SunOS リファレンスマニュアル 3 : 拡張ライブラリ関数

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 816–3991–10
2002 年 5 月

原典: *man pages section 3 : Extended Library Functions*

Part No: 816-0217-10

Revision A

Adobe PostScript

020319@2851

目次

はじめに

概要

SunOS リファレンスマニュアルは、初めて SunOS を使用するユーザーやすでにある程度の知識を持っているユーザーのどちらでも対応できるように解説されています。このマニュアルを構成するマニュアルページは一般に参照マニュアルとして作られており、チュートリアルな要素は含んでいません。それぞれのコマンドを実行すると、どのような結果が得られるかについて、詳しく説明されています。なお、各マニュアルページの内容はオンラインでも参照することができます。

このマニュアルは、マニュアルページの内容によっていくつかのセクションに分かれています。各セクションについて以下に簡単に説明します。

- セクション 1 は、オペレーティングシステムで使えるコマンドを説明します。
- セクション 1M は、システム保守や管理用として主に使われるコマンドを説明します。
- セクション 2 は、すべてのシステムコールについて説明します。ほとんどのシステムコールに 1 つまたは複数のエラーがあります。エラーの場合、通常ありえない戻り値が返されます。
- セクション 3 は、さまざまなライブラリ中の関数について説明します。ただし、UNIX システムプリミティブを直接呼び出す関数については、セクション 2 で説明しています。
- セクション 5 は、文字セットテーブルなど他のセクションには該当しないものについて説明します。

以下に、このマニュアルの項目を表記されている順に説明します。ほとんどのマニュアルページが下記の項目からなる共通の書式で書かれていますが、必要でない項目については省略されています。たとえば、記述すべきバグがコマンドにない場合などは、「使用上の留意点」という項目はありません。各マニュアルページの詳細は各セクションの `intro` を、マニュアルページの一般的な情報については `man`(1) を参照してください。

名前　コマンドや関数の名称と概略が示されています。

形式　コマンドや関数の構文が示されています。標準パスにコマンドやファイルが存在しない場合は、フルパス名が示されます。字体は、コマンド、オプションなどの定数にはボールド体 (bold) を、引数、パラメータ、置換文字などの変数にはイタリック体 (Italic) または <日本語訳> を使用しています。オプションと引数の順番は、アルファベット順です。特別な指定が必要な場合を除いて、1 文字の引数、引数のついたオプションの順に書かれています。

以下の文字がそれぞれの項目で使われています。

- [ ]　このかっこに囲まれたオプションや引数は省略できます。このかっこが付いていない場合には、引数を必ず指定する必要があります。
- . . .　省略符号。前の引数に変数を付けたり、引数を複数指定したりできることを意味します (例: ‘ `filename` . . . ’)。
- |　区切り文字 (セパレータ)。この文字で分割されている引数のうち 1 つだけを指定できます。
- { }　この大かっこに囲まれた複数のオプションや引数は省略できます。かっこ内を 1 組として扱います。

プロトコル　この項が使われているのは、プロトコルが記述されているファイルを示すサブセクション 3R だけです。パス名は常にボールド体 (bold) で示されています。

機能説明　コマンドの機能とその動作について説明します。実行時の詳細を説明していますが、オプションの説明や使用例はここでは示されていません。対話形式のコマンド、サブコマンド、リクエスト、マクロ、関数などに関しては「使用法」で説明します。

IOCTL　セクション 7 だけに使用される項です。`ioctl`(2) システムコールへのパラメータは `ioctl` と呼ばれ、適切なパラメータを持つデバイスクラスのマニュアルページだけに記載されています。特定のデバイスに関する `ioctl` は、(そのデバイスのマニュアルページに) アルファベット順に記述されています。デバイスの特定のクラスに関する `ioctl` は、`mtio`(7I) のように `io` で終わる名前が付いているデバイスクラスのマニュアルページに記載されています。

オプション　各オプションがどのように実行されるかを説明しています。「形式」で示されている順に記述されています。オプションの引数はこの項目で説明され、必要な場合はデフォルト値を示します。

オペランド　コマンドのオペランドを一覧表示し、各オペランドがコマンドの動作にどのように影響を及ぼすかを説明しています。

出力　コマンドによって生成される出力 (標準出力、標準エラー、または出力ファイル) を説明しています。

戻り値　値を返す関数の場合、その値を示し、値が返される時の条件を説明しています。関数が 0 や -1 のような一定の値だけを返す場合は、値と説明の形で示され、その他の場合は各関数の戻り値について簡単に説明しています。void として宣言された関数はこの項では扱いません。

エラー　失敗の場合、ほとんどの関数はその理由を示すエラーコードを `errno` 変数の中に設定します。この項ではエラーコードをアルファベット順に記述し、各エラーの原因となる条件について説明します。同じエラーの原因となる条件が複数ある場合は、エラーコードの下にそれぞれの条件を別々のパラグラフで説明しています。

使用法　この項では、使用する際の手がかりとなる説明が示されています。特定の決まりや機能、詳しい説明の必要なコマンドなどが示されています。組み込み機能については、以下の小項目で説明しています。

コマンド
修飾子
変数
式
入力文法

使用例　コマンドや関数の使用例または使用方法を説明しています。できるだけ実際に入力するコマンド行とスクリーンに表示される内容を例にしています。例の中には必ず `example%` のプロンプトが出てきます。スーパーユーザーの場合は `example#` のプロンプトになります。例では、その説明、変数置換の方法、戻り値が示され、それらのほとんどが「形式」、「機能説明」、「オプション」、「使用法」の項からの実例となっています。

環境　コマンドや関数が影響を与える環境変数を記述し、その影響について簡単に説明しています。

終了ステータス　コマンドが呼び出しプログラムまたはシェルに返す値と、その状態を説明しています。通常、正常終了には 0 が返され、0 以外の値はそれぞれのエラー状態を示します。

ファイル　マニュアルページが参照するファイル、関連ファイル、およびコマンドが作成または必要とするファイルを示し、各ファイルについて簡単に説明しています。

属性　属性タイプとその対応する値を定義することにより、コマンド、ユーティリティ、およびデバイスドライバの特性を一覧しています。詳細は `attributes`(5) を参照してください。

関連項目　関連するマニュアルページ、当社のマニュアル、および一般の出版物が示されています。

診断	エラーの発生状況と診断メッセージが示されています。メッセージはボールド体 (bold) で、変数はイタリック体 (Italic) または <日本語訳> で示されており、C ロケール時の表示形式です。
警告	作業に支障を与えるような現象について説明しています。診断メッセージではありません。
注意事項	それぞれの項に該当しない追加情報が示されています。マニュアルページの内容とは直接関係のない事柄も参照用に扱っています。ここでは重要な情報については説明していません。
使用上の留意点	すでに発見されているバグについて説明しています。可能な場合は対処法も示しています。

SunOS リファレンスマニュアル 3 : 拡張ライブラリ関数

名前 | regexpr, compile, step, advance – 正規表現のコンパイルおよび一致ルーチン

形式 | **cc** [*flag*...] [*file*...] -lgen [*library*...]

```
#include <regexpr.h>

char *compile(char *instring, char *expbuf, const char *endbuf);

int step(const char *string, const char *expbuf);

int advance(const char *string, const char *expbuf);

extern char *loc1, loc2, locs;

extern int nbra, regerrno, reglength;

extern char *braslist[], *braelist[];
```

機能説明 | これらのルーチンは、正規表現をコンパイルして、コンパイルした表現を行と一致する場合に使用します。コンパイルされた正規表現は、ed(1) によって使用される形式になります。

instring パラメータは、正規表現を表す NULL で終了する文字列です。

expbuf パラメータは、コンパイルされた正規表現が格納される場所を指しています。*expbuf* が NULL であれば、compile() は malloc(3C) を使用してコンパイルした正規表現にメモリー空間を割り当てます。エラーが発生すると、このメモリー空間は解放されます。コンパイルした正規表現が必要なくなったときに不要なメモリー空間を解放するのは、ユーザーの役割となります。

endbuf パラメータはコンパイルされた正規表現が格納される最高位アドレスより 1 つ上のアドレスです。この引数は *expbuf* が NULL であると無視されます。コンパイルした表現が (*endbuf–expbuf*) バイトに収まらない場合は、compile() は NULL を返し、regerrno (以下を参照) は 50 に設定されます。

string パラメータは、一致について検査が行われる文字列へのポインタです。この文字列は NULL で終わる必要があります。

expbuf パラメータは、compile() 関数を呼び出して得られるコンパイルした正規表現です。

step() 関数は、指定した文字列が正規表現と一致していると 0 以外を返し、一致していないと 0 を返します。一致していると、step() を呼び出す際の副作用として 2 つの外部文字ポインタが設定されます。step() の変数セットは loc1 と loc2 です。loc1 は、正規表現と一致した最初の文字へのポインタです。変数 loc2 は、正規表現に一致する最後の文字の後に来る文字を指しています。したがって、正規表現が行全体と一致していると、loc1 は *string* の最初の文字を指し、loc2 は *string* の最後の NULL を指しています。

step() の目的は、一致を見つけるか、*string* が最後になるまで、*string* 引数を読みながら処理することです。正規表現が ^ から始まっている場合、step() は文字列の先頭でだけ正規表現を一致させようとします。

advance() 関数は step() と同様ですが、advance() は loc2 だけを設定し、つねに文字列の先頭で正規表現を一致させます。

あるユーザーが同一の文字列内で連続する一致を検索している場合には、locs は loc2 と同じに設定し、step() は loc2 と同じ *string* を用いて呼び出す必要があります。s/y*//g のようなグローバル置換が永久にループしないように、またデフォルトでは NULL になるように、locs は ed コマンドおよび sed のようなコマンドで使用されます。

nbra 外部変数は、コンパイルされた正規表現における副表現数を決定する場合に使用します。braslist および braelist は、一致した文字列内の nbra 副表現の先頭と終わりを指している文字列ポインタの配列です。たとえば、文字列 sabcdefg および正規表現 \(abcdef\) を用いて step() または advance() を呼び出すと、braslist[0] は a を指し、braelist[0] は g を指しています。これらの配列は、副表現の *n* 表記法を含む代替置換パターンについて ed および sed のようなコマンドで使用されます。

文字列が正規表現と一致するかどうかだけ検査している場合は、regerrno、nbra、loc1、loc2、locs、braelist、および braslist の各外部変数を使用する必要はないことに注意してください。

使用例

例 **1** 以下の例は grep からの正規表現コードと同じです。

```
#include <regexpr.h>
. . .
if(compile(*argv, (char *)0, (char *)0) == (char *)0)
    regerr(regerrno);
. . .
if (step(linebuf, expbuf))
    succeed( );
```

戻り値

compile() は、成功すると、その値が *expbuf* で異なる非 NULL ポインタを返します。*expbuf* が非 NULL の場合は、compile() はコンパイルした正規表現の最終バイトの後のバイトへのポインタを返します。コンパイルした正規表現の長さは reglength に格納されます。それ以外の場合は、compile() は malloc によって割り当てられたメモリー空間へのポインタを返します。

step() 関数および advance() 関数は、指定した文字列が正規表現に一致する場合には 0 以外を返し、一致しない場合には 0 を返します。

エラー

正規表現のコンパイル中にエラーが検出されると、compile() から NULL ポインタが返され、regerrno は以下に示した 0 でないエラー番号の 1 つに設定されます。

エラー	説明
11	範囲が広すぎます。
16	番号が間違っています。

エラー	説明
25	"\digit" が範囲外です。
36	区切り文字が指定されていないか、間違っています。
41	検索文字列が記憶されていません。
42	\(と \) が合っていません。
43	\(が多すぎます。
44	\{ と \} に 3 つ以上の数が指定されています。
45	\ の後に } がありません。
46	\{ と \} において最初の数字が 2 番目の数を超えています。
49	[と] が合っていません。
50	正規表現がオーバフローしました。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目

ed(1), grep(1), sed(1), malloc(3C), attributes(5), regexp(5)

注意事項

マルチスレッドアプリケーションを コンパイルする際に、 _REENTRANT フラグはコンパイル行に定義しなければなりません。このフラグはマルチスレッドアプリケーションの際にのみ使います。

名前 | regexpr, compile, step, advance – 正規表現のコンパイルおよび一致ルーチン

形式 | **cc** [*flag*...] [*file*...] -lgen [*library*...]

```
#include <regexpr.h>

char *compile(char *instring, char *expbuf, const char *endbuf);

int step(const char *string, const char *expbuf);

int advance(const char *string, const char *expbuf);

extern char *loc1, loc2, locs;

extern int nbra, regerrno, reglength;

extern char *braslist[], *braelist[];
```

機能説明 | これらのルーチンは、正規表現をコンパイルして、コンパイルした表現を行と一致する場合に使用します。コンパイルされた正規表現は、ed(1) によって使用される形式になります。

instring パラメータは、正規表現を表す NULL で終了する文字列です。

expbuf パラメータは、コンパイルされた正規表現が格納される場所を指しています。*expbuf* が NULL であれば、compile() は malloc(3C) を使用してコンパイルした正規表現にメモリー空間を割り当てます。エラーが発生すると、このメモリー空間は解放されます。コンパイルした正規表現が必要なくなったときに不要なメモリー空間を解放するのは、ユーザーの役割となります。

endbuf パラメータはコンパイルされた正規表現が格納される最高位アドレスより 1 つ上のアドレスです。この引数は *expbuf* が NULL であると無視されます。コンパイルした表現が (*endbuf–expbuf*) バイトに収まらない場合は、compile() は NULL を返し、regerrno (以下を参照) は 50 に設定されます。

string パラメータは、一致について検査が行われる文字列へのポインタです。この文字列は NULL で終わる必要があります。

expbuf パラメータは、compile() 関数を呼び出して得られるコンパイルした正規表現です。

step() 関数は、指定した文字列が正規表現と一致していると 0 以外を返し、一致していないと 0 を返します。一致していると、step() を呼び出す際の副作用として 2 つの外部文字ポインタが設定されます。step() の変数セットは loc1 と loc2 です。loc1 は、正規表現と一致した最初の文字へのポインタです。変数 loc2 は、正規表現に一致する最後の文字の後に来る文字を指しています。したがって、正規表現が行全体と一致していると、loc1 は *string* の最初の文字を指し、loc2 は *string* の最後の NULL を指しています。

step() の目的は、一致を見つけるか、*string* が最後になるまで、*string* 引数を読みながら処理することです。正規表現が ^ から始まっている場合、step() は文字列の先頭でだけ正規表現を一致させようとします。

advance() 関数は step() と同様ですが、advance() は loc2 だけを設定し、つねに文字列の先頭で正規表現を一致させます。

あるユーザーが同一の文字列内で連続する一致を検索している場合には、locs は loc2 と同じに設定し、step() は loc2 と同じ *string* を用いて呼び出す必要があります。s/y*//g のようなグローバル置換が永久にループしないように、またデフォルトでは NULL になるように、locs は ed コマンドおよび sed のようなコマンドで使用されます。

nbra 外部変数は、コンパイルされた正規表現における副表現数を決定する場合に使用します。braslist および braelist は、一致した文字列内の nbra 副表現の先頭と終わりを指している文字列ポインタの配列です。たとえば、文字列 sabcdefg および正規表現 \(abcdef\) を用いて step() または advance() を呼び出すと、braslist[0] は a を指し、braelist[0] は g を指しています。これらの配列は、副表現の *n* 表記法を含む代替置換パターンについて ed および sed のようなコマンドで使用されます。

文字列が正規表現と一致するかどうかだけ検査している場合は、regerrno、nbra、loc1、loc2、locs、braelist、および braslist の各外部変数を使用する必要はないことに注意してください。

使用例

例 **1** 以下の例は grep からの正規表現コードと同じです。

```
#include <regexpr.h>
. . .
if(compile(*argv, (char *)0, (char *)0) == (char *)0)
    regerr(regerrno);
. . .
if (step(linebuf, expbuf))
    succeed( );
```

戻り値

compile() は、成功すると、その値が *expbuf* で異なる非 NULL ポインタを返します。*expbuf* が非 NULL の場合は、compile() はコンパイルした正規表現の最終バイトの後のバイトへのポインタを返します。コンパイルした正規表現の長さは reglength に格納されます。それ以外の場合は、compile() は malloc によって割り当てられたメモリー空間へのポインタを返します。

step() 関数および advance() 関数は、指定した文字列が正規表現に一致する場合には 0 以外を返し、一致しない場合には 0 を返します。

エラー

正規表現のコンパイル中にエラーが検出されると、compile() から NULL ポインタが返され、regerrno は以下に示した 0 でないエラー番号の 1 つに設定されます。

エラー	説明
11	範囲が広すぎます。
16	番号が間違っています。

エラー	説明
25	"\digit" が範囲外です。
36	区切り文字が指定されていないか、間違っています。
41	検索文字列が記憶されていません。
42	\(と \) が合っていません。
43	\(が多すぎます。
44	\{ と \} に 3 つ以上の数が指定されています。
45	\ の後に } がありません。
46	\{ と \} において最初の数字が 2 番目の数を超えています。
49	[と] が合っていません。
50	正規表現がオーバフローしました。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目

ed(1), grep(1), sed(1), malloc(3C), attributes(5), regexp(5)

注意事項

マルチスレッドアプリケーションを コンパイルする際に、_REENTRANT フラグはコンパイル行に定義しなければなりません。このフラグはマルチスレッドアプリケーションの際にのみ使います。

名前

config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – 構成の管理

形式

```
cc [ flag ] file -lcfgadm  [ library... ]
#include <config_admin.h>

cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int
     num_ap_ids, char * const *ap_ids, const char *options, struct cfga
     _confirm *confp, struct cfga_msg *msgp, char **errstring, cfga
     _flags_t flags);

cfga_err_t config_private_func(const char *function, int num_ap_ids,
     char * const *ap_ids, const char *options, struct cfga_confirm
     *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags)
     ;

cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const
     char *options, struct cfga_msg *msgp, char **errstring, cfga_flags
     _t flags);

cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids,
     struct cfga_list_data **ap_id_list, int *nlist, const char *options,
     const char *listops, char **errstring, cfga_flags_t flags);

int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
     ap_id2);

void config_unload_libs();

const char *config_strerror(cfga_err_t cfgerrnum);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct
     cfga_stat_data *buf, const char *options, char **errstring);

cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist,
     const char *options, char **errstring);
```

ハードウェアに依存するライブラリの形式

config_admin ライブラリは、動的再構成 (DR: Dynamic Reconfiguration) のための汎用的なインタフェースです。DR に対応している各ハードウェアは、ここに一覧されているエントリポイントを含むハードウェア固有の組み込みライブラリを提供する必要があります。この汎用ライブラリは、操作を実現するために、適切なライブラリを見つけ、接続します。ここにあるインタフェースは、汎用ライブラリのユーザーからは隠されています。ただし、ハードウェア固有の組み込みライブラリを作成するには、これらのインタフェースについて知っておく必要があります。

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char
     *ap_id, const char *options, struct cfga_confirm *confp, struct
     cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_private_func(const char *function, const char *ap_id,
     const char *options, struct cfga_confirm *confp, struct cfga_msg
```

```
    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data *
    *ap_id_list, int *nlist, const char *options, char **errstring);
```

機能説明

config_*() 関数は、ハードウェア固有のシステム構成管理機能に対して、ハードウェアに依存しないインタフェースを提供します。 cfga_*() 関数は、構成管理機能をハードウェア固有の方法で処理するために動的に読み込まれた、ハードウェア固有のライブラリによって提供されます。

libcfgadm ライブラリは、cfgadm(1M) コマンドのサービスを提供するために使用されます。libcfgadm ハードウェア固有のライブラリは /usr/platform/${machine}/lib/cfgadm、/usr/platform/${arch}/lib/cfgadm、/usr/lib/cfgadm にあります。ハードウェア固有のライブラリ名は、接続点を特定するデバイスツリーノードのドライバ名またはクラス名から派生します。

config_change_state() 関数は、システム構成の状態を変更する操作を行います。*state_change_cmd* 引数は、CFGA_CMD_INSERT、CFGA_CMD_REMOVE、CFGA_CMD_DISCONNECT、CFGA_CMD_CONNECT、CFGA_CMD_CONFIGURE、CFGA_CMD_UNCONFIGURE、のいずれかになります。 *state_change_cmd* CFGA_CMD_INSERT は、占有装置の手動による追加の準備をしたり、自動追加を有効にするために使用します。 *state_change_cmd* CFGA_CMD_REMOVE は、占有装置の手動による削除の準備をしたり、自動削除を有効にするために使用します。 *state_change_cmd* CFGA_CMD_DISCONNECT は、受容体に接続されている占有装置との通常の通信を無効にするために使用します。 *state_change_cmd* CFGA_CMD_CONNECTは、受容体に接続されている占有装置との通常の通信を有効にするために使用します。 *state_change_cmd* CFGA_CMD_CONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域に追加し、システムで使用できるようにします。*state_change_cmd* CFGA_CMD_UNCONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域から削除し、システムで使用できないようにします。

flags 引数には、定義済みのフラグである CFGA_FLAG_FORCE とCFGA_FLAG_VERBOSE のいずれかまたは両方を指定することができます。CFGA_FLAG_FORCE フラグを指定すると、特定の安全検査が無効になります。たとえば、このフラグは、failed 状態の占有装置が構成されることを許しませんが、failing 状態の占有装置が構成されることは許す場合があります。強制を受け付けるかどうかは、ハードウェアに依存します。CFGA_FLAG_VERBOSE フラグを指定すると、操作に関係するハードウェア固有の詳細情報が cfga_msg の方法で出力されます。

config_private_func() 関数は、専用ハードウェア固有の関数を起動します。

config_test() 関数は、特定の接続点の検査を開始するために使用します。

num_ap_ids 引数は、*ap_ids* 配列内の *ap_id* の数を特定します。 *ap_ids* 引数は、*ap_id* による配列を示します。

ap_id 引数は、単独の *ap_id* を示します。

function と *options* の文字列は、getsubopt(3C) 構文の規約に従い、ハードウェア固有の関数やオプション情報を提供するために使用します。ハードウェアに依存しない汎用的な関数やオプションは定義されていません。

confp によって参照される cfga_confirm 構造体は、要求された操作がサービスに対する顕著な割り込みを必要とする場合に、継続する許可を得るためのコールバックインタフェースを提供します。 cfga_confirm 構造体には以下のメンバーが含まれます。

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

confirm() 関数は、汎用ポインタの *appdata_ptr* と、確認を必要とするものの詳細を示すメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ *appdata_ptr* は、cfga_confirm 構造体のメンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で confirm 関数の呼び出しをconfig_* の呼び出しに関連付けるために使用されます。confirm 関数は、操作の継続が認められた場合は 1 を返し、認められなかった場合は 0 を返します。

msgp によって参照される cfga_msg 構造体は、ハードウェア固有のライブラリからのメッセージを出力するためのコールバックインタフェースを提供します。CFGA_FLAG_VERBOSE フラグが存在することによって、これらのメッセージは情報メッセージにもなりますが、このフラグが存在しない場合は、エラーメッセージに限定されます。cfga_msg 構造体には、以下のメンバーが含まれます。

```
void (*message_routine)(void *appdata_ptr, const char *message);
void    *appdata_ptr;
```

message_routine() 関数は、汎用ポインタの *appdata_ptr* とメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ*appdata_ptr* は、cfga_confirm 構造体メンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で message_routine() 関数の呼び出しを config_*() の呼び出しに関連付けるために使用されます。このメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

いくつかの一般的なエラーについて、ハードウェア固有のエラーメッセージが返されることがあります。エラーメッセージの文字列(終わりの NULL 文字を含む) の記憶領域は、malloc(3C) と、*errstring* を通じて返される、この記憶領域に対するポインタを使用して config_* 関数によって割り当てられます。*errstring* がNULL の場合は、エラーメッセージは作成または返されることはありません。*errstring* が NULL ではなく、エラーメッセージが作成されない場合は、*errstring* によって参照されるポインタが NULLに設定されます。free(3C) を使用して、返された記憶領域の割り当ての解除は、config_*() を呼び出している関数が行います。このエラーメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

config_list_ext() 関数は、リスト出力用インタフェースです。最初の 2 つの引数を使用して *ap_id*s にリストが指定されていると、指定されている接続点ごとに cfga_list_data_t 構造体の配列を返します。最初の 2 つの引数がそれぞれ 0 と NULL の場合は、デバイスツリーの接続点をすべて出力します。また、フラグの引数を使用して CFGA_FLAG_LIST_ALL フラグを渡した場合は、接続点を動的に展開して動的な接続点を出力する必要があります。返した配列の stat 構造の記憶領域は、config_list_ext() 関数がmalloc(3C) を使用して割り当てます。この記憶領域は、config_list_ext() の呼び出し元が free(3C) を使用して解放しなければなりません。

cfga_list_data 構造体には、以下のメンバーが含まれます。

```
cfga_log_ext_t     ap_log_id;        /* Attachment point logical id */
cfga_phys_ext_t    ap_phys_id;       /* Attachment point physical id */
cfga_class_t       ap_class;         /* Attachment point class */
cfga_stat_t        ap_r_state;       /* Receptacle state */
cfga_stat_t        ap_o_state;       /* Occupant state */
cfga_cond_t        ap_cond;          /* Attachment point condition */
cfga_busy_t        ap_busy;          /* Busy indicator */
time_t             ap_status_time;   /* Attachment point last change*/
cfga_info_t        ap_info;          /* Miscellaneous information */
cfga_type_t        ap_type;          /* Occupant type */
```

種類は以下のように定義されます。

```
typedef char cfga_log_ext_t[CFGA_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];
typedef char cfga_class_t[CFGA_CLASS_LEN];
typedef char cfga_info_t[CFGA_INFO_LEN];
typedef char cfga_type_t[CFGA_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;
```

config_list_ext() の *listopts* 引数は、getsubopt (3C) 構文に準拠しており、サブオプションのリストを渡すときに使用します。現在、サポートされているサブオプションは、*class*=class_name だけです。このリストオプションは、class_name クラスの接続点だけを出力します。

cfga_list_ext() の *listopts* 引数は、今後の使用のために予約されこの引数がNULL の場合は、ハードウェア固有のライブラリはこの引数を無視する必要がありま す。*listopts* 引数がNULL 以外のときに、ハードウェア固有のライブラリでサポートされていない場合は、適切なエラーコードを返す必要があります。

ap_log_id と ap_phys_id メンバーは、接続点に対するハードウェア固有の論理名と物理名を示します。ap_busy メンバーは、状態 (state) や条件 (condition) の変更が生じるような動作があることを示します。ap_status_time メンバーは、接続点のap_r_state、ap_o_state、ap_cond フィールドのいずれかが最後に変わった時間を示します。ap_info メンバーは、接続点に関する追加情報を得るためにハードウェア固有のコードで使用することができます。ap_class メンバーには、接続点の接続点クラスが含まれます (存在する場合)。ap_class メンバーは、汎用ライブラリから組み込まれます。ap_log_id および ap_phys_id メンバーがハードウェア固有のライブラリから組み込まれなかった場合は、汎用ライブラリから汎用形式で組み込まれます。残りのメンバーは、対応するハードウェア固有のライブラリから組み込まれます。

ap_log_id、ap_phys_id、ap_info、ap_class、および ap_type メンバーは、固定長文字列です。実際の文字列がメンバーのサイズよりも短い場合には、この文字列は空文字で終了します。そのため、プログラムは、文字列が空文字で終了していることを前提としてはなりません。これらのフィールドを印刷する場合は、以下の形式を使用してください。

printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);

config_stat()、config_list()、cfga_stat()、cfga_list() 関数およびcfga_stat_data データ構造体は、推奨されていないインタフェースで、下位互換性を維持するためにだけ提供されています。これらのインタフェースはできるだけ使用しないでください。

config_ap_id_cmp 関数は、2 つの *ap_id* に関する、ハードウェアに依存する比較を行い、strcmp(3C) の形式に従って、等しい、小さい、大きい、の結果を返します。各引数 は、cfga_ap_id_t とNULL で終了する文字列のいずれかになります。この関数は、qsort(3C) などによって *ap_id* の一覧をソートする場合や、config_list 関数の呼び出しの結果からエントリを選択する場合に使用することができます。

config_unload_libs 関数は、これまで読み込んだハードウェア固有のライブラリすべてのリンクを解除します。

config_strerror 関数を使用して、エラーメッセージの文字列に対してエラーの戻り値を割り当てることができます。「戻り値」を参照してください。返された文字列は、上書きされてはなりません。 *cfgerrnum* が範囲外にある場合に、config _strerror は NULL を返します。

cfga_help 関数を使用して、ハードウェア固有のライブラリに対して現地仕様のヘルプメッセージを表示するように要求することができます。

戻り値

以下は、config_*() および cfga_*() 関数によって返される戻り値の一覧です。戻り値が CFGA_OK ではない場合は、追加のエラー情報が *errstring* を通して返されます。詳細は、「機能説明」を参照してください。

CFGA_BUSY	システム構成管理システムの 1 要素が使用中であるために、コマンドは完了しませんでした。
CFGA_ATTR_INVAL	指定された属性の接続点は存在しません。
CFGA_ERROR	要求された操作の処理中にエラーが発生しました。このエラーコードは、ハードウェア固有のコードによるコマンド引数の妥当性検査を含みます。
CFGA_INSUFFICIENT_CONDITION	接続点の条件によって、操作が失敗しました。
CFGA_INVAL	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_LIB_ERROR	記憶域やファイル記述子などの処理用資源の獲得の失敗など、ライブラリで手続き上のエラーが発生しました。
CFGA_NACK	*confp*->confirm 関数からの否定応答を受けたために、コマンドは完了しませんでした。
CFGA_NO_LIB	指定された *ap_id* を使用して、ハードウェア固有のライブラリが見つかりませんでした。
CFGA_NOTSUPP	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_OK	要求通りにコマンドが完了しました。
CFGA_OPNOTSUPP	この接続点は、システム構成管理操作に対応していません。
CFGA_PRIV	呼び出し元は要求された処理の特権を所有していません。たとえば、構成管理がデバイスドライバを介して実行される場合は、デバイスノードのアクセス権によってアクセスが制御されます。
CFGA_SYSTEM_BUSY	コマンドがサービスの中断を要求したが、システムの一部が休止できなかったために完了しませんでした。

エラー

システム構成管理機能から返されるエラーの多くは、ハードウェア固有のものです。*errstring* を通して返される文字列には以下があります。

`attachment point` *ap_id* `not known`
エラーメッセージで説明されている接続点は存在しません。

`unknown hardware option` *option* `for` *operation*
未知のオプションが *options* 文字列の中で検出されました。

`hardware option` *option*option `requires a value`
options 文字列で指定されるオプションは、*option=value* の形式である必要があります。

`listing option` *list_option* `requires a value`
listopts 文字列で指定されるオプションは、*option*=`value` 形式である必要があります。

`hardware option` *option* `does not require a value`
options 文字列で指定されるオプションは、単純なオプションである必要があります。

`attachment point` *ap_id* `is not configured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、`CFGA_CMD _UNCONFIGURE` を行う *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `is not unconfigured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、未構成の占有装置を必要とする *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `condition not satisfactory.`
操作の妨げとなる条件を持つ接続点に対して、*config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `in condition` *condition* `cannot be used`
ハードウェアに依存する検査に通らない状態の接続点に対して、強制指定された *config_change_state* 操作が適用されました。

属性

以下の属性については、`attributes`(5) を参照してください。

属性の種類	属性の値
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

関連項目

`cfgadm`(1M), `devinfo`(1M), `dlopen`(3DL), `dlsym`(3DL), `free`(3C), `getsubopt` (3C) , `malloc`(3C), `qsort`(3C), `setlocale`(3C), `strcmp`(3C), `libcfgadm`(3LIB), `attributes`(5)

注意事項

このライブラリを使用するアプリケーションは、実際の実装環境が、外部の `errno` 変数の内容を変更するシステムサービスやファイル記述リソースを使用する可能性があることを考慮したものである必要があります。

以下のコードは、config_*() が CFGA_OK 以外の値を返した場合のエラー処理です。

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

システム構成管理で使用することができる機能に関する詳細は、ハードウェア固有のマニュアルを参照してください。

名前

config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – 構成の管理

形式

```
cc [ flag ] file -lcfgadm  [ library... ]
#include <config_admin.h>

cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int
     num_ap_ids, char * const *ap_ids, const char *options, struct cfga
     _confirm *confp, struct cfga_msg *msgp, char **errstring, cfga
     _flags_t flags);

cfga_err_t config_private_func(const char *function, int num_ap_ids,
     char * const *ap_ids, const char *options, struct cfga_confirm
     *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags)
     ;

cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const
     char *options, struct cfga_msg *msgp, char **errstring, cfga_flags
     _t flags);

cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids,
     struct cfga_list_data **ap_id_list, int *nlist, const char *options,
     const char *listops, char **errstring, cfga_flags_t flags);

int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
     ap_id2);

void config_unload_libs();

const char *config_strerror(cfga_err_t cfgerrnum);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct
     cfga_stat_data *buf, const char *options, char **errstring);

cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist,
     const char *options, char **errstring);
```

ハードウェアに依存するライブラリの形式

config_admin ライブラリは、動的再構成 (DR: Dynamic Reconfiguration) のための汎用的なインタフェースです。DR に対応している各ハードウェアは、ここに一覧されているエントリポイントを含むハードウェア固有の組み込みライブラリを提供する必要があります。この汎用ライブラリは、操作を実現するために、適切なライブラリを見つけ、接続します。ここにあるインタフェースは、汎用ライブラリのユーザーからは隠されています。ただし、ハードウェア固有の組み込みライブラリを作成するには、これらのインタフェースについて知っておく必要があります。

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char
     *ap_id, const char *options, struct cfga_confirm *confp, struct
     cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_private_func(const char *function, const char *ap_id,
     const char *options, struct cfga_confirm *confp, struct cfga_msg
```

```
    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data *
    *ap_id_list, int *nlist, const char *options, char **errstring);
```

機能説明

config_*() 関数は、ハードウェア固有のシステム構成管理機能に対して、ハードウェアに依存しないインタフェースを提供します。 cfga_*() 関数は、構成管理機能をハードウェア固有の方法で処理するために動的に読み込まれた、ハードウェア固有のライブラリによって提供されます。

libcfgadm ライブラリは、cfgadm(1M) コマンドのサービスを提供するために使用されます。libcfgadm ハードウェア固有のライブラリは /usr/platform/${machine}/lib/cfgadm、/usr/platform/${arch}/lib/cfgadm、/usr/lib/cfgadm にあります。ハードウェア固有のライブラリ名は、接続点を特定するデバイスツリーノードのドライバ名またはクラス名から派生します。

config_change_state() 関数は、システム構成の状態を変更する操作を行います。*state_change_cmd* 引数は、CFGA_CMD_INSERT、CFGA_CMD_REMOVE、CFGA_CMD_DISCONNECT、CFGA_CMD_CONNECT、CFGA_CMD_CONFIGURE、CFGA_CMD_UNCONFIGURE、のいずれかになります。 *state_change_cmd* CFGA_CMD_INSERT は、占有装置の手動による追加の準備をしたり、自動追加を有効にするために使用します。 *state_change_cmd* CFGA_CMD_REMOVE は、占有装置の手動による削除の準備をしたり、自動削除を有効にするために使用します。 *state_change_cmd* CFGA_CMD_DISCONNECT は、受容体に接続されている占有装置との通常の通信を無効にするために使用します。 *state_change_cmd* CFGA_CMD_CONNECTは、受容体に接続されている占有装置との通常の通信を有効にするために使用します。 *state_change_cmd* CFGA_CMD_CONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域に追加し、システムで使用できるようにします。*state_change_cmd* CFGA_CMD_UNCONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域から削除し、システムで使用できないようにします。

flags 引数には、定義済みのフラグである CFGA_FLAG_FORCE とCFGA_FLAG_VERBOSE のいずれかまたは両方を指定することができます。CFGA_FLAG_FORCE フラグを指定すると、特定の安全検査が無効になります。たとえば、このフラグは、failed 状態の占有装置が構成されることを許しませんが、failing 状態の占有装置が構成されることは許す場合があります。強制を受け付けるかどうかは、ハードウェアに依存します。CFGA_FLAG_VERBOSE フラグを指定すると、操作に関係するハードウェア固有の詳細情報が cfga_msg の方法で出力されます。

config_private_func() 関数は、専用ハードウェア固有の関数を起動します。

config_test() 関数は、特定の接続点の検査を開始するために使用します。

num_ap_ids 引数は、*ap_ids* 配列内の *ap_id* の数を特定します。 *ap_ids* 引数は、*ap_id* による配列を示します。

ap_id 引数は、単独の *ap_id* を示します。

function と *options* の文字列は、getsubopt(3C) 構文の規約に従い、ハードウェア固有の関数やオプション情報を提供するために使用します。ハードウェアに依存しない汎用的な関数やオプションは定義されていません。

confp によって参照される cfga_confirm 構造体は、要求された操作がサービスに対する顕著な割り込みを必要とする場合に、継続する許可を得るためのコールバックインタフェースを提供します。 cfga_confirm 構造体には以下のメンバーが含まれます。

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

confirm() 関数は、汎用ポインタの *appdata_ptr* と、確認を必要とするものの詳細を示すメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ *appdata_ptr* は、cfga_confirm 構造体のメンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で confirm 関数の呼び出しをconfig_* の呼び出しに関連付けるために使用されます。confirm 関数は、操作の継続が認められた場合は 1 を返し、認められなかった場合は 0 を返します。

msgp によって参照される cfga_msg 構造体は、ハードウェア固有のライブラリからのメッセージを出力するためのコールバックインタフェースを提供します。CFGA_FLAG_VERBOSE フラグが存在することによって、これらのメッセージは情報メッセージにもなりますが、このフラグが存在しない場合は、エラーメッセージに限定されます。cfga_msg 構造体には、以下のメンバーが含まれます。

```
void (*message_routine)(void *appdata_ptr, const char *message);
void    *appdata_ptr;
```

message_routine() 関数は、汎用ポインタの *appdata_ptr* とメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ*appdata_ptr* は、cfga_confirm 構造体メンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で message_routine() 関数の呼び出しを config_*() の呼び出しに関連付けるために使用されます。このメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

いくつかの一般的なエラーについて、ハードウェア固有のエラーメッセージが返されることがあります。エラーメッセージの文字列(終わりの NULL 文字を含む) の記憶領域は、malloc(3C) と、*errstring* を通じて返される、この記憶領域に対するポインタを使用して config_* 関数によって割り当てられます。*errstring* がNULL の場合は、エラーメッセージは作成または返されることはありません。*errstring* が NULL ではなく、エラーメッセージが作成されない場合は、*errstring* によって参照されるポインタが NULLに設定されます。free(3C) を使用して、返された記憶領域の割り当ての解除は、config_*() を呼び出している関数が行います。このエラーメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

config_list_ext() 関数は、リスト出力用インタフェースです。最初の 2 つの引数を使用して *ap_id*s にリストが指定されていると、指定されている接続点ごとに cfga_list_data_t 構造体の配列を返します。最初の 2 つの引数がそれぞれ 0 と NULL の場合は、デバイスツリーの接続点をすべて出力します。また、フラグの引数を使用して CFGA_FLAG_LIST_ALL フラグを渡した場合は、接続点を動的に展開して動的な接続点を出力する必要があります。返した配列の stat 構造の記憶領域は、config_list_ext() 関数がmalloc(3C) を使用して割り当てます。この記憶領域は、config_list_ext() の呼び出し元が free(3C) を使用して解放しなければなりません。

cfga_list_data 構造体には、以下のメンバーが含まれます。

```
cfga_log_ext_t     ap_log_id;        /* Attachment point logical id */
cfga_phys_ext_t    ap_phys_id;       /* Attachment point physical id */
cfga_class_t       ap_class;         /* Attachment point class */
cfga_stat_t        ap_r_state;       /* Receptacle state */
cfga_stat_t        ap_o_state;       /* Occupant state */
cfga_cond_t        ap_cond;          /* Attachment point condition */
cfga_busy_t        ap_busy;          /* Busy indicator */
time_t             ap_status_time;   /* Attachment point last change*/
cfga_info_t        ap_info;          /* Miscellaneous information */
cfga_type_t        ap_type;          /* Occupant type */
```

種類は以下のように定義されます。

```
typedef char cfga_log_ext_t[CFGA_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];
typedef char cfga_class_t[CFGA_CLASS_LEN];
typedef char cfga_info_t[CFGA_INFO_LEN];
typedef char cfga_type_t[CFGA_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;
```

config_list_ext() の *listopts* 引数は、getsubopt (3C) 構文に準拠しており、サブオプションのリストを渡すときに使用します。現在、サポートされているサブオプションは、*class*=class_name だけです。このリストオプションは、class_name クラスの接続点だけを出力します。

cfga_list_ext() の *listopts* 引数は、今後の使用のために予約されこの引数がNULL の場合は、ハードウェア固有のライブラリはこの引数を無視する必要がありま す。*listopts* 引数がNULL 以外のときに、ハードウェア固有のライブラリでサポートされていない場合は、適切なエラーコードを返す必要があります。

ap_log_id と ap_phys_id メンバーは、接続点に対するハードウェア固有の論理名と物理名を示します。ap_busy メンバーは、状態 (state) や条件 (condition) の変更が生じるような動作があることを示します。ap_status_time メンバーは、接続点のap_r_state、ap_o_state、ap_cond フィールドのいずれかが最後に変わった時間を示します。ap_info メンバーは、接続点に関する追加情報を得るためにハードウェア固有のコードで使用することができます。ap_class メンバーには、接続点の接続点クラスが含まれます (存在する場合)。ap_class メンバーは、汎用ライブラリから組み込まれます。ap_log_id および ap_phys_id メンバーがハードウェア固有のライブラリから組み込まれなかった場合は、汎用ライブラリから汎用形式で組み込まれます。残りのメンバーは、対応するハードウェア固有のライブラリから組み込まれます。

ap_log_id、ap_phys_id、ap_info、ap_class、および ap_type メンバーは、固定長文字列です。実際の文字列がメンバーのサイズよりも短い場合には、この文字列は空文字で終了します。そのため、プログラムは、文字列が空文字で終了していることを前提としてはなりません。これらのフィールドを印刷する場合は、以下の形式を使用してください。

```
printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);
```

config_stat()、config_list()、cfga_stat()、cfga_list() 関数およびcfga_stat_data データ構造体は、推奨されていないインタフェースで、下位互換性を維持するためにだけ提供されています。これらのインタフェースはできるだけ使用しないでください。

config_ap_id_cmp 関数は、2 つの *ap_id* に関する、ハードウェアに依存する比較を行い、strcmp(3C) の形式に従って、等しい、小さい、大きい、の結果を返します。各引数 は、cfga_ap_id_t とNULL で終了する文字列のいずれかになります。この関数は、qsort(3C) などによって *ap_id* の一覧をソートする場合や、config_list 関数の呼び出しの結果からエントリを選択する場合に使用することができます。

config_unload_libs 関数は、これまで読み込んだハードウェア固有のライブラリすべてのリンクを解除します。

config_strerror 関数を使用して、エラーメッセージの文字列に対してエラーの戻り値を割り当てることができます。「戻り値」を参照してください。返された文字列は、上書きされてはなりません。 *cfgerrnum* が範囲外にある場合に、config _strerror は NULL を返します。

cfga_help 関数を使用して、ハードウェア固有のライブラリに対して現地仕様のヘルプメッセージを表示するように要求することができます。

戻り値

以下は、config_*() および cfga_*() 関数によって返される戻り値の一覧です。戻り値が CFGA_OK ではない場合は、追加のエラー情報が *errstring* を通して返されます。詳細は、「機能説明」を参照してください。

CFGA_BUSY	システム構成管理システムの 1 要素が使用中であるために、コマンドは完了しませんでした。
CFGA_ATTR_INVAL	指定された属性の接続点は存在しません。
CFGA_ERROR	要求された操作の処理中にエラーが発生しました。このエラーコードは、ハードウェア固有のコードによるコマンド引数の妥当性検査を含みます。
CFGA_INSUFFICIENT_CONDITION	接続点の条件によって、操作が失敗しました。
CFGA_INVAL	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_LIB_ERROR	記憶域やファイル記述子などの処理用資源の獲得の失敗など、ライブラリで手続き上のエラーが発生しました。
CFGA_NACK	*confp*->confirm 関数からの否定応答を受けたために、コマンドは完了しませんでした。
CFGA_NO_LIB	指定された *ap_id* を使用して、ハードウェア固有のライブラリが見つかりませんでした。
CFGA_NOTSUPP	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_OK	要求通りにコマンドが完了しました。
CFGA_OPNOTSUPP	この接続点は、システム構成管理操作に対応していません。
CFGA_PRIV	呼び出し元は要求された処理の特権を所有していません。たとえば、構成管理がデバイスドライバを介して実行される場合は、デバイスノードのアクセス権によってアクセスが制御されます。
CFGA_SYSTEM_BUSY	コマンドがサービスの中断を要求したが、システムの一部が休止できなかったために完了しませんでした。

エラー

システム構成管理機能から返されるエラーの多くは、ハードウェア固有のものです。*errstring* を通して返される文字列には以下があります。

`attachment point` *ap_id* `not known`
エラーメッセージで説明されている接続点は存在しません。

`unknown hardware option` *option* `for` *operation*
未知のオプションが *options* 文字列の中で検出されました。

`hardware option` *option*option `requires a value`
options 文字列で指定されるオプションは、*option=value* の形式である必要があります。

`listing option` *list_option* `requires a value`
listopts 文字列で指定されるオプションは、*option*=`value` 形式である必要があります。

`hardware option` *option* `does not require a value`
options 文字列で指定されるオプションは、単純なオプションである必要があります。

`attachment point` *ap_id* `is not configured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、`CFGA_CMD _UNCONFIGURE` を行う *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `is not unconfigured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、未構成の占有装置を必要とする *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `condition not satisfactory.`
操作の妨げとなる条件を持つ接続点に対して、*config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `in condition` *condition* `cannot be used`
ハードウェアに依存する検査に通らない状態の接続点に対して、強制指定された *config_change_state* 操作が適用されました。

属性

以下の属性については、`attributes`(5) を参照してください。

属性の種類	属性の値
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

関連項目

`cfgadm`(1M), `devinfo`(1M), `dlopen`(3DL), `dlsym`(3DL), `free`(3C), `getsubopt` (3C) , `malloc`(3C), `qsort`(3C), `setlocale`(3C), `strcmp`(3C), `libcfgadm`(3LIB), `attributes`(5)

注意事項

このライブラリを使用するアプリケーションは、実際の実装環境が、外部の `errno` 変数の内容を変更するシステムサービスやファイル記述リソースを使用する可能性があることを考慮したものである必要があります。

以下のコードは、config_*() が CFGA_OK 以外の値を返した場合のエラー処理です。

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

システム構成管理で使用することができる機能に関する詳細は、ハードウェア固有のマニュアルを参照してください。

名前

config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – 構成の管理

形式

```
cc [ flag ] file -lcfgadm  [ library... ]
#include <config_admin.h>

cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int
     num_ap_ids, char * const *ap_ids, const char *options, struct cfga
     _confirm *confp, struct cfga_msg *msgp, char **errstring, cfga
     _flags_t flags);

cfga_err_t config_private_func(const char *function, int num_ap_ids,
     char * const *ap_ids, const char *options, struct cfga_confirm
     *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags)
     ;

cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const
     char *options, struct cfga_msg *msgp, char **errstring, cfga_flags
     _t flags);

cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids,
     struct cfga_list_data **ap_id_list, int *nlist, const char *options,
     const char *listops, char **errstring, cfga_flags_t flags);

int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
     ap_id2);

void config_unload_libs();

const char *config_strerror(cfga_err_t cfgerrnum);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct
     cfga_stat_data *buf, const char *options, char **errstring);

cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist,
     const char *options, char **errstring);
```

ハードウェアに依存するライブラリの形式

config_admin ライブラリは、動的再構成 (DR: Dynamic Reconfiguration) のための汎用的なインタフェースです。DR に対応している各ハードウェアは、ここに一覧されているエントリポイントを含むハードウェア固有の組み込みライブラリを提供する必要があります。この汎用ライブラリは、操作を実現するために、適切なライブラリを見つけ、接続します。ここにあるインタフェースは、汎用ライブラリのユーザーからは隠されています。ただし、ハードウェア固有の組み込みライブラリを作成するには、これらのインタフェースについて知っておく必要があります。

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char
     *ap_id, const char *options, struct cfga_confirm *confp, struct
     cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_private_func(const char *function, const char *ap_id,
     const char *options, struct cfga_confirm *confp, struct cfga_msg
```

```
    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data *
    *ap_id_list, int *nlist, const char *options, char **errstring);
```

機能説明

config_*() 関数は、ハードウェア固有のシステム構成管理機能に対して、ハードウェアに依存しないインタフェースを提供します。 cfga_*() 関数は、構成管理機能をハードウェア固有の方法で処理するために動的に読み込まれた、ハードウェア固有のライブラリによって提供されます。

libcfgadm ライブラリは、cfgadm(1M) コマンドのサービスを提供するために使用されます。libcfgadm ハードウェア固有のライブラリは /usr/platform/${machine}/lib/cfgadm、/usr/platform/${arch}/lib/cfgadm、/usr/lib/cfgadm にあります。ハードウェア固有のライブラリ名は、接続点を特定するデバイスツリーノードのドライバ名またはクラス名から派生します。

config_change_state() 関数は、システム構成の状態を変更する操作を行います。*state_change_cmd* 引数は、CFGA_CMD_INSERT、CFGA_CMD_REMOVE、CFGA_CMD_DISCONNECT、CFGA_CMD_CONNECT、CFGA_CMD_CONFIGURE、CFGA_CMD_UNCONFIGURE、のいずれかになります。 *state_change_cmd* CFGA_CMD_INSERT は、占有装置の手動による追加の準備をしたり、自動追加を有効にするために使用します。 *state_change_cmd* CFGA_CMD_REMOVE は、占有装置の手動による削除の準備をしたり、自動削除を有効にするために使用します。 *state_change_cmd* CFGA_CMD_DISCONNECT は、受容体に接続されている占有装置との通常の通信を無効にするために使用します。 *state_change_cmd* CFGA_CMD_CONNECTは、受容体に接続されている占有装置との通常の通信を有効にするために使用します。 *state_change_cmd* CFGA_CMD_CONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域に追加し、システムで使用できるようにします。*state_change_cmd* CFGA_CMD_UNCONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域から削除し、システムで使用できないようにします。

flags 引数には、定義済みのフラグである CFGA_FLAG_FORCE とCFGA_FLAG_VERBOSE のいずれかまたは両方を指定することができます。CFGA_FLAG_FORCE フラグを指定すると、特定の安全検査が無効になります。たとえば、このフラグは、failed 状態の占有装置が構成されることを許しませんが、failing 状態の占有装置が構成されることは許す場合があります。強制を受け付けるかどうかは、ハードウェアに依存します。CFGA_FLAG_VERBOSE フラグを指定すると、操作に関係するハードウェア固有の詳細情報が cfga_msg の方法で出力されます。

config_private_func() 関数は、専用ハードウェア固有の関数を起動します。

config_test() 関数は、特定の接続点の検査を開始するために使用します。

num_ap_ids 引数は、*ap_ids* 配列内の *ap_id* の数を特定します。 *ap_ids* 引数は、*ap_id* による配列を示します。

ap_id 引数は、単独の *ap_id* を示します。

function と *options* の文字列は、getsubopt(3C) 構文の規約に従い、ハードウェア固有の関数やオプション情報を提供するために使用します。ハードウェアに依存しない汎用的な関数やオプションは定義されていません。

confp によって参照される cfga_confirm 構造体は、要求された操作がサービスに対する顕著な割り込みを必要とする場合に、継続する許可を得るためのコールバックインタフェースを提供します。 cfga_confirm 構造体には以下のメンバーが含まれます。

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

confirm() 関数は、汎用ポインタの *appdata_ptr* と、確認を必要とするものの詳細を示すメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ *appdata_ptr* は、cfga_confirm 構造体のメンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で confirm 関数の呼び出しをconfig_* の呼び出しに関連付けるために使用されます。confirm 関数は、操作の継続が認められた場合は 1 を返し、認められなかった場合は 0 を返します。

msgp によって参照される cfga_msg 構造体は、ハードウェア固有のライブラリからのメッセージを出力するためのコールバックインタフェースを提供します。CFGA_FLAG_VERBOSE フラグが存在することによって、これらのメッセージは情報メッセージにもなりますが、このフラグが存在しない場合は、エラーメッセージに限定されます。cfga_msg 構造体には、以下のメンバーが含まれます。

```
void (*message_routine)(void *appdata_ptr, const char *message);
void    *appdata_ptr;
```

message_routine() 関数は、汎用ポインタの *appdata_ptr* とメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ*appdata_ptr* は、cfga_confirm 構造体メンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で message_routine() 関数の呼び出しを config_*() の呼び出しに関連付けるために使用されます。このメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

いくつかの一般的なエラーについて、ハードウェア固有のエラーメッセージが返されることがあります。エラーメッセージの文字列(終わりの NULL 文字を含む) の記憶領域は、malloc(3C) と、*errstring* を通じて返される、この記憶領域に対するポインタを使用して config_* 関数によって割り当てられます。*errstring* がNULL の場合は、エラーメッセージは作成または返されることはありません。*errstring* が NULL ではなく、エラーメッセージが作成されない場合は、*errstring* によって参照されるポインタが NULLに設定されます。free(3C) を使用して、返された記憶領域の割り当ての解除は、config_*() を呼び出している関数が行います。このエラーメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

config_list_ext() 関数は、リスト出力用インタフェースです。最初の 2 つの引数を使用して *ap_id*s にリストが指定されていると、指定されている接続点ごとに cfga_list_data_t 構造体の配列を返します。最初の 2 つの引数がそれぞれ 0 と NULL の場合は、デバイスツリーの接続点をすべて出力します。また、フラグの引数を使用して CFGA_FLAG_LIST_ALL フラグを渡した場合は、接続点を動的に展開して動的な接続点を出力する必要があります。返した配列の stat 構造の記憶領域は、config_list_ext() 関数がmalloc(3C) を使用して割り当てます。この記憶領域は、config_list_ext() の呼び出し元が free(3C) を使用して解放しなければなりません。

cfga_list_data 構造体には、以下のメンバーが含まれます。

```
cfga_log_ext_t     ap_log_id;        /* Attachment point logical id */
cfga_phys_ext_t    ap_phys_id;       /* Attachment point physical id */
cfga_class_t       ap_class;         /* Attachment point class */
cfga_stat_t        ap_r_state;       /* Receptacle state */
cfga_stat_t        ap_o_state;       /* Occupant state */
cfga_cond_t        ap_cond;          /* Attachment point condition */
cfga_busy_t        ap_busy;          /* Busy indicator */
time_t             ap_status_time;   /* Attachment point last change*/
cfga_info_t        ap_info;          /* Miscellaneous information */
cfga_type_t        ap_type;          /* Occupant type */
```

種類は以下のように定義されます。

```
typedef char cfga_log_ext_t[CFGA_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];
typedef char cfga_class_t[CFGA_CLASS_LEN];
typedef char cfga_info_t[CFGA_INFO_LEN];
typedef char cfga_type_t[CFGA_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;
```

config_list_ext() の *listopts* 引数は、getsubopt (3C) 構文に準拠しており、サブオプションのリストを渡すときに使用します。現在、サポートされているサブオプションは、*class*=class_name だけです。このリストオプションは、class_name クラスの接続点だけを出力します。

cfga_list_ext() の *listopts* 引数は、今後の使用のために予約されこの引数がNULL の場合は、ハードウェア固有のライブラリはこの引数を無視する必要がありま す。*listopts* 引数がNULL 以外のときに、ハードウェア固有のライブラリでサポートされていない場合は、適切なエラーコードを返す必要があります。

ap_log_id と ap_phys_id メンバーは、接続点に対するハードウェア固有の論理名と物理名を示します。ap_busy メンバーは、状態 (state) や条件 (condition) の変更が生じるような動作があることを示します。ap_status_time メンバーは、接続点のap_r_state、ap_o_state、ap_cond フィールドのいずれかが最後に変わった時間を示します。ap_info メンバーは、接続点に関する追加情報を得るためにハードウェア固有のコードで使用することができます。ap_class メンバーには、接続点の接続点クラスが含まれます (存在する場合)。ap_class メンバーは、汎用ライブラリから組み込まれます。ap_log_id および ap_phys_id メンバーがハードウェア固有のライブラリから組み込まれなかった場合は、汎用ライブラリから汎用形式で組み込まれます。残りのメンバーは、対応するハードウェア固有のライブラリから組み込まれます。

ap_log_id、ap_phys_id、ap_info、ap_class、および ap_type メンバーは、固定長文字列です。実際の文字列がメンバーのサイズよりも短い場合には、この文字列は空文字で終了します。そのため、プログラムは、文字列が空文字で終了していることを前提としてはなりません。これらのフィールドを印刷する場合は、以下の形式を使用してください。

printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);

config_stat()、config_list()、cfga_stat()、cfga_list() 関数およびcfga_stat_data データ構造体は、推奨されていないインタフェースで、下位互換性を維持するためにだけ提供されています。これらのインタフェースはできるだけ使用しないでください。

config_ap_id_cmp 関数は、2 つの *ap_id* に関する、ハードウェアに依存する比較を行い、strcmp(3C) の形式に従って、等しい、小さい、大きい、の結果を返します。各引数 は、cfga_ap_id_t とNULL で終了する文字列のいずれかになります。この関数は、qsort(3C) などによって *ap_id* の一覧をソートする場合や、config_list 関数の呼び出しの結果からエントリを選択する場合に使用することができます。

config_unload_libs 関数は、これまで読み込んだハードウェア固有のライブラリすべてのリンクを解除します。

config_strerror 関数を使用して、エラーメッセージの文字列に対してエラーの戻り値を割り当てることができます。「戻り値」を参照してください。返された文字列は、上書きされてはなりません。 *cfgerrnum* が範囲外にある場合に、config _strerror は NULL を返します。

cfga_help 関数を使用して、ハードウェア固有のライブラリに対して現地仕様のヘルプメッセージを表示するように要求することができます。

戻り値

以下は、config_*() および cfga_*() 関数によって返される戻り値の一覧です。戻り値が CFGA_OK ではない場合は、追加のエラー情報が *errstring* を通して返されます。詳細は、「機能説明」を参照してください。

CFGA_BUSY	システム構成管理システムの 1 要素が使用中であるために、コマンドは完了しませんでした。
CFGA_ATTR_INVAL	指定された属性の接続点は存在しません。
CFGA_ERROR	要求された操作の処理中にエラーが発生しました。このエラーコードは、ハードウェア固有のコードによるコマンド引数の妥当性検査を含みます。
CFGA_INSUFFICIENT_CONDITION	接続点の条件によって、操作が失敗しました。
CFGA_INVAL	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_LIB_ERROR	記憶域やファイル記述子などの処理用資源の獲得の失敗など、ライブラリで手続き上のエラーが発生しました。
CFGA_NACK	*confp*->confirm 関数からの否定応答を受けたために、コマンドは完了しませんでした。
CFGA_NO_LIB	指定された *ap_id* を使用して、ハードウェア固有のライブラリが見つかりませんでした。
CFGA_NOTSUPP	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_OK	要求通りにコマンドが完了しました。
CFGA_OPNOTSUPP	この接続点は、システム構成管理操作に対応していません。
CFGA_PRIV	呼び出し元は要求された処理の特権を所有していません。たとえば、構成管理がデバイスドライバを介して実行される場合は、デバイスノードのアクセス権によってアクセスが制御されます。
CFGA_SYSTEM_BUSY	コマンドがサービスの中断を要求したが、システムの一部が休止できなかったために完了しませんでした。

エラー

システム構成管理機能から返されるエラーの多くは、ハードウェア固有のものです。*errstring* を通して返される文字列には以下があります。

`attachment point` *ap_id* `not known`
エラーメッセージで説明されている接続点は存在しません。

`unknown hardware option` *option* `for` *operation*
未知のオプションが *options* 文字列の中で検出されました。

`hardware option` *option*option `requires a value`
options 文字列で指定されるオプションは、*option=value* の形式である必要があります。

`listing option` *list_option* `requires a value`
listopts 文字列で指定されるオプションは、*option*=`value` 形式である必要があります。

`hardware option` *option* `does not require a value`
options 文字列で指定されるオプションは、単純なオプションである必要があります。

`attachment point` *ap_id* `is not configured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、`CFGA_CMD _UNCONFIGURE` を行う *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `is not unconfigured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、未構成の占有装置を必要とする *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `condition not satisfactory.`
操作の妨げとなる条件を持つ接続点に対して、*config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `in condition` *condition* `cannot be used`
ハードウェアに依存する検査に通らない状態の接続点に対して、強制指定された *config_change_state* 操作が適用されました。

属性

以下の属性については、`attributes`(5) を参照してください。

属性の種類	属性の値
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

関連項目

`cfgadm`(1M), `devinfo`(1M), `dlopen`(3DL), `dlsym`(3DL), `free`(3C), `getsubopt` (3C) , `malloc`(3C), `qsort`(3C), `setlocale`(3C), `strcmp`(3C), `libcfgadm`(3LIB), `attributes`(5)

注意事項

このライブラリを使用するアプリケーションは、実際の実装環境が、外部の `errno` 変数の内容を変更するシステムサービスやファイル記述リソースを使用する可能性があることを考慮したものである必要があります。

以下のコードは、config_*() が CFGA_OK 以外の値を返した場合のエラー処理です。

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

システム構成管理で使用することができる機能に関する詳細は、ハードウェア固有のマニュアルを参照してください。

名前

config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – 構成の管理

形式

```
cc [ flag ] file -lcfgadm  [ library... ]
#include <config_admin.h>

cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int
     num_ap_ids, char * const *ap_ids, const char *options, struct cfga
     _confirm *confp, struct cfga_msg *msgp, char **errstring, cfga
     _flags_t flags);

cfga_err_t config_private_func(const char *function, int num_ap_ids,
     char * const *ap_ids, const char *options, struct cfga_confirm
     *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags)
     ;

cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const
     char *options, struct cfga_msg *msgp, char **errstring, cfga_flags
     _t flags);

cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids,
     struct cfga_list_data **ap_id_list, int *nlist, const char *options,
     const char *listops, char **errstring, cfga_flags_t flags);

int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
     ap_id2);

void config_unload_libs();

const char *config_strerror(cfga_err_t cfgerrnum);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct
     cfga_stat_data *buf, const char *options, char **errstring);

cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist,
     const char *options, char **errstring);
```

ハードウェアに依存するライブラリの形式

config_admin ライブラリは、動的再構成 (DR: Dynamic Reconfiguration) のための汎用的なインタフェースです。DR に対応している各ハードウェアは、ここに一覧されているエントリポイントを含むハードウェア固有の組み込みライブラリを提供する必要があります。この汎用ライブラリは、操作を実現するために、適切なライブラリを見つけ、接続します。ここにあるインタフェースは、汎用ライブラリのユーザーからは隠されています。ただし、ハードウェア固有の組み込みライブラリを作成するには、これらのインタフェースについて知っておく必要があります。

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char
     *ap_id, const char *options, struct cfga_confirm *confp, struct
     cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_private_func(const char *function, const char *ap_id,
     const char *options, struct cfga_confirm *confp, struct cfga_msg
```

```
    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data *
    *ap_id_list, int *nlist, const char *options, char **errstring);
```

機能説明

config_*() 関数は、ハードウェア固有のシステム構成管理機能に対して、ハードウェアに依存しないインタフェースを提供します。 cfga_*() 関数は、構成管理機能をハードウェア固有の方法で処理するために動的に読み込まれた、ハードウェア固有のライブラリによって提供されます。

libcfgadm ライブラリは、cfgadm(1M) コマンドのサービスを提供するために使用されます。libcfgadm ハードウェア固有のライブラリは /usr/platform/${machine}/lib/cfgadm、/usr/platform/${arch}/lib/cfgadm、/usr/lib/cfgadm にあります。ハードウェア固有のライブラリ名は、接続点を特定するデバイスツリーノードのドライバ名またはクラス名から派生します。

config_change_state() 関数は、システム構成の状態を変更する操作を行います。*state_change_cmd* 引数は、CFGA_CMD_INSERT、CFGA_CMD_REMOVE、CFGA_CMD_DISCONNECT、CFGA_CMD_CONNECT、CFGA_CMD_CONFIGURE、CFGA_CMD_UNCONFIGURE、のいずれかになります。 *state_change_cmd* CFGA_CMD_INSERT は、占有装置の手動による追加の準備をしたり、自動追加を有効にするために使用します。 *state_change_cmd* CFGA_CMD_REMOVE は、占有装置の手動による削除の準備をしたり、自動削除を有効にするために使用します。 *state_change_cmd* CFGA_CMD_DISCONNECT は、受容体に接続されている占有装置との通常の通信を無効にするために使用します。 *state_change_cmd* CFGA_CMD_CONNECTは、受容体に接続されている占有装置との通常の通信を有効にするために使用します。 *state_change_cmd* CFGA_CMD_CONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域に追加し、システムで使用できるようにします。*state_change_cmd* CFGA_CMD_UNCONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域から削除し、システムで使用できないようにします。

flags 引数には、定義済みのフラグである CFGA_FLAG_FORCE とCFGA_FLAG_VERBOSE のいずれかまたは両方を指定することができます。CFGA_FLAG_FORCE フラグを指定すると、特定の安全検査が無効になります。たとえば、このフラグは、failed 状態の占有装置が構成されることを許しませんが、failing 状態の占有装置が構成されることは許す場合があります。強制を受け付けるかどうかは、ハードウェアに依存します。CFGA_FLAG_VERBOSE フラグを指定すると、操作に関係するハードウェア固有の詳細情報が cfga_msg の方法で出力されます。

config_private_func() 関数は、専用ハードウェア固有の関数を起動します。

config_test() 関数は、特定の接続点の検査を開始するために使用します。

num_ap_ids 引数は、*ap_ids* 配列内の *ap_id* の数を特定します。 *ap_ids* 引数は、*ap_id* による配列を示します。

ap_id 引数は、単独の *ap_id* を示します。

function と *options* の文字列は、getsubopt(3C) 構文の規約に従い、ハードウェア固有の関数やオプション情報を提供するために使用します。ハードウェアに依存しない汎用的な関数やオプションは定義されていません。

confp によって参照される cfga_confirm 構造体は、要求された操作がサービスに対する顕著な割り込みを必要とする場合に、継続する許可を得るためのコールバックインタフェースを提供します。 cfga_confirm 構造体には以下のメンバーが含まれます。

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

confirm() 関数は、汎用ポインタの *appdata_ptr* と、確認を必要とするものの詳細を示すメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ *appdata_ptr* は、cfga_confirm 構造体のメンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で confirm 関数の呼び出しをconfig_* の呼び出しに関連付けるために使用されます。confirm 関数は、操作の継続が認められた場合は 1 を返し、認められなかった場合は 0 を返します。

msgp によって参照される cfga_msg 構造体は、ハードウェア固有のライブラリからのメッセージを出力するためのコールバックインタフェースを提供します。CFGA_FLAG_VERBOSE フラグが存在することによって、これらのメッセージは情報メッセージにもなりますが、このフラグが存在しない場合は、エラーメッセージに限定されます。cfga_msg 構造体には、以下のメンバーが含まれます。

```
void (*message_routine)(void *appdata_ptr, const char *message);
void    *appdata_ptr;
```

message_routine() 関数は、汎用ポインタの *appdata_ptr* とメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ*appdata_ptr* は、cfga_confirm 構造体メンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で message_routine() 関数の呼び出しを config_*() の呼び出しに関連付けるために使用されます。このメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

いくつかの一般的なエラーについて、ハードウェア固有のエラーメッセージが返されることがあります。エラーメッセージの文字列(終わりの NULL 文字を含む) の記憶領域は、malloc(3C) と、*errstring* を通じて返される、この記憶領域に対するポインタを使用して config_* 関数によって割り当てられます。*errstring* がNULL の場合は、エラーメッセージは作成または返されることはありません。*errstring* が NULL ではなく、エラーメッセージが作成されない場合は、*errstring* によって参照されるポインタが NULLに設定されます。free(3C) を使用して、返された記憶領域の割り当ての解除は、config_*() を呼び出している関数が行います。このエラーメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

config_list_ext() 関数は、リスト出力用インタフェースです。最初の 2 つの引数を使用して *ap_id*s にリストが指定されていると、指定されている接続点ごとに cfga_list_data_t 構造体の配列を返します。最初の 2 つの引数がそれぞれ 0 と NULL の場合は、デバイスツリーの接続点をすべて出力します。また、フラグの引数を使用して CFGA_FLAG_LIST_ALL フラグを渡した場合は、接続点を動的に展開して動的な接続点を出力する必要があります。返した配列の stat 構造の記憶領域は、config_list_ext() 関数がmalloc(3C) を使用して割り当てます。この記憶領域は、config_list_ext() の呼び出し元が free(3C) を使用して解放しなければなりません。

cfga_list_data 構造体には、以下のメンバーが含まれます。

```
cfga_log_ext_t     ap_log_id;        /* Attachment point logical id */
cfga_phys_ext_t    ap_phys_id;       /* Attachment point physical id */
cfga_class_t       ap_class;         /* Attachment point class */
cfga_stat_t        ap_r_state;       /* Receptacle state */
cfga_stat_t        ap_o_state;       /* Occupant state */
cfga_cond_t        ap_cond;          /* Attachment point condition */
cfga_busy_t        ap_busy;          /* Busy indicator */
time_t             ap_status_time;   /* Attachment point last change*/
cfga_info_t        ap_info;          /* Miscellaneous information */
cfga_type_t        ap_type;          /* Occupant type */
```

種類は以下のように定義されます。

```
typedef char cfga_log_ext_t[CFGA_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];
typedef char cfga_class_t[CFGA_CLASS_LEN];
typedef char cfga_info_t[CFGA_INFO_LEN];
typedef char cfga_type_t[CFGA_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;
```

config_list_ext() の *listopts* 引数は、getsubopt (3C) 構文に準拠しており、サブオプションのリストを渡すときに使用します。現在、サポートされているサブオプションは、*class*=class_name だけです。このリストオプションは、class_name クラスの接続点だけを出力します。

cfga_list_ext() の *listopts* 引数は、今後の使用のために予約されこの引数がNULL の場合は、ハードウェア固有のライブラリはこの引数を無視する必要がありま す。*listopts* 引数がNULL 以外のときに、ハードウェア固有のライブラリでサポートされていない場合は、適切なエラーコードを返す必要があります。

ap_log_id と ap_phys_id メンバーは、接続点に対するハードウェア固有の論理名と物理名を示します。ap_busy メンバーは、状態 (state) や条件 (condition) の変更が生じるような動作があることを示します。ap_status_time メンバーは、接続点のap_r_state、ap_o_state、ap_cond フィールドのいずれかが最後に変わった時間を示します。ap_info メンバーは、接続点に関する追加情報を得るためにハードウェア固有のコードで使用することができます。ap_class メンバーには、接続点の接続点クラスが含まれます (存在する場合)。ap_class メンバーは、汎用ライブラリから組み込まれます。ap_log_id および ap_phys_id メンバーがハードウェア固有のライブラリから組み込まれなかった場合は、汎用ライブラリから汎用形式で組み込まれます。残りのメンバーは、対応するハードウェア固有のライブラリから組み込まれます。

ap_log_id、ap_phys_id、ap_info、ap_class、および ap_type メンバーは、固定長文字列です。実際の文字列がメンバーのサイズよりも短い場合には、この文字列は空文字で終了します。そのため、プログラムは、文字列が空文字で終了していることを前提としてはなりません。これらのフィールドを印刷する場合は、以下の形式を使用してください。

printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);

config_stat()、config_list()、cfga_stat()、cfga_list() 関数およびcfga_stat_data データ構造体は、推奨されていないインタフェースで、下位互換性を維持するためにだけ提供されています。これらのインタフェースはできるだけ使用しないでください。

config_ap_id_cmp 関数は、2 つの *ap_id* に関する、ハードウェアに依存する比較を行い、strcmp(3C) の形式に従って、等しい、小さい、大きい、の結果を返します。各引数 は、cfga_ap_id_t とNULL で終了する文字列のいずれかになります。この関数は、qsort(3C) などによって *ap_id* の一覧をソートする場合や、config_list 関数の呼び出しの結果からエントリを選択する場合に使用することができます。

config_unload_libs 関数は、これまで読み込んだハードウェア固有のライブラリすべてのリンクを解除します。

config_strerror 関数を使用して、エラーメッセージの文字列に対してエラーの戻り値を割り当てることができます。「戻り値」を参照してください。返された文字列は、上書きされてはなりません。 *cfgerrnum* が範囲外にある場合に、config _strerror は NULL を返します。

cfga_help 関数を使用して、ハードウェア固有のライブラリに対して現地仕様のヘルプメッセージを表示するように要求することができます。

戻り値

以下は、config_*() および cfga_*() 関数によって返される戻り値の一覧です。戻り値が CFGA_OK ではない場合は、追加のエラー情報が *errstring* を通して返されます。詳細は、「機能説明」を参照してください。

CFGA_BUSY	システム構成管理システムの 1 要素が使用中であるために、コマンドは完了しませんでした。
CFGA_ATTR_INVAL	指定された属性の接続点は存在しません。
CFGA_ERROR	要求された操作の処理中にエラーが発生しました。このエラーコードは、ハードウェア固有のコードによるコマンド引数の妥当性検査を含みます。
CFGA_INSUFFICIENT_CONDITION	接続点の条件によって、操作が失敗しました。
CFGA_INVAL	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_LIB_ERROR	記憶域やファイル記述子などの処理用資源の獲得の失敗など、ライブラリで手続き上のエラーが発生しました。
CFGA_NACK	*confp*->confirm 関数からの否定応答を受けたために、コマンドは完了しませんでした。
CFGA_NO_LIB	指定された *ap_id* を使用して、ハードウェア固有のライブラリが見つかりませんでした。
CFGA_NOTSUPP	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_OK	要求通りにコマンドが完了しました。
CFGA_OPNOTSUPP	この接続点は、システム構成管理操作に対応していません。
CFGA_PRIV	呼び出し元は要求された処理の特権を所有していません。たとえば、構成管理がデバイスドライバを介して実行される場合は、デバイスノードのアクセス権によってアクセスが制御されます。
CFGA_SYSTEM_BUSY	コマンドがサービスの中断を要求したが、システムの一部が休止できなかったために完了しませんでした。

エラー

システム構成管理機能から返されるエラーの多くは、ハードウェア固有のものです。*errstring* を通して返される文字列には以下があります。

`attachment point` *ap_id* `not known`
エラーメッセージで説明されている接続点は存在しません。

`unknown hardware option` *option* `for` *operation*
未知のオプションが *options* 文字列の中で検出されました。

`hardware option` *option*option `requires a value`
options 文字列で指定されるオプションは、*option=value* の形式である必要があります。

`listing option` *list_option* `requires a value`
listopts 文字列で指定されるオプションは、*option*=`value` 形式である必要があります。

`hardware option` *option* `does not require a value`
options 文字列で指定されるオプションは、単純なオプションである必要があります。

`attachment point` *ap_id* `is not configured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、`CFGA_CMD_UNCONFIGURE` を行う *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `is not unconfigured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、未構成の占有装置を必要とする *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `condition not satisfactory.`
操作の妨げとなる条件を持つ接続点に対して、*config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `in condition` *condition* `cannot be used`
ハードウェアに依存する検査に通らない状態の接続点に対して、強制指定された *config_change_state* 操作が適用されました。

属性

以下の属性については、`attributes`(5) を参照してください。

属性の種類	属性の値
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

関連項目

`cfgadm`(1M), `devinfo`(1M), `dlopen`(3DL), `dlsym`(3DL), `free`(3C), `getsubopt` (3C), `malloc`(3C), `qsort`(3C), `setlocale`(3C), `strcmp`(3C), `libcfgadm`(3LIB), `attributes`(5)

注意事項

このライブラリを使用するアプリケーションは、実際の実装環境が、外部の `errno` 変数の内容を変更するシステムサービスやファイル記述リソースを使用する可能性があることを考慮したものである必要があります。

以下のコードは、config_*() が CFGA_OK 以外の値を返した場合のエラー処理です。

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

システム構成管理で使用することができる機能に関する詳細は、ハードウェア固有のマニュアルを参照してください。

名前

config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – 構成の管理

形式

```
cc [ flag ] file -lcfgadm  [ library... ]
#include <config_admin.h>

cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int
     num_ap_ids, char * const *ap_ids, const char *options, struct cfga
     _confirm *confp, struct cfga_msg *msgp, char **errstring, cfga
     _flags_t flags);

cfga_err_t config_private_func(const char *function, int num_ap_ids,
     char * const *ap_ids, const char *options, struct cfga_confirm
     *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags)
     ;

cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const
     char *options, struct cfga_msg *msgp, char **errstring, cfga_flags
     _t flags);

cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids,
     struct cfga_list_data **ap_id_list, int *nlist, const char *options,
     const char *listops, char **errstring, cfga_flags_t flags);

int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
     ap_id2);

void config_unload_libs();

const char *config_strerror(cfga_err_t cfgerrnum);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct
     cfga_stat_data *buf, const char *options, char **errstring);

cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist,
     const char *options, char **errstring);
```

ハードウェアに依存するライブラリの形式

config_admin ライブラリは、動的再構成 (DR: Dynamic Reconfiguration) のための汎用的なインタフェースです。DR に対応している各ハードウェアは、ここに一覧されているエントリポイントを含むハードウェア固有の組み込みライブラリを提供する必要があります。この汎用ライブラリは、操作を実現するために、適切なライブラリを見つけ、接続します。ここにあるインタフェースは、汎用ライブラリのユーザーからは隠されています。ただし、ハードウェア固有の組み込みライブラリを作成するには、これらのインタフェースについて知っておく必要があります。

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char
     *ap_id, const char *options, struct cfga_confirm *confp, struct
     cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_private_func(const char *function, const char *ap_id,
     const char *options, struct cfga_confirm *confp, struct cfga_msg
```

```
    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data *
    *ap_id_list, int *nlist, const char *options, char **errstring);
```

機能説明

config_*() 関数は、ハードウェア固有のシステム構成管理機能に対して、ハードウェアに依存しないインタフェースを提供します。 cfga_*() 関数は、構成管理機能をハードウェア固有の方法で処理するために動的に読み込まれた、ハードウェア固有のライブラリによって提供されます。

libcfgadm ライブラリは、cfgadm(1M) コマンドのサービスを提供するために使用されます。libcfgadm ハードウェア固有のライブラリは /usr/platform/${machine}/lib/cfgadm、/usr/platform/${arch}/lib/cfgadm、/usr/lib/cfgadm にあります。ハードウェア固有のライブラリ名は、接続点を特定するデバイスツリーノードのドライバ名またはクラス名から派生します。

config_change_state() 関数は、システム構成の状態を変更する操作を行います。*state_change_cmd* 引数は、CFGA_CMD_INSERT、CFGA_CMD_REMOVE、CFGA_CMD_DISCONNECT、CFGA_CMD_CONNECT、CFGA_CMD_CONFIGURE、CFGA_CMD_UNCONFIGURE、のいずれかになります。 *state_change_cmd* CFGA_CMD_INSERT は、占有装置の手動による追加の準備をしたり、自動追加を有効にするために使用します。 *state_change_cmd* CFGA_CMD_REMOVE は、占有装置の手動による削除の準備をしたり、自動削除を有効にするために使用します。 *state_change_cmd* CFGA_CMD_DISCONNECT は、受容体に接続されている占有装置との通常の通信を無効にするために使用します。 *state_change_cmd* CFGA_CMD_CONNECTは、受容体に接続されている占有装置との通常の通信を有効にするために使用します。 *state_change_cmd* CFGA_CMD_CONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域に追加し、システムで使用できるようにします。*state_change_cmd* CFGA_CMD_UNCONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域から削除し、システムで使用できないようにします。

flags 引数には、定義済みのフラグである CFGA_FLAG_FORCE とCFGA_FLAG_VERBOSE のいずれかまたは両方を指定することができます。CFGA_FLAG_FORCE フラグを指定すると、特定の安全検査が無効になります。たとえば、このフラグは、failed 状態の占有装置が構成されることを許しませんが、failing 状態の占有装置が構成されることは許す場合があります。強制を受け付けるかどうかは、ハードウェアに依存します。CFGA_FLAG_VERBOSE フラグを指定すると、操作に関係するハードウェア固有の詳細情報が cfga_msg の方法で出力されます。

config_private_func() 関数は、専用ハードウェア固有の関数を起動します。

config_test() 関数は、特定の接続点の検査を開始するために使用します。

num_ap_ids 引数は、*ap_ids* 配列内の *ap_id* の数を特定します。 *ap_ids* 引数は、*ap_id* による配列を示します。

ap_id 引数は、単独の *ap_id* を示します。

function と *options* の文字列は、getsubopt(3C) 構文の規約に従い、ハードウェア固有の関数やオプション情報を提供するために使用します。ハードウェアに依存しない汎用的な関数やオプションは定義されていません。

confp によって参照される cfga_confirm 構造体は、要求された操作がサービスに対する顕著な割り込みを必要とする場合に、継続する許可を得るためのコールバックインタフェースを提供します。 cfga_confirm 構造体には以下のメンバーが含まれます。

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

confirm() 関数は、汎用ポインタの *appdata_ptr* と、確認を必要とするものの詳細を示すメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ *appdata_ptr* は、cfga_confirm 構造体のメンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で confirm 関数の呼び出しをconfig_* の呼び出しに関連付けるために使用されます。confirm 関数は、操作の継続が認められた場合は 1 を返し、認められなかった場合は 0 を返します。

msgp によって参照される cfga_msg 構造体は、ハードウェア固有のライブラリからのメッセージを出力するためのコールバックインタフェースを提供します。CFGA_FLAG_VERBOSE フラグが存在することによって、これらのメッセージは情報メッセージにもなりますが、このフラグが存在しない場合は、エラーメッセージに限定されます。cfga_msg 構造体には、以下のメンバーが含まれます。

```
void (*message_routine)(void *appdata_ptr, const char *message);
void    *appdata_ptr;
```

message_routine() 関数は、汎用ポインタの *appdata_ptr* とメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ*appdata_ptr* は、cfga_confirm 構造体メンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で message_routine() 関数の呼び出しを config_*() の呼び出しに関連付けるために使用されます。このメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

いくつかの一般的なエラーについて、ハードウェア固有のエラーメッセージが返されることがあります。エラーメッセージの文字列(終わりの NULL 文字を含む) の記憶領域は、malloc(3C) と、*errstring* を通じて返される、この記憶領域に対するポインタを使用して config_* 関数によって割り当てられます。*errstring* がNULL の場合は、エラーメッセージは作成または返されることはありません。*errstring* が NULL ではなく、エラーメッセージが作成されない場合は、*errstring* によって参照されるポインタが NULLに設定されます。free(3C) を使用して、返された記憶領域の割り当ての解除は、config_*() を呼び出している関数が行います。このエラーメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

config_list_ext() 関数は、リスト出力用インタフェースです。最初の 2 つの引数を使用して *ap_id*s にリストが指定されていると、指定されている接続点ごとに cfga_list_data_t 構造体の配列を返します。最初の 2 つの引数がそれぞれ 0 と NULL の場合は、デバイスツリーの接続点をすべて出力します。また、フラグの引数を使用して CFGA_FLAG_LIST_ALL フラグを渡した場合は、接続点を動的に展開して動的な接続点を出力する必要があります。返した配列の stat 構造の記憶領域は、config_list_ext() 関数がmalloc(3C) を使用して割り当てます。この記憶領域は、config_list_ext() の呼び出し元が free(3C) を使用して解放しなければなりません。

cfga_list_data 構造体には、以下のメンバーが含まれます。

```
cfga_log_ext_t     ap_log_id;        /* Attachment point logical id */
cfga_phys_ext_t    ap_phys_id;       /* Attachment point physical id */
cfga_class_t       ap_class;         /* Attachment point class */
cfga_stat_t        ap_r_state;       /* Receptacle state */
cfga_stat_t        ap_o_state;       /* Occupant state */
cfga_cond_t        ap_cond;          /* Attachment point condition */
cfga_busy_t        ap_busy;          /* Busy indicator */
time_t             ap_status_time;   /* Attachment point last change*/
cfga_info_t        ap_info;          /* Miscellaneous information */
cfga_type_t        ap_type;          /* Occupant type */
```

種類は以下のように定義されます。

```
typedef char cfga_log_ext_t[CFGA_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];
typedef char cfga_class_t[CFGA_CLASS_LEN];
typedef char cfga_info_t[CFGA_INFO_LEN];
typedef char cfga_type_t[CFGA_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;
```

config_list_ext() の *listopts* 引数は、getsubopt (3C) 構文に準拠しており、サブオプションのリストを渡すときに使用します。現在、サポートされているサブオプションは、*class*=class_name だけです。このリストオプションは、class_name クラスの接続点だけを出力します。

cfga_list_ext() の *listopts* 引数は、今後の使用のために予約されこの引数がNULL の場合は、ハードウェア固有のライブラリはこの引数を無視する必要がありま す。*listopts* 引数がNULL 以外のときに、ハードウェア固有のライブラリでサポートされていない場合は、適切なエラーコードを返す必要があります。

ap_log_id と ap_phys_id メンバーは、接続点に対するハードウェア固有の論理名と物理名を示します。ap_busy メンバーは、状態 (state) や条件 (condition) の変更が生じるような動作があることを示します。ap_status_time メンバーは、接続点のap_r_state、ap_o_state、ap_cond フィールドのいずれかが最後に変わった時間を示します。ap_info メンバーは、接続点に関する追加情報を得るためにハードウェア固有のコードで使用することができます。ap_class メンバーには、接続点の接続点クラスが含まれます (存在する場合)。ap_class メンバーは、汎用ライブラリから組み込まれます。ap_log_id および ap_phys_id メンバーがハードウェア固有のライブラリから組み込まれなかった場合は、汎用ライブラリから汎用形式で組み込まれます。残りのメンバーは、対応するハードウェア固有のライブラリから組み込まれます。

ap_log_id、ap_phys_id、ap_info、ap_class、および ap_type メンバーは、固定長文字列です。実際の文字列がメンバーのサイズよりも短い場合には、この文字列は空文字で終了します。そのため、プログラムは、文字列が空文字で終了していることを前提としてはなりません。これらのフィールドを印刷する場合は、以下の形式を使用してください。

printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);

config_stat()、config_list()、cfga_stat()、cfga_list() 関数およびcfga_stat_data データ構造体は、推奨されていないインタフェースで、下位互換性を維持するためにだけ提供されています。これらのインタフェースはできるだけ使用しないでください。

config_ap_id_cmp 関数は、2 つの *ap_id* に関する、ハードウェアに依存する比較を行い、strcmp(3C) の形式に従って、等しい、小さい、大きい、の結果を返します。各引数 は、cfga_ap_id_t とNULL で終了する文字列のいずれかになります。この関数は、qsort(3C) などによって *ap_id* の一覧をソートする場合や、config_list 関数の呼び出しの結果からエントリを選択する場合に使用することができます。

config_unload_libs 関数は、これまで読み込んだハードウェア固有のライブラリすべてのリンクを解除します。

config_strerror 関数を使用して、エラーメッセージの文字列に対してエラーの戻り値を割り当てることができます。「戻り値」を参照してください。返された文字列は、上書きされてはなりません。 *cfgerrnum* が範囲外にある場合に、config _strerror は NULL を返します。

cfga_help 関数を使用して、ハードウェア固有のライブラリに対して現地仕様のヘルプメッセージを表示するように要求することができます。

戻り値

以下は、config_*() および cfga_*() 関数によって返される戻り値の一覧です。戻り値が CFGA_OK ではない場合は、追加のエラー情報が *errstring* を通して返されます。詳細は、「機能説明」を参照してください。

CFGA_BUSY	システム構成管理システムの 1 要素が使用中であるために、コマンドは完了しませんでした。
CFGA_ATTR_INVAL	指定された属性の接続点は存在しません。
CFGA_ERROR	要求された操作の処理中にエラーが発生しました。このエラーコードは、ハードウェア固有のコードによるコマンド引数の妥当性検査を含みます。
CFGA_INSUFFICIENT_CONDITION	接続点の条件によって、操作が失敗しました。
CFGA_INVAL	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_LIB_ERROR	記憶域やファイル記述子などの処理用資源の獲得の失敗など、ライブラリで手続き上のエラーが発生しました。
CFGA_NACK	*confp*->confirm 関数からの否定応答を受けたために、コマンドは完了しませんでした。
CFGA_NO_LIB	指定された *ap_id* を使用して、ハードウェア固有のライブラリが見つかりませんでした。
CFGA_NOTSUPP	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_OK	要求通りにコマンドが完了しました。
CFGA_OPNOTSUPP	この接続点は、システム構成管理操作に対応していません。
CFGA_PRIV	呼び出し元は要求された処理の特権を所有していません。たとえば、構成管理がデバイスドライバを介して実行される場合は、デバイスノードのアクセス権によってアクセスが制御されます。
CFGA_SYSTEM_BUSY	コマンドがサービスの中断を要求したが、システムの一部が休止できなかったために完了しませんでした。

エラー | システム構成管理機能から返されるエラーの多くは、ハードウェア固有のものです。*errstring* を通して返される文字列には以下があります。

`attachment point` *ap_id* `not known`
エラーメッセージで説明されている接続点は存在しません。

`unknown hardware option` *option* `for` *operation*
未知のオプションが *options* 文字列の中で検出されました。

`hardware option` *option*option `requires a value`
options 文字列で指定されるオプションは、*option=value* の形式である必要があります。

`listing option` *list_option* `requires a value`
listopts 文字列で指定されるオプションは、*option*=`value` 形式である必要があります。

`hardware option` *option* `does not require a value`
options 文字列で指定されるオプションは、単純なオプションである必要があります。

`attachment point` *ap_id* `is not configured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、`CFGA_CMD _UNCONFIGURE` を行う *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `is not unconfigured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、未構成の占有装置を必要とする *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `condition not satisfactory.`
操作の妨げとなる条件を持つ接続点に対して、*config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `in condition` *condition* `cannot be used`
ハードウェアに依存する検査に通らない状態の接続点に対して、強制指定された *config_change_state* 操作が適用されました。

属性 | 以下の属性については、`attributes`(5) を参照してください。

属性の種類	属性の値
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

関連項目 | `cfgadm`(1M), `devinfo`(1M), `dlopen`(3DL), `dlsym`(3DL), `free`(3C), `getsubopt` (3C), `malloc`(3C), `qsort`(3C), `setlocale`(3C), `strcmp`(3C), `libcfgadm`(3LIB), `attributes`(5)

注意事項 | このライブラリを使用するアプリケーションは、実際の実装環境が、外部の `errno` 変数の内容を変更するシステムサービスやファイル記述リソースを使用する可能性があることを考慮したものである必要があります。

以下のコードは、config_*() が CFGA_OK 以外の値を返した場合のエラー処理です。

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

システム構成管理で使用することができる機能に関する詳細は、ハードウェア固有のマニュアルを参照してください。

名前

config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – 構成の管理

形式

```
cc [ flag ] file -lcfgadm  [ library... ]
#include <config_admin.h>

cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int
     num_ap_ids, char * const *ap_ids, const char *options, struct cfga
     _confirm *confp, struct cfga_msg *msgp, char **errstring, cfga
     _flags_t flags);

cfga_err_t config_private_func(const char *function, int num_ap_ids,
     char * const *ap_ids, const char *options, struct cfga_confirm
     *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags)
     ;

cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const
     char *options, struct cfga_msg *msgp, char **errstring, cfga_flags
     _t flags);

cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids,
     struct cfga_list_data **ap_id_list, int *nlist, const char *options,
     const char *listops, char **errstring, cfga_flags_t flags);

int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
     ap_id2);

void config_unload_libs();

const char *config_strerror(cfga_err_t cfgerrnum);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct
     cfga_stat_data *buf, const char *options, char **errstring);

cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist,
     const char *options, char **errstring);
```

ハードウェアに依存するライブラリの形式

config_admin ライブラリは、動的再構成 (DR: Dynamic Reconfiguration) のための汎用的なインタフェースです。DR に対応している各ハードウェアは、ここに一覧されているエントリポイントを含むハードウェア固有の組み込みライブラリを提供する必要があります。この汎用ライブラリは、操作を実現するために、適切なライブラリを見つけ、接続します。ここにあるインタフェースは、汎用ライブラリのユーザーからは隠されています。ただし、ハードウェア固有の組み込みライブラリを作成するには、これらのインタフェースについて知っておく必要があります。

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char
     *ap_id, const char *options, struct cfga_confirm *confp, struct
     cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_private_func(const char *function, const char *ap_id,
     const char *options, struct cfga_confirm *confp, struct cfga_msg
```

```
    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data *
    *ap_id_list, int *nlist, const char *options, char **errstring);
```

機能説明

config_*() 関数は、ハードウェア固有のシステム構成管理機能に対して、ハードウェアに依存しないインタフェースを提供します。 cfga_*() 関数は、構成管理機能をハードウェア固有の方法で処理するために動的に読み込まれた、ハードウェア固有のライブラリによって提供されます。

libcfgadm ライブラリは、cfgadm(1M) コマンドのサービスを提供するために使用されます。libcfgadm ハードウェア固有のライブラリは /usr/platform/${machine}/lib/cfgadm、/usr/platform/${arch}/lib/cfgadm、/usr/lib/cfgadm にあります。ハードウェア固有のライブラリ名は、接続点を特定するデバイスツリーノードのドライバ名またはクラス名から派生します。

config_change_state() 関数は、システム構成の状態を変更する操作を行います。*state_change_cmd* 引数は、CFGA_CMD_INSERT、CFGA_CMD_REMOVE、CFGA_CMD_DISCONNECT、CFGA_CMD_CONNECT、CFGA_CMD_CONFIGURE、CFGA_CMD_UNCONFIGURE、のいずれかになります。 *state_change_cmd* CFGA_CMD_INSERT は、占有装置の手動による追加の準備をしたり、自動追加を有効にするために使用します。 *state_change_cmd* CFGA_CMD_REMOVE は、占有装置の手動による削除の準備をしたり、自動削除を有効にするために使用します。 *state_change_cmd* CFGA_CMD_DISCONNECT は、受容体に接続されている占有装置との通常の通信を無効にするために使用します。 *state_change_cmd* CFGA_CMD_CONNECTは、受容体に接続されている占有装置との通常の通信を有効にするために使用します。 *state_change_cmd* CFGA_CMD_CONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域に追加し、システムで使用できるようにします。*state_change_cmd* CFGA_CMD_UNCONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域から削除し、システムで使用できないようにします。

flags 引数には、定義済みのフラグである CFGA_FLAG_FORCE とCFGA_FLAG_VERBOSE のいずれかまたは両方を指定することができます。CFGA_FLAG_FORCE フラグを指定すると、特定の安全検査が無効になります。たとえば、このフラグは、failed 状態の占有装置が構成されることを許しませんが、failing 状態の占有装置が構成されることは許す場合があります。強制を受け付けるかどうかは、ハードウェアに依存します。CFGA_FLAG_VERBOSE フラグを指定すると、操作に関係するハードウェア固有の詳細情報が cfga_msg の方法で出力されます。

config_private_func() 関数は、専用ハードウェア固有の関数を起動します。

config_test() 関数は、特定の接続点の検査を開始するために使用します。

num_ap_ids 引数は、*ap_ids* 配列内の *ap_id* の数を特定します。 *ap_ids* 引数は、*ap_id* による配列を示します。

ap_id 引数は、単独の *ap_id* を示します。

function と *options* の文字列は、getsubopt(3C) 構文の規約に従い、ハードウェア固有の関数やオプション情報を提供するために使用します。ハードウェアに依存しない汎用的な関数やオプションは定義されていません。

confp によって参照される cfga_confirm 構造体は、要求された操作がサービスに対する顕著な割り込みを必要とする場合に、継続する許可を得るためのコールバックインタフェースを提供します。 cfga_confirm 構造体には以下のメンバーが含まれます。

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

confirm() 関数は、汎用ポインタの *appdata_ptr* と、確認を必要とするものの詳細を示すメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ *appdata_ptr* は、cfga_confirm 構造体のメンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で confirm 関数の呼び出しをconfig_* の呼び出しに関連付けるために使用されます。confirm 関数は、操作の継続が認められた場合は 1 を返し、認められなかった場合は 0 を返します。

msgp によって参照される cfga_msg 構造体は、ハードウェア固有のライブラリからのメッセージを出力するためのコールバックインタフェースを提供します。CFGA_FLAG_VERBOSE フラグが存在することによって、これらのメッセージは情報メッセージにもなりますが、このフラグが存在しない場合は、エラーメッセージに限定されます。cfga_msg 構造体には、以下のメンバーが含まれます。

```
void (*message_routine)(void *appdata_ptr, const char *message);
void    *appdata_ptr;
```

message_routine() 関数は、汎用ポインタの *appdata_ptr* とメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ*appdata_ptr* は、cfga_confirm 構造体メンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で message_routine() 関数の呼び出しを config_*() の呼び出しに関連付けるために使用されます。このメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

いくつかの一般的なエラーについて、ハードウェア固有のエラーメッセージが返されることがあります。エラーメッセージの文字列(終わりの NULL 文字を含む) の記憶領域は、malloc(3C) と、*errstring* を通じて返される、この記憶領域に対するポインタを使用して config_* 関数によって割り当てられます。*errstring* がNULL の場合は、エラーメッセージは作成または返されることはありません。*errstring* が NULL ではなく、エラーメッセージが作成されない場合は、*errstring* によって参照されるポインタが NULLに設定されます。free(3C) を使用して、返された記憶領域の割り当ての解除は、config_*() を呼び出している関数が行います。このエラーメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

config_list_ext() 関数は、リスト出力用インタフェースです。最初の 2 つの引数を使用して *ap_id*s にリストが指定されていると、指定されている接続点ごとに cfga_list_data_t 構造体の配列を返します。最初の 2 つの引数がそれぞれ 0 と NULL の場合は、デバイスツリーの接続点をすべて出力します。また、フラグの引数を使用して CFGA_FLAG_LIST_ALL フラグを渡した場合は、接続点を動的に展開して動的な接続点を出力する必要があります。返した配列の stat 構造の記憶領域は、config_list_ext() 関数がmalloc(3C) を使用して割り当てます。この記憶領域は、config_list_ext() の呼び出し元が free(3C) を使用して解放しなければなりません。

cfga_list_data 構造体には、以下のメンバーが含まれます。

```
cfga_log_ext_t     ap_log_id;        /* Attachment point logical id */
cfga_phys_ext_t    ap_phys_id;       /* Attachment point physical id */
cfga_class_t       ap_class;         /* Attachment point class */
cfga_stat_t        ap_r_state;       /* Receptacle state */
cfga_stat_t        ap_o_state;       /* Occupant state */
cfga_cond_t        ap_cond;          /* Attachment point condition */
cfga_busy_t        ap_busy;          /* Busy indicator */
time_t             ap_status_time;   /* Attachment point last change*/
cfga_info_t        ap_info;          /* Miscellaneous information */
cfga_type_t        ap_type;          /* Occupant type */
```

種類は以下のように定義されます。

```
typedef char cfga_log_ext_t[CFGA_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];
typedef char cfga_class_t[CFGA_CLASS_LEN];
typedef char cfga_info_t[CFGA_INFO_LEN];
typedef char cfga_type_t[CFGA_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;
```

config_list_ext() の *listopts* 引数は、getsubopt (3C) 構文に準拠しており、サブオプションのリストを渡すときに使用します。現在、サポートされているサブオプションは、*class*=class_name だけです。このリストオプションは、class_name クラスの接続点だけを出力します。

cfga_list_ext() の *listopts* 引数は、今後の使用のために予約されこの引数がNULL の場合は、ハードウェア固有のライブラリはこの引数を無視する必要がありま す。*listopts* 引数がNULL 以外のときに、ハードウェア固有のライブラリでサポートされていない場合は、適切なエラーコードを返す必要があります。

ap_log_id と ap_phys_id メンバーは、接続点に対するハードウェア固有の論理名と物理名を示します。ap_busy メンバーは、状態 (state) や条件 (condition) の変更が生じるような動作があることを示します。ap_status_time メンバーは、接続点のap_r_state、ap_o_state、ap_cond フィールドのいずれかが最後に変わった時間を示します。ap_info メンバーは、接続点に関する追加情報を得るためにハードウェア固有のコードで使用することができます。ap_class メンバーには、接続点の接続点クラスが含まれます (存在する場合)。ap_class メンバーは、汎用ライブラリから組み込まれます。ap_log_id および ap_phys_id メンバーがハードウェア固有のライブラリから組み込まれなかった場合は、汎用ライブラリから汎用形式で組み込まれます。残りのメンバーは、対応するハードウェア固有のライブラリから組み込まれます。

ap_log_id、ap_phys_id、ap_info、ap_class、および ap_type メンバーは、固定長文字列です。実際の文字列がメンバーのサイズよりも短い場合には、この文字列は空文字で終了します。そのため、プログラムは、文字列が空文字で終了していることを前提としてはなりません。これらのフィールドを印刷する場合は、以下の形式を使用してください。

printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);

config_stat()、config_list()、cfga_stat()、cfga_list() 関数およびcfga_stat_data データ構造体は、推奨されていないインタフェースで、下位互換性を維持するためにだけ提供されています。これらのインタフェースはできるだけ使用しないでください。

config_ap_id_cmp 関数は、2 つの *ap_id* に関する、ハードウェアに依存する比較を行い、strcmp(3C) の形式に従って、等しい、小さい、大きい、の結果を返します。各引数 は、cfga_ap_id_t とNULL で終了する文字列のいずれかになります。この関数は、qsort(3C) などによって *ap_id* の一覧をソートする場合や、config_list 関数の呼び出しの結果からエントリを選択する場合に使用することができます。

config_unload_libs 関数は、これまで読み込んだハードウェア固有のライブラリすべてのリンクを解除します。

config_strerror 関数を使用して、エラーメッセージの文字列に対してエラーの戻り値を割り当てることができます。「戻り値」を参照してください。返された文字列は、上書きされてはなりません。 *cfgerrnum* が範囲外にある場合に、config _strerror は NULL を返します。

cfga_help 関数を使用して、ハードウェア固有のライブラリに対して現地仕様のヘルプメッセージを表示するように要求することができます。

戻り値

以下は、config_*() および cfga_*() 関数によって返される戻り値の一覧です。戻り値が CFGA_OK ではない場合は、追加のエラー情報が *errstring* を通して返されます。詳細は、「機能説明」を参照してください。

CFGA_BUSY	システム構成管理システムの 1 要素が使用中であるために、コマンドは完了しませんでした。
CFGA_ATTR_INVAL	指定された属性の接続点は存在しません。
CFGA_ERROR	要求された操作の処理中にエラーが発生しました。このエラーコードは、ハードウェア固有のコードによるコマンド引数の妥当性検査を含みます。
CFGA_INSUFFICIENT_CONDITION	接続点の条件によって、操作が失敗しました。
CFGA_INVAL	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_LIB_ERROR	記憶域やファイル記述子などの処理用資源の獲得の失敗など、ライブラリで手続き上のエラーが発生しました。
CFGA_NACK	*confp*->confirm 関数からの否定応答を受けたために、コマンドは完了しませんでした。
CFGA_NO_LIB	指定された *ap_id* を使用して、ハードウェア固有のライブラリが見つかりませんでした。
CFGA_NOTSUPP	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_OK	要求通りにコマンドが完了しました。
CFGA_OPNOTSUPP	この接続点は、システム構成管理操作に対応していません。
CFGA_PRIV	呼び出し元は要求された処理の特権を所有していません。たとえば、構成管理がデバイスドライバを介して実行される場合は、デバイスノードのアクセス権によってアクセスが制御されます。
CFGA_SYSTEM_BUSY	コマンドがサービスの中断を要求したが、システムの一部が休止できなかったために完了しませんでした。

エラー

システム構成管理機能から返されるエラーの多くは、ハードウェア固有のものです。*errstring* を通して返される文字列には以下があります。

`attachment point` *ap_id* `not known`
エラーメッセージで説明されている接続点は存在しません。

`unknown hardware option` *option* `for` *operation*
未知のオプションが *options* 文字列の中で検出されました。

`hardware option` *option*option `requires a value`
options 文字列で指定されるオプションは、*option=value* の形式である必要があります。

`listing option` *list_option* `requires a value`
listopts 文字列で指定されるオプションは、*option*=`value` 形式である必要があります。

`hardware option` *option* `does not require a value`
options 文字列で指定されるオプションは、単純なオプションである必要があります。

`attachment point` *ap_id* `is not configured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、`CFGA_CMD _UNCONFIGURE` を行う *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `is not unconfigured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、未構成の占有装置を必要とする *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `condition not satisfactory.`
操作の妨げとなる条件を持つ接続点に対して、*config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `in condition` *condition* `cannot be used`
ハードウェアに依存する検査に通らない状態の接続点に対して、強制指定された *config_change_state* 操作が適用されました。

属性

以下の属性については、`attributes`(5) を参照してください。

属性の種類	属性の値
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

関連項目

`cfgadm`(1M), `devinfo`(1M), `dlopen`(3DL), `dlsym`(3DL), `free`(3C), `getsubopt` (3C) , `malloc`(3C), `qsort`(3C), `setlocale`(3C), `strcmp`(3C), `libcfgadm`(3LIB), `attributes`(5)

注意事項

このライブラリを使用するアプリケーションは、実際の実装環境が、外部の `errno` 変数の内容を変更するシステムサービスやファイル記述リソースを使用する可能性があることを考慮したものである必要があります。

以下のコードは、config_*() が CFGA_OK 以外の値を返した場合のエラー処理です。

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

システム構成管理で使用することができる機能に関する詳細は、ハードウェア固有のマニュアルを参照してください。

名前

config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – 構成の管理

形式

```
cc [ flag ] file -lcfgadm  [ library... ]
#include <config_admin.h>

cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int
     num_ap_ids, char * const *ap_ids, const char *options, struct cfga
     _confirm *confp, struct cfga_msg *msgp, char **errstring, cfga
     _flags_t flags);

cfga_err_t config_private_func(const char *function, int num_ap_ids,
     char * const *ap_ids, const char *options, struct cfga_confirm
     *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags)
     ;

cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const
     char *options, struct cfga_msg *msgp, char **errstring, cfga_flags
     _t flags);

cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids,
     struct cfga_list_data **ap_id_list, int *nlist, const char *options,
     const char *listops, char **errstring, cfga_flags_t flags);

int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
     ap_id2);

void config_unload_libs();

const char *config_strerror(cfga_err_t cfgerrnum);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct
     cfga_stat_data *buf, const char *options, char **errstring);

cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist,
     const char *options, char **errstring);
```

ハードウェアに依存するライブラリの形式

config_admin ライブラリは、動的再構成 (DR: Dynamic Reconfiguration) のための汎用的なインタフェースです。DR に対応している各ハードウェアは、ここに一覧されているエントリポイントを含むハードウェア固有の組み込みライブラリを提供する必要があります。この汎用ライブラリは、操作を実現するために、適切なライブラリを見つけ、接続します。ここにあるインタフェースは、汎用ライブラリのユーザーからは隠されています。ただし、ハードウェア固有の組み込みライブラリを作成するには、これらのインタフェースについて知っておく必要があります。

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char
     *ap_id, const char *options, struct cfga_confirm *confp, struct
     cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_private_func(const char *function, const char *ap_id,
     const char *options, struct cfga_confirm *confp, struct cfga_msg
```

```
    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data *
    *ap_id_list, int *nlist, const char *options, char **errstring);
```

機能説明

config_*() 関数は、ハードウェア固有のシステム構成管理機能に対して、ハードウェアに依存しないインタフェースを提供します。 cfga_*() 関数は、構成管理機能をハードウェア固有の方法で処理するために動的に読み込まれた、ハードウェア固有のライブラリによって提供されます。

libcfgadm ライブラリは、cfgadm(1M) コマンドのサービスを提供するために使用されます。libcfgadm ハードウェア固有のライブラリは /usr/platform/${machine}/lib/cfgadm、/usr/platform/${arch}/lib/cfgadm、/usr/lib/cfgadm にあります。ハードウェア固有のライブラリ名は、接続点を特定するデバイスツリーノードのドライバ名またはクラス名から派生します。

config_change_state() 関数は、システム構成の状態を変更する操作を行います。*state_change_cmd* 引数は、CFGA_CMD_INSERT、CFGA_CMD_REMOVE、CFGA_CMD_DISCONNECT、CFGA_CMD_CONNECT、CFGA_CMD_CONFIGURE、CFGA_CMD_UNCONFIGURE、のいずれかになります。 *state_change_cmd* CFGA_CMD_INSERT は、占有装置の手動による追加の準備をしたり、自動追加を有効にするために使用します。 *state_change_cmd* CFGA_CMD_REMOVE は、占有装置の手動による削除の準備をしたり、自動削除を有効にするために使用します。 *state_change_cmd* CFGA_CMD_DISCONNECT は、受容体に接続されている占有装置との通常の通信を無効にするために使用します。 *state_change_cmd* CFGA_CMD_CONNECTは、受容体に接続されている占有装置との通常の通信を有効にするために使用します。 *state_change_cmd* CFGA_CMD_CONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域に追加し、システムで使用できるようにします。*state_change_cmd* CFGA_CMD_UNCONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域から削除し、システムで使用できないようにします。

flags 引数には、定義済みのフラグである CFGA_FLAG_FORCE とCFGA_FLAG_VERBOSE のいずれかまたは両方を指定することができます。CFGA_FLAG_FORCE フラグを指定すると、特定の安全検査が無効になります。たとえば、このフラグは、failed 状態の占有装置が構成されることを許しませんが、failing 状態の占有装置が構成されることは許す場合があります。強制を受け付けるかどうかは、ハードウェアに依存します。CFGA_FLAG_VERBOSE フラグを指定すると、操作に関係するハードウェア固有の詳細情報が cfga_msg の方法で出力されます。

config_private_func() 関数は、専用ハードウェア固有の関数を起動します。

config_test() 関数は、特定の接続点の検査を開始するために使用します。

num_ap_ids 引数は、*ap_ids* 配列内の *ap_id* の数を特定します。 *ap_ids* 引数は、*ap_id* による配列を示します。

ap_id 引数は、単独の *ap_id* を示します。

function と *options* の文字列は、getsubopt(3C) 構文の規約に従い、ハードウェア固有の関数やオプション情報を提供するために使用します。ハードウェアに依存しない汎用的な関数やオプションは定義されていません。

confp によって参照される cfga_confirm 構造体は、要求された操作がサービスに対する顕著な割り込みを必要とする場合に、継続する許可を得るためのコールバックインタフェースを提供します。 cfga_confirm 構造体には以下のメンバーが含まれます。

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

confirm() 関数は、汎用ポインタの *appdata_ptr* と、確認を必要とするものの詳細を示すメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ *appdata_ptr* は、cfga_confirm 構造体のメンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で confirm 関数の呼び出しをconfig_* の呼び出しに関連付けるために使用されます。confirm 関数は、操作の継続が認められた場合は 1 を返し、認められなかった場合は 0 を返します。

msgp によって参照される cfga_msg 構造体は、ハードウェア固有のライブラリからのメッセージを出力するためのコールバックインタフェースを提供します。CFGA_FLAG_VERBOSE フラグが存在することによって、これらのメッセージは情報メッセージにもなりますが、このフラグが存在しない場合は、エラーメッセージに限定されます。cfga_msg 構造体には、以下のメンバーが含まれます。

```
void (*message_routine)(void *appdata_ptr, const char *message);
void    *appdata_ptr;
```

message_routine() 関数は、汎用ポインタの *appdata_ptr* とメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ*appdata_ptr* は、cfga_confirm 構造体メンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で message_routine() 関数の呼び出しを config_*() の呼び出しに関連付けるために使用されます。このメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

いくつかの一般的なエラーについて、ハードウェア固有のエラーメッセージが返されることがあります。エラーメッセージの文字列(終わりの NULL 文字を含む) の記憶領域は、malloc(3C) と、*errstring* を通じて返される、この記憶領域に対するポインタを使用して config_* 関数によって割り当てられます。*errstring* がNULL の場合は、エラーメッセージは作成または返されることはありません。*errstring* が NULL ではなく、エラーメッセージが作成されない場合は、*errstring* によって参照されるポインタが NULLに設定されます。free(3C) を使用して、返された記憶領域の割り当ての解除は、config_*() を呼び出している関数が行います。このエラーメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

config_list_ext() 関数は、リスト出力用インタフェースです。最初の 2 つの引数を使用して *ap_id*s にリストが指定されていると、指定されている接続点ごとに cfga_list_data_t 構造体の配列を返します。最初の 2 つの引数がそれぞれ 0 と NULL の場合は、デバイスツリーの接続点をすべて出力します。また、フラグの引数を使用して CFGA_FLAG_LIST_ALL フラグを渡した場合は、接続点を動的に展開して動的な接続点を出力する必要があります。返した配列の stat 構造の記憶領域は、config_list_ext() 関数がmalloc(3C) を使用して割り当てます。この記憶領域は、config_list_ext() の呼び出し元が free(3C) を使用して解放しなければなりません。

cfga_list_data 構造体には、以下のメンバーが含まれます。

```
cfga_log_ext_t     ap_log_id;        /* Attachment point logical id */
cfga_phys_ext_t    ap_phys_id;       /* Attachment point physical id */
cfga_class_t       ap_class;         /* Attachment point class */
cfga_stat_t        ap_r_state;       /* Receptacle state */
cfga_stat_t        ap_o_state;       /* Occupant state */
cfga_cond_t        ap_cond;          /* Attachment point condition */
cfga_busy_t        ap_busy;          /* Busy indicator */
time_t             ap_status_time;   /* Attachment point last change*/
cfga_info_t        ap_info;          /* Miscellaneous information */
cfga_type_t        ap_type;          /* Occupant type */
```

種類は以下のように定義されます。

```
typedef char cfga_log_ext_t[CFGA_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];
typedef char cfga_class_t[CFGA_CLASS_LEN];
typedef char cfga_info_t[CFGA_INFO_LEN];
typedef char cfga_type_t[CFGA_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;
```

config_list_ext() の *listopts* 引数は、getsubopt (3C) 構文に準拠しており、サブオプションのリストを渡すときに使用します。現在、サポートされているサブオプションは、*class*=class_name だけです。このリストオプションは、class_name クラスの接続点だけを出力します。

cfga_list_ext() の *listopts* 引数は、今後の使用のために予約されこの引数がNULL の場合は、ハードウェア固有のライブラリはこの引数を無視する必要がありま す。*listopts* 引数がNULL 以外のときに、ハードウェア固有のライブラリでサポートされていない場合は、適切なエラーコードを返す必要があります。

ap_log_id と ap_phys_id メンバーは、接続点に対するハードウェア固有の論理名と物理名を示します。ap_busy メンバーは、状態 (state) や条件 (condition) の変更が生じるような動作があることを示します。ap_status_time メンバーは、接続点のap_r_state、ap_o_state、ap_cond フィールドのいずれかが最後に変わった時間を示します。ap_info メンバーは、接続点に関する追加情報を得るためにハードウェア固有のコードで使用することができます。ap_class メンバーには、接続点の接続点クラスが含まれます (存在する場合)。ap_class メンバーは、汎用ライブラリから組み込まれます。ap_log_id および ap_phys_id メンバーがハードウェア固有のライブラリから組み込まれなかった場合は、汎用ライブラリから汎用形式で組み込まれます。残りのメンバーは、対応するハードウェア固有のライブラリから組み込まれます。

ap_log_id、ap_phys_id、ap_info、ap_class、および ap_type メンバーは、固定長文字列です。実際の文字列がメンバーのサイズよりも短い場合には、この文字列は空文字で終了します。そのため、プログラムは、文字列が空文字で終了していることを前提としてはなりません。これらのフィールドを印刷する場合は、以下の形式を使用してください。

printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);

config_stat()、config_list()、cfga_stat()、cfga_list() 関数およびcfga_stat_data データ構造体は、推奨されていないインタフェースで、下位互換性を維持するためにだけ提供されています。これらのインタフェースはできるだけ使用しないでください。

config_ap_id_cmp 関数は、2 つの *ap_id* に関する、ハードウェアに依存する比較を行い、strcmp(3C) の形式に従って、等しい、小さい、大きい、の結果を返します。各引数 は、cfga_ap_id_t とNULL で終了する文字列のいずれかになります。この関数は、qsort(3C) などによって *ap_id* の一覧をソートする場合や、config_list 関数の呼び出しの結果からエントリを選択する場合に使用することができます。

config_unload_libs 関数は、これまで読み込んだハードウェア固有のライブラリすべてのリンクを解除します。

config_strerror 関数を使用して、エラーメッセージの文字列に対してエラーの戻り値を割り当てることができます。「戻り値」を参照してください。返された文字列は、上書きされてはなりません。 *cfgerrnum* が範囲外にある場合に、config _strerror は NULL を返します。

cfga_help 関数を使用して、ハードウェア固有のライブラリに対して現地仕様のヘルプメッセージを表示するように要求することができます。

戻り値

以下は、`config_*()` および `cfga_*()` 関数によって返される戻り値の一覧です。戻り値が `CFGA_OK` ではない場合は、追加のエラー情報が *errstring* を通して返されます。詳細は、「機能説明」を参照してください。

`CFGA_BUSY`	システム構成管理システムの 1 要素が使用中であるために、コマンドは完了しませんでした。
`CFGA_ATTR_INVAL`	指定された属性の接続点は存在しません。
`CFGA_ERROR`	要求された操作の処理中にエラーが発生しました。このエラーコードは、ハードウェア固有のコードによるコマンド引数の妥当性検査を含みます。
`CFGA_INSUFFICIENT_CONDITION`	接続点の条件によって、操作が失敗しました。
`CFGA_INVAL`	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
`CFGA_LIB_ERROR`	記憶域やファイル記述子などの処理用資源の獲得の失敗など、ライブラリで手続き上のエラーが発生しました。
`CFGA_NACK`	*confp*`->confirm` 関数からの否定応答を受けたために、コマンドは完了しませんでした。
`CFGA_NO_LIB`	指定された *ap_id* を使用して、ハードウェア固有のライブラリが見つかりませんでした。
`CFGA_NOTSUPP`	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
`CFGA_OK`	要求通りにコマンドが完了しました。
`CFGA_OPNOTSUPP`	この接続点は、システム構成管理操作に対応していません。
`CFGA_PRIV`	呼び出し元は要求された処理の特権を所有していません。たとえば、構成管理がデバイスドライバを介して実行される場合は、デバイスノードのアクセス権によってアクセスが制御されます。
`CFGA_SYSTEM_BUSY`	コマンドがサービスの中断を要求したが、システムの一部が休止できなかったために完了しませんでした。

エラー

システム構成管理機能から返されるエラーの多くは、ハードウェア固有のものです。*errstring* を通して返される文字列には以下があります。

`attachment point` *ap_id* `not known`
エラーメッセージで説明されている接続点は存在しません。

`unknown hardware option` *option* `for` *operation*
未知のオプションが *options* 文字列の中で検出されました。

`hardware option` *option*option `requires a value`
options 文字列で指定されるオプションは、*option=value* の形式である必要があります。

`listing option` *list_option* `requires a value`
listopts 文字列で指定されるオプションは、*option*=`value` 形式である必要があります。

`hardware option` *option* `does not require a value`
options 文字列で指定されるオプションは、単純なオプションである必要があります。

`attachment point` *ap_id* `is not configured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、`CFGA_CMD_UNCONFIGURE` を行う *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `is not unconfigured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、未構成の占有装置を必要とする *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `condition not satisfactory.`
操作の妨げとなる条件を持つ接続点に対して、*config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `in condition` *condition* `cannot be used`
ハードウェアに依存する検査に通らない状態の接続点に対して、強制指定された *config_change_state* 操作が適用されました。

属性

以下の属性については、`attributes`(5) を参照してください。

属性の種類	属性の値
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

関連項目

`cfgadm`(1M), `devinfo`(1M), `dlopen`(3DL), `dlsym`(3DL), `free`(3C), `getsubopt` (3C), `malloc`(3C), `qsort`(3C), `setlocale`(3C), `strcmp`(3C), `libcfgadm`(3LIB), `attributes`(5)

注意事項

このライブラリを使用するアプリケーションは、実際の実装環境が、外部の `errno` 変数の内容を変更するシステムサービスやファイル記述リソースを使用する可能性があることを考慮したものである必要があります。

以下のコードは、config_*() が CFGA_OK 以外の値を返した場合のエラー処理です。

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

システム構成管理で使用することができる機能に関する詳細は、ハードウェア固有のマニュアルを参照してください。

名前

config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – 構成の管理

形式

```
cc [ flag ] file -lcfgadm  [ library... ]
#include <config_admin.h>

cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int
     num_ap_ids, char * const *ap_ids, const char *options, struct cfga
     _confirm *confp, struct cfga_msg *msgp, char **errstring, cfga
     _flags_t flags);

cfga_err_t config_private_func(const char *function, int num_ap_ids,
     char * const *ap_ids, const char *options, struct cfga_confirm
     *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags)
     ;

cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const
     char *options, struct cfga_msg *msgp, char **errstring, cfga_flags
     _t flags);

cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids,
     struct cfga_list_data **ap_id_list, int *nlist, const char *options,
     const char *listops, char **errstring, cfga_flags_t flags);

int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
     ap_id2);

void config_unload_libs();

const char *config_strerror(cfga_err_t cfgerrnum);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct
     cfga_stat_data *buf, const char *options, char **errstring);

cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist,
     const char *options, char **errstring);
```

ハードウェアに依存するライブラリの形式

config_admin ライブラリは、動的再構成 (DR: Dynamic Reconfiguration) のための汎用的なインタフェースです。DR に対応している各ハードウェアは、ここに一覧されているエントリポイントを含むハードウェア固有の組み込みライブラリを提供する必要があります。この汎用ライブラリは、操作を実現するために、適切なライブラリを見つけ、接続します。ここにあるインタフェースは、汎用ライブラリのユーザーからは隠されています。ただし、ハードウェア固有の組み込みライブラリを作成するには、これらのインタフェースについて知っておく必要があります。

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char
     *ap_id, const char *options, struct cfga_confirm *confp, struct
     cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_private_func(const char *function, const char *ap_id,
     const char *options, struct cfga_confirm *confp, struct cfga_msg
```

```
    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data *
    *ap_id_list, int *nlist, const char *options, char **errstring);
```

機能説明

config_*() 関数は、ハードウェア固有のシステム構成管理機能に対して、ハードウェアに依存しないインタフェースを提供します。 cfga_*() 関数は、構成管理機能をハードウェア固有の方法で処理するために動的に読み込まれた、ハードウェア固有のライブラリによって提供されます。

libcfgadm ライブラリは、cfgadm(1M) コマンドのサービスを提供するために使用されます。libcfgadm ハードウェア固有のライブラリは /usr/platform/${machine}/lib/cfgadm、/usr/platform/${arch}/lib/cfgadm、/usr/lib/cfgadm にあります。ハードウェア固有のライブラリ名は、接続点を特定するデバイスツリーノードのドライバ名またはクラス名から派生します。

config_change_state() 関数は、システム構成の状態を変更する操作を行います。*state_change_cmd* 引数は、CFGA_CMD_INSERT、CFGA_CMD_REMOVE、CFGA_CMD_DISCONNECT、CFGA_CMD_CONNECT、CFGA_CMD_CONFIGURE、CFGA_CMD_UNCONFIGURE、のいずれかになります。 *state_change_cmd* CFGA_CMD_INSERT は、占有装置の手動による追加の準備をしたり、自動追加を有効にするために使用します。 *state_change_cmd* CFGA_CMD_REMOVE は、占有装置の手動による削除の準備をしたり、自動削除を有効にするために使用します。 *state_change_cmd* CFGA_CMD_DISCONNECT は、受容体に接続されている占有装置との通常の通信を無効にするために使用します。 *state_change_cmd* CFGA_CMD_CONNECTは、受容体に接続されている占有装置との通常の通信を有効にするために使用します。 *state_change_cmd* CFGA_CMD_CONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域に追加し、システムで使用できるようにします。*state_change_cmd* CFGA_CMD_UNCONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域から削除し、システムで使用できないようにします。

flags 引数には、定義済みのフラグである CFGA_FLAG_FORCE とCFGA_FLAG_VERBOSE のいずれかまたは両方を指定することができます。CFGA_FLAG_FORCE フラグを指定すると、特定の安全検査が無効になります。たとえば、このフラグは、failed 状態の占有装置が構成されることを許しませんが、failing 状態の占有装置が構成されることは許す場合があります。強制を受け付けるかどうかは、ハードウェアに依存します。CFGA_FLAG_VERBOSE フラグを指定すると、操作に関係するハードウェア固有の詳細情報が cfga_msg の方法で出力されます。

config_private_func() 関数は、専用ハードウェア固有の関数を起動します。

config_test() 関数は、特定の接続点の検査を開始するために使用します。

num_ap_ids 引数は、*ap_ids* 配列内の *ap_id* の数を特定します。 *ap_ids* 引数は、*ap_id* による配列を示します。

ap_id 引数は、単独の *ap_id* を示します。

function と *options* の文字列は、getsubopt(3C) 構文の規約に従い、ハードウェア固有の関数やオプション情報を提供するために使用します。ハードウェアに依存しない汎用的な関数やオプションは定義されていません。

confp によって参照される cfga_confirm 構造体は、要求された操作がサービスに対する顕著な割り込みを必要とする場合に、継続する許可を得るためのコールバックインタフェースを提供します。 cfga_confirm 構造体には以下のメンバーが含まれます。

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

confirm() 関数は、汎用ポインタの *appdata_ptr* と、確認を必要とするものの詳細を示すメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ *appdata_ptr* は、cfga_confirm 構造体のメンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で confirm 関数の呼び出しをconfig_* の呼び出しに関連付けるために使用されます。confirm 関数は、操作の継続が認められた場合は 1 を返し、認められなかった場合は 0 を返します。

msgp によって参照される cfga_msg 構造体は、ハードウェア固有のライブラリからのメッセージを出力するためのコールバックインタフェースを提供します。CFGA_FLAG_VERBOSE フラグが存在することによって、これらのメッセージは情報メッセージにもなりますが、このフラグが存在しない場合は、エラーメッセージに限定されます。cfga_msg 構造体には、以下のメンバーが含まれます。

```
void (*message_routine)(void *appdata_ptr, const char *message);
void    *appdata_ptr;
```

message_routine() 関数は、汎用ポインタの *appdata_ptr* とメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ*appdata_ptr* は、cfga_confirm 構造体メンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で message_routine() 関数の呼び出しを config_*() の呼び出しに関連付けるために使用されます。このメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

いくつかの一般的なエラーについて、ハードウェア固有のエラーメッセージが返されることがあります。エラーメッセージの文字列(終わりの NULL 文字を含む) の記憶領域は、malloc(3C) と、*errstring* を通じて返される、この記憶領域に対するポインタを使用して config_* 関数によって割り当てられます。*errstring* がNULL の場合は、エラーメッセージは作成または返されることはありません。*errstring* が NULL ではなく、エラーメッセージが作成されない場合は、*errstring* によって参照されるポインタが NULLに設定されます。free(3C) を使用して、返された記憶領域の割り当ての解除は、config_*() を呼び出している関数が行います。このエラーメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

config_list_ext() 関数は、リスト出力用インタフェースです。最初の 2 つの引数を使用して *ap_id*s にリストが指定されていると、指定されている接続点ごとに cfga_list_data_t 構造体の配列を返します。最初の 2 つの引数がそれぞれ 0 と NULL の場合は、デバイスツリーの接続点をすべて出力します。また、フラグの引数を使用して CFGA_FLAG_LIST_ALL フラグを渡した場合は、接続点を動的に展開して動的な接続点を出力する必要があります。返した配列の stat 構造の記憶領域は、config_list_ext() 関数がmalloc(3C) を使用して割り当てます。この記憶領域は、config_list_ext() の呼び出し元が free(3C) を使用して解放しなければなりません。

cfga_list_data 構造体には、以下のメンバーが含まれます。

```
cfga_log_ext_t     ap_log_id;        /* Attachment point logical id */
cfga_phys_ext_t    ap_phys_id;       /* Attachment point physical id */
cfga_class_t       ap_class;         /* Attachment point class */
cfga_stat_t        ap_r_state;       /* Receptacle state */
cfga_stat_t        ap_o_state;       /* Occupant state */
cfga_cond_t        ap_cond;          /* Attachment point condition */
cfga_busy_t        ap_busy;          /* Busy indicator */
time_t             ap_status_time;   /* Attachment point last change*/
cfga_info_t        ap_info;          /* Miscellaneous information */
cfga_type_t        ap_type;          /* Occupant type */
```

種類は以下のように定義されます。

```
typedef char cfga_log_ext_t[CFGA_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];
typedef char cfga_class_t[CFGA_CLASS_LEN];
typedef char cfga_info_t[CFGA_INFO_LEN];
typedef char cfga_type_t[CFGA_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;
```

config_list_ext() の *listopts* 引数は、getsubopt (3C) 構文に準拠しており、サブオプションのリストを渡すときに使用します。現在、サポートされているサブオプションは、*class*=class_name だけです。このリストオプションは、class_name クラスの接続点だけを出力します。

cfga_list_ext() の *listopts* 引数は、今後の使用のために予約されこの引数がNULL の場合は、ハードウェア固有のライブラリはこの引数を無視する必要がありま す。*listopts* 引数がNULL 以外のときに、ハードウェア固有のライブラリでサポートされていない場合は、適切なエラーコードを返す必要があります。

ap_log_id と ap_phys_id メンバーは、接続点に対するハードウェア固有の論理名と物理名を示します。ap_busy メンバーは、状態 (state) や条件 (condition) の変更が生じるような動作があることを示します。ap_status_time メンバーは、接続点のap_r_state、ap_o_state、ap_cond フィールドのいずれかが最後に変わった時間を示します。ap_info メンバーは、接続点に関する追加情報を得るためにハードウェア固有のコードで使用することができます。ap_class メンバーには、接続点の接続点クラスが含まれます (存在する場合)。ap_class メンバーは、汎用ライブラリから組み込まれます。ap_log_id および ap_phys_id メンバーがハードウェア固有のライブラリから組み込まれなかった場合は、汎用ライブラリから汎用形式で組み込まれます。残りのメンバーは、対応するハードウェア固有のライブラリから組み込まれます。

ap_log_id、ap_phys_id、ap_info、ap_class、および ap_type メンバーは、固定長文字列です。実際の文字列がメンバーのサイズよりも短い場合には、この文字列は空文字で終了します。そのため、プログラムは、文字列が空文字で終了していることを前提としてはなりません。これらのフィールドを印刷する場合は、以下の形式を使用してください。

printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);

config_stat()、config_list()、cfga_stat()、cfga_list() 関数およびcfga_stat_data データ構造体は、推奨されていないインタフェースで、下位互換性を維持するためにだけ提供されています。これらのインタフェースはできるだけ使用しないでください。

config_ap_id_cmp 関数は、2 つの *ap_id* に関する、ハードウェアに依存する比較を行い、strcmp(3C) の形式に従って、等しい、小さい、大きい、の結果を返します。各引数 は、cfga_ap_id_t とNULL で終了する文字列のいずれかになります。この関数は、qsort(3C) などによって *ap_id* の一覧をソートする場合や、config_list 関数の呼び出しの結果からエントリを選択する場合に使用することができます。

config_unload_libs 関数は、これまで読み込んだハードウェア固有のライブラリすべてのリンクを解除します。

config_strerror 関数を使用して、エラーメッセージの文字列に対してエラーの戻り値を割り当てることができます。「戻り値」を参照してください。返された文字列は、上書きされてはなりません。 *cfgerrnum* が範囲外にある場合に、config _strerror は NULL を返します。

cfga_help 関数を使用して、ハードウェア固有のライブラリに対して現地仕様のヘルプメッセージを表示するように要求することができます。

戻り値

以下は、config_*() および cfga_*() 関数によって返される戻り値の一覧です。戻り値が CFGA_OK ではない場合は、追加のエラー情報が *errstring* を通して返されます。詳細は、「機能説明」を参照してください。

CFGA_BUSY	システム構成管理システムの 1 要素が使用中であるために、コマンドは完了しませんでした。
CFGA_ATTR_INVAL	指定された属性の接続点は存在しません。
CFGA_ERROR	要求された操作の処理中にエラーが発生しました。このエラーコードは、ハードウェア固有のコードによるコマンド引数の妥当性検査を含みます。
CFGA_INSUFFICIENT_CONDITION	接続点の条件によって、操作が失敗しました。
CFGA_INVAL	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_LIB_ERROR	記憶域やファイル記述子などの処理用資源の獲得の失敗など、ライブラリで手続き上のエラーが発生しました。
CFGA_NACK	*confp*->confirm 関数からの否定応答を受けたために、コマンドは完了しませんでした。
CFGA_NO_LIB	指定された *ap_id* を使用して、ハードウェア固有のライブラリが見つかりませんでした。
CFGA_NOTSUPP	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_OK	要求通りにコマンドが完了しました。
CFGA_OPNOTSUPP	この接続点は、システム構成管理操作に対応していません。
CFGA_PRIV	呼び出し元は要求された処理の特権を所有していません。たとえば、構成管理がデバイスドライバを介して実行される場合は、デバイスノードのアクセス権によってアクセスが制御されます。
CFGA_SYSTEM_BUSY	コマンドがサービスの中断を要求したが、システムの一部が休止できなかったために完了しませんでした。

エラー

システム構成管理機能から返されるエラーの多くは、ハードウェア固有のものです。*errstring* を通して返される文字列には以下があります。

`attachment point` *ap_id* `not known`
エラーメッセージで説明されている接続点は存在しません。

`unknown hardware option` *option* `for` *operation*
未知のオプションが *options* 文字列の中で検出されました。

`hardware option` *option*option `requires a value`
options 文字列で指定されるオプションは、*option=value* の形式である必要があります。

`listing option` *list_option* `requires a value`
listopts 文字列で指定されるオプションは、*option*=`value` 形式である必要があります。

`hardware option` *option* `does not require a value`
options 文字列で指定されるオプションは、単純なオプションである必要があります。

`attachment point` *ap_id* `is not configured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、`CFGA_CMD_UNCONFIGURE` を行う *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `is not unconfigured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、未構成の占有装置を必要とする *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `condition not satisfactory.`
操作の妨げとなる条件を持つ接続点に対して、*config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `in condition` *condition* `cannot be used`
ハードウェアに依存する検査に通らない状態の接続点に対して、強制指定された *config_change_state* 操作が適用されました。

属性

以下の属性については、`attributes`(5) を参照してください。

属性の種類	属性の値
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

関連項目

`cfgadm`(1M), `devinfo`(1M), `dlopen`(3DL), `dlsym`(3DL), `free`(3C), `getsubopt` (3C), `malloc`(3C), `qsort`(3C), `setlocale`(3C), `strcmp`(3C), `libcfgadm`(3LIB), `attributes`(5)

注意事項

このライブラリを使用するアプリケーションは、実際の実装環境が、外部の `errno` 変数の内容を変更するシステムサービスやファイル記述リソースを使用する可能性があることを考慮したものである必要があります。

以下のコードは、config_*() が CFGA_OK 以外の値を返した場合のエラー処理です。

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

システム構成管理で使用することができる機能に関する詳細は、ハードウェア固有のマニュアルを参照してください。

名前

config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – 構成の管理

形式

```
cc [ flag ] file -lcfgadm  [ library... ]
#include <config_admin.h>

cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int
     num_ap_ids, char * const *ap_ids, const char *options, struct cfga
     _confirm *confp, struct cfga_msg *msgp, char **errstring, cfga
     _flags_t flags);

cfga_err_t config_private_func(const char *function, int num_ap_ids,
     char * const *ap_ids, const char *options, struct cfga_confirm
     *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags)
     ;

cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const
     char *options, struct cfga_msg *msgp, char **errstring, cfga_flags
     _t flags);

cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids,
     struct cfga_list_data **ap_id_list, int *nlist, const char *options,
     const char *listops, char **errstring, cfga_flags_t flags);

int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
     ap_id2);

void config_unload_libs();

const char *config_strerror(cfga_err_t cfgerrnum);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct
     cfga_stat_data *buf, const char *options, char **errstring);

cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist,
     const char *options, char **errstring);
```

ハードウェアに依存するライブラリの形式

config_admin ライブラリは、動的再構成 (DR: Dynamic Reconfiguration) のための汎用的なインタフェースです。DR に対応している各ハードウェアは、ここに一覧されているエントリポイントを含むハードウェア固有の組み込みライブラリを提供する必要があります。この汎用ライブラリは、操作を実現するために、適切なライブラリを見つけ、接続します。ここにあるインタフェースは、汎用ライブラリのユーザーからは隠されています。ただし、ハードウェア固有の組み込みライブラリを作成するには、これらのインタフェースについて知っておく必要があります。

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char
     *ap_id, const char *options, struct cfga_confirm *confp, struct
     cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_private_func(const char *function, const char *ap_id,
     const char *options, struct cfga_confirm *confp, struct cfga_msg
```

```
    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data *
    *ap_id_list, int *nlist, const char *options, char **errstring);
```

機能説明

config_*() 関数は、ハードウェア固有のシステム構成管理機能に対して、ハードウェアに依存しないインタフェースを提供します。 cfga_*() 関数は、構成管理機能をハードウェア固有の方法で処理するために動的に読み込まれた、ハードウェア固有のライブラリによって提供されます。

libcfgadm ライブラリは、cfgadm(1M) コマンドのサービスを提供するために使用されます。libcfgadm ハードウェア固有のライブラリは /usr/platform/${machine}/lib/cfgadm、/usr/platform/${arch}/lib/cfgadm、/usr/lib/cfgadm にあります。ハードウェア固有のライブラリ名は、接続点を特定するデバイスツリーノードのドライバ名またはクラス名から派生します。

config_change_state() 関数は、システム構成の状態を変更する操作を行います。*state_change_cmd* 引数は、CFGA_CMD_INSERT、CFGA_CMD_REMOVE、CFGA_CMD_DISCONNECT、CFGA_CMD_CONNECT、CFGA_CMD_CONFIGURE、CFGA_CMD_UNCONFIGURE、のいずれかになります。 *state_change_cmd* CFGA_CMD_INSERT は、占有装置の手動による追加の準備をしたり、自動追加を有効にするために使用します。 *state_change_cmd* CFGA_CMD_REMOVE は、占有装置の手動による削除の準備をしたり、自動削除を有効にするために使用します。 *state_change_cmd* CFGA_CMD_DISCONNECT は、受容体に接続されている占有装置との通常の通信を無効にするために使用します。 *state_change_cmd* CFGA_CMD_CONNECTは、受容体に接続されている占有装置との通常の通信を有効にするために使用します。 *state_change_cmd* CFGA_CMD_CONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域に追加し、システムで使用できるようにします。*state_change_cmd* CFGA_CMD_UNCONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域から削除し、システムで使用できないようにします。

flags 引数には、定義済みのフラグである CFGA_FLAG_FORCE とCFGA_FLAG_VERBOSE のいずれかまたは両方を指定することができます。CFGA_FLAG_FORCE フラグを指定すると、特定の安全検査が無効になります。たとえば、このフラグは、failed 状態の占有装置が構成されることを許しませんが、failing 状態の占有装置が構成されることは許す場合があります。強制を受け付けるかどうかは、ハードウェアに依存します。CFGA_FLAG_VERBOSE フラグを指定すると、操作に関係するハードウェア固有の詳細情報が cfga_msg の方法で出力されます。

config_private_func() 関数は、専用ハードウェア固有の関数を起動します。

config_test() 関数は、特定の接続点の検査を開始するために使用します。

num_ap_ids 引数は、*ap_ids* 配列内の *ap_id* の数を特定します。 *ap_ids* 引数は、*ap_id* による配列を示します。

ap_id 引数は、単独の *ap_id* を示します。

function と *options* の文字列は、getsubopt(3C) 構文の規約に従い、ハードウェア固有の関数やオプション情報を提供するために使用します。ハードウェアに依存しない汎用的な関数やオプションは定義されていません。

confp によって参照される cfga_confirm 構造体は、要求された操作がサービスに対する顕著な割り込みを必要とする場合に、継続する許可を得るためのコールバックインタフェースを提供します。 cfga_confirm 構造体には以下のメンバーが含まれます。

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

confirm() 関数は、汎用ポインタの *appdata_ptr* と、確認を必要とするものの詳細を示すメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ *appdata_ptr* は、cfga_confirm 構造体のメンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で confirm 関数の呼び出しをconfig_* の呼び出しに関連付けるために使用されます。confirm 関数は、操作の継続が認められた場合は 1 を返し、認められなかった場合は 0 を返します。

msgp によって参照される cfga_msg 構造体は、ハードウェア固有のライブラリからのメッセージを出力するためのコールバックインタフェースを提供します。CFGA_FLAG_VERBOSE フラグが存在することによって、これらのメッセージは情報メッセージにもなりますが、このフラグが存在しない場合は、エラーメッセージに限定されます。cfga_msg 構造体には、以下のメンバーが含まれます。

```
void (*message_routine)(void *appdata_ptr, const char *message);
void    *appdata_ptr;
```

message_routine() 関数は、汎用ポインタの *appdata_ptr* とメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ*appdata_ptr* は、cfga_confirm 構造体メンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で message_routine() 関数の呼び出しを config_*() の呼び出しに関連付けるために使用されます。このメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

いくつかの一般的なエラーについて、ハードウェア固有のエラーメッセージが返されることがあります。エラーメッセージの文字列(終わりの NULL 文字を含む) の記憶領域は、malloc(3C) と、*errstring* を通じて返される、この記憶領域に対するポインタを使用して config_* 関数によって割り当てられます。*errstring* がNULL の場合は、エラーメッセージは作成または返されることはありません。*errstring* が NULL ではなく、エラーメッセージが作成されない場合は、*errstring* によって参照されるポインタが NULLに設定されます。free(3C) を使用して、返された記憶領域の割り当ての解除は、config_*() を呼び出している関数が行います。このエラーメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

config_list_ext() 関数は、リスト出力用インタフェースです。最初の 2 つの引数を使用して *ap_id*s にリストが指定されていると、指定されている接続点ごとに cfga_list_data_t 構造体の配列を返します。最初の 2 つの引数がそれぞれ 0 と NULL の場合は、デバイスツリーの接続点をすべて出力します。また、フラグの引数を使用して CFGA_FLAG_LIST_ALL フラグを渡した場合は、接続点を動的に展開して動的な接続点を出力する必要があります。返した配列の stat 構造の記憶領域は、config_list_ext() 関数がmalloc(3C) を使用して割り当てます。この記憶領域は、config_list_ext() の呼び出し元が free(3C) を使用して解放しなければなりません。

cfga_list_data 構造体には、以下のメンバーが含まれます。

```
cfga_log_ext_t     ap_log_id;        /* Attachment point logical id */
cfga_phys_ext_t    ap_phys_id;       /* Attachment point physical id */
cfga_class_t       ap_class;         /* Attachment point class */
cfga_stat_t        ap_r_state;       /* Receptacle state */
cfga_stat_t        ap_o_state;       /* Occupant state */
cfga_cond_t        ap_cond;          /* Attachment point condition */
cfga_busy_t        ap_busy;          /* Busy indicator */
time_t             ap_status_time;   /* Attachment point last change*/
cfga_info_t        ap_info;          /* Miscellaneous information */
cfga_type_t        ap_type;          /* Occupant type */
```

種類は以下のように定義されます。

```
typedef char cfga_log_ext_t[CFGA_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];
typedef char cfga_class_t[CFGA_CLASS_LEN];
typedef char cfga_info_t[CFGA_INFO_LEN];
typedef char cfga_type_t[CFGA_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;
```

config_list_ext() の *listopts* 引数は、getsubopt (3C) 構文に準拠しており、サブオプションのリストを渡すときに使用します。現在、サポートされているサブオプションは、*class*=class_name だけです。このリストオプションは、class_name クラスの接続点だけを出力します。

cfga_list_ext() の *listopts* 引数は、今後の使用のために予約されこの引数がNULL の場合は、ハードウェア固有のライブラリはこの引数を無視する必要がありま す。*listopts* 引数がNULL 以外のときに、ハードウェア固有のライブラリでサポートされていない場合は、適切なエラーコードを返す必要があります。

ap_log_id と ap_phys_id メンバーは、接続点に対するハードウェア固有の論理名と物理名を示します。ap_busy メンバーは、状態 (state) や条件 (condition) の変更が生じるような動作があることを示します。ap_status_time メンバーは、接続点のap_r_state、ap_o_state、ap_cond フィールドのいずれかが最後に変わった時間を示します。ap_info メンバーは、接続点に関する追加情報を得るためにハードウェア固有のコードで使用することができます。ap_class メンバーには、接続点の接続点クラスが含まれます (存在する場合)。ap_class メンバーは、汎用ライブラリから組み込まれます。ap_log_id および ap_phys_id メンバーがハードウェア固有のライブラリから組み込まれなかった場合は、汎用ライブラリから汎用形式で組み込まれます。残りのメンバーは、対応するハードウェア固有のライブラリから組み込まれます。

ap_log_id、ap_phys_id、ap_info、ap_class、および ap_type メンバーは、固定長文字列です。実際の文字列がメンバーのサイズよりも短い場合には、この文字列は空文字で終了します。そのため、プログラムは、文字列が空文字で終了していることを前提としてはなりません。これらのフィールドを印刷する場合は、以下の形式を使用してください。

printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);

config_stat()、config_list()、cfga_stat()、cfga_list() 関数およびcfga_stat_data データ構造体は、推奨されていないインタフェースで、下位互換性を維持するためにだけ提供されています。これらのインタフェースはできるだけ使用しないでください。

config_ap_id_cmp 関数は、2 つの *ap_id* に関する、ハードウェアに依存する比較を行い、strcmp(3C) の形式に従って、等しい、小さい、大きい、の結果を返します。各引数 は、cfga_ap_id_t とNULL で終了する文字列のいずれかになります。この関数は、qsort(3C) などによって *ap_id* の一覧をソートする場合や、config_list 関数の呼び出しの結果からエントリを選択する場合に使用することができます。

config_unload_libs 関数は、これまで読み込んだハードウェア固有のライブラリすべてのリンクを解除します。

config_strerror 関数を使用して、エラーメッセージの文字列に対してエラーの戻り値を割り当てることができます。「戻り値」を参照してください。返された文字列は、上書きされてはなりません。 *cfgerrnum* が範囲外にある場合に、config _strerror は NULL を返します。

cfga_help 関数を使用して、ハードウェア固有のライブラリに対して現地仕様のヘルプメッセージを表示するように要求することができます。

戻り値

以下は、config_*() および cfga_*() 関数によって返される戻り値の一覧です。戻り値が CFGA_OK ではない場合は、追加のエラー情報が *errstring* を通して返されます。詳細は、「機能説明」を参照してください。

CFGA_BUSY	システム構成管理システムの 1 要素が使用中であるために、コマンドは完了しませんでした。
CFGA_ATTR_INVAL	指定された属性の接続点は存在しません。
CFGA_ERROR	要求された操作の処理中にエラーが発生しました。このエラーコードは、ハードウェア固有のコードによるコマンド引数の妥当性検査を含みます。
CFGA_INSUFFICIENT_CONDITION	接続点の条件によって、操作が失敗しました。
CFGA_INVAL	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_LIB_ERROR	記憶域やファイル記述子などの処理用資源の獲得の失敗など、ライブラリで手続き上のエラーが発生しました。
CFGA_NACK	*confp*->confirm 関数からの否定応答を受けたために、コマンドは完了しませんでした。
CFGA_NO_LIB	指定された *ap_id* を使用して、ハードウェア固有のライブラリが見つかりませんでした。
CFGA_NOTSUPP	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_OK	要求通りにコマンドが完了しました。
CFGA_OPNOTSUPP	この接続点は、システム構成管理操作に対応していません。
CFGA_PRIV	呼び出し元は要求された処理の特権を所有していません。たとえば、構成管理がデバイスドライバを介して実行される場合は、デバイスノードのアクセス権によってアクセスが制御されます。
CFGA_SYSTEM_BUSY	コマンドがサービスの中断を要求したが、システムの一部が休止できなかったために完了しませんでした。

エラー

システム構成管理機能から返されるエラーの多くは、ハードウェア固有のものです。*errstring* を通して返される文字列には以下があります。

`attachment point` *ap_id* `not known`
エラーメッセージで説明されている接続点は存在しません。

`unknown hardware option` *option* `for` *operation*
未知のオプションが *options* 文字列の中で検出されました。

`hardware option` *option*option `requires a value`
options 文字列で指定されるオプションは、*option=value* の形式である必要があります。

`listing option` *list_option* `requires a value`
listopts 文字列で指定されるオプションは、*option*=`value` 形式である必要があります。

`hardware option` *option* `does not require a value`
options 文字列で指定されるオプションは、単純なオプションである必要があります。

`attachment point` *ap_id* `is not configured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、`CFGA_CMD _UNCONFIGURE` を行う *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `is not unconfigured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、未構成の占有装置を必要とする *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `condition not satisfactory.`
操作の妨げとなる条件を持つ接続点に対して、*config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `in condition` *condition* `cannot be used`
ハードウェアに依存する検査に通らない状態の接続点に対して、強制指定された *config_change_state* 操作が適用されました。

属性

以下の属性については、`attributes`(5) を参照してください。

属性の種類	属性の値
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

関連項目

`cfgadm`(1M), `devinfo`(1M), `dlopen`(3DL), `dlsym`(3DL), `free`(3C), `getsubopt` (3C) , `malloc`(3C), `qsort`(3C), `setlocale`(3C), `strcmp`(3C), `libcfgadm`(3LIB), `attributes`(5)

注意事項

このライブラリを使用するアプリケーションは、実際の実装環境が、外部の `errno` 変数の内容を変更するシステムサービスやファイル記述リソースを使用する可能性があることを考慮したものである必要があります。

以下のコードは、config_*() が CFGA_OK 以外の値を返した場合のエラー処理です。

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

システム構成管理で使用することができる機能に関する詳細は、ハードウェア固有のマニュアルを参照してください。

名前

config_admin, config_change_state, config_private_func, config_test, config_stat, config_list, config_list_ext, config_ap_id_cmp, config_unload_libs, config_strerror – 構成の管理

形式

```
cc [ flag ] file -lcfgadm  [ library... ]
#include <config_admin.h>

cfga_err_t config_change_state(cfga_cmd_t state_change_cmd, int
     num_ap_ids, char * const *ap_ids, const char *options, struct cfga
     _confirm *confp, struct cfga_msg *msgp, char **errstring, cfga
     _flags_t flags);

cfga_err_t config_private_func(const char *function, int num_ap_ids,
     char * const *ap_ids, const char *options, struct cfga_confirm
     *confp, struct cfga_msg *msgp, char **errstring, cfga_flags_t flags)
     ;

cfga_err_t config_test(int num_ap_ids, char * const *ap_ids, const
     char *options, struct cfga_msg *msgp, char **errstring, cfga_flags
     _t flags);

cfga_err_t config_list_ext(int num_ap_ids, char * const *ap_ids,
     struct cfga_list_data **ap_id_list, int *nlist, const char *options,
     const char *listops, char **errstring, cfga_flags_t flags);

int config_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
     ap_id2);

void config_unload_libs();

const char *config_strerror(cfga_err_t cfgerrnum);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t config_stat(int num_ap_ids, char * const *ap_ids, struct
     cfga_stat_data *buf, const char *options, char **errstring);

cfga_err_t config_list(struct cfga_stat_data **ap_id_list, int *nlist,
     const char *options, char **errstring);
```

ハードウェアに依存するライブラリの形式

config_admin ライブラリは、動的再構成 (DR: Dynamic Reconfiguration) のための汎用的なインタフェースです。DR に対応している各ハードウェアは、ここに一覧されているエントリポイントを含むハードウェア固有の組み込みライブラリを提供する必要があります。この汎用ライブラリは、操作を実現するために、適切なライブラリを見つけ、接続します。ここにあるインタフェースは、汎用ライブラリのユーザーからは隠されています。ただし、ハードウェア固有の組み込みライブラリを作成するには、これらのインタフェースについて知っておく必要があります。

```
cfga_err_t cfga_change_state(cfga_cmd_t state_change_cmd, const char
     *ap_id, const char *options, struct cfga_confirm *confp, struct
     cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_private_func(const char *function, const char *ap_id,
     const char *options, struct cfga_confirm *confp, struct cfga_msg
```

```
    *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_test(const char *ap_id, const char *options, struct
    cfga_msg *msgp, char **errstring, cfga_flags_t flags);

cfga_err_t cfga_list_ext(const char *ap_id, struct cfga_list_data
    **ap_id_list, int *nlist, const char *options, const char *listopts,
    char **errstring, cfga_flags_t flags);

cfga_err_t cfga_help(struct cfga_msg *msgp, const char *options,
    cfga_flags_t flags);

int cfga_ap_id_cmp(const cfga_ap_id_t ap_id1, const cfga_ap_id_t
    ap_id2);
```

推奨されないインタフェース

次のインタフェースは推奨されていないため、できるだけ使用しないでください。

```
cfga_err_t cfga_stat(const char *ap_id, struct cfga_stat_data *buf,
    const char *options, char **errstring);

cfga_err_t cfga_list(const char *ap_id, struct cfga_stat_data *
    *ap_id_list, int *nlist, const char *options, char **errstring);
```

機能説明

config_*() 関数は、ハードウェア固有のシステム構成管理機能に対して、ハードウェアに依存しないインタフェースを提供します。 cfga_*() 関数は、構成管理機能をハードウェア固有の方法で処理するために動的に読み込まれた、ハードウェア固有のライブラリによって提供されます。

libcfgadm ライブラリは、cfgadm(1M) コマンドのサービスを提供するために使用されます。libcfgadm ハードウェア固有のライブラリは /usr/platform/${machine}/lib/cfgadm、/usr/platform/${arch}/lib/cfgadm、/usr/lib/cfgadm にあります。ハードウェア固有のライブラリ名は、接続点を特定するデバイスツリーノードのドライバ名またはクラス名から派生します。

config_change_state() 関数は、システム構成の状態を変更する操作を行います。*state_change_cmd* 引数は、CFGA_CMD_INSERT、CFGA_CMD_REMOVE、CFGA_CMD_DISCONNECT、CFGA_CMD_CONNECT、CFGA_CMD_CONFIGURE、CFGA_CMD_UNCONFIGURE、のいずれかになります。 *state_change_cmd* CFGA_CMD_INSERT は、占有装置の手動による追加の準備をしたり、自動追加を有効にするために使用します。 *state_change_cmd* CFGA_CMD_REMOVE は、占有装置の手動による削除の準備をしたり、自動削除を有効にするために使用します。 *state_change_cmd* CFGA_CMD_DISCONNECT は、受容体に接続されている占有装置との通常の通信を無効にするために使用します。 *state_change_cmd* CFGA_CMD_CONNECTは、受容体に接続されている占有装置との通常の通信を有効にするために使用します。 *state_change_cmd* CFGA_CMD_CONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域に追加し、システムで使用できるようにします。*state_change_cmd* CFGA_CMD_UNCONFIGURE は、占有装置に含まれているか接続されているハードウェア資源を Solaris の領域から削除し、システムで使用できないようにします。

flags 引数には、定義済みのフラグである CFGA_FLAG_FORCE とCFGA_FLAG_VERBOSE のいずれかまたは両方を指定することができます。CFGA_FLAG_FORCE フラグを指定すると、特定の安全検査が無効になります。たとえば、このフラグは、failed 状態の占有装置が構成されることを許しませんが、failing 状態の占有装置が構成されることは許す場合があります。強制を受け付けるかどうかは、ハードウェアに依存します。CFGA_FLAG_VERBOSE フラグを指定すると、操作に関係するハードウェア固有の詳細情報が cfga_msg の方法で出力されます。

config_private_func() 関数は、専用ハードウェア固有の関数を起動します。

config_test() 関数は、特定の接続点の検査を開始するために使用します。

num_ap_ids 引数は、*ap_ids* 配列内の *ap_id* の数を特定します。 *ap_ids* 引数は、*ap_id* による配列を示します。

ap_id 引数は、単独の *ap_id* を示します。

function と *options* の文字列は、getsubopt(3C) 構文の規約に従い、ハードウェア固有の関数やオプション情報を提供するために使用します。ハードウェアに依存しない汎用的な関数やオプションは定義されていません。

confp によって参照される cfga_confirm 構造体は、要求された操作がサービスに対する顕著な割り込みを必要とする場合に、継続する許可を得るためのコールバックインタフェースを提供します。 cfga_confirm 構造体には以下のメンバーが含まれます。

```
int  (*confirm)(void *appdata_ptr, const char *message);
void *appdata_ptr;
```

confirm() 関数は、汎用ポインタの *appdata_ptr* と、確認を必要とするものの詳細を示すメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ *appdata_ptr* は、cfga_confirm 構造体のメンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で confirm 関数の呼び出しをconfig_* の呼び出しに関連付けるために使用されます。confirm 関数は、操作の継続が認められた場合は 1 を返し、認められなかった場合は 0 を返します。

msgp によって参照される cfga_msg 構造体は、ハードウェア固有のライブラリからのメッセージを出力するためのコールバックインタフェースを提供します。CFGA_FLAG_VERBOSE フラグが存在することによって、これらのメッセージは情報メッセージにもなりますが、このフラグが存在しない場合は、エラーメッセージに限定されます。cfga_msg 構造体には、以下のメンバーが含まれます。

```
void (*message_routine)(void *appdata_ptr, const char *message);
void    *appdata_ptr;
```

message_routine() 関数は、汎用ポインタの *appdata_ptr* とメッセージの 2 つの引数を指定して呼び出されます。汎用ポインタ*appdata_ptr* は、cfga_confirm 構造体メンバーの appdata_ptr の中に渡される値に設定され、グラフィカルユーザーインタフェースの中で message_routine() 関数の呼び出しを config_*() の呼び出しに関連付けるために使用されます。このメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

いくつかの一般的なエラーについて、ハードウェア固有のエラーメッセージが返されることがあります。エラーメッセージの文字列(終わりの NULL 文字を含む) の記憶領域は、malloc(3C) と、*errstring* を通じて返される、この記憶領域に対するポインタを使用して config_* 関数によって割り当てられます。*errstring* がNULL の場合は、エラーメッセージは作成または返されることはありません。*errstring* が NULL ではなく、エラーメッセージが作成されない場合は、*errstring* によって参照されるポインタが NULLに設定されます。free(3C) を使用して、返された記憶領域の割り当ての解除は、config_*() を呼び出している関数が行います。このエラーメッセージは、LC_MESSAGES ロケールカテゴリで指定された言語である必要があります (setlocale(3C) を参照)。

config_list_ext() 関数は、リスト出力用インタフェースです。最初の 2 つの引数を使用して *ap_id*s にリストが指定されていると、指定されている接続点ごとに cfga_list_data_t 構造体の配列を返します。最初の 2 つの引数がそれぞれ 0 と NULL の場合は、デバイスツリーの接続点をすべて出力します。また、フラグの引数を使用して CFGA_FLAG_LIST_ALL フラグを渡した場合は、接続点を動的に展開して動的な接続点を出力する必要があります。返した配列の stat 構造の記憶領域は、config_list_ext() 関数がmalloc(3C) を使用して割り当てます。この記憶領域は、config_list_ext() の呼び出し元が free(3C) を使用して解放しなければなりません。

cfga_list_data 構造体には、以下のメンバーが含まれます。

```
cfga_log_ext_t     ap_log_id;        /* Attachment point logical id */
cfga_phys_ext_t    ap_phys_id;       /* Attachment point physical id */
cfga_class_t       ap_class;         /* Attachment point class */
cfga_stat_t        ap_r_state;       /* Receptacle state */
cfga_stat_t        ap_o_state;       /* Occupant state */
cfga_cond_t        ap_cond;          /* Attachment point condition */
cfga_busy_t        ap_busy;          /* Busy indicator */
time_t             ap_status_time;   /* Attachment point last change*/
cfga_info_t        ap_info;          /* Miscellaneous information */
cfga_type_t        ap_type;          /* Occupant type */
```

種類は以下のように定義されます。

```
typedef char cfga_log_ext_t[CFGA_LOG_EXT_LEN];
typedef char cfga_phys_ext_t[CFGA_PHYS_EXT_LEN];
typedef char cfga_class_t[CFGA_CLASS_LEN];
typedef char cfga_info_t[CFGA_INFO_LEN];
typedef char cfga_type_t[CFGA_TYPE_LEN];
typedef enum cfga_cond_t;
typedef enum cfga_stat_t;
typedef enum cfga_busy_t;
typedef int cfga_flags_t;
```

config_list_ext() の *listopts* 引数は、getsubopt (3C) 構文に準拠しており、サブオプションのリストを渡すときに使用します。現在、サポートされているサブオプションは、*class*=class_name だけです。このリストオプションは、class_name クラスの接続点だけを出力します。

cfga_list_ext() の *listopts* 引数は、今後の使用のために予約されこの引数がNULL の場合は、ハードウェア固有のライブラリはこの引数を無視する必要がありま す。*listopts* 引数がNULL 以外のときに、ハードウェア固有のライブラリでサポートされていない場合は、適切なエラーコードを返す必要があります。

ap_log_id と ap_phys_id メンバーは、接続点に対するハードウェア固有の論理名と物理名を示します。ap_busy メンバーは、状態 (state) や条件 (condition) の変更が生じるような動作があることを示します。ap_status_time メンバーは、接続点のap_r_state、ap_o_state、ap_cond フィールドのいずれかが最後に変わった時間を示します。ap_info メンバーは、接続点に関する追加情報を得るためにハードウェア固有のコードで使用することができます。ap_class メンバーには、接続点の接続点クラスが含まれます (存在する場合)。ap_class メンバーは、汎用ライブラリから組み込まれます。ap_log_id および ap_phys_id メンバーがハードウェア固有のライブラリから組み込まれなかった場合は、汎用ライブラリから汎用形式で組み込まれます。残りのメンバーは、対応するハードウェア固有のライブラリから組み込まれます。

ap_log_id、ap_phys_id、ap_info、ap_class、および ap_type メンバーは、固定長文字列です。実際の文字列がメンバーのサイズよりも短い場合には、この文字列は空文字で終了します。そのため、プログラムは、文字列が空文字で終了していることを前提としてはなりません。これらのフィールドを印刷する場合は、以下の形式を使用してください。

printf("%.*s", sizeof(p->ap_log_id), p->ap_log_id);

config_stat()、config_list()、cfga_stat()、cfga_list() 関数およびcfga_stat_data データ構造体は、推奨されていないインタフェースで、下位互換性を維持するためにだけ提供されています。これらのインタフェースはできるだけ使用しないでください。

config_ap_id_cmp 関数は、2 つの *ap_id* に関する、ハードウェアに依存する比較を行い、strcmp(3C) の形式に従って、等しい、小さい、大きい、の結果を返します。各引数 は、cfga_ap_id_t とNULL で終了する文字列のいずれかになります。この関数は、qsort(3C) などによって *ap_id* の一覧をソートする場合や、config_list 関数の呼び出しの結果からエントリを選択する場合に使用することができます。

config_unload_libs 関数は、これまで読み込んだハードウェア固有のライブラリすべてのリンクを解除します。

config_strerror 関数を使用して、エラーメッセージの文字列に対してエラーの戻り値を割り当てることができます。「戻り値」を参照してください。返された文字列は、上書きされてはなりません。 *cfgerrnum* が範囲外にある場合に、config _strerror は NULL を返します。

cfga_help 関数を使用して、ハードウェア固有のライブラリに対して現地仕様のヘルプメッセージを表示するように要求することができます。

戻り値

以下は、config_*() および cfga_*() 関数によって返される戻り値の一覧です。戻り値が CFGA_OK ではない場合は、追加のエラー情報が *errstring* を通して返されます。詳細は、「機能説明」を参照してください。

CFGA_BUSY	システム構成管理システムの 1 要素が使用中であるために、コマンドは完了しませんでした。
CFGA_ATTR_INVAL	指定された属性の接続点は存在しません。
CFGA_ERROR	要求された操作の処理中にエラーが発生しました。このエラーコードは、ハードウェア固有のコードによるコマンド引数の妥当性検査を含みます。
CFGA_INSUFFICIENT_CONDITION	接続点の条件によって、操作が失敗しました。
CFGA_INVAL	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_LIB_ERROR	記憶域やファイル記述子などの処理用資源の獲得の失敗など、ライブラリで手続き上のエラーが発生しました。
CFGA_NACK	*confp*->confirm 関数からの否定応答を受けたために、コマンドは完了しませんでした。
CFGA_NO_LIB	指定された *ap_id* を使用して、ハードウェア固有のライブラリが見つかりませんでした。
CFGA_NOTSUPP	要求されたシステム設定管理操作は、指定された接続点に対しては使用することができません。
CFGA_OK	要求通りにコマンドが完了しました。
CFGA_OPNOTSUPP	この接続点は、システム構成管理操作に対応していません。
CFGA_PRIV	呼び出し元は要求された処理の特権を所有していません。たとえば、構成管理がデバイスドライバを介して実行される場合は、デバイスノードのアクセス権によってアクセスが制御されます。
CFGA_SYSTEM_BUSY	コマンドがサービスの中断を要求したが、システムの一部が休止できなかったために完了しませんでした。

エラー

システム構成管理機能から返されるエラーの多くは、ハードウェア固有のものです。*errstring* を通して返される文字列には以下があります。

`attachment point` *ap_id* `not known`
エラーメッセージで説明されている接続点は存在しません。

`unknown hardware option` *option* `for` *operation*
未知のオプションが *options* 文字列の中で検出されました。

`hardware option` *option*option `requires a value`
options 文字列で指定されるオプションは、*option=value* の形式である必要があります。

`listing option` *list_option* `requires a value`
listopts 文字列で指定されるオプションは、*option*=`value` 形式である必要があります。

`hardware option` *option* `does not require a value`
options 文字列で指定されるオプションは、単純なオプションである必要があります。

`attachment point` *ap_id* `is not configured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、`CFGA_CMD_UNCONFIGURE` を行う *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `is not unconfigured`
占有装置が `CFGA_STAT_CONFIGURED` 状態にない接続点に対して、未構成の占有装置を必要とする *config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `condition not satisfactory.`
操作の妨げとなる条件を持つ接続点に対して、*config_change_state* コマンドが適用されました。

`attachment point` *ap_id* `in condition` *condition* `cannot be used`
ハードウェアに依存する検査に通らない状態の接続点に対して、強制指定された *config_change_state* 操作が適用されました。

属性

以下の属性については、`attributes`(5) を参照してください。

属性の種類	属性の値
Availability	SUNWcsu, SUNWkvm
MT-Level	Safe

関連項目

`cfgadm`(1M), `devinfo`(1M), `dlopen`(3DL), `dlsym`(3DL), `free`(3C), `getsubopt` (3C), `malloc`(3C), `qsort`(3C), `setlocale`(3C), `strcmp`(3C), `libcfgadm`(3LIB), `attributes`(5)

注意事項

このライブラリを使用するアプリケーションは、実際の実装環境が、外部の `errno` 変数の内容を変更するシステムサービスやファイル記述リソースを使用する可能性があることを考慮したものである必要があります。

以下のコードは、config_*() が CFGA_OK 以外の値を返した場合のエラー処理です。

```
void
emit_error(int cfgerrnum, char *estrp)
{
    const char *ep;
    ep = config_strerror(cfgerrnum);
    if (ep == NULL)
        ep = gettext("configuration administration unknown error");
    if (estrp != NULL && *estrp != '\0') {
        (void) fprintf(stderr, "%s: %s\n", ep, estrp);
    } else {
        (void) fprintf(stderr, "%s\n", ep);
    }
    if (estrp != NULL)
        free((void *)estrp);
}
```

システム構成管理で使用することができる機能に関する詳細は、ハードウェア固有のマニュアルを参照してください。

名前 | isencrypt – 文字のバッファが暗号化されているかの判別

形式 | **cc** [*flag*...] *file*... -lgen [*library*...]

```
#include<libgen.h>

int isencrypt(const char *fbuf, size_t ninbuf);
```

機能説明 | isencrypt() は、ヒューリスティックを使用して文字のバッファが 暗号化されているかどうかを判別します。この関数には 2 つの引数が必要になります。1 つは文字配列へのポインタで、もう 1 つはバッファの中の文字数です。

isencrypt() は、最初のブロックの文字すべてが ASCII 文字ならば、そのファイルは暗号化されていないと見なします。最初の *ninbuf* 文字に非 ASCII 文字があると、isencrypt() は setlocale() LC_CTYPE カテゴリが C または ascii に設定されている場合に、バッファが暗号化されていると見なします。

LC_CTYPE カテゴリが C または ascii 以外の値に設定されていると、isencrypt() はヒューリスティックを組み合わせて、バッファが暗号化されているかどうかを判別します。*ninbuf* に少なくとも 64 文字あれば、カイ二乗分布を用いてバッファのバイトが均一に分布してるかを判別します。 バイトが均一に分布している場合には、isencrypt() は、バッファが暗号化されていると見なします。バッファに格納される文字数が 64 文字よりも少なければ、NULL 文字および復帰改行による終端について検査を行なって、バッファが暗号化されているかを判別します。

戻り値 | バッファが暗号化されている場合は 1 を返し、それ以外の場合には 0 を返します。

属性 | 次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目 | setlocale(3C), attributes(5)

注意事項 | マルチスレッドアプリケーションを コンパイルする際に、 _REENTRANT フラグをコンパイル行に定義する必要があります。このフラグはマルチスレッドアプリケーションの際にのみ使います。

名前 | regexpr, compile, step, advance – 正規表現のコンパイルおよび一致ルーチン

形式 | **cc** [*flag*...] [*file*...] -lgen [*library*...]

```
#include <regexpr.h>

char *compile(char *instring, char *expbuf, const char *endbuf);

int step(const char *string, const char *expbuf);

int advance(const char *string, const char *expbuf);

extern char *loc1, loc2, locs;

extern int nbra, regerrno, reglength;

extern char *braslist[], *braelist[];
```

機能説明 | これらのルーチンは、正規表現をコンパイルして、コンパイルした表現を行と一致する場合に使用します。コンパイルされた正規表現は、ed(1) によって使用される形式になります。

instring パラメータは、正規表現を表す NULL で終了する文字列です。

expbuf パラメータは、コンパイルされた正規表現が格納される場所を指しています。*expbuf* が NULL であれば、compile() は malloc(3C) を使用してコンパイルした正規表現にメモリー空間を割り当てます。エラーが発生すると、このメモリー空間は解放されます。コンパイルした正規表現が必要なくなったときに不要なメモリー空間を解放するのは、ユーザーの役割となります。

endbuf パラメータはコンパイルされた正規表現が格納される最高位アドレスより 1 つ上のアドレスです。この引数は *expbuf* が NULL であると無視されます。コンパイルした表現が (*endbuf–expbuf*) バイトに収まらない場合は、compile() は NULL を返し、regerrno (以下を参照) は 50 に設定されます。

string パラメータは、一致について検査が行われる文字列へのポインタです。この文字列は NULL で終わる必要があります。

expbuf パラメータは、compile() 関数を呼び出して得られるコンパイルした正規表現です。

step() 関数は、指定した文字列が正規表現と一致していると 0 以外を返し、一致していないと 0 を返します。一致していると、step() を呼び出す際の副作用として 2 つの外部文字ポインタが設定されます。step() の変数セットは loc1 と loc2 です。loc1 は、正規表現と一致した最初の文字へのポインタです。変数 loc2 は、正規表現に一致する最後の文字の後に来る文字を指しています。したがって、正規表現が行全体と一致していると、loc1 は *string* の最初の文字を指し、loc2 は *string* の最後の NULL を指しています。

step() の目的は、一致を見つけるか、*string* が最後になるまで、*string* 引数を読みながら処理することです。正規表現が ^ から始まっている場合、step() は文字列の先頭でだけ正規表現を一致させようとします。

advance() 関数は step() と同様ですが、advance() は loc2 だけを設定し、つねに文字列の先頭で正規表現を一致させます。

あるユーザーが同一の文字列内で連続する一致を検索している場合には、locs は loc2 と同じに設定し、step() は loc2 と同じ *string* を用いて呼び出す必要があります。s/y*//g のようなグローバル置換が永久にループしないように、またデフォルトでは NULL になるように、locs は ed コマンドおよび sed のようなコマンドで使用されます。

nbra 外部変数は、コンパイルされた正規表現における副表現数を決定する場合に使用します。braslist および braelist は、一致した文字列内の nbra 副表現の先頭と終わりを指している文字列ポインタの配列です。たとえば、文字列 sabcdefg および正規表現 \(abcdef\) を用いて step() または advance() を呼び出すと、braslist[0] は a を指し、braelist[0] は g を指しています。これらの配列は、副表現の *n* 表記法を含む代替置換パターンについて ed および sed のようなコマンドで使用されます。

文字列が正規表現と一致するかどうかだけ検査している場合は、regerrno、nbra、loc1、loc2、locs、braelist、および braslist の各外部変数を使用する必要はないことに注意してください。

使用例

例 **1** 以下の例は grep からの正規表現コードと同じです。

```
#include <regexpr.h>
. . .
if(compile(*argv, (char *)0, (char *)0) == (char *)0)
    regerr(regerrno);
. . .
if (step(linebuf, expbuf))
    succeed( );
```

戻り値

compile() は、成功すると、その値が *expbuf* で異なる非 NULL ポインタを返します。*expbuf* が非 NULL の場合は、compile() はコンパイルした正規表現の最終バイトの後のバイトへのポインタを返します。コンパイルした正規表現の長さは reglength に格納されます。それ以外の場合は、compile() は malloc によって割り当てられたメモリー空間へのポインタを返します。

step() 関数および advance() 関数は、指定した文字列が正規表現に一致する場合には 0 以外を返し、一致しない場合には 0 を返します。

エラー

正規表現のコンパイル中にエラーが検出されると、compile() から NULL ポインタが返され、regerrno は以下に示した 0 でないエラー番号の 1 つに設定されます。

エラー	説明
11	範囲が広すぎます。
16	番号が間違っています。

エラー	説明
25	"\digit" が範囲外です。
36	区切り文字が指定されていないか、間違っています。
41	検索文字列が記憶されていません。
42	\(と \) が合っていません。
43	\(が多すぎます。
44	\{ と \} に 3 つ以上の数が指定されています。
45	\ の後に } がありません。
46	\{ と \} において最初の数字が 2 番目の数を超えています。
49	[と] が合っていません。
50	正規表現がオーバフローしました。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目

ed(1), grep(1), sed(1), malloc(3C), attributes(5), regexp(5)

注意事項

マルチスレッドアプリケーションを コンパイルする際に、_REENTRANT フラグはコンパイル行に定義しなければなりません。このフラグはマルチスレッドアプリケーションの際にのみ使います。

名前 | regexpr, compile, step, advance – 正規表現のコンパイルおよび一致ルーチン

形式 | **cc** [*flag*...] [*file*...] -lgen [*library*...]

```
#include <regexpr.h>

char *compile(char *instring, char *expbuf, const char *endbuf);

int step(const char *string, const char *expbuf);

int advance(const char *string, const char *expbuf);

extern char *loc1, loc2, locs;

extern int nbra, regerrno, reglength;

extern char *braslist[], *braelist[];
```

機能説明 | これらのルーチンは、正規表現をコンパイルして、コンパイルした表現を行と一致する場合に使用します。コンパイルされた正規表現は、ed(1) によって使用される形式になります。

instring パラメータは、正規表現を表す NULL で終了する文字列です。

expbuf パラメータは、コンパイルされた正規表現が格納される場所を指しています。*expbuf* が NULL であれば、compile() は malloc(3C) を使用してコンパイルした正規表現にメモリー空間を割り当てます。エラーが発生すると、このメモリー空間は解放されます。コンパイルした正規表現が必要なくなったときに不要なメモリー空間を解放するのは、ユーザーの役割となります。

endbuf パラメータはコンパイルされた正規表現が格納される最高位アドレスより 1 つ上のアドレスです。この引数は *expbuf* が NULL であると無視されます。コンパイルした表現が (*endbuf–expbuf*) バイトに収まらない場合は、compile() は NULL を返し、regerrno (以下を参照) は 50 に設定されます。

string パラメータは、一致について検査が行われる文字列へのポインタです。この文字列は NULL で終わる必要があります。

expbuf パラメータは、compile() 関数を呼び出して得られるコンパイルした正規表現です。

step() 関数は、指定した文字列が正規表現と一致していると 0 以外を返し、一致していないと 0 を返します。一致していると、step() を呼び出す際の副作用として 2 つの外部文字ポインタが設定されます。step() の変数セットは loc1 と loc2 です。loc1 は、正規表現と一致した最初の文字へのポインタです。変数 loc2 は、正規表現に一致する最後の文字の後に来る文字を指しています。したがって、正規表現が行全体と一致していると、loc1 は *string* の最初の文字を指し、loc2 は *string* の最後の NULL を指しています。

step() の目的は、一致を見つけるか、*string* が最後になるまで、*string* 引数を読みながら処理することです。正規表現が ^ から始まっている場合、step() は文字列の先頭でだけ正規表現を一致させようとします。

advance() 関数は step() と同様ですが、advance() は loc2 だけを設定し、つねに文字列の先頭で正規表現を一致させます。

あるユーザーが同一の文字列内で連続する一致を検索している場合には、locs は loc2 と同じに設定し、step() は loc2 と同じ *string* を用いて呼び出す必要があります。s/y*//g のようなグローバル置換が永久にループしないように、またデフォルトでは NULL になるように、locs は ed コマンドおよび sed のようなコマンドで使用されます。

nbra 外部変数は、コンパイルされた正規表現における副表現数を決定する場合に使用します。braslist および braelist は、一致した文字列内の nbra 副表現の先頭と終わりを指している文字列ポインタの配列です。たとえば、文字列 sabcdefg および正規表現 \(abcdef\) を用いて step() または advance() を呼び出すと、braslist[0] は a を指し、braelist[0] は g を指しています。これらの配列は、副表現の *n* 表記法を含む代替置換パターンについて ed および sed のようなコマンドで使用されます。

文字列が正規表現と一致するかどうかだけ検査している場合は、regerrno、nbra、loc1、loc2、locs、braelist、および braslist の各外部変数を使用する必要はないことに注意してください。

使用例

例 **1** 以下の例は grep からの正規表現コードと同じです。

```
#include <regexpr.h>
. . .
if(compile(*argv, (char *)0, (char *)0) == (char *)0)
    regerr(regerrno);
. . .
if (step(linebuf, expbuf))
    succeed( );
```

戻り値

compile() は、成功すると、その値が *expbuf* で異なる非 NULL ポインタを返します。*expbuf* が非 NULL の場合は、compile() はコンパイルした正規表現の最終バイトの後のバイトへのポインタを返します。コンパイルした正規表現の長さは reglength に格納されます。それ以外の場合は、compile() は malloc によって割り当てられたメモリー空間へのポインタを返します。

step() 関数および advance() 関数は、指定した文字列が正規表現に一致する場合には 0 以外を返し、一致しない場合には 0 を返します。

エラー

正規表現のコンパイル中にエラーが検出されると、compile() から NULL ポインタが返され、regerrno は以下に示した 0 でないエラー番号の 1 つに設定されます。

エラー	説明
11	範囲が広すぎます。
16	番号が間違っています。

エラー	説明
25	"\digit" が範囲外です。
36	区切り文字が指定されていないか、間違っています。
41	検索文字列が記憶されていません。
42	\(と \) が合っていません。
43	\(が多すぎます。
44	\{ と \} に 3 つ以上の数が指定されています。
45	\ の後に } がありません。
46	\{ と \} において最初の数字が 2 番目の数を超えています。
49	[と] が合っていません。
50	正規表現がオーバフローしました。

属性

次の属性については attributes(5) のマニュアルページを参照してください。

属性タイプ	属性値
MT レベル	MT-Safe

関連項目

ed(1), grep(1), sed(1), malloc(3C), attributes(5), regexp(5)

注意事項

マルチスレッドアプリケーションを コンパイルする際に、_REENTRANT フラグはコンパイル行に定義しなければなりません。このフラグはマルチスレッドアプリケーションの際にのみ使います。