

Dcs_(Ver. 2.11)

リファレンス・マニュアル

青字はバージョン2.00, 2.10, 2.11の新機能

目次

A. Dcs API

概要	6
<code>get_version()</code>	
文字コード判定／変換テーブル	6
文字コード判定／変換関数	7
<code>dcs_judges_buffer()</code> <code>dcs_judges_stream()</code> <code>dcs_judges_file()</code>	
<code>dcs_convert_buffer()</code> <code>dcs_convert_stream()</code> <code>dcs_convert_file()</code>	
Dcs API 関数使用例	
文字コード判定プログラム	8
<code>create_dcs()</code> <code>dcs_judges_buffer()</code> <code>dcs_typedto_name()</code> <code>release_dcs()</code>	
文字コード変換プログラム	9
<code>dcs_typedto_name()</code> <code>create_dcs()</code> <code>dcs_convert_buffer()</code> <code>dcs_judges_buffer()</code>	
<code>dcs_typedto_name()</code> <code>release_dcs()</code>	
テキスト 1 行入力プログラム	11
<code>dcs_jis_strln()</code> <code>dcs_sjis_strln()</code>	
Dcs 関数リファレンス 関数名の後に * のあるものはマクロであることを表す：例 <code>get_ltype()</code> *	
A- 1. Dcs 情報生成／開放	13
<code>create_dcs()</code> <code>release_dcs()</code>	
A- 2. 改行コード・タイプの取得／設定	13
<code>get_ltype()</code> * <code>set_ltype()</code> *	
A- 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	14
<code>dcs_is_utfmac_to()</code> * <code>dcs_set_utfmac_to()</code> * <code>dcs_utfmac_exists()</code> *	
A- 4. 不正文字出力抑止の判定／設定	15
<code>dcs_is_suppress()</code> * <code>dcs_set_suppress()</code> *	
A- 5. コード判定（バッファ／ストリーム／ファイル）	15
<code>dcs_judges_buffer()</code> <code>dcs_judges_stream()</code> <code>dcs_judges_file()</code>	
A- 6. コード変換（バッファ／ストリーム／ファイル）	17
<code>dcs_convert_buffer()</code> <code>dcs_convert_stream()</code> <code>dcs_convert_file()</code>	
A- 7. ファイル・アクセス時のエラー番号／メッセージ取得	18
<code>dcs_get_errno()</code> * <code>dcs_get_strerror()</code>	
A- 8. 処理ステータスのメッセージ取得	19
<code>dcs_statusto_message()</code>	
A- 9. 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
<code>get_no_char_count()</code> * <code>get_first_no_char()</code> * <code>get_last_no_char()</code> *	
<code>get_first_no_char_offset()</code> * <code>get_last_no_char_offset()</code> *	
A-10. テキストの 1 行入力	20
<code>dcs_jis_strln()</code> <code>dcs_sjis_strln()</code> <code>dcs_euc_strln()</code>	
<code>dcs_unibe_strln()</code> <code>dcs_unile_strln()</code> <code>dcs_utf8_strln()</code>	

A-11. Dcs API バージョン取得	21
dcs_get_version()	
A-12. コード・タイプ／コード名取得	21
dcs_typedto_name() dcs_nameto_type() dcs_multi_type_names()	
dcs_multi_type_names() dcs_all_type_names() dcs_all_type_names_length()	
A-13. BOMの追加／削除判定	22
dcs_cmp_bom()	
A-14. 変換不能コードの判定	23
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()*	
A-15. 半角カナ判定	23
is_jis_kana()* is_sjis_kana()* is_unibe_kana()*	
A-16. 半角カナ変換	24
jis_kana_to_sjis()* jis_kana_to_unibe()* sjis_kana_to_jis()*	
sjis_kana_to_unibe()* unibe_kana_to_jis()* unibe_kana_to_sjis()*	
A-17. 全角仮名判定	24
is_jis_zenkana()* is_sjis_zenkana()* is_euc_zenkana()* is_unibe_zenkana()*	
A-18. UTF-16 間変換	24
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()*	
A-19. UTF-16 から UTF-8 への変換	25
unibe_to_utf8_char()* unibe_to_utf8_short()* unile_to_utf8_char()*	
unile_to_utf8_short()*	
A-20. UTF-8(4byte) から UTF-16BE への変換	25
utf8_to_unibe_4bytes()*	
A-21. サロゲート・ペアからの変換	26
unibe_surrogate_to_utf8()* unibe_surrogate_to_cd()*	
A-22. 変換後も ASCII である出力タイプの判定	26
remains_ascii()*	

B. DcsIO

概要	27
get_version()	
文字コード判定／変換／仮想ファイル型関数	27
dcs_io_path() dcs_io_convert_file() dcs_io_open()	
dcs_io_close() dcs_io_get_stype() dcs_io_get_line()	
Dcs API 関数使用例	
一括処理型プログラム	28
dcs_io_convert_file() dcs_io_path() dcs_typedto_name()	
仮想ファイル型プログラム	30
dcs_io_open() dcs_io_get_line() dcs_io_close() dcs_io_get_stype()	
dcs_typedto_name() dcs_io_get_dtype()	

DcsIO 関数リファレンス

B- 1. ファイルの文字コード判定.....	33
dcs_io_path()	
B- 2. ファイルのコード変換.....	33
dcs_io_convert_file()	
B- 3. 処理ステータスのメッセージ取得.....	34
dcs_io_statusto_message()	
B- 4. コード・タイプ／コード名取得.....	34
dcs_io_typeto_name() dcs_io_nameto_type() dcs_io_multi_type_names()	
dcs_io_all_type_names() dcs_io_all_type_names_length()	
B- 5. BOMの追加／削除判定.....	36
dcs_io_cmp_bom()	
B- 6. 入力モード判定.....	36
dcs_io_is_can_read()	
B- 7. 出力モード判定.....	36
dcs_io_is_can_write()	
B- 8. 仮想ファイル・オープン.....	37
dcs_io_open()	
B- 9. 仮想ファイル・クローズ.....	37
dcs_io_close()	
B-10. 仮想ファイル1行入力.....	38
dcs_io_get_line()	
B-11. 仮想ファイル巻き戻し.....	38
dcs_io_rewind()	
B-12. コード変換後サイズ取得.....	38
dcs_io_get_size()	
B-13. 入力文字コード・タイプ取得.....	39
dcs_io_get_stype()	
B-14. 出力文字コード・タイプ取得.....	39
dcs_io_get_dtype()	
B-15. エラー番号取得.....	39
dcs_io_get_errno()	
B-16. エラー・メッセージ取得.....	40
dcs_io_get_strerror()	
B-17. 処理タイプ取得.....	40
dcs_io_get_ope_type()	
B-18. 判定結果の取得.....	40
dcs_io_is_decided()	
B-19. 判定文字コード・タイプ取得.....	41
dcs_io_get_ctype()	

C. Dcs クラス

概要.....	42
Dcs クラス関数使用例.....	42
IoNametoType() IoOpen() IoGetLine() IGetStype() IoTypetoName()	
IoIsCanRead() IoIsCanWrite() IoCmpBom() IGetErrno() IoGetStrerror()	
IoGetSize() IoGetDtype() IoClose() IoIsDecided() IoGetCtype() IoGetOpeType()	

Dcs クラス・リファレンス

C- 1. ファイルの文字コード判定	46
IoPath()	
C- 2. ファイルのコード変換	46
IoConvertFile()	
C- 3. 処理ステータスのメッセージ取得	47
IoStatustoMessage()	
C- 4. コード・タイプ／コード名取得	47
IoTypeetoName() IoNametoType() IoMultiTypeNames() IoAllTypeNames()	
IoAllTypeNamesLength()	
C- 5. BOMの追加／削除判定	49
IoCmpBom()	
C- 6. 入力モード判定	49
IoIsCanRead()	
C- 7. 出力モード判定	49
IoIsCanWrite()	
C- 8. 仮想ファイル・オープン	50
IoOpen()	
C- 9. 仮想ファイル・クローズ	50
IoClose()	
C-10. 仮想ファイル1行入力	51
IoGetLine()	
C-11. 仮想ファイル巻き戻し	51
IoRewind()	
C-12. コード変換後サイズ取得	51
IoGetSize()	
C-13. 入力文字コード・タイプ取得	52
IoGetType()	
C-14. 出力文字コード・タイプ取得	52
IoGetDtype()	
C-15. エラー番号取得	53
IoGetErrno()	
C-16. エラー・メッセージ取得	53
IoGetStrerror()	
C-17. 処理タイプ取得	53
IoGetOpeType()	
C-18. 判定結果の取得	54
IoIsDecided()	
C-19. 判定文字コード・タイプ取得	54
IoGetCtype()	
C-20. コード・タイプからMSの Charset 名を取得	54
GetMSCharset()	

付録

a-1. Dcs 定義	55
a-2. DcsIO 定義	59
a-3. Dcs クラス定義	59

※サンプルで特に断りのないものはOSに Linux を使用した。

A. Dcs API

概要

Dcs (DetectCharacterSets) 日本語文字コード判定／変換 API（以後は単に Dcs API 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。Dcs API 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。

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 3 種類のうち 2 つ (CP20932, eucJP-ms) 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、すべて合わせると以下の 20 本になる。

1. UTF16-BE 定義 (日本語 UTF8-MAC 定義を兼ねる)

ユニコードから他のコードへの変換

2. UTF16-BE => JIS (CP50221)
3. **UTF16-BE => JIS (CP50221) シフト・コード**
4. **UTF16-BE => JIS (ISO-2022-JP-2004)**
5. **UTF16-BE => JIS (同上) シフト・コード**
6. **UTF16-BE => JIS (同上) サロゲート文字**
7. UTF16-BE => SJIS (CP932)
8. UTF16-BE => EUC
9. UTF16-BE => EUC1 (CP20932)
10. UTF16-BE => EUC2 (CP51932)
11. UTF16-BE => EUC3 (eucJP-ms)

他のコードからユニコードへの変換

- (ユニコード以外の定義を兼ねる)
12. JIS (CP50221) => UTF16-BE
 13. **JIS2 (ISO-2022-JP-2004) => UTF16-BE**
 14. SJIS (CP932) => UTF16-BE
 15. EUC => UTF16-BE
 16. EUC1 (CP20932) => UTF16-BE
 17. EUC1 (CP20932) 3byte => UTF16-BE
 18. EUC2 (CP51932) => UTF16-BE
 19. EUC3 (eucJP-ms) => UTF16-BE
 20. EUC3 (eucJP-ms) 3byte => UTF16-BE

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、先にあげたテーブル仕様の変更にともない、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

以下にこれらの判定／変換関数の使用例として入力バッファーのものを見ていく。先に文字コード判定の簡単な例を示す。

```
/*
 * 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 (バッファー) */
        sts = dcs_judges_buffer(dcs, NULL, NULL, buffer, sizeof(buffer)-1, DCS_UNKNOWN);

        /* 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)
```


文字コード判定プログラム hello の解説

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。

なお、この例ではコード判定関数 `dcs_judges_buffer()` の戻り値のチェックは省略したが、一般的には正常値であることを確認する必要がある。次に、文字コードを変換する例を示す（少し長いので、説明のために行番号を付けた）。

```

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: /* des_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: /* des_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\x8b3\x82\xfb1\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:         /* バッファーによるコード変換(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:                 *(_cptr - 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:         }
87:
88:         /* Dcs 解放 */
89:         release_dcs(dcs);
90:     }
91:     return 0;
92: }

```

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)
```

文字コード変換プログラム `hello2` の解説

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 `release_dcs()` により Dcs 情報を開放して (89) プログラムを終了する。

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)。

テキストの 1 行入力の解説

コード・タイプの判明しているデータを行単位に抽出する。JIS, SJIS, EUC, UTF-16BE, UTF-16LE, UTF-8 の 6 本あるが JIS だけはシフト・コードの状態を保持する引数 (`int *shtype`) を余分に持つ。

テキストの 1 行入力

```
int dcs_jis_strln(Dcs *dcs, char *dbuf, size_t dsize, size_t *pdsiz, char **sbuf, size_t ssize, int *shtype);
int dcs_sjis_strln(Dcs *dcs, char *dbuf, size_t dsize, size_t *pdsiz, char **sbuf, size_t ssize);
```

以下に JIS での例を示す。

```

1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <sys/stat.h>
4: #include "dcs.h"
5:
6: int jis_proc(Dcs *dcs, char *path) {
7:     int shtype = 0;
8:     int sts, cent = 0;
9:     size_t rsize, dsize;
10:    char *cp, *ep;
11:    char *sptr, dest[256];
12:    struct stat stbuf;
13:    FILE *fp = fopen(path, "rb");
14:
15:    if (fp == NULL) {
16:        return DCS_EROPEN;
17:    }
18:    stat(path, &stbuf);
19:    rsize = stbuf.st_size;
20:    sptr = malloc(rsize);
21:    if (sptr == NULL) {
22:        fclose(fp);
23:        return DCS_EALLOC;
24:    }
25:    fread(sptr, 1, rsize, fp);
26:    if (ferror(fp)) {
27:        fclose(fp);
28:        return DCS_EREAD;
29:    }
30:    fclose(fp);
31:    cp = sptr;
32:    ep = sptr + rsize;
33:
34:    while(cp < ep) {
35:        sts = dcs_jis_strln(dcs, dest, sizeof(dest), &dsize, &cp, ep - cp, &shtype);
36:        if (sts < 0) {
37:            break;
38:        }
39:        cent += sts;
40:        .
41:        . /* 何らかの処理 */
42:        .
43:    }
44:    free(sptr);
45:    return(sts < 0? sts: cent);
46: }

```

Dcs 関数リファレンス

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

A-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。

A-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れない。

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:

 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 dcs_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。

A-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

```
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:

 dcs_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。

```
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:

 dcs_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。

```
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:

 dcs_judges_buffer() 等によるコード判定や dcs_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。

A-4. 不正文字出力抑止の判定／設定

```
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。

```
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。

A-5. 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: 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:

`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。

`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。

`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。

A-6. 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: des - 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:

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、stype や実際の文字コードが JIS や BOM 付きの UTF の場合は無効となる。

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。

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。

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。

A-7. 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。

A-8. 処理ステータ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 - 対象のステータス
 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タス 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。

A-9. 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:

get_no_char_count()
 変換ができなかった文字数が返る。

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。

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。

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。

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。

A-10. テキストの 1 行入力 (JIS, SJIS, EUC, UTF-16BE, UTF-16LE, UTF-8)

JIS のテキストから 1 行入力

```
int dcs_jis_strln(Dcs *dcs, char *dbuf, size_t dsize, size_t *pdsiz,
                  char **sbuf, size_t ssize, int *stype);
```

SJIS のテキストから 1 行入力

```
int dcs_sjis_strln(Dcs *dcs, char *dbuf, size_t dsize, size_t *pdsiz,
                   char **sbuf, size_t ssize);
```

EUC のテキストから 1 行入力

```
int dcs_euc_strln(Dcs *dcs, char *dbuf, size_t dsize, size_t *pdsiz,
                  char **sbuf, size_t ssize);
```

UTF-16BE のテキストから 1 行入力

```
int dcs_unibe_strln(Dcs *dcs, char *dbuf, size_t dsize, size_t *pdsiz,
                    char **sbuf, size_t ssize);
```

UTF-16LE のテキストから 1 行入力

```
int dcs_unile_strln(Dcs *dcs, char *dbuf, size_t dsize, size_t *pdsiz,
                    char **sbuf, size_t ssize);
```

UTF-8 のテキストから 1 行入力

```
int dcs_utf8_strln(Dcs *dcs, char *dbuf, size_t dsize, size_t *pdsiz,
                   char **sbuf, size_t ssize);
```

引き数: `dcs` - Dcs 情報
 `dbuf` - 出力バッファ
 `dsize` - 出力サイズ
 `pdsiz` - 出力データのバイト数
 `sbuf` - 入力バッファ
 `ssize` - 入力サイズ
 `stype` - J I S シフト・コード

戻り値: 抽出が成功した時は出力データの文字数（バイト数ではない）、不正なコードがあった時は `DCS_ECHAR` が返る。

機能:

入力バッファ `sbuf` から 1 行ずつ、バッファの最後までを出力バッファ `dbuf`, `dsize` に取得していく。出力されたバイト数は `pdsiz` に設定される。改行文字は含まれない。リターン後の `sbuf` は読み進んだ位置にあるので `ssize` を再計算する。

`stype` は解析中の J I S シフト・コードを表わし、初期値はゼロを設定する。解析中はアプリケーションで操作することはない。

A-11. Dcs API バージョン取得

```
float dcs_get_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。

A-12. コード・タイプ／コード名取得

文字コード名取得

```
int dcs_typeto_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);
```

引き数: **names** - 取得したコード名（ヌル終端）
 size - **names** のサイズ（ヌル終端文字を含む）
 ctype - 対象の文字コード・タイプ

戻り値: 取得した **names** の長さが返る。ヌル終端文字は含まない。

機能:

dcs_multi_type_names()
 ctype の値が一意の時は 1 つ、不特定のタイプが複数ある時は複数のコード名を取得する。
names が複数の時は **DCS_SEPARATE_CHAR** で区切られる。

dcs_all_type_names()
 すべてのコード名を取得する。**names** は **DCS_SEPARATE_CHAR** で区切られる。

すべてのコード名の長さを取得
int dcs_all_type_names_length();

引き数: なし

戻り値: すべてのコード名の長さ（ヌル終端文字を含む）

機能:

dcs_all_type_names() に渡すバッファのサイズを決めるための、ヌル終端文字を含む、すべてのコード名の長さを取得する。

A-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:

dcs_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_NON** が返る。

A-14. 変換不能コードの判定

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

`dcs_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。

A-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。

A-16. 半角カナ変換

<code>int jis_kana_to_sjis(int c);*</code>	- JIS から SJIS への半角カナ変換
<code>int jis_kana_to_unibe(int c);*</code>	- JIS から UTF-16BE への半角カナ変換
<code>int sjis_kana_to_jis(int c);*</code>	- SJIS から JIS への半角カナ変換
<code>int sjis_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。

A-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。

A-18. UTF-16 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>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

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。

A-19. UTF-16 から UTF-8 への変換

```
void unibe_to_utf8_char(char *dest, char *src, int bytes);*
    - UTF-16BE (文字列) から UTF-8 (文字列) への変換
void unibe_to_utf8_short(char *dest, int c, int bytes);*
    - UTF-16BE (数値) から UTF-8 (文字列) への変換
void unile_to_utf8_char(char *dest, char *src, int bytes);*
    - UTF-16LE (文字列) から UTF-8 (文字列) への変換
void unile_to_utf8_short(char *dest, int c, int bytes);*
    - 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。

A-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_EUEXPED が設定される。

A-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。

A-22. 変換後も ASCII である出力タイプの判定

```
bool remains_ascii(int dtype);*
```

引き数: dtype 判定する出力タイプ

戻り値: 入力データがASCII の場合、変換後も ASCII のまま変わらない出力タイプの時は真、それ以外は偽が返る。

B. DcsIO

概要

DcsIO は C言語や C++ 以外の環境から Dcs API にアクセスするためのラッパー・ライブラリーである。DcsIO は `dcs_io.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 DcsIO のすべての関数群を使用することができる。文字コードの判定や変換には独自の関数があり、入力ファイルを1回読んで処理を終えるものと、仮想ファイルとして変換データをメモリーにため込んだ後、アプリケーションから随時読み込むものがある。それらの関数の主なものを以下に示す。

文字コード判定／変換／仮想ファイル型関数

文字コード判定／変換関数

1. ファイルの文字コード判定

```
long dcs_io_path(const char *spath, long stype);
```

2. ファイルのコード変換

```
long dcs_io_convert_file(const char *spath, long stype, const char *dpath, long dtype);
```

仮想ファイル型関数

1. 仮想ファイル入出力オープン

```
long dcs_io_open(const char *path, long mode, long stype, long dtype, long *pstype);
```

2. テキスト入出力クローズ

```
long dcs_io_close(long fd);
```

3. 入力文字コード・タイプ取得

```
long dcs_io_get_stype(long fd);
```

4. 1行入力

```
long dcs_io_get_line(long fd, char *str, long size, long *psize);
```

DcsIO API 関数使用例

以上の関数を用いた一括処理型と仮想ファイル型の例を以下に示す。まずは一括処理型のプログラムから。

```

1: /*
2:  * 一括処理型プログラム
3:  */
4: #include <stdio.h>
5: #include <stdlib.h>
6: #include "dcs_io.h"
7:
8: int main(int argc, char *argv[]) {
9:     int i, sts;
10:    FILE *fp;
11:    char *spath, *dpath, str[256];
12:
13:    if (argc < 3) {
14:        fprintf(stderr, "Specify the I / O file path!");
15:        exit(1);
16:    }
17:    spath = argv[1];
18:    dpath = argv[2];
19:
20:    /* ファイルのコード変換 */
21:    sts = (int)dcs_io_convert_file(spath, DCS_UNKNOWN, dpath, DCS_UTF8, false);
22:    if (sts != DCS_SUCCESS) {
23:        /* 異常終了した時 */
24:
25:        fprintf(stderr, "io_convert_file() error!(%d)¥n", sts);
26:        exit(1);
27:    } else {
28:        /* 正常終了した時 */
29:
30:        /* 変換出力したファイルの頭を表示 */
31:        fp = fopen(dpath, "r");
32:        if (fp == NULL) {
33:            fprintf(stderr, "%s: can't open!", dpath);
34:            exit(1);
35:        }
36:        for(i=0;i<10;i++) {
37:
38:            /* ファイルの1行入力 */
39:            if (fgets(str, sizeof(str), fp) == NULL) {
40:                if (ferror(fp)) {
41:                    fprintf(stderr, "%s: read error!", dpath);
42:                    fclose(fp);
43:                    exit(1);
44:                }

```

```

45:         break;
46:     }
47:     if (fputs(str, stdout) == EOF) {
48:         fprintf(stderr, "stdout: write error!");
49:         fclose(fp);
50:         exit(1);
51:     }
52: }
53: fclose(fp);
54:
55: /* 入力ファイルの文字コード判定 */
56: sts = (int)dcs_io_path(spath, DCS_UNKNOWN);
57:
58: /* 入力文字コード名取得 */
59: dcs_typedto_name(str, sts);
60: printf("%n%Yn%s (%d: %s) => ", spath, sts, str);
61:
62: /* 出力ファイルの文字コード判定 */
63: sts = (int)dcs_io_path(dpath, DCS_UNKNOWN);
64:
65: /* 出力文字コード名取得 */
66: dcs_typedto_name(str, sts);
67: printf("%s (%d: %s) %Yn", dpath, sts, str);
68: }
69: return 0;
70: }

```

このプログラムを `tdcsiol` として実行すると以下の結果となった。ただし入力ファイルはインターネット図書館「青空文庫」にある夏目漱石「吾輩は猫である」に多少編集を施したものを使用した。テキストの文字コードは SJIS である。画面は UTF-8、80桁で折り返すものとする。

```

$ tdcsiol wagahaiwa_nekodearu.txt wagahaiwa_nekodearu_u8.txt
吾輩は猫である
夏目漱石

```

[# 8 字下げ] ー [# 「一」は中見出し]

吾輩《わがはい》は猫である。名前はまだ無い。

どこで生れたかとんと見当《けんとう》がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番「獯悪《どうあく》」な種族であったそう。この書生というのは時々我々を捕《つかま》えて煮《に》て食うという話である。しかしその当時は何という考もなかったから別段恐しいとも思わなかった。ただ彼の掌《てのひら》に載せられてスーと持ち上げられた時何だかフワフワした感じがあったばかりである。掌の上で少し落ちついて書生の顔を見たのがいわゆる人間というものの見始《みはじめ》であろう。この時妙なものだと思った感じが今でも残っている。第一毛をもって装飾されべきはずの顔がつるつるしてまるで

```
wagahaiwa_nekodearu.txt(4: SJIS) => wagahaiwa_nekodearu_u8.txt(15: UTF8)
```

一括処理型プログラム `tdcsio1` の解説

`DcsIO` の関数は内部で `Dcs` を生成し、処理を行なうので、アプリケーションが直接扱うことはない。引き数で渡された入出力ファイル・パスにより出力文字コードを UTF-8 として、ファイルのコード変換 `dcs_io_convert_file()` を呼び出す (21)。変換処理が成功したことを確認し (22)、変換出力したファイルの頭を表示する (31-54 `fopen()` ~ `fclose()`)。最後に入出力ファイルの文字コード判定を行なう (56-67)。次に、仮想ファイル型のプログラム例を示す。

```

1: /*
2:  * 仮想ファイル型プログラム
3:  */
4: #include <stdlib.h>
5: #include "dcs_io.h"
6:
7: int main(int argc, char *argv[]) {
8:     int i, sts;
9:     long fd, stype, dsize;
10:    char *path, str[128];
11:
12:
13:    if (argc < 2) {
14:        fprintf(stderr, "Specify the file path!");
15:        exit(1);
16:    }
17:    path = argv[1];
18:
19:    /* 仮想ファイル入出力オープン */
20:    fd = dcs_io_open(path, DMD_READ, DCS_UNKNOWN, DCS_UTF8, false, &stype);
21:    if (fd < 0) {
22:        /* 仮想ファイルのオープンに失敗した時 */
23:
24:        fprintf(stderr, "io_dcs_io_open() error! (%ld) %n", fd);
25:        exit(1);
26:    } else {
27:        /* 仮想ファイルが正常にオープンした時 */
28:
29:        for (i=0; i<15; i++) {
30:
31:            /* 仮想ファイル1行入力 */
32:            sts = (int)dcs_io_get_line(fd, str, 120, &dsize);
33:            if (sts == DCS_EOF) {
34:                /* EOF の時 */
35:                break;
36:            } else if (sts < 0) {
37:                fprintf(stderr, "dcs_io_get_line error! (%d) %n", sts);
38:                dcs_io_close(fd);
39:                exit(1);
40:            }

```

```

41:         /* 出力データをヌル・ターミネート */
42:         *(str + dsize) = 0;
43:         if (puts(str) == EOF) {
44:             fprintf(stderr, "stdout: write error!");
45:             dcs_io_close(fd);
46:             exit(1);
47:         }
48:     }
49:     /* 入力文字コード・タイプ取得 */
50:     sts = (int)dcs_io_get_stype(fd);
51:
52:     /* 入力文字コード名取得 */
53:     dcs_typedto_name(str, sts);
54:     printf("%n%s(%d: %s) => ", path, sts, str);
55:
56:     /* 出力文字コード・タイプ取得 */
57:     sts = (int)dcs_io_get_dtype(fd);
58:
59:     /* 出力文字コード名取得 */
60:     dcs_typedto_name(str, sts);
61:     printf("stdout(%d: %s)%n", sts, str);
62:
63:     /* 仮想ファイルのクローズ */
64:     dcs_io_close(fd);
65: }
66: return 0;
67: }

```

このプログラムを `tdcsio2` として実行すると以下の結果となった。ただし入力ファイルは `tdcsio1` と同じ夏目漱石「吾輩は猫である」。

```

$ tdcio2 wagahaiwa_nekodearu.txt
吾輩は猫である
夏目漱石

```

[# 8 字下げ] 一 [# 「一」は中見出し]

吾輩《わがはい》は猫である。名前はまだ無い。

どこで生れたかとうと見当《けんとう》がつかぬ。何でも薄暗いじめじめした所でニャーニャー泣いていた事だけは記憶している。吾輩はここで始めて人間というものを見た。しかもあとで聞くとそれは書生という人間中で一番 | 獐悪《どうあく》な種族であったそうだ。この書生というのは時々我々を捕《つかま》えて煮《に》て食うという話である。しかしその当時は何という考もなかったから別段恐しいとも思わなかった。ただ彼の掌《てのひら》に載せられてスーと持ち上げられた時何だかフワフワした感じがあったばかりである。掌の上で少し落ちついて書生の顔を見たのがいわゆる人間というものの見始《みはじめ》であろう。この時妙なものだと思った感じが今でも残っている。第一毛をもって装飾されべきはずの顔がつるつるしてまるで薬缶《やかん》だ。その後《ご》猫にもだい

```
wagahaiwa_nekodearu.s.txt(4: SJIS) => stdout(15: UTF8)
```

仮想ファイル型プログラム `tdcsio2` の解説

引き数で渡された入力ファイル・パスにより出力文字コードを UTF-8 として、仮想ファイル入出力オープン `dcs_io_open()` を呼び出す (20)。変換処理が成功したことを確認し (21)、仮想ファイルの 1 行入力 `dcs_io_get_line()`、`puts()` によりデータの頭を表示し (32-47)、`dcs_io_get_stype()`、`dcs_io_get_dtype()` により入出力データの文字コードを取得 (50-61)。最後に `dcs_io_close()` により仮想ファイルを閉じる (64)。

今回は `dcs_io_get_line()` による入力データのバイト数を調整し、80 桁ずつ改行を入れて表示した。UTF-8 の全角は 1 文字 3 バイトなので、1 行分の入力バッファサイズは $120 (3 \times 80 / 2)$ バイトとなる。

DcsIO 関数リファレンス

B-1. ファイルの文字コード判定

```
long dcs_io_path(const char *spath, long stype);
```

引き数: spath - 判定対象の入力ファイル・パス
 stype - 入力データの文字コード・タイプ

呼び出す Dcs 関数: dcs_judges_file()

戻り値: 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:

引き数 spath の入力ファイルのデータから文字コードの判定を行なう。
 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。

B-2. ファイルのコード変換

```
long dcs_io_convert_file(const char *spath, long stype, const char *dpath, long dtype);
```

引き数: spath - 入力ファイル・パス
 stype - 入力データの文字コード・タイプ
 dpath - 出力ファイル・パス
 dtype - 出力データの文字コード・タイプ

呼び出す Dcs 関数: dcs_convert_file()

戻り値: 変換が成功した場合は DCS_SUCCESS、コード判定エラー (CONFIRMED_CODE_TYPE(sts) が偽) の時はその値、メモリー・アロケート・エラーの時は DCS_EALLOC、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。

機能:

初めに入力データのコード判定を行い、一意に決定した場合は引き数 spath の入力ファイルのデータを指定された dtype の文字コードに変換し、dpath に出力する。

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。

B-3. 処理ステータスのメッセージ取得

```
long dcs_io_statusto_message(char *str, long sts, long ctype);
```

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

呼び出す Dcs 関数: dcs_statusto_message()

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タス 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。

B-4. コード・タイプ／コード名取得

文字コード名取得

```
long dcs_io_typeto_name(char *name, long ctype);
```

引き数: name - 取得した文字コード名 (ヌル終端)
 type - 対象の文字コード・タイプ

呼び出す Dcs 関数: dcs_typeto_name()

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・タイプから名前を取得する。

文字コード・タイプ取得

```
long dcs_io_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。

呼び出す Dcs 関数: dcs_nameto_type()

機能:

文字コード名から文字コード・タイプを取得する。

不特定の文字コード名を取得

```
long dcs_io_multi_type_names(char *names, long size, long ctype);
```

すべての文字コード名を取得

```
long dcs_io_all_type_names(char *names, long size);
```

引き数: names - 取得した文字コード名 (ヌル終端)
 size - names のサイズ (ヌル終端文字を含む)
 ctype - 対象の文字コード・タイプ

呼び出す Dcs 関数: dcs_multi_type_names(), dcs_all_type_names()

戻り値: 取得した names の長さが返る。ヌル終端文字は含まない。

機能:

```
dcs_io_multi_type_names()
```

ctype が一意の時は 1 つ、不特定のタイプが複数ある時は複数のコード名を取得する。names が複数の時は DCS_SEPARATE_CHAR で区切られる。

```
dcs_io_all_type_names()
```

すべてのコード名を取得する。names は DCS_SEPARATE_CHAR で区切られる。

すべてのコード名の長さを取得

```
long dcs_io_all_type_names_length();
```

引き数: なし

呼び出す Dcs 関数: dcs_all_type_names_length()

戻り値: すべてのコード名の長さ (ヌル終端文字を含む)

機能:

dcs_io_all_type_names() に渡すバッファのサイズを決めるための、すべてのコード名の長さを取得する。

B-5. BOMの追加／削除判定

```
int dcs_io_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

呼び出す Dcs 関数: dcs_cmp_bom()

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:

dcs_io_convert_file() 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_NON が返る。

.....

B-6. 入力モード判定

```
long dcs_io_is_can_read(long mode);
```

引き数: mode - 判定する入力モード

戻り値: mode が DMD_READ 又は DMD_MODIFY の時は真、それ以外は偽が返る。

機能:

dcs_io_open() に渡す入力モードの有効な値を判定する。

.....

B-7. 出力モード判定

```
long dcs_io_is_can_write(long mode);
```

引き数: mode - 判定する出力モード

戻り値: mode が DMD_WRITE 又は DMD_APPEND、DMD_MODIFY の時は真、それ以外は偽が返る。

機能:

dcs_io_open() に渡す出力モードの有効な値を判定する。

B-8. 仮想ファイル入出力オープン

```
long dcs_io_open(const char *path, long mode, long stype, long dtype, long *pstype);
```

引き数: path - 入出力対象のファイル・パス
 mode - 入出力モード
 stype - 入力データの文字コード・タイプ
 dtype - 出力文字コード・タイプ
 pstype - 判定された入力文字コード

呼び出す Dcs 関数: `create_dcs()`, `dcs_convert_file()`

戻り値: 正しくオープンされた時は仮想ファイル記述子、それ以外は負のエラー・ステータスが返る。

機能:

初めに path の入力ファイルのデータのコード判定を行い、一意に決定した場合はデータを指定された dtype の文字コードに変換し、mode で指定された入出力モードで仮想オープンする（ただし、仮想ファイルによるアクセスではバージョン2.11 現在、入力のみで出力機能はない）。

stype は `dcs_io_path()` 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。

判定結果は pstype に書かれる。不要ならば `NULL` を設定する。

B-9. テキスト入出力クローズ

```
long dcs_io_close(long fd);
```

引き数: fd - 仮想ファイル記述子

呼び出す Dcs 関数: `release_dcs()`

戻り値: 正しくクローズされた時は `DCS_SUCCESS`、それ以外は負のエラー・ステータスが返る。

機能:

引き数 fd に対する資源をすべて開放する。

B-10. 仮想ファイル 1 行入力

```
long dcs_io_get_line(long fd, char *str, long size, long *psize);
```

引き数: fd - 仮想ファイル記述子
 str - 入力バッファ
 size - 入力サイズ
 psize - 入力したバイト数。不要ならば NULL を設定する。

呼び出す Des 関数: dcs_jis_strln(), dcs_sjis_strln(), dcs_euc_strln(),
 dcs_unibe_strln(), dcs_unile_strln(), dcs_utf8_strln()

戻り値: 正しく入力されたら一行の文字数（空行はゼロ）を返し、データの終わりに達したら
 DCS_EOF、それ以外は負のエラー・ステータスが返る。

機能:

引き数 fd の現在位置から 1 行を str, size に読み込む。改行文字は含まれない。

B-11. 入力ファイル巻き戻し

```
long dcs_io_rewind(long fd);
```

引き数: fd - 仮想ファイル記述子

戻り値: 正しく巻き戻された時は DCS_SUCCSES、それ以外は負のエラー・ステータスが返る。

機能:

引き数 fd の入力位置を先頭に戻し、入力ファイルの巻き戻しを行なう。

B-12. コード変換後サイズ取得

```
long dcs_io_get_size(long fd);
```

引き数: fd - 仮想ファイル記述子

戻り値: dcs_io_open() で正しく変換されていれば変換後のサイズ、それ以外は負のエラー・ス
 テータスが返る。

機能:

引き数 fd のコード変換後のサイズを取得する。

B-13. 入力文字コード・タイプ取得

```
long dcs_io_get_stype(long fd);
```

引き数: fd - 仮想ファイル記述子

戻り値: dcs_io_open() で正しく変換されていれば入力文字コード・タイプ、それ以外は負のエラー・ステータスが返る。

機能:

引き数 fd の入力文字コード・タイプを取得する。

B-14. 出力文字コード・タイプ取得

```
long dcs_io_get_dtype(long fd);
```

引き数: fd - 仮想ファイル記述子

戻り値: dcs_io_open() で正しく変換されていれば出力文字コード・タイプ、それ以外は負のエラー・ステータスが返る。

機能:

引き数 fd の出力文字コード・タイプを取得する。

B-15. エラー番号取得

```
long dcs_io_get_errno(long fd);
```

引き数: fd - 仮想ファイル記述子

呼び出す Dcs 関数: dcs_get_errno()

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。

B-16. エラー・メッセージ取得

```
long dcs_io_get_strerror(long fd, long err_no, char *str, long size, long ctype);
```

引き数: fd - 仮想ファイル記述子
 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

呼び出す Dcs 関数: dcs_get_strerror()

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。

B-17. 処理タイプ取得

```
long dcs_io_get_ope_type();
```

引き数: なし

戻り値: コード変換の処理タイプが返る。

機能:

dcs_convert_file(), dcs_io_open() により最後に行なわれたコード変換の処理タイプが返る。

B-18. 判定結果の取得

```
long dcs_io_is_decided();
```

引き数: なし

戻り値: 入力文字コード・タイプの判定結果が返る。

機能:

dcs_convert_file(), dcs_io_open() により最後に行なわれたコード変換前の判定で、入力文字コード・タイプが特定できた時は真、それ以外は偽が返る。

B-19. 判定文字コード・タイプ取得

```
long dcs_io_get_ctype();
```

引き数: なし

戻り値: 判定の文字コード・タイプが返る。

機能:

`dcs_convert_file()`, `dcs_io_open()` により最後に行なわれたコード変換前の判定での文字コード・タイプが返る。

C. Dcs クラス

概要

Dcs クラスは VBA から DcsIO にアクセスするための VBA クラスである。DcsIO 19関数すべてに対応しており、加えて Dcs コード・タイプからMSの Charset 名を取得する関数が含まれる。対応する19関数の DcsIO からDcs クラスへの変換規則は以下の通り。

1. プレフィックス "dcs" 削除
2. 単語の頭文字を大文字に置換
3. アンダーバー "_" 削除

例：dcs_io_open => IoOpen

Dcs クラス関数使用例

以下は Dcs ソフトに付属する、Dcs クラスのテスト用シート dcs_tester.xlsm から抜粋した関数 PrintText() である。あらかじめ設定されているコード変換用のファイル・パスを引き数として呼び出され、IoOpen() によりコード変換を行ない、IoGetLine() で1行80桁として先頭から100行表示する。

```

1: ' テキスト・ファイルの表示
2: Function PrintText(ByRef pctype As Long, ByRef pcname As String, path As String) As String
3:     On Error GoTo ErrorHandler
4:
5:     Dim str As String
6:     Dim lineno As Long
7:
8:     Dim sts As Long, fd As Long, rcs As Long, stype As Long
9:     Dim pdc As Dcs
10:
11:     fd = -1
12:     If gdc Is Nothing Then
13:         ' 生成されていなければ、Dcs を生成
14:         Set gdc = New Dcs
15:     End If
16:
17:     Set pdc = gdc
18:     ' コード名からタイプを取得
19:     stype = pdc.IoNameToType(Range("B3").Value)
20:     ' 名前が不正の時、UNKNOWNとする
21:     If stype < 0 Then stype = pdc.UNKNOWN
22:

```

```

23:      ' 仮想ファイル・オープン
24:      fd = pdc$.IoOpen(path, pdc$.DMD_READ, stype, pdc$.SJIS, False, res)
25:
26:      ' エラーの時、抜ける
27:      If fd < 0 Then Err.Raise fd
28:
29:      ' 100行クリア
30:      bClearClicked
31:
32:      Dim DCS_EOF As Long
33:      DCS_EOF = pdc$.EOF
34:
35:  '' 変換データ 100件をシートに表示
36:      Do While True
37:          ' 100行書いたら、抜ける
38:          If lineno >= 100 Then Exit Do
39:
40:          ' 1行読み込み
41:          sts = pdc$.IoGetLine(fd, str, 80)
42:
43:          ' EOF の時、抜ける
44:          If sts = DCS_EOF Then Exit Do
45:
46:          ' エラーの時、抜ける
47:          If sts < 0 Then Err.Raise sts
48:
49:          lineno = lineno + 1
50:
51:      '' 変換データをシートに表示
52:      ActiveSheet.Cells(lineno + 2, 4).Value = str
53:  Loop
54:
55:      Dim sctype As Long, dctype As Long
56:      Dim cs As Long
57:      Dim errorNo As Long
58:      Dim message As String
59:      Dim sname As String, dname As String
60:      Dim cname As String
61:
62:  '' 入力文字コード判定結果取得
63:      sctype = pdc$.IoGetType(fd)
64:  '' 判定結果を返却パラメータに設定
65:      pctype = sctype
66:  '' 入力文字コード・タイプから名前を返却パラメータに設定
67:      Call pdc$.IoTypeToName(pcname, sctype)
68:  '' MSの Charset 名を戻り値に設定
69:      PrintText = pdc$.GetMSCharset(sctype)
70:
71:  '' 入出力モード取得
72:      Range("B10").Value = pdc$.IoIsCanRead(pdc$.DMD_READ)
73:      Range("B11").Value = pdc$.IoIsCanWrite(pdc$.DMD_READ)
74:  '' BOMの追加／削除判定取得
75:      Range("B12").Value = pdc$.IoCmpBom(stype, pdc$.SJIS)
76:  '' 入力ファイル・エラー番号／文字列取得
77:      errorNo = pdc$.IoGetErrno(fd)
78:      Call pdc$.IoGetStrerror(fd, errorNo, message)
79:      Range("B13").Value = errorNo & ":" & message
80:
81:  '' コード変換後サイズ取得

```

```

82:     Range("B14").Value = pdc$.IoGetSize(fd)
83:
84: '' 出力文字コード取得
85:     dtype = pdc$.IoGetType(fd)
86: '' 出力文字コード・タイプから名前を取得
87:     Call pdc$.IoTypeToName(dname, dtype)
88: '' 入出力タイプ名
89:     Range("B15").Value = sctype & ":" & pname
90:     Range("B16").Value = dtype & ":" & dname
91:
92: '' ファイル・クローズ
93:     sts = pdc$.IoClose(fd)
94:
95: '' 直近の情報を取得（クローズ後可）
96:
97: '' 判定結果の取得（入力文字コードを特定できたら TRUE）
98:     Range("B17").Value = pdc$.IoIsDecided()
99: '' 判定コード取得
100:     cs = pdc$.IoGetType()
101: '' コード・タイプから名前を取得
102:     Call pdc$.IoTypeToName(cname, cs)
103:     Range("B18").Value = cs & ":" & cname
104: '' 処理タイプ
105:     Range("B19").Value = pdc$.IoGetType()
106:
107: ' エラーの時、抜ける
108:     If sts < 0 Then Err.Raise sts
109:
110:     Exit Function
111:
112: ErrorHandler:
113:     If Err.Number = pdc$.ECTYPE Then
114:         ' コード・タイプ誤りの時
115:         Dim ds As Long
116:         Dim errors As String
117:
118:         PrintText = pdc$.GetMSCharset(res)
119:         CSError res
120:     Else
121:         MsgBox "PrintText: " & path & ": error(" & Err.Number & ") " _
122:             & Err.Description, vbYes + vbCritical, "dcs_test: エラー"
123:     End If
124:     ' オープンしていたらクローズ
125:     If fd >= 0 Then pdc$.IoClose(fd)
126:
127: End Function ' PrintText

```

Dcs クラスのテスト用関数 `PrintText()` の解説

初めて処理する時は Dcs クラスを生成 (12, 14)。シートに指定されている入力データの文字コード名から `IoNameToType()` により文字コード・タイプを取得 (19)。引き数で渡された `path` と、取得した入力文字コード・タイプ `stype`、出力文字コード・タイプ `Dcs.SJIS` により仮想ファイル・オープン `IoOpen()` を呼び出す (24)。表示領域を、クリア・ボタンのコール・バック関数 `bClearClicked()` を直接呼び出してクリア (30)。入力バッファ80桁として `IoGetLine()` を呼出しながら100行表示する (36~53)。`IoGetType()` により入力文字コード判定結果を返却パラメータ `pctype` に設定 (63, 65)。判定された入力文字コードから `IoTypeToName()` により名前を返却パラメータ `pcname` に設定 (67)。同じく入力文字コードから `GetMSCharset()` によりMSの Charset 名を戻り値に設定 (69)。その他の変換情報をシートに設定 (72~90)。`IoClose()` により仮想ファイルをクローズ (93)。クローズ後に取得できる直近の情報をシートに設定 (98~105)。

Dcs クラス・リファレンス

C-1. ファイルの文字コード判定

Public Function IoPath(ByVal path As String, ByVal ctype As Long) As Long

引き数: path - 判定対象の入力ファイル・パス
 ctype - 入力データの文字コード・タイプ

呼び出す DcsIO 関数: dcs_io_path()

戻り値: 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:

引き数 path の入力ファイルのデータから文字コードの判定を行なう。
 ctype 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。

C-2. ファイルのコード変換

**Public Function IoConvertFile(ByVal spath As String, ByVal stype As Long, _
 ByVal dpath As String, ByVal dtype As Long, ByVal isForced As Boolean) As Long**

引き数: spath - 入力ファイル・パス
 stype - 入力データの文字コード・タイプ
 dpath - 出力ファイル・パス
 dtype - 出力データの文字コード・タイプ
 isForced - 入力データの文字コード・タイプを強制的に stype とする

呼び出す DcsIO 関数: dcs_io_convert_file()

戻り値: 変換が成功した場合は Dcs.SUCCESS、コード判定エラー (Dcs.CONFIRMED_CODE_TYPE(sts) が偽) の時はその値、メモリー・アロケート・エラーの時は Dcs.EALLOC、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。

機能:

初めに入力データのコード判定を行い、一意に決定した場合は引き数 spath の入力ファイルのデータを指定された dtype の文字コードに変換し、dpath に出力する。

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。isForced 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、stype や実際の文字コードが JIS や BOM 付きの UTF の場合は無効となる。

C-3. 処理ステータスのメッセージ取得

Public Function IoStatustoMessage (ByRef str As String, ByVal sts As Long) As Long

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

呼び出す DcsIO 関数: dcs_io_statusto_message()

戻り値: 取得されたメッセージの長さが返る。

機能:

引き数で渡された処理ステータス 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。

C-4. コード・タイプ／コード名取得

コード・タイプ名取得

Public Function IoTypetoName (ByRef name As String, ByVal ctype As Long) As Long

引き数: name - 取得した文字コード名
 ctype - 対象の文字コード・タイプ

呼び出す DcsIO 関数: dcs_io_typeto_name()

戻り値: 取得した name の長さが返る。

機能:

文字コード・タイプから名前を取得する。

文字コード・タイプ取得

Public Function IoNametoType (ByVal name As String) As Long

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。

呼び出す DcsIO 関数: dcs_io_nameto_type()

機能:

文字コード名から文字コード・タイプを取得する。

不特定の文字コード名を取得

Public Function IoMultiTypeNames (ByRef names As String, ByVal ctype As Long) As Long

すべての文字コード名を取得

Public Function IoAllTypeNames (ByRef names As String) As Long

引き数: names - 取得した文字コード名 (ヌル終端)
size - names のサイズ (ヌル終端文字を含む)
ctype - 対象の文字コード・タイプ

呼び出す DcsIO 関数: dcs_io_multi_type_names(), dcs_io_all_type_names()

戻り値: 取得した names の長さが返る。

機能:

IoMultiTypeNames ()

ctype が一意の時は 1 つ、不特定のタイプが複数ある時は複数のコード名を取得する。names が複数の時は Dcs.SEPARATE_CHAR で区切られる。

IoAllTypeNames ()

すべてのコード名を取得する。names は Dcs.SEPARATE_CHAR で区切られる。

すべてのコード名の長さを取得

Public Function IoAllTypeNamesLength () As Long

引き数: なし

呼び出す DcsIO 関数: dcs_io_all_type_names_length()

戻り値: すべてのコード名の長さ (ヌル終端文字を含む)

機能:

すべてのコード名の長さを取得する。

C-5. BOMの追加／削除判定

Public Function IoCmpBom (ByVal stype As Long, ByVal dtype As Long) As Long

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

呼び出す DcsIO 関数: dcs_io_cmp_bom()

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:

IoConvertFile() 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_NON が返る。

C-6. 入力モード判定

Public Function IoIsCanRead (ByVal mode As Long) As Boolean

引き数: mode - 判定する入力モード

呼び出す DcsIO 関数: dcs_io_is_can_read()

戻り値: mode が DMD_READ 又は DMD_MODIFY の時は真、それ以外は偽が返る。

機能:

IoOpen() に渡す入力モードの有効な値を判定する。

C-7. 出力モード判定

Public Function IoIsCanWrite (ByVal mode As Long) As Boolean

引き数: mode - 判定する出力モード

呼び出す DcsIO 関数: dcs_io_is_can_write()

戻り値: mode が DMD_WRITE 又は DMD_APPEND、DMD_MODIFY の時は真、それ以外は偽が返る。

機能:

IoOpen() に渡す出力モードの有効な値を判定する。

C-8. 仮想ファイル・オープン

Public Function IoOpen(ByVal path As String, ByVal mode As Long, ByVal stype As Long, _
ByVal dtype As Long, ByVal isForced As Boolean, ByRef pstype As Long) As Long

引き数: path - 入出力対象のファイル・パス
mode - 入出力モード
stype - 入力データの文字コード・タイプ
dtype - 出力文字コード・タイプ
isForced - 入力データの文字コード・タイプを強制的に stype とする
pstype - 判定された入力文字コード。

呼び出す DcsIO 関数: dcs_io_open()

戻り値: 正しくオープンされた時は仮想ファイル記述子、それ以外は負のエラー・ステータスが返る。

機能:

初めに path の入力ファイルのデータのコード判定を行い、一意に決定した場合はデータを指定された dtype の文字コードに変換し、mode で指定された入出力モードで仮想オープンする（ただし、仮想ファイルによるアクセスではバージョン2.10 現在、入力のみで出力機能はない）。

stype は IoPath() 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。isForced 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、stype や実際の文字コードが JIS や BOM 付きの UTF の場合は無効となる。

判定結果は pstype に書かれる。不要ならば Null を設定する。

C-9. 仮想ファイル・クローズ

Public Function IoClose(ByVal fd As Long) As Long

引き数: fd - 仮想ファイル記述子

呼び出す DcsIO 関数: dcs_io_close()

戻り値: 正しくクローズされた時は Dcs.SUCCESES、それ以外は負のエラー・ステータスが返る。

機能:

引き数 fd に対する資源をすべて開放する。

C-10. 仮想ファイル 1 行入力

```
Public Function IoGetLine (ByVal fd As Long, ByVal buf As String, ByVal size As Long) _  
As Long
```

引き数: fd - 仮想ファイル記述子
 str - 入力バッファ
 size - 入力サイズ

呼び出す DcsIO 関数: dcs_io_get_line()

戻り値: 正しく入力されたら一行の文字数（空行はゼロ）を返し、データの終わりに達したら
 Dcs.EOF、それ以外は負のエラー・ステータスが返る。

機能:

引き数 fd の現在位置から 1 行を str, size に読み込む。改行文字は含まれない。

C-11. 仮想ファイル巻き戻し

```
Public Function IoRewind (ByVal fd As Long) As Long
```

引き数: fd - 仮想ファイル記述子

呼び出す DcsIO 関数: dcs_io_rewind()

戻り値: 正しく巻き戻された時は Dcs.SUCCESES、それ以外は負のエラー・ステータスが返る。

機能:

引き数 fd の入力位置を先頭に戻し、入力ファイルの巻き戻しを行なう。

C-12. コード変換後サイズ取得

```
Public Function IoGetSize (ByVal fd As Long) As Long
```

引き数: fd - 仮想ファイル記述子

呼び出す DcsIO 関数: dcs_io_get_size()

戻り値: IoOpen() で正しく変換されていれば変換後のサイズ、それ以外は負のエラー・ステータ
 スが返る。

機能:

引き数 fd のコード変換後のサイズを取得する。

.....

C-13. 入力文字コード・タイプ取得

Public Function IoGetStype (ByVal fd As Long) As Long

引き数: fd - 仮想ファイル記述子

呼び出す DcsIO 関数: dcs_io_get_stype()

戻り値: IoOpen() で正しく変換されていれば入力文字コード・タイプ、それ以外は負のエラー・ステータスが返る。

機能:

引き数 fd の入力文字コード・タイプを取得する。

.....

C-14. 出力文字コード・タイプ取得

Public Function IoGetDtype (ByVal fd As Long) As Long

引き数: fd - 仮想ファイル記述子

呼び出す DcsIO 関数: dcs_io_get_dtype()

戻り値: IoOpen() で正しく変換されていれば出力文字コード・タイプ、それ以外は負のエラー・ステータスが返る。

機能:

引き数 fd の出力文字コード・タイプを取得する。

C-15. エラー番号取得

Public Function IoGetErrno (ByVal fd As Long) As Long

引き数: fd - 仮想ファイル記述子

呼び出す DcsIO 関数: dcs_io_get_errno()

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。

C-16. エラー・メッセージ取得

**Public Function IoGetStrerror (ByVal fd As Long, ByVal errNo As Long, _
ByRef str As String) As Long**

引き数: fd - 仮想ファイル記述子

errNo - 対象のエラー番号

str - 出力バッファ

呼び出す DcsIO 関数: dcs_io_get_strerror()

戻り値: 取得されたメッセージの長さが返る。

機能:

引き数で渡されたエラー番号 errNo から、Dcs.SJIS に変換されたメッセージを取得する。

C-17. 処理タイプ取得

Public Function IoGetOpeType () As Long

引き数: なし

戻り値: コード変換の処理タイプが返る。

機能:

IoConvertFile(), IoOpen() により最後に行なわれたコード変換の処理タイプが返る。

C-18. 判定結果の取得

Public Function IoIsDecided() As Boolean

引き数: なし

呼び出す DcsIO 関数: dcs_io_is_decided()

戻り値: 入力文字コード・タイプの判定結果が返る。

機能:

IoConvertFile(), IoOpen() により最後に行なわれたコード変換前の判定で、入力文字コード・タイプが特定できた時は真、それ以外は偽が返る。

C-19. 判定文字コード・タイプ取得

Public Function IoGetCtype() As Long

引き数: なし

呼び出す DcsIO 関数: dcs_io_get_ctype()

戻り値: 判定の文字コード・タイプが返る。

機能:

IoConvertFile(), IoOpen() により最後に行なわれたコード変換前の判定での文字コード・タイプが返る。

C-20. コード・タイプからMSの Charset 名を取得

Public Function GetMSCharset(ctype As Long) As String

引き数: ctype - 文charset 名を取得する文字コード・タイプ

戻り値: MSの Charset 名

機能:

DCS で定義されている文字コード・タイプから、対応するMSの Charset 名を取得する。未定義の場合はアスキー "us-ascii" が返る。

付録

a-1. Dcs 定義

以下にヘッダー・ファイル dcs.h に定義されている値を示す。

DCS バージョン

```
#define DCS_VERSION ((float)2.11)
```

DCS 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 CP50221 */
#define DCS_JIS2             3 /* JIS ISO-2022-JP-2004 */
#define DCS_SJIS             4 /* SJIS */
#define DCS_EUC              5 /* EUC */
#define DCS_EUC1             6 /* EUC CP20932 */
#define DCS_EUC2             7 /* EUC CP51932 */
#define DCS_EUC3             8 /* EUC-MS */
#define DCS_UTF16BE          11 /* UTF-16 ビッグ・エンディアン */
#define DCS_UTF16BE_BOM      12 /* UTF-16 ビッグ・エンディアン (BOM付) */
#define DCS_UTF16LE          13 /* UTF-16 リトル・エンディアン・ */
#define DCS_UTF16LE_BOM      14 /* UTF-16 リトル・エンディアン (BOM付) */
#define DCS_UTF8             15 /* UTF-8 */
#define DCS_UTF8_BOM         16 /* UTF-8 (BOM付) */
#define DCS_NORMAL           00000000077 /* 確定コード・タイプの最大値 */

```

未確定コード・タイプ (マスク値)

```

#define DCS_ND_ASCII         00000000100 /* ASCII */
#define DCS_ND_JIS           00000000200 /* JIS CP50221 */
#define DCS_ND_JIS2          00000000400 /* JIS ISO-2022-JP-2004 */
#define DCS_ND_JIS_ALL       00000000600 /* JIS ALL */
#define DCS_ND_SJIS          00000001000 /* SJIS */
#define DCS_ND_EUC           00000002000 /* EUC-JP */
#define DCS_ND_EUC1          00000004000 /* EUC CP20932 */
#define DCS_ND_EUC2          00000010000 /* EUC CP51932 */
#define DCS_ND_EUC3          00000020000 /* eucJP-ms */
#define DCS_ND_EUC_ALL       00000036000 /* EUC ALL */
#define DCS_ND_UTF16BE       00000200000 /* UTF-16 ビッグ・エンディアン */
#define DCS_ND_UTF16BE_BOM   00000400000 /* UTF-16 ビッグ・エンディアン (BOM付) */
#define DCS_ND_UTF16LE       00001000000 /* UTF-16 リトル・エンディアン・ */
#define DCS_ND_UTF16LE_BOM   00002000000 /* UTF-16 リトル・エンディアン (BOM付) */
#define DCS_ND_UTF8          00004000000 /* UTF-8 */
#define DCS_ND_UTF8_BOM      00010000000 /* UTF-8 (BOM付) */
#define DCS_ND_UTF_BOM       00012400000 /* UTF (BOM付) */
#define DCS_ND_ALL           00017777700 /* ND ALL */

```

確定コード・タイプ

```

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

```


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             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 */
```

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-16BE BOM

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

UTF-8 BOM

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

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  /* 終了片仮名 */
#define SJIS_KKANA_B     0x8340  /* 開始平仮名 */
#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  /* 終了片仮名 */
#define EUC_KKANA_B      0xa5a1  /* 開始平仮名 */
#define EUC_KKANA_E      0xa5f3  /* 終了片仮名 */
```

UTF-16BE 全角仮名領域

```
#define UNIBE_HKANA_B     0x3041  /* 開始平仮名 */
#define UNIBE_HKANA_E     0x3093  /* 終了片仮名 */
#define UNIBE_KKANA_B     0x30a1  /* 開始平仮名 */
#define UNIBE_KKANA_E     0x30f3  /* 終了片仮名 */
#define UNIBE_ODORIJI_B   0x3031  /* 開始範囲外踊り字 (く ぐ / へ \) */
#define UNIBE_ODORIJI_E   0x3035  /* 終了範囲外踊り字 */
```

a-2. DcsIO 定義

以下にヘッダー・ファイル dcs_io.h に定義されている値を示す。

最大ファイル・オープン数

```
#define DCS_MAX_TIO      100
```

オープン・モード

```
#define DMD_READ          0    /* 入力モード("rt" "rb") */
#define DMD_WRITE         1    /* 出力モード("wt" "wb") */
#define DMD_APPEND        2    /* 追加モード("at" "ab") */
#define DMD_MODIFY         3    /* 更新モード("r+t" "r+b") */
```

処理タイプ

```
#define DOP_NOTHING       0    /* 未処理 */
#define DOP_COPY          1    /* コピー */
#define DOP_ADDBOM        2    /* BOM 追加 */
#define DOP_DELBOM        3    /* BOM 削除 */
#define DOP_CONV           4    /* コード変換 */
```

.....

a-3. Dcs クラス定義

以下にDcs クラス定義されている定数関数を示す。

```
Public Function DCS_BITS() As Integer    '' Dcs ビット数
    Dcs.BITS = 64
```

```
Public Function SEPARATE_CHAR() As String '' 文字列の区切り文字
    SEPARATE_CHAR = ";"
```

処理ステータス

```
Public Function SUCCESS() As Long        '' 処理成功
    SUCCESS = 0
Public Function EOF() As Long            '' 入力ファイルの終わり
    EOF = -1
Public Function ENOTEXISTS() As Long     '' ファイルは存在せず
    ENOTEXISTS = -2
Public Function ENOTFILE() As Long       '' 通常ファイルではない
    ENOTFILE = -3
Public Function EALLOC() As Long         '' メモリー・アロケート・エラー
    EALLOC = -4
```

Public Function EROPEN() As Long EROPEN = -5	'' 入力ファイル・オープン・エラー
Public Function EWOPEN() As Long EWOPEN = -6	'' 出力ファイル・オープン・エラー
Public Function EREAD() As Long EREAD = -7	'' 入力エラー
Public Function EWRITE() As Long EWRITE = -8	'' 出力エラー
Public Function EIOMODE() As Long EIOMODE = -10	'' 入出力モード誤り
Public Function EIOFDRANGE() As Long EIOFDRANGE = -11	'' ファイル記述子範囲誤り
Public Function EIONOTOPEN() As Long EIONOTOPEN = -12	'' 未オープン・ファイルへのアクセス
Public Function EIOMAX() As Long EIOMAX = -13	'' 上限を超えたファイル・オープン
Public Function ECTYPE() As Long ECTYPE = -20	'' コード・タイプ誤り
Public Function EBOM() As Long EBOM = -21	'' BOM 不正（未使用）
Public Function EUEXPEOD() As Long EUEXPEOD = -22	'' 予期せぬデータの終わり
Public Function ECHAR() As Long ECHAR = -23	'' コード値エラー
コード判定ステータス	
Public Function UNKNOWN() As Long UNKNOWN = -1	'' コード・タイプ不明
Public Function EEMPTY() As Long EEMPTY = 0	'' 空データ
Public Function ASCII() As Long ASCII = 1	'' ASCII コード・タイプ
Public Function JIS() As Long JIS = 2	'' JIS コード・タイプ
Public Function JIS2() As Long JIS2 = 3	'' JIS2 コード・タイプ
Public Function SJIS() As Long SJIS = 4	'' SJIS コード・タイプ
Public Function EUC() As Long EUC = 5	'' EUC コード・タイプ
Public Function EUC1() As Long EUC1 = 6	'' EUC CP20932 コード・タイプ

```

Public Function EUC2() As Long          '' EUC CP51932 コード・タイプ
    EUC2 = 7
Public Function EUC3() As Long          '' EUC-MS コード・タイプ
    EUC3 = 8
Public Function UTF16BE() As Long       '' UTF-16 ビッグ・エンディアン・コード・タイプ
    UTF16BE = 11
Public Function UTF16BE_BOM() As Long   '' UTF-16 ビッグ・エンディアン (BOM付) コード・タイプ
    UTF16BE_BOM = 12
Public Function UTF16LE() As Long       '' UTF-16 リトル・エンディアン・コード・タイプ
    UTF16LE = 13
Public Function UTF16LE_BOM() As Long   '' UTF-16 リトル・エンディアン (BOM付) コード・タイプ
    UTF16LE_BOM = 14
Public Function UTF8() As Long          '' UTF-8 コード・タイプ
    UTF8 = 15
Public Function UTF8_BOM() As Long      '' UTF-8 (BOM付) コード・タイプ
    UTF8_BOM = 16
Public Function NORMAL() As Long        '' 確定コード・タイプの最大値
    NORMAL = &077

'' 確定コード・タイプ
Public Function CONFIRMED_CODE_TYPE(ByVal sts As Long) As Boolean
    CONFIRMED_CODE_TYPE = (UNKNOWN < sts And sts <= NORMAL)

```

未確定コード・タイプ (マスク値)

```

Public Function ND_ASCII() As Long      '' ASCII コード・タイプ
    ND_ASCII = &0100
Public Function ND_JIS() As Long        '' JIS コード・タイプ
    ND_JIS = &0200
Public Function ND_JIS2() As Long       '' JIS2 コード・タイプ
    ND_JIS = &02000000
Public Function ND_SJIS() As Long       '' SJIS コード・タイプ
    ND_SJIS = &0400
Public Function ND_EUC() As Long        '' EUC コード・タイプ
    ND_EUC = &01000
Public Function ND_EUC1() As Long       '' EUC CP20932 コード・タイプ
    ND_EUC1 = &02000
Public Function ND_EUC2() As Long       '' EUC CP51932 コード・タイプ
    ND_EUC2 = &04000
Public Function ND_EUC3() As Long       '' EUC-MS コード・タイプ
    ND_EUC3 = &010000
Public Function ND_UTF16BE() As Long     '' UTF-16 ビッグ・エンディアン・コード・タイプ
    ND_UTF16BE = &020000

```

```

Public Function ND_UTF16BE_BOM() As Long    '' UTF-16 ビッグ・エンディアン(BOM付)コード・タイプ
    ND_UTF16BE = &040000
Public Function ND_UTF16LE() As Long        '' UTF-16 リトル・エンディアン・コード・タイプ
    ND_UTF16LE = &0100000
Public Function ND_UTF16LE_BOM() As Long    '' UTF-16 リトル・エンディアン(BOM付)コード・タイプ
    ND_UTF16LE = &0200000
Public Function ND_UTF8() As Long           '' UTF-8 コード・タイプ
    ND_UTF8 = &0400000
Public Function ND_UTF8_BOM() As Long       '' UTF-8 (BOM付)コード・タイプ
    ND_UTF8 = &01000000

```

DcsIO

```

Public Function MAX_TIO() As Long           '' 最大ファイル・オープン数
    MAX_TIO = 100

Public Function DMD_READ() As Long          '' 入力モード("rt" "rb")
    DMD_READ = 0
Public Function DMD_WRITE() As Long         '' 出力モード("wt" "wb")
    DMD_WRITE = 1
Public Function DMD_APPEND() As Long        '' 追加モード("at" "ab")
    DMD_APPEND = 2
Public Function DMD_MODIFY() As Long        '' 更新モード("r+t" "r+b")
    DMD_MODIFY = 3

```

処理タイプ

```

Public Function DOP_NOTHING() As Long       '' 未処理
    DOP_NOTHING = 0
Public Function DOP_COPY() As Long          '' コピー
    DOP_COPY = 1
Public Function DOP_ADDBOM() As Long        '' BOM 追加
    DOP_ADDBOM = 2
Public Function DOP_DELBOM() As Long        '' BOM 削除
    DOP_DELBOM = 3
Public Function DOP_CONV() As Long          '' コード変換
    DOP_CONV = 4

```