

# Dcs<sub>(Ver. 3.00)</sub> API

## リファレンス・マニュアル



# 目次

## Dcs API

概要.....	5
get_version()	
文字コード判定／変換テーブル.....	5
マルチスレッド対応.....	5
文字コード判定／変換関数.....	6
dcs_judges_buffer() dcs_judges_stream() dcs_judges_file()	
dcs_convert_buffer() dcs_convert_stream() dcs_convert_file()	

## Dcs API 関数使用例

文字コード判定プログラム.....	7
create_dcs() dcs_judges_buffer() dcs_typedto_name() release_dcs()	
文字コード変換プログラム.....	8
dcs_typedto_name() create_dcs() dcs_convert_buffer() dcs_judges_buffer()	
dcs_typedto_name() release_dcs()	

## Dcs API 関数リファレンス

1. Dcs 情報生成／開放.....	11
create_dcs() release_dcs()	
2. 改行コード・タイプの取得／設定.....	11
get_ltype()* set_ltype()*	
3. 日本語 UTF8-MAC 1 文字変換の判定／設定及び、UTF8-MAC コード判定.....	12
dcs_is_utfmac_to()* dcs_set_utfmac_to()* dcs_utfmac_exists()*	
4. 出力コードの半角カナから全角変換の判定／設定.....	13
dcs_is_hankana_to()* dcs_set_hankana_to()*	
5. 不正文字出力抑止の判定／設定.....	14
dcs_is_suppress()* dcs_set_suppress()*	
6. コード判定（バッファ／ストリーム／ファイル） .....	14
dcs_judges_buffer() dcs_judges_stream() dcs_judges_file()	
7. コード変換（バッファ／ストリーム／ファイル） .....	16
dcs_convert_buffer() dcs_convert_stream() dcs_convert_file()	
8. ファイル・アクセス時のエラー番号／メッセージ取得.....	18
dcs_get_errno()* dcs_get_strerror()	
9. 処理ステータスのメッセージ取得.....	18
dcs_statusto_message()	

10. 変換不能文字情報取得（文字数、先頭／末尾文字、先頭／末尾位置） .....	19
get_no_char_count()* get_first_no_char()* get_last_no_char()* get_first_no_char_offset()* get_last_no_char_offset()* .....	
11. Dcs API バージョン取得.....	20
dcs_get_version() .....	
12. コード・タイプ／コード名取得.....	20
dcs_typedto_name() dcs_nameto_type() dcs_multi_type_names() dcs_multi_type_names() dcs_all_type_names() dcs_all_type_names_length() .....	
13. B O Mの追加／削除判定.....	21
dcs_cmp_bom() .....	
14. 変換不能コードの判定.....	21
is_jis_no_char()* is_jis_no_short()* is_sjis_no_char()* is_sjis_no_short()* is_euc_no_char()* is_euc_no_short()* is_unibe_no_char()* is_unibe_no_short()* is_unile_no_char()* is_unile_no_short()* .....	
15. 半角カナ判定.....	22
is_jis_kana()* is_sjis_kana()* is_unibe_kana()* .....	
16. 半角カナ変換.....	22
jis_kana_to_sjis()* jis_kana_to_unibe()* sjis_kana_to_jis()* sjis_kana_to_unibe()* unibe_kana_to_jis()* unibe_kana_to_sjis()* .....	
17. 全角仮名判定.....	22
is_jis_zenkana()* is_sjis_zenkana()* is_euc_zenkana()* is_unibe_zenkana()* .....	
18. UTF-16 間、UTF-32 間変換.....	23
short_to_char()* char_to_short()* unibe_char_to_short()* unile_char_to_short()* unibe_to_unile_char()* unile_to_unibe_char()* unibe_to_unile_short()* unile_to_unibe_short()* .....	
19. UTF-16 から UTF-8 への変換.....	23
unibe_utf8_char()* unibe_to_utf8_short()* unile_to_utf8_char()* unile_to_utf8_short()* .....	
20. UTF-8(4byte)から UTF-16BE への変換.....	24
utf8_to_unibe_4bytes()* .....	
21. サロゲート・ペアからの変換.....	24
unibe_surrogate_to_utf8()* unibe_surrogate_to_cd()* .....	

## 付録

Dcs API 定義.....	25
-----------------	----

# Dcs API

## 概要

**Dcs(DetectCharacterSets)日本語文字コード判定／変換 API**（以後は単に DcsAPI と記す）は C 言語で作成され、C 言語、C++ で使用するためのものである。DcsAPI は **dcs.h** で定義されており、アプリケーションはこのヘッダー・ファイルをインクルードして Dcs のすべての関数群を使用することができる。

Dcs API で使用する Dcs 情報(Dcs \*)はそのほとんどの変数の名前がアンダーバー '\_' で始まるが、これらには使用する関数が用意されており、アプリケーションから直接参照してはならない。参照できるのは関数の処理ステータス sts とバージョン番号 version の 2 つだけである。ただし、処理ステータス sts は呼び出した関数の戻り値と同じ値であり、後から参照するためのものである。バージョン番号 version は、別にスタティックな値として Dcs 情報を生成せずにそれを確認できる関数 `get_version()` がある。

## マルチスレッド対応

バージョン 3.00 からはスレッドごとに Dcs 情報を生成することにより他のスレッドに独立してすべての API を利用することができる。

## 文字コード判定／変換テーブル

コードの判定と変換には、ユニコード内の変換を除きテーブルが使用される。ユニコードの判定は独自の定義テーブルを持つが、他のコードではユニコードに変換するテーブルが定義テーブルとなる。変換はユニコードを基準としており、たとえば JIS から EUC への変換は JIS => UTF16-BE => EUC へと 2 段階の変換が内部的に行なわれる。ユニコードから JIS への変換にはシフト・コード用のテーブル、さらに JIS (ISO-2022-JP-2004) への変換にはサロゲート文字定義テーブルがある。EUC からユニコードへの変換 4 種類のうち 3 つ (CP20932, eucJP-ms, JIS-2004) は 3 バイト・コード用のテーブルがある。これらのコード判定、変換テーブルは、すべて合わせると 36 本になる。ただしバージョン 2.00 からは、これらのテーブルは利便性や性能改善のために内部仕様として扱われ、アプリケーションから操作することはない。

## 文字コード判定／変換関数

Dcs API の主な機能は文字コードの判定と変換であり、それらはバージョン 2.00 において、先に説明したテーブル仕様の変更にもとない、判定と変換のどちらも入力データの形式（バッファ、ストリーム、ファイル）による 3 つ、計 6 つの関数にまとめられた。以下にそれらを示す。

### 文字コード判定関数

#### 1. 入力バッファ用

```
int dcs_judges_ buffer(Dcs *dcs, int *atype, int *ln_type,  
    char *buffer, size_t size, int stype);
```

#### 2. 入力ストリーム用

```
int dcs_judges_stream(Dcs *dcs, int *atype, int *ln_type, FILE *sfp, int stype);
```

#### 3. 入力ファイル用

```
int dcs_judges_file(Dcs *dcs, int *atype, int *ln_type, const char *spath, int stype);
```

### 文字コード変換関数

#### 4. 入力バッファ用

```
int dcs_convert_buffer(Dcs *dcs, const char *buffer, size_t size, int stype, int dtype,  
    bool is_forced, int (*observer)(int sts, void *user_info),  
    int (*writer)(const char *buf, size_t size, void *user_info), void *user_info);
```

#### 5. 入力ストリーム用

```
int dcs_convert_stream(Dcs *dcs, FILE *sfp, int stype, int dtype,  
    bool is_forced, int (*observer)(int sts, void *user_info),  
    int (*writer)(const char *buf, size_t size, void *user_info), void *user_info);
```

#### 6. 入力ファイル用

```
int dcs_convert_file(Dcs *dcs, const char *spath, int stype, int dtype,  
    bool is_forced, int (*observer)(int sts, void *user_info),  
    int (*writer)(const char *buf, size_t size, void *user_info), void *user_info);
```

# Dcs API 関数使用例

## 文字コード判定プログラム

```

/*
 * 文字コード判定プログラム
 */
#include "dcs.h"

int main() {
    int sts;
    char buffer[] = "みなさん、今日は！\n";
    char str[256];

    /* Dcs 生成 */
    Dcs *dcs = create_dcs();

    if (dcs == NULL) {
        /* Dcs の生成が失敗した時 */
        fprintf(stderr, "create_dcs() error!\n");
        exit(1);
    } else {
        /* Dcs が正常に生成された時 */

        /* コード判定 (バッファ) */
        sts = dcs_judges_buffer(dcs, NULL, NULL,
                               buffer, sizeof(buffer)-1, DCS_UNKNOWN);

        if (sts < DCS_UNKNOWN) {
            dcs_statusto_message(NULL, str, sts, 0);
            printf("(%d: %s)", sts, str);
        } else {
            /* 文字コード名取得 */
            dcs_typedto_name(str, sts);
            printf("%s(%d: %s)", buffer, sts, str);
        }

        /* Dcs 解放 */
        release_dcs(dcs);
    }
    return 0;
}

```

このプログラムを hello とし実行すると、以下の内容で標準出力に表示される。

```

$ hello
みなさん、今日は！
(15: UTF8)

```

解説：

プログラムの初めに `create_dcs()` で Dcs 情報を生成する。返却値が `NULL` ならば異常終了する。次に文字列用のコード判定関数 `dcs_judges_buffer()` を呼び出し、引き数で与えた文字列のコード判定を行なう。さらに取得した数値コード `sts` から `dcs_typedto_name()` により文字コード名を取得する。これらを標準出力に表示して `release_dcs()` により Dcs 情報を開放してプログラムを終了する。`dcs_judges_buffer()` の戻り値は `DCS_UNKNOWN` 以上が正常であり、それより小さい値は異常とし、`dcs_statusto_message()` によりエラー・メッセージを取得しエラー表示する。

## 文字コード変換プログラム

```

1:  /*
2:   * 文字コード変換プログラム
3:   */
4:  #include <stdlib.h>
5:  #include <string.h>
6:  #include "dcs.h"
7:
8:  /* 出力バッファ、ポインター */
9:  char _convert_str[256], *_cptr, *_eptr;
10:
11:  /* dcs_convert_buffer() から呼び出される監視関数 */
12:  /* sts: コード判定結果 */
13:  int h_observer(int sts, void *user_info) {
14:      char str[256];
15:
16:      /* 文字コード名取得 */
17:      dcs_typedto_name(str, sts);
18:      printf("Observed sts %d[%s]\n", sts, str);
19:
20:      /* 処理を継続 */
21:      return DCS_OBS_CONTINUE;
22:  }
23:
24:  /* dcs_convert_buffer() から呼び出されるライター */
25:  int h_writer(const char *buf, size_t size, void *user_info) {
26:      if (_cptr < _eptr) {
27:          /* バッファに余りがある時 */
28:
29:          if (_cptr + size > _eptr) {
30:              /* バッファを超える時 */
31:              size = _eptr - _cptr;
32:          }
33:
34:          /* バッファに出力 */
35:          memcpy(_cptr, buf, size);
36:          _cptr += size;
37:      }
38:      /* 成功ステータスを返す */
39:      return DCS_SUCCESS;
40:  }
41:
42:  int main() {
43:      int sts;
44:      /* "みなさん、今日は！\n" SJIS コード */
45:      char buffer[] =
46:          "\x82\xdd\x82\xc8\x82\xb3\x82\xf1\x81\x41\x8d\xa1\x93\xfa\x82\xcd\x81\x49\x0a";
47:      char str[256];
48:

```



```

49:  /* Dcs 生成 */
50:  Dcs *dcs = create_dcs();
51:
52:  if (dcs == NULL) {
53:  /* Dcs の生成が失敗した時 */
54:
55:      fprintf(stderr, "create_dcs() error!\n");
56:      exit(1);
57:  } else {
58:  /* Dcs が正常に生成された時 */
59:
60:      /* 出力バッファ・ポインターの初期化 */
61:      _cptr = _convert_str;
62:      _eptr = _convert_str + sizeof(_convert_str);
63:
64:      /* バッファによるコード変換(SJIS => UTF-8) */
65:      sts = dcs_convert_buffer(dcs, buffer, sizeof(buffer)-1, DCS_UNKNOWN, DCS_UTF8,
66:          false, h_observer, h_writer, NULL);
67:
68:      if (sts == DCS_SUCCESS) {
69:      /* 変換が成功した時 */
70:
71:          /* 出力バッファをヌル・ターム */
72:          if (_cptr >= _eptr) {
73:          /* バッファ・フルの時 */
74:              *(_eptr - 1) = 0;
75:          } else {
76:              *_cptr = 0;
77:          }
78:
79:          /* 変換後のコード判定 */
80:          sts = dcs_judges_buffer(dcs, NULL, NULL,
81:              _convert_str, _cptr - _convert_str, DCS_UNKNOWN);
82:
83:          /* 文字コード名取得 */
84:          dcs_typedto_name(str, sts);
85:          printf("%s(%d: %s)", _convert_str, sts, str);
86:      } else {
87:          dcs_statusto_message(NULL, str, sts, 0);
88:          printf("(%d: %s)", sts, str);
89:      }
90:
91:      /* Dcs 解放 */
92:      release_dcs(dcs);
93:  }
94:  return 0;
95: }

```

このプログラムを hello2 とし実行すると、以下の内容で標準出力に表示される。

```

$ hello2
Observed sts 4[SJIS]
みなさん、今日は！
(15: UTF8)

```

## 解説：

判定プログラムと同様、初めに `create_dcs()` で Dcs 情報を生成する(50)。返却値が NULL ならば異常終了する(52)。次に広域変数である出力バッファ・ポインタの初期化を行なう(61, 62)。次に文字列用のコード変換関数 `dcs_convert_buffer()` を呼び出し、引き数で与えた文字列のコード変換を行なう(65)。変換が成功したことを確認(68)した後、文字列である出力バッファをヌル・タムする(72~77)。以後は判定プログラムと同様に、変換後のバッファのコード判定関数 `dcs_judges_buffer()` を呼び出し(80)、取得した数値コード `sts` から `dcs_typedto_name()` により文字コード名を取得する(84)。これらを標準出力に表示して(85)、`release_dcs()` により Dcs 情報を開放して(92)プログラムを終了する。`dcs_judges_buffer()` の戻り値は `DCS_SUCCESS` が正常であり、それ以外の値は異常とし、`dcs_statusto_message()` によりエラー・メッセージを取得しエラー表示する(87, 88)。

コード変換関数 `dcs_convert_buffer()` はリターンするまでに、引き数で渡された監視関数 `h_observer()` と出力関数 `h_writer()` を呼び出している。コード判定が終わるとコード変換関数 `dcs_convert_buffer()` は判定結果を引き数にして監視関数を 1 度だけ呼び出す。監視関数から継続のステータスが返ると変換関数 `dcs_convert_buffer()` は変換処理を始める。編集バッファがフルになる度に今度は出力関数を呼び出し、正常ステータスが返ると変換を継続して行く。このプログラム `hello2` では監視関数 `h_observer()` が判定結果から文字コード名を取得し(17)画面に表示した後、継続ステータスを返している(21)。そして出力関数 `h_writer()` は引数で与えられた出力サイズ `size` によりバッファの空きサイズを確認／調整して(26, 29, 31)、データ `buf` を出力バッファに書き出してから(35, 36)成功ステータスを返す(39)。

# Dcs API 関数リファレンス

プロトタイプの末尾に \* のあるものはマクロであることを表す：例 `int get_ltype(Dcs *dcs);*`

## 1. Dcs 情報生成／開放

**Dcs \*create\_dcs();**

引き数: なし

戻り値: 生成された Dcs 情報が返る。メモリー・アロケート・エラーの時は NULL。

機能:

文字コード判定や文字コード変換等を行なうための Dcs 情報を生成する。

**void release\_dcs(Dcs \*dcs);**

引き数: dcs      - Dcs 情報

戻り値: なし

機能:

不要になった Dcs 情報を解放する。

---

## 2. 改行コード・タイプの取得／設定

**int get \_ ltype(Dcs \*dcs);\***

引き数: dcs      - Dcs 情報

戻り値: 設定されている改行コード・タイプ

機能:

`dcs_convert_buffer()` 等のコード変換時に行なわれる、改行コード編集の改行コード・タイプを取得する。以下の4つの何れかが返る。初期値は `LTYPE_NON` で、改行コードの編集は行なわない。なお定義されている値は5つあるが、残りの1つ `LTYPE_ALL(07)` は `dcs_judges_buffer()`

等のコード判定時に判定される値であり、get\_ltype() では返らない。

```
#define LTYPE_NON      000000000 /* 改行なし */
#define LTYPE_LF       000000001 /* UNIX 系 */
#define LTYPE_CR       000000002 /* Mac OS9 */
#define LTYPE_CRLF     000000004 /* Win */
```

**void set\_ltype(Dcs \*dcs, int ltype);\***

引き数: dcs        - Dcs 情報  
          ltype    - 設定する改行コード・タイプ

戻り値: なし

機能:

dc\_convert\_buffer() 等のコード変換時に行なわれる、改行コード編集の改行コード・タイプを設定する。get\_ltype() で示された 4 つの何れかを設定する。

なお、残りの 1 つ LTYPE\_ALL(07) は dc\_judges\_buffer() 等のコード判定時に判定される値であり、上記 4 つの値以外は set\_ltype() では無効である。

### 3. 日本語 UTF8-MAC 1 文字変換の判定／設定及び、UTF8-MAC コード判定

日本語 UTF8-MAC 1 文字変換の判定

**bool dcs\_is\_utfmac\_to(Dcs \*dcs);\***

引き数: dcs        - Dcs 情報

戻り値: 設定されている日本語 UTF8-MAC 1 文字変換フラグ

機能:

dc\_convert\_buffer() 等によるコード変換時に行なう日本語 UTF8-MAC 1 文字変換の真偽を判定する。初期値は偽であり、日本語 UTF8-MAC 1 文字変換を行なわない。

日本語 UTF8-MAC 1 文字変換の設定

**void dcs\_set\_utfmac\_to(Dcs \*dcs, bool flag);\***

引き数: dcs        - Dcs 情報  
          flag       - 日本語 UTF8-MAC 1 文字変換の可、不可を設定する。

戻り値: なし

機能:

dc\_convert\_buffer() 等によるコード変換時に行なう日本語 UTF8-MAC 1 文字変換の真偽を設定する。日本語 UTF8-MAC 1 文字変換する時は真、それ以外は偽を設定する。

#### UTF8-MAC コード判定

**bool dcs\_utfmac\_exists(Dcs \*dcs);\***

引き数: dcs      - Dcs 情報

戻り値: 設定されている入力データの日本語 UTF8-MAC コード存在フラグ

機能:

dc\_judges\_buffer() 等によるコード判定や dc\_convert\_buffer() 等によるコード変換の後の、入力データの日本語 UTF8-MAC コード有無の判定を行なう。

## 4. 出力コードの半角カナから全角変換の判定／設定

#### 出力コードの半角カナから全角変換の判定

**bool dcs\_is\_hankana\_to(Dcs \*dcs);\***

引き数: dcs      - Dcs 情報

戻り値: 設定されている半角カナから全角変換フラグ

機能:

dc\_convert\_buffer() 等によるコード変換時に行なう半角カナから全角変換の真偽を判定する。初期値は偽であり、半角カナから全角への変換を行なわない。

#### 出力コードを半角カナから全角変換の設定

**void dcs\_set\_hankana\_to(Dcs \*dcs, bool flag);\***

引き数: dcs      - Dcs 情報  
           flag      - 半角カナから全角変換の可、不可を設定する。

戻り値: なし

機能:

dc\_convert\_buffer() 等によるコード変換時に行なう半角カナから全角変換の真偽を設定する。半角カナから全角変換する時は真、それ以外は偽を設定する。

## 5. 不正文字出力抑止の判定／設定

### 不正文字出力抑止の判定

```
bool dcs_is_suppress(Dcs *dcs);*
```

引き数: dcs        - Dcs 情報

戻り値: 設定されている不正文字出力抑止フラグ

機能:

    dcs\_convert\_buffer() 等によるコード変換時に発生する不正文字抑止の真偽を判定する。初期値は偽であり、不正文字を出力する。

### 不正文字出力抑止の設定

```
void dcs_set_suppress(Dcs *dcs, bool flag);*
```

引き数: dcs        - Dcs 情報  
        flag        - 不正文字出力抑止の可、不可を設定する。

戻り値: なし

機能:

    dcs\_convert\_buffer() 等によるコード変換時に発生する不正文字出力抑止の真偽を設定する。出力を抑止する時は真、それ以外は偽を設定する。

---

## 6. コード判定 (バッファ／ストリーム／ファイル)

```
int dcs_judges_buffer(Dcs *dcs,  
                      int *atype, int *ln_type, char *buffer, size_t size, int stype);  
int dcs_judges_stream(Dcs *dcs, int *atype, int *ln_type, FILE *sfp, int stype);  
int dcs_judges_file(Dcs *dcs, int *atype, int *ln_type, const char *spath, int stype);
```

引き数: dcs        - Dcs 情報  
        atype      - 未確定の文字コード・タイプ  
        ln\_type    - 改行コード・タイプ  
        buffer     - 入力バッファ  
        size       - 入力サイズ  
        sfp        - 入力ファイル・ポインター  
        spath      - 入力ファイル・パス  
        stype      - 入力データの文字コード・タイプ

戻り値: 正常の場合は判定した文字コード・タイプが返る。メモリー・アロケート・エラーの時は DCS\_EALLOC、ファイル入力エラーの時は DCS\_EREAD が返る。

機能:

バッファからのコード判定

#### **dcs\_judges\_buffer()**

入力バッファ buffer、入力サイズ size のデータから文字コードの判定を行なう。

atype は未確定の文字コード・タイプ (DCS\_CONFIRMED\_CODE\_TYPE(sts)が偽) がある時にはその値、一意に決まればリターン値と同じになり、エラーの場合は DCS\_UNKNOWN(-1) が設定される。不要ならば NULL を設定する。

ln\_type は入力データの改行コード・タイプであり、以下の5つの何れかが混在していた時はそれらの OR 値が設定される。不要ならば NULL を設定する。

/\* 改行コード・タイプ \*/

```
#define LTYPE_NON      000000000 /* 改行なし */
#define LTYPE_LF       000000001 /* UNIX 系 */
#define LTYPE_CR       000000002 /* Mac OS9 */
#define LTYPE_CRLF     000000004 /* Win */
#define LTYPE_ALL      000000007 /* すべての改行 */
```

stype は文字コードが一意に決まらなかった場合に、同一の文字コード・タイプがあれば、その値に決定する。不要ならば DCS\_UNKNOWN を設定する。

ストリームからのコード判定

#### **dcs\_judges\_stream()**

ファイル・ポインター sfp のデータから文字コードの判定を行なう。sfp は呼び出し側でクローズする。入力エラーの時は DCS\_EREAD が返る。その他の機能は dcs\_judges\_buffer() と同じである。

ファイルからのコード判定

#### **dcs\_judges\_file()**

入力ファイル spath のデータから文字コードの判定を行なう。ファイル・オープン・エラーの時は DCS\_EROPEN、入力エラーの時は DCS\_EREAD が返る。その他の機能は dcs\_judges\_buffer() と同じである。

## 7. コード変換 (バッファ／ストリーム／ファイル)

```
int dcs_convert_buffer(Dcs *dcs, const char *buffer, size_t size, int stype, int dtype,
    bool is_forced, int (*observer)(int sts, void *user_info),
    int (*writer)(const char *buf, size_t size, void *user_info), void *user_info);
```

```
int dcs_convert_stream(Dcs *dcs, FILE *sfp, int stype, int dtype,
    bool is_forced, int (*observer)(int sts, void *user_info),
    int (*writer)(const char *buf, size_t size, void *user_info), void *user_info);
```

```
int dcs_convert_file(Dcs *dcs, const char *spath, int stype, int dtype,
    bool is_forced, int (*observer)(int sts, void *user_info),
    int (*writer)(const char *buf, size_t size, void *user_info), void *user_info);
```

引き数: dcs	- Dcs 情報
buffer	- 入力バッファ
size	- 入力サイズ
sfp	- 入力ファイル・ポインター
spath	- 入力ファイル・パス
stype	- 入力データの文字コード・タイプ
dtype	- 出力データの文字コード・タイプ
is_forced	- 入力データの文字コード・タイプを強制的に stype とする
observer	- 判定結果の監視関数 (ユーザー定義)
writer	- 変換データの出力関数 (ユーザー定義)
user_info	- observer, writer に渡されるユーザー情報

戻り値: 変換が成功した場合は DCS\_SUCCESS、判定終了時に observer を呼び出して中止した場合は DCS\_ECANOBS、コード判定エラー (DCS\_CONFIRMED\_CODE\_TYPE(sts)が偽) の時はその値が返る。メモリー・アロケート・エラーの時は DCS\_EALLOC。

機能:

バッファからのコード変換

**dcs\_convert\_buffer()**

初めに入力バッファ buffer、入力サイズ size の文字コード判定を行い、一意に決定した場合は、指定された dtype の文字コードに変換する。

stype は dcs\_judges\_buffer() 等のコード判定だけだけを行なう関数と同様に、文字コードが一意に決まらなかった場合に、同一の文字コード・タイプがあれば、その値に決定する。不要ならば DCS\_UNKNOWN を設定する。is\_forced が真ならば、DCS\_UNKNOWN 以外の stype の値が強制的に採られる。ただし、実際の文字コードが JIS や BOM 付きの UTF、又は不正コードの多い場合は無効となる。

observer は変換前のコード判定が終了した時点で、渡される判定結果 sts により必ず 1 回呼び



出される。継続ならば DCS\_OBS\_CONTINUE、中止する時は DCS\_OBS\_STOP で戻る。致命的なエラーや入力データの文字コード・タイプが確定できなかった

(CONFIRMED\_CODE\_TYPE(sts)が偽) 場合は判定結果が dcs\_convert\_buffer() の返却値となり、observer が DCS\_OBS\_CONTINUE を返しても継続されない。変換結果をファイルに出力する場合は observer がそのファイルをオープンし、dcs\_convert\_buffer() の返った後、呼び出し側でクローズする。observer が不要ならば NULL を設定する。

writer は変換中に編集バッファがフルになる度に呼び出され、渡される出力バッファ buffer、出力サイズ size により実際の出力を行なう。出力が成功したら DCS\_SUCCESS、ファイル出力エラーなら DCS\_EWRITE、メモリー・アロケート・エラーならば DCS\_EALLOC で戻る。writer の返却値が DCS\_SUCCESS ならば継続され、それ以外の値なら中断される。最後の呼び出しの場合、これらは dcs\_convert\_buffer() の返却値となる。writer が不要ならば NULL を設定する。

user\_info は observer() や writer() に渡されるユーザー情報であり、不要ならば NULL を設定する。

#### ストリームからのコード変換

##### **dcs\_convert\_stream()**

ファイル・ポインター sfp のデータを指定された文字コードに変換する。sfp は呼び出し側でクローズする。入力エラーの時は DCS\_EREAD が返る。その他の機能は dcs\_convert\_buffer() と同じである。

#### ファイルからのコード変換

##### **dcs\_convert\_file()**

入力ファイル spath のデータを指定された文字コードに変換する。ファイル・オープン・エラーの時は DCS\_EROPEN、入力エラーの時は DCS\_EREAD が返る。その他の機能は dcs\_convert\_buffer() と同じである。

## 8. ファイル・アクセス時のエラー番号／メッセージ取得

**int dcs\_get\_errno(Dcs \*dcs);\***

引き数: dcs        - Dcs 情報

戻り値: 最後にアクセスしたファイルの errno が返る。

機能:

ファイル・アクセス時のエラー番号を取得する。

**int dcs\_get\_strerror(Dcs \*dcs, int err\_no, char \*str, size\_t size, int ctype);**

引き数: dcs        - Dcs 情報  
      err\_no    - 対象のエラー番号  
      str        - 出力バッファ  
      size       - 出力サイズ  
      ctype      - 変換する文字コード・タイプ

戻り値: 取得されたメッセージの長さが返る。ヌル終端文字は含まない。

機能:

引き数で渡されたエラー番号 err\_no から、指定された文字コードに変換されたメッセージを取得する。変換する文字コード・タイプ ctype には DCS\_JIS, DCS\_SJIS, DCS\_EUC, DCS\_UTF8 の何かを指定する。変換の必要がない場合は dcs に NULL、変換する文字コード・タイプ ctype は不要であるため 0 を設定する。

---

## 9. 処理ステースのメッセージ取得

**int dcs\_statusto\_message(Dcs \*dcs, char \*str, int sts, int ctype);**

引き数: dcs        - Dcs 情報  
      str        - 出力バッファ  
      sts        - 対象のステース  
      ctype      - 変換する文字コード・タイプ

戻り値: 取得されたメッセージの長さが返る。ヌル終端文字は含まない。

機能:

引き数で渡された処理ステース sts から、指定された文字コードに変換されたメッセージを取得する。変換する文字コード・タイプ ctype には DCS\_JIS, DCS\_SJIS, DCS\_EUC, DCS\_UTF8 の何かを指定する。変換の必要がない場合は dcs に NULL、変換する文字コード・タイプ ctype は不要であるため 0 を設定する。

## 10. 変換不能文字情報取得（文字数、先頭／末尾文字、先頭／末尾位置）

```
int get_no_char_count(Dcs *dcs);*
int64_t get_first_no_char(Dcs *dcs);*
int64_t get_last_no_char(Dcs *dcs);*
long get_first_no_char_offset(Dcs *dcs);*
long get_last_no_char_offset(Dcs *dcs);*
```

引き数: dcs        - Dcs 情報

戻り値:

変換不能文字数取得

**get\_no\_char\_count()**

変換ができなかった文字数が返る。

変換不能先頭文字取得

**get\_first\_no\_char()**

変換ができなかった文字があれば最初の文字、すべて変換できていた時は -1 が返る。

変換不能末尾文字取得

**get\_last\_no\_char()**

変換ができなかった文字があれば最後の文字、すべて変換できていた時は -1 が返る。

変換不能先頭位置取得

**get\_first\_no\_char\_offset()**

変換ができなかった文字があれば最初のバイト位置（ゼロ・オリジン）、すべて変換できた時は -1L が返る。

変換不能末尾位置取得

**get\_last\_no\_char\_offset()**

変換ができなかった文字があれば最後のバイト位置（ゼロ・オリジン）、すべて変換できた時は -1L が返る。

機能:

dcs\_convert\_buffer() 等によるコード変換で、変換できなかった文字の数と、それらの先頭／末尾文字、先頭／末尾位置を取得する。

## 11. Dcs API バージョン取得

**float dcs\_et\_version();**

引き数: なし

戻り値: Dcs API のバージョンが返る。

機能:

Dcs API のバージョンを取得する。DCS\_VERSION と同値。

---

## 12. コード・タイプ／コード名取得

文字コード名取得

**int dcs\_typedto\_name(char \*name, int ctype);**

引き数: name    - 取得した文字コード名（ヌル終端）

          ctype    - 対象の文字コード・タイプ

戻り値: 取得した name の長さが返る。ヌル終端文字は含まない。

機能:

文字コード・タイプから文字コード名を取得する。

文字コード・タイプ取得

**int dcs\_nameto\_type(const char \*name);**

引き数: name    - 対象の文字コード名

戻り値: 取得した文字コード・タイプが返る。

機能:

文字コード名から文字コード・タイプを取得する。

不特定のコード名取得

**int dcs\_multi\_type\_names(char \*names, size\_t size, int ctype);**

すべてのコード名を取得

**int dcs\_all\_type\_names(char \*names, size\_t size);**

### 13. BOMの追加／削除判定

**int dcs\_cmp\_bom(int stype, int dtype);**

引き数: stype    - 変換前の文字コード・タイプ  
          dtype    - 変換後の文字コード・タイプ

戻り値: BOM追加／削除の判定値 CBOM\_NON（なし）、CBOM\_ADD（追加）、CBOM\_DELETE（削除）の何れかが返る。

機能:

dcsc\_convert\_buffer() 等によるコード変換で起きる、UTF系BOMの追加／削除の有無を判定する。同じ文字コードであれば追加や削除になり、それ以外は非となる。DCS\_UTF16BE から DCS\_UTF16BE\_BOM への変換は CBOM\_ADD、DCS\_UTF16BE から DCS\_UTF16LE\_BOM への変換は CBOM\_DELETE が返る。

### 14. 変換不能コードの判定

<b>bool is_jis_no_char(char *buf);*</b>	- JIS 変換不能コードの判定（文字列）
<b>bool is_jis_no_short(int c);*</b>	- JIS 変換不能コードの判定（数値）
<b>bool is_sjis_no_char(char *buf);*</b>	- SJIS 変換不能コードの判定（文字列）
<b>bool is_sjis_no_short(int c);*</b>	- SJIS 変換不能コードの判定（数値）
<b>bool is_euc_no_char(char *buf);*</b>	- EUC 変換不能コードの判定（文字列）
<b>bool is_euc_no_short(int c);*</b>	- EUC 変換不能コードの判定（数値）
<b>bool is_utf8_no_char(char *buf);*</b>	- UTF-8BE 変換不能コードの判定（文字列）
<b>bool is_utf8_no_short(int c);*</b>	- UTF-8BE 変換不能コードの判定（数値）
<b>bool is_unibe_no_char(char *buf);*</b>	- UTF-16BE 変換不能コードの判定（文字列）
<b>bool is_unibe_no_short(int c);*</b>	- UTF-16BE 変換不能コードの判定（数値）
<b>bool is_unile_no_char(char *buf);*</b>	- UTF-16LE 変換不能コードの判定（文字列）
<b>bool is_unile_no_short(int c);*</b>	- UTF-16LE 変換不能コードの判定（数値）

引き数: buf        - 判定対象コードの先頭アドレス  
          c            - 判定対象コード

戻り値: 引き数 c が変換不能コードの時は真、それ以外は偽が返る。

機能:

dcsc\_convert\_buffer() 等によるコード変換で出力される変換不能コードの値を判定する。変換不能コードの値は J I S が全角'?'(NO\_CHAR1, NO\_CHAR2)、それ以外の文字コードは半角の'?'(NO\_CHAR)

ここからは Dcs API を使用するために必須ではないが、文字コードを扱う上で有用なものを解説する。

## 15. 半角カナ判定

<code>bool is_jis_kana(int c);*</code>	- JIS 半角カナの判定
<code>bool is_sjis_kana(int c);*</code>	- SJIS 半角カナの判定
<code>bool is_unibe_kana(int c);*</code>	- UTF-16BE 半角カナの判定

引き数: c - 真偽を判定する文字コード

戻り値: 指定した文字コードがこの文字コードセットの半角カナに含まれる時は真、それ以外は偽が返る。

---

## 16. 半角カナ変換

<code>int lis_kana_to_sjis(int c);*</code>	- JIS から SJIS への半角カナ変換
<code>int lis_kana_to_unibe(int c);*</code>	- JIS から UTF-16BE への半角カナ変換
<code>int slis_kana_to_jis(int c);*</code>	- SJIS から JIS への半角カナ変換
<code>int slis_kana_to_unibe(int c);*</code>	- SJIS から UTF-16BE への半角カナ変換
<code>int unibe_kana_to_jis(int c);*</code>	- UTF-16BE から JIS への半角カナ変換
<code>int unibe_kana_to_sjis(int c);*</code>	- UTF-16BE から SJIS への半角カナ変換

引き数: c - 変換対象の文字コード

戻り値: 変換後の文字コード。該当がない時は不正値が返る。

---

## 17. 全角仮名判定

<code>bool is_jis_zenkana(int c);*</code>	- JIS 全角仮名判定
<code>bool is_sjis_zenkana(int c);*</code>	- SJIS 全角仮名判定
<code>bool is_euc_zenkana(int c);*</code>	- EUC 全角仮名判定
<code>bool is_unibe_zenkana(int c);*</code>	- UTF-16BE 全角仮名判定

引き数: c - 真偽を判定する文字コード。

戻り値: 指定した文字コードがこの文字コードセットの全角平仮名、又は全角片仮名に含まれる時は真、それ以外は偽が返る。

## 18. UTF-16 間、UTF-32 間変換

<code>void short_to_char(char *dest, int c);*</code>	- 数値から文字列への変換
<code>int char_to_short(const char *src);*</code>	- 文字列から数値への変換
<code>int unibe_char_to_short(const char *src);*</code>	- UTF-16BE(文字列)から UTF-16BE(数値)への変換
<code>long unibe_char_to_long(const char *src);*</code>	- UTF-16BE(文字列)から UTF-32BE(数値)への変換
<code>int unile_char_to_short(const char *src);*</code>	- UTF-16LE(文字列)から UTF-16LE(数値)への変換
<code>int unibe_to_unile_char(const char *src);*</code>	- UTF-16BE(文字列)から UTF-16LE(数値)への変換
<code>int unile_to_unibe_char(const char *src);*</code>	- UTF-16LE(文字列)から UTF-16BE(数値)への変換
<code>int unibe_to_unile_short(int c);*</code>	- UTF-16BE(数値)から UTF-16LE(数値)への変換
<code>int unile_to_unibe_short(int c);*</code>	- UTF-16LE(数値)から UTF-16BE(数値)への変換
<code>int utf32be_to_utf32le_char(const char *src);*</code>	- UTF-32BE(文字列)から UTF-32LE(数値)への変換
<code>int utf32le_to_utf32be_char(const char *src);*</code>	- UTF-32LE(文字列)から UTF-32BE(数値)への変換
<code>int utf32be_to_utf32le_long(int c);*</code>	- UTF-32BE(数値)から UTF-32LE(数値)への変換
<code>int utf32le_to_utf32be_long(int c);*</code>	- UTF-32LE(数値)から UTF-32BE(数値)への変換

引き数: `c`            - 変換対象の文字コード  
           `src`        - 変換対象の文字列

戻り値: 変換後の文字コード。該当がない時は不正値が返る。

---

## 19. UTF-16 から UTF-8 への変換

<code>void unibe_utf8_char(char *dest, char *src, int bytes);*</code>	- UTF-16BE (文字列) から UTF-8 (文字列) への変換
<code>void unibe_to_utf8_short(char *dest, int c, int bytes);*</code>	- UTF-16BE (数値) から UTF-8 (文字列) への変換
<code>void unile_to_utf8_char(char *dest, char *src, int bytes);*</code>	- UTF-16LE (文字列) から UTF-8 (文字列) への変換
<code>void unile_to_utf8_short(char *dest, int c, int bytes);*</code>	- UTF-16LE (数値) から UTF-8 (文字列) への変換

引き数: `dest`    - 出力バッファ  
           `src`        - 変換対象の文字列  
           `c`        - 変換対象の文字コード

戻り値: なし

機能:

入力コードは数値 `c` と文字列 `src` があり、変換されたコードはすべて `dest` に出力され、出力サイズは `bytes` に設定される。

## 20. UTF-8(4byte)から UTF-16BE への変換

```
void utf8_to_unibe_4bytes(char *dest, int c1, int c2,int c3,int c4,int bytes,int sts);*
```

引き数: dest     - 出力バッファ  
           c1     - 変換前の文字コード第 1 バイト  
           c2     - 変換前の文字コード第 2 バイト  
           c3     - 変換前の文字コード第 3 バイト  
           c4     - 変換前の文字コード第 4 バイト  
           bytes   - dest へ格納したバイト数  
           sts     - 変換結果のステータス

戻り値: なし

機能:

入力は 1～4 バイトまでであり、ゼロより小さければそこで EOD となる。入力バイト数は bytes に設定される。入力バイト数が 4 バイトならばサロゲート文字であり出力も 4 バイトとなり、その他の出力はすべて 2 バイトである。sts には変換が成功した時は DCS\_SUCCESS、不正なコードがあった時は DCS\_ECHAR、コードの途中で終わっている時は DCS\_EUEXPEOD が設定される。

---

## 21. サロゲート・ペアからの変換

```
void unibe_surrogate_to_utf8(char *dest, int hcd, int lcd);*
```

- UTF-16BE サロゲート・ペアから UTF-8 (文字列) への変換

```
void unibe_surrogate_to_cd(int dcd, int hcd, int lcd);*
```

- UTF-16BE サロゲート・ペアからコードへの変換

引き数: dest     - 出力バッファ  
           hcd     - 変換対象の文字コード (上位)  
           lcd     - 変換対象の文字コード (下位)  
           dcd     - 出力コード

戻り値: なし

機能:

入力コードは上位数値 hcd(0xd800～0xdbff)と、下位数値 lcd(0xdc00～0xdfff)を設定する。変換されたコード dest のサイズは常に 4 バイトである。出力コード dcd には 0x10000 以上の値が設定される。



# 付録

## *Dcs API 定義*

ヘッダー・ファイル dcs.h に定義されている処理ステータス、コード・タイプ等の主な値を示す。

D C S バージョン

```
#define DCS_VERSION ((float)3.00)
```

D C S バージョン情報 (文字列)

```
#define DCS_VERSION_INFO
```

文字列の区切り文字

```
#define DCS_SEPARATE_CHAR ';' ;
```

```
#define min(a, b) ((a) < (b)? (a): (b))
```

```
#define max(a, b) ((a) > (b)? (a): (b))
```

処理ステータス

```
#define DCS_SUCCESS 0 /* 処理成功 */
#define DCS_EOF -1 /* 入力ファイルの終わり */
#define DCS_ENOTEXISTS -2 /* ファイルは存在せず */
#define DCS_ENOTFILE -3 /* 通常ファイルではない */
#define DCS_EALLOC -4 /* メモリー・アロケート・エラー */
#define DCS_EROPEN -5 /* 入力ファイル・オープン・エラー */
#define DCS_EWOPEN -6 /* 出力ファイル・オープン・エラー */
#define DCS_EREAD -7 /* 入力エラー */
#define DCS_EWRITE -8 /* 出力エラー */
#define DCS_EIOMODE -10 /* 入出力モード誤り */
#define DCS_EIOFDRANGE -11 /* ファイル記述子範囲誤り */
#define DCS_EIONOTOPEN -12 /* 未オープン・ファイルへのアクセス */
#define DCS_EIOMAX -13 /* 上限を超えたファイル・オープン */
#define DCS_ECTYPE -20 /* コード・タイプ誤り */
#define DCS_EBOM -21 /* BOM 不正 (未使用) */
#define DCS_EUEXPEOD -22 /* 予期せぬデータの終わり */
#define DCS_ECHAR -23 /* コード値エラー */
#define DCS_ELTYPE -24 /* 改行コード・タイプ誤り */
#define DCS_ECANOBS -30 /* 監視者による取り消し */
```

## コード・タイプ (コード判定ステータス)

```

#define DCS_UNKNOWN          -1 /* コード・タイプ不明 */
#define DCS_EMPTY            0 /* 空データ */
#define DCS_ASCII            1 /* ASCII */
#define DCS_JIS              2 /* JIS ISO-2022-JP */
#define DCS_JIS1             3 /* JIS CP50221 */
#define DCS_JIS2             4 /* JIS ISO-2022-JP-2004 */
#define DCS_JIS3             5 /* JIS ISO-2022-JP-3 */
#define DCS_SJIS             6 /* Shift_JIS */
#define DCS_SJIS1            7 /* SJIS CP932 */
#define DCS_SJIS2            8 /* Shift_JIS-2004 */
#define DCS_EUC              9 /* EUC-JP */
#define DCS_EUC1            10 /* EUC CP20932 */
#define DCS_EUC2            11 /* EUC CP51932 */
#define DCS_EUC3            12 /* eucJP-ms */
#define DCS_EUC4            13 /* eucJP-2004 */
#define DCS_UTF8            20 /* UTF-8 */
#define DCS_UTF8_BOM        21 /* UTF-8 (BOM 付) */
#define DCS_UTF16BE         22 /* UTF-16 ビッグ・エンディアン */
#define DCS_UTF16BE_BOM    23 /* UTF-16 ビッグ・エンディアン (BOM 付) */
#define DCS_UTF16LE         24 /* UTF-16 リトル・エンディアン */
#define DCS_UTF16LE_BOM    25 /* UTF-16 リトル・エンディアン (BOM 付) */
#define DCS_UTF32BE         26 /* UTF-32 ビッグ・エンディアン */
#define DCS_UTF32BE_BOM    27 /* UTF-32 ビッグ・エンディアン (BOM 付) */
#define DCS_UTF32LE         28 /* UTF-32 リトル・エンディアン */
#define DCS_UTF32LE_BOM    29 /* UTF-32 リトル・エンディアン (BOM 付) */
#define DCS_NORMAL          00000000077 /* 確定コード・タイプの最大値 */

```

## 各コード範囲判定

```

#define IS_JIS(stype)        (DCS_JIS  <= (stype) && (stype) <= DCS_JIS3)
#define IS_SJIS(stype)      (DCS_SJIS <= (stype) && (stype) <= DCS_SJIS2)
#define IS_EUC(stype)       (DCS_EUC  <= (stype) && (stype) <= DCS_EUC4)
#define IS_UTF(stype)       (DCS_UTF8  <= (stype) && (stype) <= DCS_UTF32LE_BOM)

```

## 確定コード・タイプ

```

#define DCS_CONFIRMED_CODE_TYPE(sts) (DCS_UNKNOWN < (sts) && (sts) <= DCS_NORMAL)

```

## 未確定コード・タイプ (マスク値)

```

#define DCS_ND_ASCII        000000000100 /* ASCII */
#define DCS_ND_JIS          000000000200 /* JIS ISO-2022-JP */
#define DCS_ND_JIS1         000000000400 /* JIS CP50221 */
#define DCS_ND_JIS2         000000001000 /* JIS ISO-2022-JP-2004 */
#define DCS_ND_JIS3         000000002000 /* JIS ISO-2022-JP-3 */
#define DCS_ND_JIS_ALL      000000003600 /* JIS ALL */
#define DCS_ND_SJIS         000000004000 /* Shift_JIS */
#define DCS_ND_SJIS1        000000010000 /* SJIS CP932 */
#define DCS_ND_SJIS2        000000020000 /* Shift_JIS-2004 */
#define DCS_ND_SJIS_ALL     000000034000 /* SJIS ALL */
#define DCS_ND_EUC          000000040000 /* EUC-JP */
#define DCS_ND_EUC1         000000100000 /* EUC CP20932 */

```

```

#define DCS_ND_EUC2          000000200000 /* EUC CP51932 */
#define DCS_ND_EUC3          000000400000 /* eucJP-ms */
#define DCS_ND_EUC4          000001000000 /* eucJP-2004 */
#define DCS_ND_EUC_ALL       000001740000 /* EUC ALL */
#define DCS_ND_UTF8          000010000000 /* UTF-8 */
#define DCS_ND_UTF8_BOM      000020000000 /* UTF-8 (BOM 付) */
#define DCS_ND_UTF16BE       000040000000 /* UTF-16 ビッグ・エンディアン */
#define DCS_ND_UTF16BE_BOM   000100000000 /* UTF-16 ビッグ・エンディアン (BOM 付) */
#define DCS_ND_UTF16LE       000200000000 /* UTF-16 リトル・エンディアン */
#define DCS_ND_UTF16LE_BOM   000400000000 /* UTF-16 リトル・エンディアン (BOM 付) */
#define DCS_ND_UTF32BE       001000000000 /* UTF-32 ビッグ・エンディアン */
#define DCS_ND_UTF32BE_BOM   002000000000 /* UTF-32 ビッグ・エンディアン (BOM 付) */
#define DCS_ND_UTF32LE       004000000000 /* UTF-32 リトル・エンディアン */
#define DCS_ND_UTF32LE_BOM   010000000000 /* UTF-32 リトル・エンディアン (BOM 付) */
#define DCS_ND_UTF_BOM      (DCS_ND_UTF8_BOM| \
                             DCS_ND_UTF16BE_BOM|DCS_ND_UTF16LE_BOM| \
                             DCS_ND_UTF32BE_BOM|DCS_ND_UTF32LE_BOM) /* UTF BOM 付 */

#define DCS_ND_ALL           017777777700 /* ND ALL */

変換監視継続、停止ステータス
#define DCS_OBS_STOP        -1      /* 変換監視停止 */
#define DCS_OBS_CONTINUE    0       /* 変換監視継続 */

改行コード・タイプ
#define LTYPE_NON           0000000000 /* 改行なし */
#define LTYPE_LF            0000000001 /* UNIX 系 */
#define LTYPE_CR            0000000002 /* Mac OS9 */
#define LTYPE_CRLF          0000000004 /* Win */
#define LTYPE_ALL           0000000007 /* すべての改行 */

BOM 追加／削除判定
#define CBOM_NON            0 /* BOM 追加削除なし */
#define CBOM_ADD            1 /* BOM 追加 */
#define CBOM_DELETE         -1 /* BOM 削除 */

未定義コード表示値
#define NO_CHAR              '?'

未定義コード表示値(JIS)
#define NO_CHAR1            0x21 /* 第 1 バイト */
#define NO_CHAR2            0x29 /* 第 2 バイト */

```

#### UTF-8 BOM

```
#define UTF8_B0          0xef
#define UTF8_B1          0xbb
#define UTF8_B2          0xbf
```

#### UTF-16BE BOM

```
#define UTF16_B0        0xfe
#define UTF16_B1        0xff
```

#### UTF-16 置換文字

```
#define REP_CHARBE      0xffffd
#define REP_CHARLE      0xfdff
```

#### JIS 全角仮名領域

```
#define JIS_HKANA_B      0x2421 /* 開始平仮名 */
#define JIS_HKANA_E      0x2473 /* 終了平仮名 */
#define JIS_KKANA_B      0x2521 /* 開始片仮名 */
#define JIS_KKANA_E      0x2573 /* 終了片仮名 */
```

#### SJIS 全角仮名領域

```
#define SJIS_HKANA_B     0x829f /* 開始平仮名 */
#define SJIS_HKANA_E     0x82f1 /* 終了平仮名 */
#define SJIS_KKANA_B     0x8340 /* 開始片仮名 */
#define SJIS_KKANA_E     0x8393 /* 終了片仮名 */
```

#### EUC 全角仮名領域

```
#define EUC_HKANA_B      0xa4a1 /* 開始平仮名 */
#define EUC_HKANA_E      0xa4f3 /* 終了平仮名 */
#define EUC_KKANA_B      0xa5a1 /* 開始片仮名 */
#define EUC_KKANA_E      0xa5f3 /* 終了片仮名 */
```

#### Unicode(BE)全角仮名領域

```
#define UNIBE_HKANA_B    0x3041 /* 開始平仮名 */
#define UNIBE_HKANA_E    0x309f /* 終了平仮名 */
#define UNIBE_KKANA_B    0x30a1 /* 開始片仮名 */
#define UNIBE_KKANA_E    0x30ff /* 終了片仮名 */
#define UNIBE_ODORIJI_B  0x3031 /* 開始範囲外踊り字（寫衰 / ㇿ） */
#define UNIBE_ODORIJI_E  0x3035 /* 終了範囲外踊り字 */
```