 言語ファイル‘cfg1_out.c’の別名を指定する。ただし、拡張子を除いた‘cfg1_out’に代わる部分のみを指定する必要がある。

--rom-image または -r　　=ファイル名
リンク後の ROM イメージのファイル名を指定する。ROM イメージは S レコード形式でなければならない。
このオプションはパス 3 でのみ使用し、他のパスでは無視される。

--symbol-table または -s =ファイル名
リンク後のシンボルテーブルのファイル名を指定する。
シンボルテーブルは、GNU NM コマンドに‘-C’オプションを付けた場合の出力結果と同等でなければならない。すなわち、シンボル名は C 言語のソースファイル上に現れる識別子と同一でなければならない。コンパイラによっては、C 言語の識別子に‘_’を付加するなどの変形を行うが、コンフィギュレータに与えるシンボルテーブルでは、そうした変形後のシンボルではなく、変形前のシンボルに戻す必要がある。

--cfg-directory または -d =ディレクトリ名
コンフィギュレータのプログラム（cfg または cfg.exe）が格納されているディレクトリを指定する。
このオプションを指定しない場合でも、自動的にコンフィギュレータの格納場所を探索するが､環境変数 PATH で設定された内容までは探索対象に含まれないため、PATH の設定に依存した呼び出しを行う場合にはこのオプションを指定する必要がある。

--msgcat-directory または -m　　=ディレクトリ名

メッセージカタログが格納されているディレクトリを指定する。
このオプションを指定しない場合、メッセージカタログはコンフィギュレータのプログラムと同じディレクトリから探索される。

--destination-directory または -n　=ディレクトリ名

コンフィギュレータが生成するファイルの出力先ディレクトリを指定する。
このオプションは将来に対する予約であり、現時点では機能しない。

--id-output-file　　=ファイル名

コンフィギュレータが割り付けたオブジェクト識別名に対応する値の一覧を、指定した名前のファイルとして出力する。
このオプションはパス 1 およびパス 2 でのみ使用し、他のパスでは無視される。

--id-input-file　　=ファイル名

特定のオブジェクト識別名に対応する値を指定するためのファイルを指定する。
ファイルの形式は、‘--id-output-file’オプションによって出力されるファイルと同じである。
このオプションに指定されなかったオブジェクト識別名に対しては、コンフィギュレータが自動的に値を割り付ける。

--alignof-fp　　=数値

旧バージョンとの互換性のためにのみ存在する。

--external-id

このオプションを指定した場合、コンフィギュレータがマクロプロセッサを呼び出す際に、変数‘USE_EXTERNAL_ID’を 1 に設定する。このオプションが指定されていない場合には、変数‘USE_EXTERNAL_ID’を 0 に設定する。
このオプションはパス 2 およびパス 3 でのみ使用し、他のパスでは無視される。

--print-dependencies または -M　　=ファイル名

ファイル名の依存関係を出力する。依存関係は GNU Make に適した形式で出力される。

--with-software-component

このオプションを指定した場合、ソフトウェア部品に対応した処理を行う。具体的な動作は、「TOPPERS 新世代カーネル用コンフィギュレータ内蔵マクロプロセッサ仕様」を参照のこと。

3. コンフィギュレータの処理モデル

新世代カーネル上のシステムを構築するには、コンフィギュレータを合計三度起動する必要がある。それぞれのコンフィギュレータの起動を、最初から順に「パス 1」、「パス 2」、「パス 3」と呼ぶ。また、パス 1 における ‘1’ のように、パスを特定する番号のことを「パス番号」と呼ぶ。

それぞれのパスでは、必要に応じてファイルを生成し、以後のパス、またはコンパイルやリンクに使用することがある。

1 パス 1 では、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解釈し、‘cfg1_out.c’ という C 言語のソースファイルを生成する。

‘cfg1_out.c’ には、システムコンフィギュレーションファイルに記述された静的 API の各パラメータ、使用する C 言語処理系の特性を調べるための式、カーネルやソフトウェア部品に依存する型情報等、およびアプリケーション定義のパラメータが埋め込まれる。

‘cfg1_out.c’ は、使用する C コンパイラ、リンカを用いてコンパイル、リンクを行い、S レコードとシンボルテーブルを生成する。このとき、S レコードには ‘cfg1_out.srec’、シンボルテーブルには ‘cfg1_out.syms’ というファイル名を付ける必要がある。ただし、--cfg1_out オプションを使用する場合には、指定したファイル名が適用される。

2 パス 2 では、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解釈するとともに、パス 1 で生成した ‘cfg1_out.c’ を基に生成された ‘cfg1_out.srec’ および ‘cfg1_out.syms’ をも解釈する。このとき、‘cfg1_out.c’ に埋め込んだ静的 API のパラメータの評価結果を ‘cfg1_out.srec’ から読み取る。同時に、‘cfg1_out.c’ に埋め込んだ他の値も読み取る。

次に、コンフィギュレータに内蔵されたマクロプロセッサを呼び出し、--template-file オプションで指定したテンプレートファイルを解釈・実行する。このとき、‘cfg1_out.srec’ から読み取った値は、対応する変数に設定することにより、マクロプロセッサに渡される。また、システムコンフィギュレーションファイルに記述された情報も、同時にマクロプロセッサに渡される。

パス 2 で解釈・実行するテンプレートファイルには、通常、‘kernel_cfg.h’ および

‘kernel_cfg.c’ という二つのファイルを生成するための雛形が記述されている。この雛形に、コンフィギュレータがマクロプロセッサに渡した情報を埋め込むことで、カーネルの構成や初期状態を定義するためのソースファイルが生成される。

3　パス 3 は、システム全体のリンクが完了した後で、カーネルの構成等が正常に行われているかどうかを判定するために実行される。そのため、パス 3 を省略したとしても、構築される内容に変わりはない。ただし、本来コンフィギュレータが検出すべきエラーをユーザーの責任において検出する必要がある。

パス 3 では、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解釈するとともに、パス 1 で生成した ‘cfg1_out.c’ を基に生成された ‘cfg1_out.srec’ および ‘cfg1_out.syms’ をも解釈する。また、システム全体をリンクして生成した S レコード（--rom-image オプションで指定）とシンボルテーブル（--symbol-table オプションで指定）を読み込む。

次に、コンフィギュレータに内蔵されたマクロプロセッサを呼び出し、--template-file オプションで指定したテンプレートファイルを解釈・実行する。このとき、パス 2 と同様に、‘cfg1_out.srec’ から読み取った値は、対応する変数に設定することにより、マクロプロセッサに渡される。また、システムコンフィギュレーションファイルに記述された情報も、同時にマクロプロセッサに渡される。

パス 3 で解釈されるテンプレートファイルには、通常、パス 2 でエラー検出することができなかった静的 API のパラメータチェック処理が記述される。また、ターゲットハードウェアのメモリマップへの適合性をチェックすることも可能である。

4　パス 4 は、メモリ保護機能を有するカーネル向けにのみ存在する。基本的にはパス 3 の同一の処理を行うが、出力メッセージが若干異なる。

4. システムコンフィギュレーションファイル

4.1 概要

カーネルやシステムサービスが管理するオブジェクトの生成情報や初期状態などを記述するファイルを、システムコンフィギュレーションファイル（system configuration file）と呼ぶ。

システムコンフィギュレーションファイルには，カーネルの静的 API、システムサービスの静的 API、コンフィギュレータに対する INCLUDE ディレクティブ（4.3.2）、コメント（4.3.1）、

C 言語の前処理指令（4.3.3）のみを記述することができる。

4.2 字句

コンフィギュレータが扱う字句は、基本的には C 言語の字句と同等である。ただし、C 言語の前処理指令に用いる‘#’を除き、前処理字句は扱わない。結果として、C 言語ではエラーになる式が、システムコンフィギュレーションファイルではエラーにならない場合がある。

例）
0xfe-1

これは、C 言語では単一の前処理数（前処理字句の一種）とみなされるため、整数定数に変換される際にエラーになる。しかし、コンフィギュレータはこれを {0xfe} {-} {1} の三つの字句と解釈するためエラーにはならない。

この字句解釈の違いにより、コンフィギュレータがエラーを報告しない場合であっても、‘cfg1_out.c’のコンパイル時に C コンパイラがエラーを報告することがある。しかし、コンフィギュレータは静的 API のパラメータの意味を理解することができないため、いずれにせよ、同様のことは頻繁に発生する。具体的には、静的 API のパラメータである定数式中に、何らかの識別子が含まれる場合、それがマクロであるのか、列挙定数であるのか、型名であるのか、あるいはオブジェクトや関数の名前であるのか、コンフィギュレータには知る方法がない。

一般定数式パラメータの式中に同様のことがあった場合、エラーの検出は‘cfg_out.c’のコンパイル時ではなく、’kernel_cfg.c’のコンパイル時まで行うことができない。

4.3 前処理

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に記述された静的 API を解釈する前に、前処理を行う。前処理では、コメントの除去、INCLUDE ディレクティブの解決、および C 言語の前処理指令の除去を行う。

C 言語とは異なり、コンフィギュレータは以下の処理を行わない。

1. 行末に逆斜線‘＼’が現れた場合、物理行の連結による論理行の生成
2. 三文字表記から対応する文字への置換

3. 多バイト文字から国際文字名への変換
4. マクロの展開

4.3.1 コメント

システムコンフィギュレーションファイルでは、C 言語のコメント形式である‘/*’で始まり‘*/’で終わるブロックコメントを記述することができる。また、C++のコメント形式である‘//’で始まり、行末で終わる行コメントも記述することができる。行コメントは、C++のものとは異なり、行末に逆斜線‘＼’が現れた場合でも次の行をコメントとはみなさい。

4.3.2 INCLUDE ディレクティブ

コンフィギュレータに対する INCLUDE ディレクティブは，システムコンフィギュレーションファイルを複数のファイルに分割して記述するために用いるもので、その文法は通りである。

```
include-directive ::= 'INCLUDE' '(' header-name ')' ';'

header-name ::= '<' h-char-sequence '>'
              | '"' q-char-sequence '"'

h-char-sequence ::= h-char
                  | h-char-sequence h-char

h-char ::= [^>]

q-char-sequence ::= q-char
                  | q-char-sequence h-char

q-char ::= [^"]
```

要約すれば次の二種類のいずれかとなり、両者の意味に実質的な違いはない。

```
INCLUDE("ファイル名");
INCLUDE(<ファイル名>);
```

なお、ファイル名の途中に改行や逆斜線 ‘＼’、円記号 ‘¥’ が現れた場合の動作は未定義である。したがって、ディレクトリの区切り子には、逆斜線や円記号ではなく、斜線‘/’を用いるべきである。

システムコンフィギュレーションファイルに INCLUDE ディレクティブが現れた場合、コンフィギュレータは内部的に、その部分を INCLUDE ディレクティブで指定したファイル名の内容に置き換える。

INCLUDE ディレクティブで指定したファイル名は、以下の手順で探索を行う。

1. カレントディレクトリ（コンフィギュレータを実行したディレクトリ）に指定した名前のファイルが存在すれば、そのファイルとする。
2. そうではなく、--include-path オプションで指定したディレクトリに指定した名前のファイルが存在すれば、そのファイルとする。--include-file オプションを複数指定した場合には、最初に指定したディレクトリから順に探索し、最初に見つかったファイルとする。
3. 上記のいずれでもファイルが見つからなかった場合はエラーを報告する。

4.3.3 C 言語の前処理指令

システムコンフィギュレーションファイルに記述することができる C 言語の前処理指令には以下のものがある。

インクルード指令

#include

条件取り込み指令

#if
#ifdef
#ifndef
#else
#elif
#endif

プラグマ指令

#pragma

インクルード指令は，静的 API のパラメータを解釈するために必要な C 言語のヘッダファイルを指定するために用いる。また、条件取り込み指令は，有効とする静的 API を選択するために用いることができる。ただし、インクルード指令は、コンフィギュレータが生成するファイルでは先頭に集められる。そのため、条件取り込み指令の中にインクルード指令を記述しても、インクルード指令は常に有効となる。

例）
システムコンフィギュレーションファイル中に、

```
#ifdef ABC
#include “abc.h”
#endif
```

のように記述した場合、ABC マクロの定義状態如何に関わらず、‘abc.h’ はインクルードされることになる。上記のような記述が必要な場合、‘abc.h’ の内部で条件取り込み指令を記述するか、いったん別のファイルで条件取り込み指令を解決してからシステムコンフィギュレーションファイルにインクルードする必要がある。

一つの静的 API の記述の途中に条件取り込み指令を記述することはできない。また、CLASS や DOMAIN を条件取り込み指令で制御する場合、必ずそれぞれの囲み全体を対象としなければならない。不適切な記述を行った場合の動作は未定義である。また、CLASS や DOMAIN の囲み全体を条件取り込み指令で囲んだ場合、条件に関わらずクラス ID やドメイン ID は割り付けられる。

例）

```
CRE_TSK(TASK1, {
#ifdef ABC   //  静的 API の記述の途中に条件取り込み指令を記述することはできない。
  TA_ACT,
#else
  TA_NULL,
#endif
0, task, MID_PRIORITY, STACK_SIZE, NULL });

#ifdef DEF       //  必ずそれぞれの囲み全体を対象としなければならない。
CLASS(CLS1) {
```

```
#endif
CRE_TSK(TASK2, { TA_NULL, 0, task, MID_PRIORITY, STACK_SIZE, NULL });
#ifdef DEF
}
#endif
```

上記のような記述はいずれも正しい結果を得られない。

プラグマ指令は、唯一 #pragma once のみ記述することができる。他のプラグマ指令を記述した場合の動作は未定義である。#pragma once は、INCLUDE ディレクティブによって同一のファイルが多重にインクルードされることを防ぐ。

4.4 静的 API

通常、カーネルやソフトウェア部品の機能を呼び出すには、サービスコール、すなわち C 言語の関数を実行時に呼び出すことで行う。しかし、カーネルオブジェクトの生成などを、実行時ではなくシステムの構築時に解決するための API が静的 API である。

静的 API は通常、C 言語の関数と類似の形式であるが、中には、他の静的 API を波括弧 '{' と '}' で表したブロックで囲むものもある。

4.2.1 静的 API の文法構造

以下に、静的 API の文法構造を BNF によって表す。

```
statement ::= simple-statement
            | compound-statement

simple-statement ::= api-name '(' parameter-list ')' ';'

compound-statement ::= block-api-name '(' identifier ')'
                       '{' statement '}'
                     | 'KERNEL_DOMAIN' '{' statement '}'

api-name ::= identifier
```

```
block-api-name ::= 'CLASS' | 'DOMAIN'

parameter-list ::= parameter
                 | parameter-list ',' parameter

parameter ::= packet | constant-expression

packet ::= '{' parameter-list '}'
```

ここで、*constant-expression* は C 言語の任意の定数式を意味する。ただし、確実に記述可能な定数式は ISO/IEC 9899:1990（以後、C90 と表記）の範疇とし、また、二文字表記および三文字表記は非対応とする。（現在の実装では、複合リテラルを除く ISO/IEC 9899:1999 の定数式に対応している。ただし、二文字表記および三文字表記は非対応、国際文字名には対応している。）処理系の独自拡張（_near, _far など）にも原則として対応していない。

4.2.2 パラメータ

静的 API のパラメータは、次の三種類に分類される。

(a) オブジェクト識別名

オブジェクトの ID 番号を指定するパラメータ、オブジェクトの名称を表す単一の識別名のみを記述することができる。

コンフィギュレータは、オブジェクト生成のための静的 API（CRE_???）を処理する際に、オブジェクトに ID 番号を割り付け、構成・初期化ヘッダファイルに、指定された識別名を割り付けた ID 番号にマクロ定義する C 言語の#define 指令を生成する。

オブジェクト生成以外の静的 API が、オブジェクトの ID 番号をパラメータに取る場合（カーネルの静的 API では，DEF_TEX の tskid パラメータのみがこれに該当）には、パラメータとして記述する識別名は、生成済みのオブジェクトの名称を表す識別名でなければならない。そうでない場合には、コンフィギュレータがエラーを報告する。

(b) 整数定数式パラメータ

オブジェクト番号や機能コード、オブジェクト属性、サイズや数、優先度など、整数値を指定するパラメータ。プログラムが配置されるアドレスに依存せずに値の決まる整数定数式を記述することができる。

(c) 一般定数式パラメータ

処理単位のエントリ番地、メモリ領域の先頭番地、拡張情報など、アドレスを指定する可能性のあるパラメータ。任意の定数式を記述することができる。

(d) 文字列定数式パラメータ

二重引用符で囲まれた文字列を指定するパラメータ。文字列には C 言語の文字列リテラルと同じ記法を用いる。ただし、ワイド文字列は使用不可。また、連続した複数の文字列リテラルを連結することもできない。

5. オブジェクトの ID 番号の指定

μITRON 4.0 仕様とは異なり、新世代カーネルのシステムコンフィギュレーションファイルでは、オブジェクト識別名に整数定数を指定することができない。そのため、特定のオブジェクト識別名に特定の ID 番号を与えることは原則としてできない。

しかし、コンフィギュレータのオプション機能として、アプリケーション設計者がオブジェクトの ID 番号を指定するための次の機能を提供する。

5.1 --id-input-file オプション

コンフィギュレータの起動時に--id-input-file オプションにより、オブジェクト識別名と ID 番号の対応表を含むファイルを渡すと、コンフィギュレータはそれに従ってオブジェクトに ID 番号を割り付ける。それに従った ID 番号割付けができない場合（ID 番号に抜けができる場合など）には、コンフィギュレータはエラーを報告する。

オブジェクト識別名と ID 番号の対応表を含むファイルは以下の書式とする。

オブジェクト識別名　　ID 番号

上記のように、オブジェクト識別名と ID 番号を対にし、両者を空白文字またはタブ文字で区切る。オブジェクト識別名と ID 番号の対は、一行につき一対だけ記述することができる。

パス 1 で--id-input-file オプションを指定可能な ID 番号は、ドメイン ID 番号だけである。

5.2 --id-output-file オプション

コンフィギュレータは、--id-output-file オプションにより、オブジェクト識別名とコンフィギュレータが割り付けた ID 番号の対応表を含むファイルを、--id-input-file オプショ

ンによりコンフィギュレータに渡すファイルと同じフォーマットで生成する。

5.3 ドメイン ID 番号の最大数

ドメイン ID 番号は、新世代カーネルにおける ACPTN 型が符号無し 32 ビット整数型であることから 32 個を最大とする。

6. 静的 API テーブル

システムコンフィギュレーションファイルに記述することができる静的 API は静的 API テーブルで定義しなければならない。静的 API テーブルは--api-table オプションによって指定する。静的 API テーブルは複数指定することができる。

6.1 静的 API テーブルの書式

静的 API テーブルは CSV（コンマ区切り）形式のファイルとする。ただし、利便性に配慮するため、RFC4180 に合致しないファイルにも対応している。具体的には、改行文字には CR LF 以外に、CR のみ、または LF のみでもかまわない。また、値には多バイト文字を使用することができる。

静的 API テーブルの各レコード（行）は次の形式とする。

[種別],[静的 API 名],[パラメータリスト],[ID 位置],[従属フラグ]

6.1.1 種別

オブジェクトの種類を表すための文字列を指定する。通常、サービスコールにおいてオブジェクトを表す 3 文字を使用する。例えば、タスクであれば tsk となる。ここで指定した種別はテンプレートファイルでパラメータ等を格納する変数名に使用される。ただし、テンプレートファイルにおける変数では常に大文字に変換される。

6.1.2 静的 API 名

システムコンフィギュレーションファイルで記述すべき静的 API の名称を文字列で指定する。

6.1.3 パラメータリスト

静的 API のパラメータの並びを文字列で指定する。各パラメータは空白文字で区切る（コンマは付けないこと）。また、波括弧（{}）も便宜上パラメータとして扱う。波括弧はコンフィギュレータがシステムコンフィギュレーションファイルを構文解析する際に必要となる。

各パラメータには、パラメータ名の先頭に次に挙げるいずれかの記号を付ける。

#	オブジェクト識別名
%	定義済みオブジェクト識別名
.	符号無し整数定数式パラメータ
+	符号付き整数定数式パラメータ
&	一般定数式パラメータ
$	文字列定数式パラメータ

定義済みオブジェクト識別名（%）は、DEF_TEX における tskid のように、他の静的 API で定義されるべきオブジェクト識別名である。

以下に、ASP カーネルにおける CRE_TSK および DEF_TEX のためのパラメータリストを示す。

```
#tskid { +tskatr &exinf &task +itskpri .stksz &stk }
%tskid { +texatr &texrtn }
```

パラメータ名の末尾に ? を付加した場合、静的 API を記述する際にそのパラメータを省略することができる。ただし、 } または) の前のパラメータを後ろから順に省略できるだけとし、途中のパラメータを抜くことはできないものとする。

パラメータの末尾に ... を付加した場合、システムコンフィギュレーションファイルでは、同種（一般定数式パラメータなど）のパラメータを 0 個以上記述することができるものとする。

6.1.4 ID 位置

パラメータリストのうち、ID 番号の位置を指定します。ここでいう ID 番号とは、オブジェクトを識別するための値のことで、オブジェクト識別名のほか、割込みハンドラ番号のような識別番号も含む。

位置は、パラメータリストの最初（一番左）の位置を 0 として、以後 1, 2, 3,…のように

数える。ATT_INI のように、ID 番号に相当するパラメータが存在しない場合は-1 を指定する。
ID 位置は省略することができる。ID 位置を省略した場合は 0 とみなす。

6.1.5 従属フラグ

従属フラグは、DEF_TEX のように、他の静的 API によって定義されたオブジェクト識別名を使用する場合に 1 を指定する。従属フラグが 1 の場合、同一のオブジェクト識別子を用いて、同一の静的 API を複数記述した場合にエラーが発生する。
従属フラグは省略することができる。従属フラグを省略した場合は 0 とみなす。

7. 値取得シンボルテーブル

C 言語の定数式の評価結果の値をテンプレートファイル内で変数として扱いたい場合には、値取得シンボルテーブルを指定する必要がある。値取得シンボルテーブルは--cfg1-def-table オプションで指定する。値取得シンボルテーブルは複数指定することができる。

7.1 値取得シンボルテーブルの書式

値取得シンボルテーブルは CSV（コンマ区切り）形式のファイルとする。ただし、利便性に配慮するため、RFC4180 に合致しないファイルにも対応している。具体的には、改行文字には CR LF 以外に、CR のみ、または LF のみでもかまわない。また、値には多バイト文字を使用することができる。
値取得シンボルテーブルの各レコード（行）は次の形式とする。

[変数名],[式][,符号付きフラグ][,式が真の場合の値][,式が偽の場合の値]

7.1.1 変数名

「変数名」は、テンプレートファイル内で参照するときの変数の名称である。

7.1.2 式

「式」は、「変数名」で指定した変数に設定する値に評価される C 言語の定数式を指定する。「式」の先頭に'#'を付加した場合、もしくは「式が真の場合の値」と「式が偽の場合の値」

のどちらかが設定されている場合、「式」は前処理指令#if の条件式として扱う。ただし、「式」が'#'で始まる場合、先頭の'#'は除去される。
「式」の先頭に'@'を付加した場合、'@'に続く内容はアドレスとみなす。この場合、型に関係なく、指定したアドレスから 8 バイトを変数の値として設定する。

7.1.3 符号付きフラグ

「符号付きフラグ」は、式の評価結果が符号付き整数型の場合には、s または signed を、それ以外は u を指定する。
「符号付きフラグ」は省略することができる。「符号付きフラグ」を省略した場合は u とみなす。

7.1.4 式が真の場合の値

「式が真の場合の値」を指定した場合、「式」は#if の条件式として扱われる。「式」が真であれば、「式が真の場合の値」が「変数名」で指定した変数の値となる。
「式が真の場合の値」は省略することができる。「式が真の場合の値」を省略した場合は 1 とみなす。

7.1.5 式が偽の場合の値

「式が偽の場合の値」を指定した場合、「式」は#if の条件式として扱われる。「式」が偽であれば、「式が偽の場合の値」が「変数名」で指定した変数の値となる。
「式が偽の場合の値」は省略することができる。「式が偽の場合の値」を省略した場合は 0 とみなす。

8. メッセージカタログ

コンフィギュレータが出力するメッセージは基本的にはすべて英語である。しかし、メッセージカタログを適切に定義することにより、異なる言語でメッセージを出力することができるようになる。現時点では、コンフィギュレータ本体とともに配布されるメッセージカタログは日本語のものだけである。

8.1 メッセージカタログの書式

コンフィギュレータが使用するメッセージカタログはテキストファイルであり、文字コ

ードには UTF-8（BOM なし）を使用しなければならない。
メッセージカタログには、コメント、msgid、および msgstr のみを記述することができる。

8.1.1 コメント

メッセージカタログでは、'#' で始まる行はコメントとみなされる。コメント行は、コンフィギュレータに単に読み飛ばされ、動作には影響しない。

8.1.2 msgid

msgid は、コンフィギュレータのソースファイルおよびテンプレートファイル中に記述した英語の文字列を指定する。コンフィギュレータのソースファイルでは _ マクロ、テンプレートファイルでは _ 関数の引数としてこの文字列を渡した場合、対応するメッセージに翻訳することができる。
msgid の書式は以下の通りである。

msgid 文字列リテラル

文字列リテラルは、二重引用符で囲まれた C 言語の文字列をそのまま指定する。

8.1.3 msgstr

msgstr は msgid の直後に記述しなければならない。コンフィギュレータのソースファイルでは _ マクロ、テンプレートファイルでは _ 関数の引数として msgid で指定した文字列を渡した場合、直後に記述した msgstr で指定した文字列に置換される。
msgstr の書式は以下の通りである。

msgstr 文字列リテラル

文字列リテラルは、二重引用符で囲まれた C 言語の文字列をそのまま指定する。

8.1.4 語順の変更について

msgid で指定する文字列の多くは、書式指定を含んでおり、コンフィギュレータ内部またはテンプレートファイルで指定したパラメータに置換される。これは、基本的には C 言語

の printf の書式指定と同じである。しかし、複数の書式指定を含む場合、異なる言語に翻訳する際に語順が問題になる。

しかし、コンフィギュレータが msgid で指定した文字列を解釈するのは printf 系関数ではなく、Boost C++ Libraries に含まれる boost::format クラスである。したがって、boost::format クラスの拡張書式を使用することができる。

多くの場合、boost::format クラスがサポートする拡張書式のうち、%番号% を使用することで語順の問題は解消される。例えば、%1%と記述すれば最初のパラメータを、%2%と指定すれば二番目のパラメータを指定することができる。以下、%3%, %4%, %5%, ..., %n%のように、特定のパラメータを指定することができる。パラメータはその型に応じて適切な書式で変換される。すなわち、文字列であればそのまま、整数値であれば%d 形式で、浮動小数点数であれば%g 形式で変換される。ここで、int 型であるか long 型であるか、double 型であるか long double 型であるかを気にする必要はなく、型に応じて適切な書式が選択される。

8.2 メッセージカタログの選択方法

コンフィギュレータがどのメッセージカタログを使用するかは、環境変数 TOPPERS_CFG_LANG によって決定する。TOPPERS_CFG_LANG の値に‘.po’を続けた名前がメッセージカタログのファイル名となる。

例）
日本語のメッセージカタログ‘ja.po’を使用するには、TOPPERS_CFG_LANG を‘ja’に設定する必要がある。

メッセージカタログは通常、コンフィギュレータのプログラムが格納されたディレクトリに格納される。しかし、--msgcat-directory オプションを指定することで、他のディレクトリにメッセージカタログを格納することもできる。

9. パラメータ計算用 C 言語ファイル

コンフィギュレータは、パス 1 でパラメータ計算用 C 言語ファイルを出力する。パラメータ計算用 C 言語ファイルは、デフォルトでは‘cfg1_out.c’という名称であるが、--cfg1_out オプションによって変更することもできる。

パラメータ計算用 C 言語ファイルには、C 言語処理系の特性、システムコンフィギュレーションファイルに記述された静的 API の整数定数式パラメータ、および値取得シンボルテー

ブルで定義された式の評価結果を取得するためのソースコードが埋め込まれる。

パラメータ計算用 C 言語ファイルは、C コンパイラでコンパイルおよびリンクし、S レコードおよびシンボルテーブルを‘cfg1_out.srec’および‘cfg1_out.syms’という名称で生成する必要がある。これらのファイルは、コンフィギュレータのパス 2 およびパス 3 で読み込まれ、テンプレートファイル内で参照可能な変数として定義される。

9.1 パラメータ計算用 C 言語ファイル内で使用するマクロ

パラメータ計算用 C 言語ファイルでは、いくつかのマクロを予約している。

9.1.1 TOPPERS_CFG1_OUT

パラメータ計算用 C 言語ファイルの先頭で、TOPPERS_CFG1_OUT というマクロが 1 に定義される。このマクロの定義状態を調べることにより、パラメータ計算用 C 言語ファイルとそれ以外で、ターゲット依存部やアプリケーション等のヘッダファイルの宣言・定義内容を切り替えることができる。

9.2 パラメータ計算用 C 言語ファイルにおける定数定義

パラメータ計算用 C 言語ファイル内では、大多数の定義は、signed_t または unsigned_t 型の const 修飾された整数型オブジェクトの形をとる。signed_t および unsigned_t 型は、64 ビット整数型が使える場合には 64 ビット、そうでない場合には 32 ビットの整数型として、パラメータ計算用 C 言語ファイル内で定義される。

C 言語処理系のバイトオーダー判別用の TOPPERS_cfg_magic_number および signed_t 型のサイズを調べるための TOPPERS_cfg_sizeof_signed_t に限り、signed_t または unsigned_t 型ではなく、uint32_t 型を使用する。

パラメータ計算用 C 言語ファイル内で定義される定数は、すべて TOPPERS_cfg_という接頭辞が付く。これは、カーネルやアプリケーションが定義した識別子と衝突しないようにするための配慮である。テンプレートファイル内で変数を指定する場合は、この接頭辞、TOPPERS_cfg_を付ける必要はない。

9.3 パラメータ計算用 C 言語ファイルからインクルードするヘッダファイル

パラメータ計算用 C 言語ファイルからはいくつかのヘッダファイルをインクルードする。

9.3.1 kernel_int.h

カーネルが内部的に使用する型、マクロ等の定義を使用するため、kernel_int.h を必ずインクルードする。

9.3.2 #include 指令で指定したヘッダファイル

システムコンフィギュレーションファイルにおいて、#include 指令で指定したヘッダファイルを記述した順にインクルードする。

9.3.3 target_cfg1_out.h

ターゲット依存部が提供する target_cfg1_out.h をインクルードする。
このヘッダファイルは、cfg1_out.c をコンパイル～リンクするために必要な宣言、定義をすべて含む必要がある。