

---

# BTRON3仕様書 Ver 3.20.00

---

TRON Architecture: Designed by Ken Sakamura

Copyright (C) 1999 TRON Association

本仕様書の著作権は、社団法人トロン協会に属しています。

TRONは、The Realtime Operating system Nucleusの略称です。  
BTRONは、Business TRONの略称です。

---

## 第1編 共有データ

## 第2編 OS仕様

---

# 第1編 共有データ 目次

---

## [第1章 基本データタイプ](#)

## [第2章 TRON コード体系](#)

### [2.1 TRON 仕様コードの概要](#)

#### [2.1.1 多国語対応](#)

#### [2.1.2 文字セット](#)

#### [2.1.3 書体セット](#)

### [2.2 既存のコード体系との関連](#)

#### [2.2.1 JIS X 0201](#)

#### [2.2.2 JIS X 0208](#)

### [2.3 TRON仕様日本文字コード](#)

#### [2.3.1 概要](#)

#### [2.3.2 TRON 仕様日本文字コード](#)

#### [2.3.3 TRON 仕様日本文字コード説明](#)

#### [2.3.4 その他](#)

## [第3章 TAD詳細仕様書](#)

### [3.1 はじめに](#)

### [3.2 TAD 実身構成](#)

#### [3.2.1 概要](#)

#### [3.2.2 実身レコード構成](#)

### [3.3 TADデータ構成の概要](#)

#### [3.3.1 TADデータ構成](#)

#### [3.3.2 文章データと図形データ](#)

#### [3.3.3 画像データ](#)

#### [3.3.4 TADデータの構造定義](#)

### [3.4 TADデータ構成の詳細](#)

#### [3.4.1 概要](#)

#### [3.4.2 管理情報セグメント](#)

#### [3.4.3 文章開始/終了セグメント](#)

#### [3.4.4 図形開始/終了セグメント](#)

#### [3.4.5 画像セグメント](#)

#### [3.4.6 仮身セグメント](#)

#### [3.4.7 機能付箋セグメント](#)

#### [3.4.8 指定付箋セグメント](#)

### [3.5 文章付箋セグメント](#)

#### [3.5.1 概要](#)

- [3.5.2 文章ページ割付け指定付箋](#)
- [3.5.3 行書式指定付箋](#)
- [3.5.4 文字指定付箋](#)
- [3.5.5 特殊文字指定付箋](#)
- [3.5.6 文字割付け指定付箋](#)
- [3.5.7 文字修飾指定付箋](#)
- [3.5.8 変数参照指定付箋](#)
- [3.5.9 文章メモ指定付箋](#)
- [3.5.10 文章アプリケーション指定付箋](#)
- [3.6 図形描画セグメント](#)
  - [3.6.1 概要](#)
  - [3.6.2 図形要素セグメント](#)
  - [3.6.3 データ定義セグメント](#)
  - [3.6.4 グループ定義セグメント](#)
  - [3.6.5 マクロ定義 / 参照セグメント](#)
  - [3.6.6 図形修飾セグメント](#)
  - [3.6.7 図形ページ割り付け指定付箋](#)
  - [3.6.8 図形メモ指定付箋](#)
  - [3.6.9 図形アプリケーション指定付箋](#)

## [付録 A TADセグメント一覧](#)

## [付録 B 準TAD規格](#)

## [第4章 フロッピーディスク形式](#)

- [4.1 概要](#)
- [4.2 ファイルシステム構成](#)
  - [4.2.1 全体構成](#)
  - [4.2.2 システムブロック構成](#)
  - [4.2.3 ファイルIDテーブル](#)
  - [4.2.4 ファイル短縮名テーブル](#)
  - [4.2.5 ファイル領域](#)
- [4.3 ファイル構成](#)
  - [4.3.1 全体構成](#)
  - [4.3.2 ヘッダブロックの構成](#)
  - [4.3.3 ファイルヘッダの構成](#)
  - [4.3.4 フラグメントテーブルの構造](#)
  - [4.3.5 レコードインデックスの構成](#)
  - [4.3.6 間接インデックスの構成](#)
  - [4.3.7 レコードインデックスの実際](#)
  - [4.3.8 データブロックの構成](#)

## [付録 A 準標準フロッピーディスク形式](#)

## [付録 B フロッピーディスク形式の実現例](#)

[次頁:図版目次にすすむ](#)

# 第1編 共有データ 図版目次

---

- [図 1 TRON 仕様日本文字コード体系](#)
- [図 2 TRON 仕様日本文字コード](#)
- [図 3 可変長セグメントの構造](#)
- [図 4 文章データ](#)
- [図 5 図形データ](#)
- [図 6 描画領域と表示領域](#)
- [図 7 図形データでのネスティング](#)
- [図 8 文章データでのネスティング](#)
- [図 9 カラー表現](#)
- [図 10 画像セグメントのデータ形式](#)
- [図 11 カラーマップ](#)
- [図 12 画像セグメント](#)
- [図 13 文章付箋の構造](#)
- [図 14 レイアウト用紙とオーバーレイマージン](#)
- [図 15 レイアウト用紙と本文マージン](#)
- [図 16 段組みレイアウト](#)
- [図 17 コラム長の均等化指定](#)
- [図 18 コラム数を変更した場合\(1\)](#)
- [図 19 コラム数を変更した場合\(2\)](#)
- [図 20 用紙オーバーレイ](#)
- [図 21 用紙オーバーレイ指定付箋](#)
- [図 22 枠あけ指定](#)
- [図 23 ページ番号指定](#)
- [図 24 条件改ページ指定](#)
- [図 25 フィールド書式の例](#)
- [図 26 行間隔指定](#)
- [図 27 行揃え指定](#)
- [図 28 タブ書式](#)
- [図 29 フィールド書式\(一般\)](#)
- [図 30 フィールド書式の例-1](#)
- [図 31 フィールド書式の例-2](#)
- [図 32 フィールド書式の例-3](#)
- [図 33 フィールド書式の例-4](#)
- [図 34 文字方向指定付箋](#)
- [図 35 フォントクラス](#)
- [図 36 フォント属性](#)
- [図 37 文字間隔指定](#)

- [図 38 充填文字指定](#)
- [図 39 文字罫線指定](#)
- [図 40 文字罫線指定\(通常罫線\)](#)
- [図 41 文字罫線指定\(行間 / 文間罫線\)](#)
- [図 42 結合指定付箋](#)
- [図 43 添字指定\(1\)](#)
- [図 44 添字指定\(2\)](#)
- [図 45 添字指定\(3\)](#)
- [図 46 図形描画セグメントの構造](#)
- [図 47 任意図形セグメントの例](#)
- [図 48 マスクデータの構成](#)
- [図 49 実際のパターンの生成](#)
- [図 50 パターンサイズとマスクサイズ](#)
- [図 51 線種定義データ](#)
- [図 52 回転角度](#)
- [図 53 画像セグメントのデータ形式](#)
- [図 54 準TAD規格での可変長セグメントの構造](#)
- [図 55 準TAD規格での文章 / 図形付箋の構造](#)
- [図 56 全体構成](#)
- [図 57 システムブロック構成](#)
- [図 58 システムヘッダの構成](#)
- [図 59 ビットマップの構成例](#)
- [図 60 ファイルIDテーブルの構成](#)
- [図 61 ファイル短縮名テーブルの構成](#)
- [図 62 NAME\[10\]](#)
- [図 63 ファイルの構成\(インデックスレベル=0\)](#)
- [図 64 ファイルの構成\(インデックスレベル=1\)](#)
- [図 65 ファイルの構成\(インデックスレベル=2\)](#)
- [図 66 ヘッダブロックの構成](#)
- [図 67 ファイルヘッダの構成](#)
- [図 68 ファイル所在データの構成](#)
- [図 69 フラグメントテーブルの構成](#)
- [図 70 レコードインデックスの構成](#)
- [図 71 間接インデックスの構成](#)
- [図 72 レコードインデックスの例](#)
- [図 73 レコード間の区切りコード](#)
- [図 74 準標準フロッピーディスク形式によるレコードインデックスの構成](#)
- [図 75 準標準フロッピーディスク形式におけるビットマップの構成](#)
- [図 76 フロッピーディスク形式の実現例](#)

---

[全目次にもどる](#)

[前頁:目次にもどる](#)

[この章の目次にもどる](#)

[前頁:第1章 基本データタイプにもどる](#)

[次頁:第3章 TAD詳細仕様書にすすむ](#)

---

## 第2章 TRON コード体系

---

### 2.1 TRON 仕様コードの概要

#### 2.1.1 多国語対応

TRON 仕様コードでは、多国語文字を扱うことを考えるが、国語によって1バイトコードで足りるものと、足りないものがある。このため、TRON 仕様では1バイトコードと2バイトコードを扱う。しかし、基本的には1バイトコードと2バイトコードを混在させて使うことは避け、1バイトコードで足りる国語については1バイトコードを使い、不足する国語では2バイトコードを使用する。

#### 2.1.2 文字セット

従来、コードに割り当てられていない文字を外字として扱っていたが、TRON 仕様コードではこの意味での外字はおこらないように、全ての文字に対してコードを割り当てる方式を取る。漢字の場合でも、45,000文字程度で全てを網羅できる。このようにして定めた、一つの文字コードの集合を文字セットと呼ぶ。会社のマークなどのパターンは、文字とは別扱いで、イメージとして扱うようにする。コードで区別すべき文字が必要となる場合は、新しく登録する。

本仕様書および JIS X 0208 で規定されていない独自の文字コードはデータの交換に用いてはならない。

#### 2.1.3 書体セット

同じ文字に対していくつかの書体が存在する。例えばゴシックと明朝のような場合である。これらは同じ文字コードを割り当て、書体が異なってもデータベース検索の場合に同一文字として扱うことができるようにする。同じ文字セットの中に存在する複数の書体の異なる文字集合を書体セットと呼ぶ。

### 2.2 既存のコード体系との関連

#### 2.2.1 JIS X 0201

JIS X 0201 では、7ビットと8ビットの情報交換符号を規定している。TRON 仕様コードでは8ビットおよび16ビットコード体系を取るため、JIS X 0201 の8ビットコードの規定からの制限を考慮する。

8ビットコード表で、C0 集合の 00/0 ~ 01/15、および 02/0、07/15 は制御コードとして使用

する。また、C1 集合の 08/0 ~ 09/15、および 10/0、15/15 は制御コード用であるが未定義領域とする。

また、日本の場合は、漢字、仮名、ラテン文字のいずれも使用するので、1 バイトコードと 2 バイトコードを頻繁に切り換えるのは不適當である。従って、日本語は全て 2 バイトコードで扱い、1 バイトコードでは扱わない。

TRON 仕様では半角、全角の概念はなく、これらは文字指定付箋を用いて扱う。文字指定付箋については[3.5.4文字指定付箋](#)を参照のこと。

## 2.2.2 JIS X 0208

JIS X 0208 では、8,836 字の領域を確保し、そのうち 6,877 字を割り当てている。このうち漢字は 6,353 字である (JIS X 0208-1983)。

TRON 仕様コードは JIS X 0208 に定義されている文字を準用する。またコードの位置もそのまま用いる。

## 2.3 TRON仕様日本文字コード

### 2.3.1 概要

TRON 仕様日本文字コード体系は、2 バイトの文字コード体系であり、JIS X 0208 のスーパーセットとする。

|         | 第 1 バイト  | 第 2 バイト  |               |
|---------|--|--|---------------|
| 制御コード   | ( 0x00 )   | ( 0x00 ~ 0xFE )  |               |
| 文字コード   | ( 0x21 ~ 0x7E )<br>( 0x80 ~ 0xFD )<br>( 0x21 ~ 0x7E )<br>( 0x80 ~ 0xFD ) | ( 0x21 ~ 0x7E )<br>( 0x21 ~ 0x7E )<br>( 0x80 ~ 0xFD )<br>( 0x80 ~ 0xFD ) | JIS X 0208に対応 |
| 言語指定    | ( 0xFE )<br>( 0xFE )   | ( 0x21 ~ 0x7E )<br>( 0x80 ~ 0xFE )                                       |               |
| 特殊コード   | ( 0xFF )   | ( 0x21 ~ 0x7E )  |               |
| エスケープ   | ( 0xFF )   | ( 0x80 ~ 0xFE )  |               |
| ( EOF ) | ( 0xFF )   | ( 0xFF )   | (特殊用途)        |

図 1 : TRON 仕様日本文字コード体系

### 2.3.2 TRON 仕様日本文字コード



第 2 バイト

|         |    |       |                                |    |                |          |
|---------|----|-------|--------------------------------|----|----------------|----------|
|         | 0  | 20 21 | 7E 7F                          | 80 | A0             | FD FE FF |
| 第 1 バイト | 0  | 制御コード | 予 約                            |    | 予 約            | 予 約      |
|         | 20 |       |                                |    |                |          |
|         | 21 | 未使用   | 文字コード<br>A ゾーン<br>(JIS X 0208) |    | 文字コード<br>C ゾーン |          |
|         | 7E |       |                                |    |                |          |
|         | 7F |       | 未 使                            |    | 用              |          |
|         | 80 | 未使用   | 文字コード<br>B ゾーン                 |    | 文字コード<br>D ゾーン |          |
|         | FD |       |                                |    |                |          |
|         | FE |       | 言語指定<br>特殊コード                  |    | 言語指定<br>エスケープ  |          |
|         | FF |       |                                |    |                |          |

図 2 : TRON 仕様日本文字コード

### 2.3.3 TRON 仕様日本文字コード説明

#### (1)TRON 仕様で特別な意味を持つ制御コード

(00)(00) ~ (00)(20),(00)(7F)のコードは制御コードとして割り当てる。(制御コードの第2バイトはTRON仕様1バイトコード系、2バイトコード系に共通である)

このうち、以下のコードはTRON仕様で特別な意味を持ち、無効コードを除き、特殊文字キーより発生可能なコードである。

- (00) 無効コード このコードがあっても、あたかも無いように処理する文字コードとしては無効であるが、通常は文字列の終了を示す目的に使用する
- (09) タブ 次タブ位置または次フィールドへの遷移
- (0A) 改段落 通常の改行または段落替え  
段落の概念を用いる場合、段落内の改行は改行を用いる
- (0B) 改コラム
- (0C) 改ページ
- (0D) 改行 フィールド内の改行または段落内の改行
- (20) セパレータ 語あるいは文節の区切れ

フィールドの定義については[3.5.3行書式指定付箋](#)を参照のこと。

#### (2)TRON 仕様文字コード

第1ブロック: (21)(21) × (7E)(7E) Aゾーン

8,836文字

JIS X 0208 に完全対応

|                    |                           |                   |
|--------------------|---------------------------|-------------------|
| 第1バイト: 0x21 ~ 0x27 | 非漢字領域 (1 ~ 7区)            | 453文字 / 658文字     |
|                    | (0x21 / 0x21 SP(漢字固定ピッチ)) |                   |
| : 0x28 ~ 0x2F      | 予約領域 (8 ~ 15区)            | / 752文字           |
| : 0x30 ~ 0x4F      | 第1水準漢字 (16 ~ 47区)         | 2,965文字 / 3,008文字 |
| : 0x50 ~ 0x74      | 第2水準漢字 (48 ~ 84区)         | 3,388文字 / 3,478文字 |
| : 0x75 ~ 0x7E      | 予約領域 (85 ~ 94区)           | / 940文字           |
| 第2ブロック:            | (80)(21) × (FD)(7E) Bゾーン  | 11,844文字          |
| 第3ブロック:            | (21)(80) × (7E)(FD) Cゾーン  | 11,844文字          |
| 第4ブロック:            | (80)(80) × (FD)(FD) Dゾーン  | 15,876文字          |

Aゾーンには、JIS X 0208 を対応付ける。

Bゾーンは、JIS X 0208 で不足している文字のうち、利用頻度の高い文字を集める。この中には、漢字以外に記号、他文字属の一部も含む。

C, Dゾーンは、A, Bゾーンよりも頻度の低い文字を集める。

### (3)言語指定コード

言語指定コード (FE)(21) ~ (FE)(7E)、(FE)(80) ~ (FE)(FE)

言語および文字属の切り換え指定を行なう。第2バイトが0xFEの場合、さらに言語指定コードが続くことを示す。2バイトで表現しうる言語指定数は220である。

### (4) TRON 仕様特殊コード

TRON 仕様特殊コード (FF)(21) ~ (FF)(7E) 94 文字

文章中に挿入される特殊コードとして用いる。

### (5) TRON 仕様エスケープ

TRON 仕様エスケープ (FF)(80) ~ (FF)(FE)

文書中に埋め込まれる指定付箋挿入等の区切り記号として用いる。区切り記号の後に付箋情報が続く(付箋情報については[3.3TADデータ構成の概要](#)を参照のこと)。

2バイトで表現しうる TRON 仕様エスケープ数は127である。

## 2.3.4 その他

- (0xFF)/(0xFF) は数値として見た場合の -1 であり、EOF として使用する。
- インデント等の書式制御用の文字コードは使用しない。  
NL, HT(TAB), SP(3種), (NULL) は文字コードとしてテキストに混在して使用する。
- 3種のスペースは次のコードとする。

日本語モードの場合:

漢字固定ピッチ SP (0x21/0x21) - 空白キー

英語モードの場合:

英字固定ピッチ SP (0x00/0xA0) - 空白キー

プロポーショナル SP (0x00/0x20) - 変換・逆変換キー

---

[この章の目次にもどる](#)

[前頁:第1章 基本データタイプにもどる](#)

[次頁:第3章 TAD詳細仕様書にすすむ](#)

# 第1章 基本データタイプ

---

共通して使われる型の定義を以下に記述する。

基本データ型:ITRON共通

```
typedef char          B;          /* 符号付き 8ビット整数 */
typedef short        H;          /* 符号付き16ビット整数 */
typedef int          W;          /* 符号付き32ビット整数 */
typedef unsigned char UB;        /* 符号無し 8ビット整数 */
typedef unsigned short UH;       /* 符号無し16ビット整数 */
typedef unsigned int UW;        /* 符号無し32ビット整数 */

typedef char          VB;        /* 不定型 8ビットデータ */
typedef short        VH;        /* 不定型16ビットデータ */
typedef int          VW;        /* 不定型32ビットデータ */

typedef void          *VP;       /* 不定型データへのポインタ */

typedef void          (*FP)();   /* 一般関数ポインタ */
```

以上のデータ型はITRONと共通である。

Volatile: コンパイラによる最適化禁止指定

```
typedef volatile W    _W;        /* 最適化禁止の符号付き 8ビット整数 */
typedef volatile H    _H;        /* 最適化禁止の符号付き16ビット整数 */
typedef volatile B    _B;        /* 最適化禁止の符号付き32ビット整数 */

typedef volatile UW   _UW;       /* 最適化禁止の符号無し 8ビット整数 */
typedef volatile UH   _UH;       /* 最適化禁止の符号無し16ビット整数 */
typedef volatile UB   _UB;       /* 最適化禁止の符号無し32ビット整数 */

typedef volatile void _void;
```

修飾型

```
#define CONST        const      /* 変数・関数の代入禁止指定 */

#define LOCAL        static     /* モジュール\(\*\)内でのみ使われる */
```

```
/* 変数・関数 */
```

```
#define EXPORT /* プログラム中の任意の箇所で参照 */  
/* 可能な変数・関数の定義 */  
/* 1個の変数・関数について EXPORT */  
/* 宣言はプログラム中に1箇所だけ */  
/* 記述 */  
#define IMPORT extern /* プログラム中の任意の箇所で参照 */  
/* 可能な変数・関数の参照宣言 */  
/* 1モジュールに1個の参照宣言で */  
/* EXPORT された変数・関数の使用が */  
/* 可能 */  
#define VOID void /* 空の型 */
```

```
* モジュール
```

プログラムはソースコードをコンパイル・リンクして作成されるが、そのソースコードは通常、複数のソースファイルに分割して記述されている。以降、本書ではこのソースファイル1個の単位をモジュールと呼ぶ。

```
数値型
```

```
typedef float FLOAT; /* 単精度(32ビット)浮動小数点 */  
typedef double DOUBLE; /* 倍精度(64ビット)浮動小数点 */  
typedef int BOOL; /* TRUE or FALSE */  
typedef W (*FUNCP)(); /* W 関数へのポインタ */
```

```
ブール値
```

```
typedef enum {  
    False = 0,  
    True = 1  
} Bool; /* ブール値 */
```

```
ID番号
```

```
typedef W ID; /* ID番号 */
```

```
時間
```

```
typedef W MSEC; /* ミリ秒 */  
typedef W STIME; /* システム時間 */  
/* 1985/01/01 00:00:00 GMT からの秒数 */
```

```
文字コード
```

```
typedef UH      TC;          /* TRON 文字コード */
#define TNULL   ((TC) 0)    /* 文字列終端の文字コード値 */
```

エラーコード

ビッグエンディアンの場合

```
typedef union {
    W      err;          /* エラーコード */
    struct {
        H   eclass;     /* エラークラス */
        UH  detail;     /* 詳細エラー   */
    } c;
} ErrCode;
```

リトルエンディアンの場合

```
typedef union {
    W      err;          /* エラーコード */
    struct {
        UH  detail;     /* 詳細エラー   */
        H   eclass;     /* エラークラス */
    } c;
} ErrCode;
```

システムコールの関数値は以下のいずれかである。

```
typedef W      ERR;      /* エラーコード */
/* エラーまたは正常のみ戻す場合に使用 */
typedef W      WERR;     /* システムコール関数値 */
/* エラーまたは意味のある値を戻す */
/* 場合に使用 */
```

エラーコードとして通常は ERR を使用し、詳細が必要な場合は ErrCode を使用する。

```
#define OK      (0)      /* 正常(OK) */
```

共通定義

```
#define CLR      0x0000   /* クリア指定   */
#define NOCLR    0x0008   /* ノークリア指定 */
#define DELEXIT  0x8000   /* 終了時削除指定 */
```

大きさ

```
typedef struct Size {
    H   h;   /* 幅 */
    H   v;   /* 高さ */
}
```

```
} SIZE;
```

点

```
typedef struct point {
    H    x;    /* 水平座標値 */
    H    y;    /* 垂直座標値 */
} PNT;
```

長方形

```
typedef union rect {
    struct _rect {
        H    left;    /* 左の座標値 */
        H    top;     /* 上の座標値 */
        H    right;   /* 右の座標値 */
        H    bottom;  /* 下の座標値 */
    } c;
    struct {
        PNT    lefttop; /* 左上の点 */
        PNT    rightbot; /* 右下の点 */
    } p;
} RECT;
```

ファイルへのリンク

```
#define L_FSNM    20    /* ファイルシステム名の長さ(文字数) */
typedef struct {
    TC    fs_name[L_FSNM]; /* ファイルシステム名 */
    UH    f_id;           /* ファイルID */
    UH    atr1;           /* 属性データ1 */
    UH    atr2;           /* 属性データ2 */
    UH    atr3;           /* 属性データ3 */
    UH    atr4;           /* 属性データ4 */
    UH    atr5;           /* 属性データ5 */
} LINK;
```

ウィンドウ

```
typedef W        WID;    /* ウィンドウID */
```

---

[この章の目次にもどる](#)

[次頁:第2章 TRON コード体系にすすむ](#)

## 第3章 TAD 詳細仕様書

### 3.1 はじめに

本仕様書では、TAD (TRON APPLICATION DATABUS) に関する詳細のデータ構造を説明することを目的としているため、基本的な BTRON 仕様の実身、仮身、付箋等の概念、および TAD の目的、概念、位置付け等に関しては特に説明していない。

また、BTRON 仕様システム、特にファイルシステムとディスプレイ・プリミティブ、および TRON 仕様コードに関する知識を有することを前提としている。

### 3.2 TAD 実身構成

#### 3.2.1 概要

TAD (TRON APPLICATION DATABUS) 実身構成は、BTRON 仕様の実身 (ファイル) の全体構成を規定するものである。

BTRON 仕様の実身 (ファイル) には、以下の情報が格納されている。

- 実身(ファイル)管理情報:  
実身 (ファイル) の所有者、所属グループ、アクセス管理、アクセス属性等の BTRON 仕様ファイルシステムとして定義される。
- アプリケーション実身タイプ:  
実身のアプリケーションレベルでの基本タイプを示すもので、ピクトグラムに 1 対 1 対応する。また、実行プログラム、デバイス実身等の識別情報も含む。
- 実身データ:  
実身のデータは、各種のレコードの1次元連鎖より構成され、各レコードは、レコードタイプにより、その内容が規定される。

#### 3.2.2 実身レコード構成

BTRON仕様実身(ファイル)はレコードの列から構成され、その各レコードは、レコードタイプに応じて、そのデータ構成が規定されている。レコードのサブタイプは、通常レコードの検索等のために使用され、その内容は特に規定されない。

以下にレコードタイプと、そのデータ構成の規定を示す。

|   | レコードタイプ    | データ構成の規定  |
|---|------------|-----------|
| 0 | リンクレコード    | 標準データタイプ  |
| 1 | TAD 主レコード  | TAD データ構成 |
| 2 | TAD 注釈レコード | TAD データ構成 |
| 3 | TAD 補助レコード | TAD データ構成 |
| 4 | 予約レコード     | -         |
| 5 | 設定付箋レコード   | TAD データ構成 |



|       |              |                |
|-------|--------------|----------------|
| 6     | 指定付箋レコード     | TAD データ構成      |
| 7     | 機能付箋レコード     | TAD データ構成      |
| 8     | 実行機能付箋レコード   | TAD データ構成      |
| 9     | 実行プログラムレコード  | 標準オブジェクト形式     |
| 10    | データボックスレコード  | データボックス定義形式    |
| 11    | フォントデータレコード  | 標準フォントデータ形式    |
| 12    | 辞書データレコード    | 標準辞書データ形式      |
| 13    | 予約レコード       | -              |
| 14    | 予約レコード       | -              |
| 15    | システムデータレコード  | システムアプリケーション定義 |
| 16~31 | アプリケーションレコード | アプリケーション定義     |

- 5~ は付箋レコードであり、9~15は標準生身付箋レコードとなる。

TAD 主レコード:

実身のメインデータであり、TAD データ形式である。基本エディタでの処理対象となる。

TAD 注釈レコード:

実身の注釈としての補助データであり、TAD データ形式である。実行プログラムファイル内のヘルプ・データ、コピーライトデータ等に使用され、基本エディタでの処理対象とならない。

TAD 補助レコード:

実身の補助データであり、TAD データ形式である。その使用方法はアプリケーションに依存し、基本エディタでの処理対象とならない。

設定付箋レコード:

実身に貼られた設定付箋である。

指定付箋レコード:

実身に貼られた指定付箋である。

機能付箋レコード:

実身に貼られた機能付箋である。

実行機能付箋レコード:

実身の実行メニューとして表示され、実行される機能付箋である。

実行プログラムレコード:

実行プログラムの標準オブジェクト形式であり、サブタイプはCPUのタイプを示す。

0x100 ~ 0x13F TRONCHIP 系 0x00 TRONCHIP32

0x140 ~ 0x14F 68000 系 0x40 68000

0x41 68010

0x42 68020

0x150 ~ 0x15F NS32000 系 0x50 32032

0x160 ~ 0x16F 86 系 0x60 8086 / 8088

0x61 80186

0x62 80286

0x63 80386

0x170 ~ 0x17F V シリーズ

データボックスレコード:

データボックスのデータ定義であり、データマネージャによりサポートされるデータ形式であ

る。

フォントデータレコード:

標準のフォントデータ形式。

辞書データレコード:

標準の辞書データ形式。

システムデータレコード:

システムで使用される標準のシステムデータ形式である。

アプリケーションレコード:

アプリケーションで定義されるデータ形式である。

## 3.3 TAD データ構成の概要

### 3.3.1 TAD データ構成

TAD データ構成は、BTRON 仕様ファイル内の TAD データレコードの内容を規定するものである。

TAD データ構成は、ファイル内のデータ形式を規定するとともに、トレー経由、またはドラッグによるアプリケーション間のデータ交換のためにも使用される。

TAD データ構成は、セグメントと呼ばれるデータ要素の次元要素連鎖であり、セグメントには、固定バイト長の固定長セグメントと、可変バイト長の可変長セグメントがある。各セグメントは、TRON 仕様多国語文字コード体系の下に定義されているため、TAD データ構成は TRON 仕様文字コード列そのものであると言える。

固定長セグメント

TRON 仕様多国語文字コードとして定義されている、広義の文字コードそのものであり、以下のものが含まれる。

- ・通常文字コード : 1または2バイト・コード
- ・制御コード : (0x00 ~ 0x20) の1バイト・コード
- ・言語指定コード : (0xFE)で始まる多バイトコード
- ・特殊コード : (0xFF)(0x21 ~ 0x7E)の2バイトコード

なお、TAD データ構成では、基本的に下記の制御コードのみを使用する。

|             |      |
|-------------|------|
| 無効コード       | 0x00 |
| タブコード       | 0x09 |
| 改段落コード      | 0x0A |
| 改コラムコード     | 0x0B |
| 改ページコード     | 0x0C |
| 改行コード       | 0x0D |
| セパレータ(スペース) | 0x20 |

- タブ書式、フィールド書式が設定されていない場合は、タブコードは無視され、改段落コードは改行コードと同一の意味を持つ。
- コラム指定が行なわれていない場合の改コラムコードは、改ページコードと同様の意味を持つ。
- 図形データ内に埋め込まれた文章データの場合、改ページコード(コラム指定が行なわれていない場合は、改コラムコードも)は改行コードと見なされる。

可変長セグメント

TRON 仕様多国語文字コードとして定義されている TRON 仕様エスケープであり、先頭バイトが

(0xFF)、第2バイトが、(0x80~0xFE)の値を持つ。第2バイトはセグメントの種別を表すため、セグメントIDと呼ばれる。セグメントIDの直後にデータ本体のバイト長が入るが、バイト長の形式として、通常セグメントとラージセグメントの2つの形式が存在し、ラージセグメントは、バイト長さが65,536バイト以上の場合にのみ使用される。バイト長は必ず偶数となるように必要に応じて"0"のバイトが最後に追加される。データは上位バイトが先にくるビッグエンディアンの形式となる。下図に可変長セグメントの形式を示す。

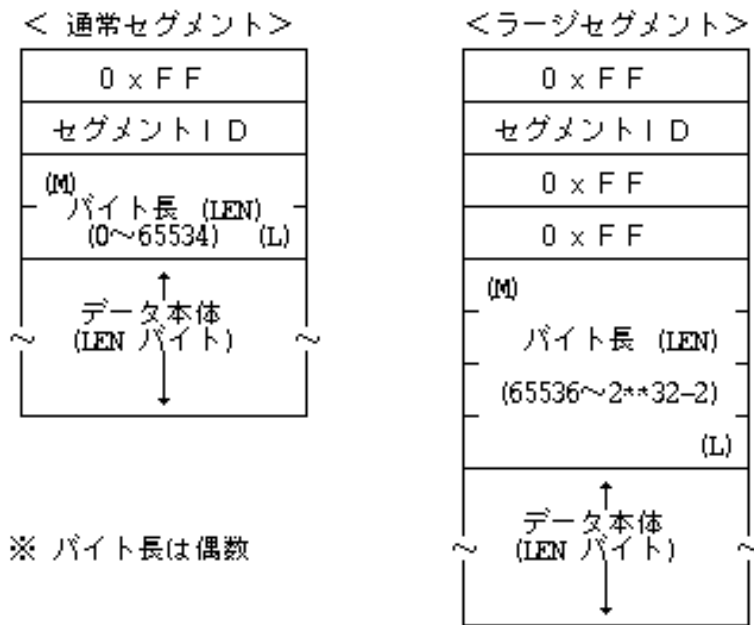


図3: 可変長セグメントの構造

標準可変長セグメントとして以下のものが定義されている。

| セグメントタイプ             | セグメントID       |
|----------------------|---------------|
| (予約)                 | (0x80 ~ 0x9F) |
| 文章付箋セグメント TS_Txxxx   | (0xA0 ~ 0xAF) |
| 図形描画セグメント TS_Fxxxx   | (0xB0 ~ 0xBF) |
| (予約)                 | (0xC0 ~ 0xDF) |
| 管理情報セグメント TS_INFO    | (0xE0)        |
| 文章開始セグメント TS_TEXT    | (0xE1)        |
| 文章終了セグメント TS_TEXTEND | (0xE2)        |
| 図形開始セグメント TS_FIG     | (0xE3)        |
| 図形終了セグメント TS_FIGEND  | (0xE4)        |
| 画像セグメント TS_IMAGE     | (0xE5)        |
| 仮身セグメント TS_VOBJ      | (0xE6)        |
| 指定付箋セグメント TS_DFUSEN  | (0xE7)        |
| 機能付箋セグメント TS_FFUSEN  | (0xE8)        |
| 設定付箋セグメント TS_SFUSEN  | (0xE9)        |
| (予約)                 | (0xEA ~ 0xFE) |

### 3.3.2 文章データと図形データ

TAD データは、大きく文章データと、図形データに分類される。

文章データは1次元データとも呼ばれ、データ要素を一次元平面上で表現したデータである。文章データの個々のデータ要素は2次元の大きさを持つ場合もあるが、位置は1次元、即ち順序関係として

表現される。

文章データには、文章データを実際に2次元平面(用紙)上に表現するための各種のレイアウト情報も含まれるが、これはあくまでもレイアウトへの入力情報であり、文章データ自体としては、レイアウトされた結果としての2次元平面上での表現を直接表しているものでない。

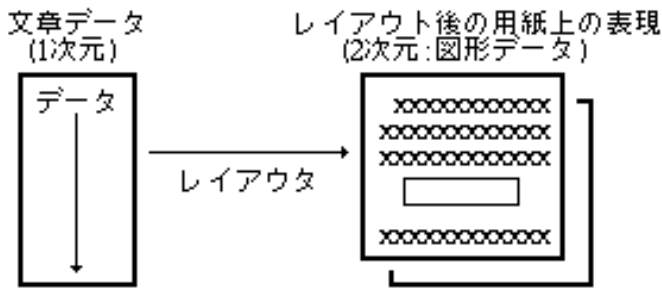


図4: 文章データ

図形データは2次元データとも呼ばれ、データ要素を2次元平面上で直接的に表現したデータである。図形データの個々のデータ要素は2次元座標平面上での大きさと位置を持つ。

図形データは左上を(0,0)とし、下/右方向に無限に広がる2次元座標平面(用紙)上の1つの長方形領域内のデータを表現しているもので、各データ要素は、その位置、大きさをその座標平面上での座標値として持っている。座標値は0~32,767の範囲の非負の整数値である。

図形データの座標系はあくまで2次元座標平面(用紙)上での表現上の座標系であり、実際の物理的な大きさを直接表現しているものではない。

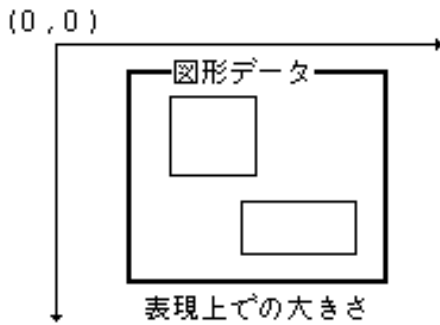


図5: 図形データ

文章データと図形データは任意にネスティング可能であり、一番外側のデータの種別により、文章データか図形データかが分類される。ネスティングした文章データ、図形データは、埋め込み文章データ、埋め込み図形データとも呼ばれる。

同種の(文章/図形)データがネスティングした場合は、原則として未定義の属性は、外側で定義された属性のうち、最も内側で定義された属性が適用される。一番外側でも未定義な場合は、データを解釈するアプリケーションに依存した適当なデフォルト値として解釈される。

別種の(図形/文章)データがネスティングしている場合は、同種のデータのうち、最も外側のものが一番外側と判断される。即ち、別種のデータがネスティングしている場合は、そこで外側からの属性の継承が切断される。ただし、座標系、領域に関しては、同種、別種に無関係にすぐ外側のものが適用される。同種、または別種のデータがネスティングしている場合、あるデータ内で定義された属性は外側のデータには適用されない。

例: 図形データ A

図形データ B -- 属性の上位スコープは A

図形データ C -- 属性の上位スコープは A

図形データ D -- 属性の上位スコープは C, A

文章データ E -- 属性の上位スコープなし  
文章データ F -- 属性の上位スコープは E  
図形データ G -- 属性の上位スコープなし

(埋め込み) 文章 / 図形データには、以下の情報が定義される。

描画領域 ( draw )

(埋め込み) 図形データの時

図形データとして描画される長方形領域、および描画のための座標系を示し、大きさ、値ともに意味を持つ。描画領域外をはみ出る部分はクリッピングされ、図形データには含まれない。

文章データの時

意味を持たない。

埋め込み文章データの時

図形データ中に埋め込まれた文章データの場合、文章データをレイアウトする長方形領域を示し、大きさのみが意味を持つ。  
文章データ中に埋め込まれた文章データの場合は、意味を持たない。

座標系単位 ( unit )

描画領域の座標系の単位を、1 cm 当たりの座標数または 1 inch 当たりの座標数により示す。文章データの場合は、中に含まれる図形データ、画像データの表示領域の単位、および書式のマージンの単位等を示すために使用される。

座標系単位の値が 0 の場合は、未定義であり、すぐ外側と同一であると見なされる。

表示領域 ( view )

図形データ、文章データ、文章データ内に埋め込まれている文章データの時

意味を持たない。

その他の時

描画領域の表現を写像する外側の描画領域内での位置と大きさを示す。従って、表示領域は外側の描画座標系 / 単位で規定される。

文章データ内に埋め込まれている場合は、大きさのみ意味を持つ。

表示領域の大きさ / 外側の描画座標単位 / 外側の倍率と、描画領域の大きさ / 描画座標単位 / 倍率により、描画領域の表現は拡大または縮小されて表示領域に写像される。

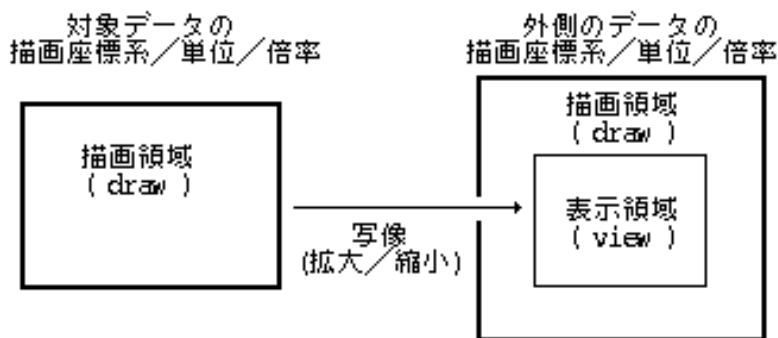


図 6 : 描画領域と表示領域

図形データ内に埋め込まれた文章データは、描画領域として指定された領域内にレイアウトされた結果としての表現を意味し、2次元データと見なされる。この場合、ページ、用紙の概念は存在しないため、文章データ中のページ、用紙のレイアウトに関する指定は無視される。

図形データ内に埋め込まれた図形データは、座標系 / 倍率が異なる場合、または、図形のグルーピングのために使用される。

文章データ内に埋め込まれた文章データは、座標系単位が異なる場合、または、グルーピングを行なうために使用できるが、埋め込まれた文章データも1次元データであるため、その順番のみが意味を持つ。2次元のレイアウト済みの表現としての文章を埋め込む場合は、埋め込み図形データ内の埋め込み文章データとする必要がある。

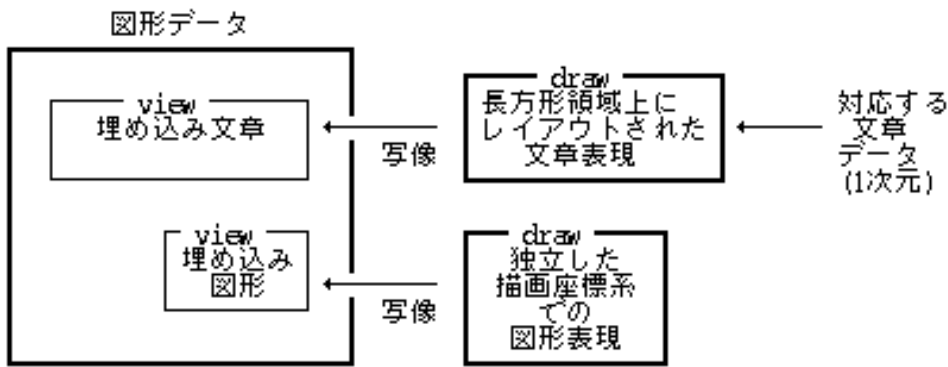


図7: 図形データでのネスティング

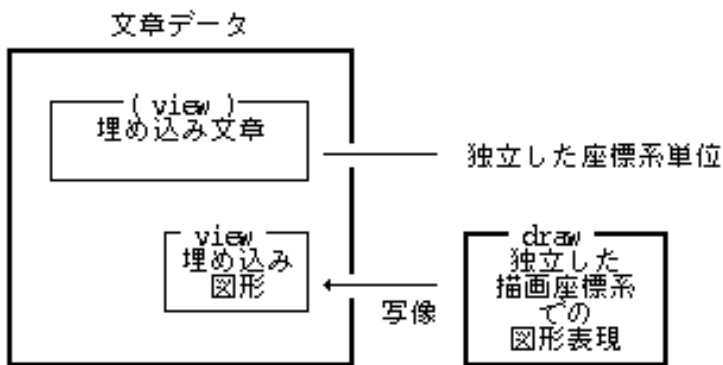


図8: 文章データでのネスティング

### 3.3.3 画像データ

画像データは、(圧縮)ビットマップ形式により表現されたイメージ・データであり、単位系、カラー表現等のデータを解釈するための情報を持っている。

画像データは、それ自体で1つの独立したデータとしての意味を持つが、TADデータ構造としては、1つのセグメントとして取り扱われ、図形データ、または文章データの1つの要素として取り扱われる。

### 3.3.4 TADデータの構造定義

TADデータ構造は、以下に示すBNF表記により定義される。

- 【TADデータ】 ::= 《管理情報セグメント》【TAD本体】
- 【TAD本体】 ::= 【文章データ】 | 【図形データ】
- 【文章データ】 ::= 《文章開始セグメント》【文章要素並び】《文章終了セグメント》
- 【文章要素並び】 ::= 《空》 | 【文章要素並び】【文章要素】
- 【文章要素】 ::= 【文字コード】 |  
 《文章付箋セグメント》 |  
 【図形データ】 |  
 【文章データ】 |  
 《画像セグメント》 |  
 《仮身セグメント》 |

《指定付箋セグメント》 |  
 《機能付箋セグメント》  
**【文字コード】** ::= 《通常文字コード》 |  
 《制御コード》 |  
 《言語指定コード》 |  
 《特殊文字コード》  
**【図形データ】** ::= 《図形開始セグメント》 **【図形要素並び】** 《図形終了セグメント》  
**【図形要素並び】** ::= 《空》 | **【図形要素並び】** **【図形要素】**  
**【図形要素】** ::= 《図形描画セグメント》 |  
**【文章データ】** |  
**【図形データ】** |  
 《画像セグメント》 |  
 《仮身セグメント》 |  
 《指定付箋セグメント》 |  
 《機能付箋セグメント》

## 3.4 TAD データ構成の詳細

### 3.4.1 概要

この章では、TAD データの要素である各セグメントの構造の詳細を説明するが、文章付箋セグメントに関しては第 4 節で、図形描画セグメントに関しては第 5 節で説明する。

各セグメントのデータ構造は、以下の形式で説明している。

ID : -- セグメント ID を示す。  
 LEN : -- データ本体のバイト長を示し、" ~ " は可変長を意味する。  
 DATA: -- データ本体の内容を示す。

各セグメントのバイト長 (LEN) は、セグメント・タイプ毎に規定されているが、将来的に拡張される可能性があるため、仕様上のバイト長ではなく、実際に格納されている LEN の値に従って処理する必要がある。

また、データ本体の内容は、以下に示すデータ表記を使用している。8 ビット以上のビット長さを持つデータは、上位バイトが先にくるビッグエンディアンの形式である。

B  
 8 ビットの符号付きデータ  
 UB  
 8 ビットの符号無しデータ  
 UB  
 文字コードを示す、8 ビットの符号無しデータ  
 H  
 16 ビットの符号付きデータ  
 UH  
 16 ビットの符号無しデータ  
 W  
 32 ビットの符号付きデータ  
 UW

## 32 ビットの符号無しデータ

### PNT

点を表す以下の構造体データ。

```
typedef struct {
  H  h  -- 水平座標値
  H  v  -- 垂直座標値
} PNT;
```

### RECT

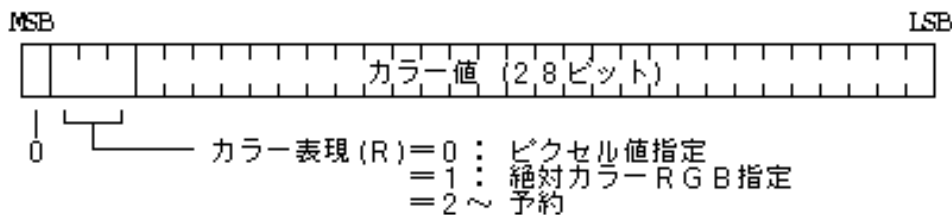
長方形領域を表す以下の構造体データ。

```
typedef struct {
  DBYTE left   -- 左の座標値
  DBYTE top    -- 上の座標値
  DBYTE right  -- 右の座標値
  DBYTE bottom -- 下の座標値
} RECT;
```

- 左と上の座標値は長方形領域に含まれるが、右と下の座標値自体は、長方形領域には含まれない (half-open property)。

### COLOR

カラーを表現する以下の 32 ビットの符号無しデータ



カラー値は以下の内容となるが、通常、R=0の表記は使用しない。  
R=0 直接ピクセル値を示す。(0は白、all 1は黒となる)  
R=1 赤/緑/青の各色の輝度を0~255の正規化されたレンジ  
で示す。すなわち、0が0%、255が100%を示す。  
(赤/緑/青がすべて0%の時、黒、すべて100%の時、  
白となる。)

X · X R · · R G · · G B · · B  
無視(4) 赤(8) 緑(8) 青(8)

図9: カラー表現

### UNITS

座標系の単位を表現する以下の 16 ビットの符号付きデータ。

- >0: 1 cm 当たりの座標数
- <0: 1 inch 当たりの座標数の負数
- =0: 未定義 (外側の座標系で定義される単位と同じ)

例 : 400 -- 1 / 400 cm 単位  
-240 -- 1 / 240 inch 単位

### CHSIZE

文字サイズを表現する以下の 16 ビットの符号無しデータ。

U USS SSSS SSSS SSSS

- U: 単位指定
- 0: 外側の座標系で定義される単位



- 1: 1/20 級 ( 1 / 80 mm ) 単位。
- 2: 1/20 ポイント ( 1 / 1440 inch ) 単位。
- 3: 予約

S: U で指定した単位での文字サイズ。0 の場合は未定義であることを意味する。

## SCALE

行間隔等を表現する以下の 16 ビットの符号無しデータ。

1NNN NNNN NNNN NNNN -- 絶対指定  
 OAAA AAAA BBBB BBBB -- 比率指定

絶対指定 : 座標系単位での値 ( N )  
 比率指定 : 基準値 × ( A / B ) の値  
 ( B = 0 の時は比率 1 と見なす )

## RATIO

比率を表現する以下の 16 ビットの符号無しデータ。

AAAA AAAA BBBB BBBB

A / B の値が比率となる ( B = 0 の時は比率 1 と見なす )

## 3.4.2 管理情報セグメント

説明:

TAD データの解釈のために必要な バージョン等の管理情報を保持するためのセグメントで、実身等の 1 つの独立した TAD データの場合は、TAD 本体 ( 文章 / 図形データ ) 前に必ずなくてはならない。このセグメントは参照情報として意味を持つが、バージョンが異なる場合の処理はインプリメントに依存する。原則として、バージョンアップがあった場合、仕様は上位互換となる。

データ形式:

```
ID : TS_INFO          -- 管理情報セグメント
LEN : ~
DATA: UH subid        -- 項目 ID
      UH sublen       -- 項目のバイト数
      UH data[]       -- 項目データ本体(sublen バイト)
      :
      :
      ;上記の繰り返し>
```

データ説明:

subid :

管理情報の項目 ID を示す。

sublen :

項目データのバイト数を示す。バイト数は必ず偶数となる。

data[ ] :

項目データの本体を示す、sublen バイトのデータ。データの内容は項目 ID に依存する。

- 管理情報セグメントは、上記の項目データの繰り返しの形式を持つ。
- 項目データとしては、以下のものが定義されている。

subid: 0 -- TAD 規格のバージョン番号  
sublen: 2  
data: UH ver -- 以下の形式の 3 桁 BCD コードによるバージョン番号

0000 AAAA BBBB CCCC : バージョン A.BC

例:バージョン1.2のとき    AAAA = 0001  
                              BBBB = 0010  
                              CCCC = 0000

subid: 1~ -- 予約

### 3.4.3 文章開始 / 終了セグメント

説明:

文章データの開始および終了を表すセグメントで、文章開始セグメントと文章終了セグメントに囲まれた部分が1つの文章データとなる。

データ形式:

ID : TS\_TEXT -- 文章開始セグメント  
LEN : 24  
DATA: RECT view -- 表示領域  
      RECT draw -- 描画領域  
      UNITS h\_unit -- 水平ユニット  
      UNITS v\_unit -- 垂直ユニット  
      UH lang -- デフォルト言語  
      UH bgpat -- 背景パターン ID  
ID : TS\_TEXTEND -- 文章終了セグメント  
LEN : 0  
DATA: なし

データ説明:

view:

外側の座標系で定義される文章データの表示領域を示し、図形データ中に埋め込まれた埋め込み文章データの場合のみ、意味を持つ。

draw:

描画領域を示し、図形データ中に埋め込まれた埋め込み文章データの場合のみ、その大きさのみが意味を持ち、レイアウトすべき領域の大きさを示す。

h\_unit:

描画領域の水平の座標単位を示す UNITS タイプのデータ。

v\_unit:

描画領域の垂直の座標単位を示す UNITS タイプのデータ。

lang:

文章データ内で使用しているデフォルト言語を示す。  
デフォルト言語とは、データ中に言語指定コードによる言語切り換えが現れるまでの言語を意味する。

日本語: 0x0021

bgpat:

文章データの背景パターン ID を示し、図形データ中に埋め込まれた埋め込み文章データの場合

のみ意味を持つ。0は背景が透明であることを示す。  
パターンIDに関しては、「[パターン定義セグメント](#)」を参照の事。

### 3.4.4 図形開始/終了セグメント

説明:

図形データの開始および終了を表すセグメントで、図形開始セグメントと図形終了セグメントに囲まれた部分が1つの図形データとなる。

データ形式:

```
ID : TS_FIG          -- 図形開始セグメント
LEN : 24
DATA: RECT   view    -- 表示領域
      RECT   draw    -- 描画領域
      UNITS  h_unit   -- 水平ユニット
      UNITS  v_unit   -- 垂直ユニット
      W     ratio    -- 倍率
ID : TS_FIGEND      -- 図形終了セグメント
LEN : 0
DATA: なし
```

データ説明:

view:

外側の座標系で定義される図形データの表示領域を示し、埋め込み図形データの場合のみ、意味をもつ。文章データ中に埋め込まれた図形データの場合、viewが空の場合は、文字サイズ比例という特別な意味を持つ。文字サイズ比例とは、描画領域の高さを文字サイズと等しいように相似変形したものの表示領域とするもので、埋め込み図形を文字として取り扱いたい場合に使用する(従来の外字機能)。

draw:

描画領域を示し、図形データの領域の大きさと座標系を定義する。

h\_unit:

描画領域の水平の座標単位を示す UNITS タイプのデータ。

v\_unit:

描画領域の垂直の座標単位を示す UNITS タイプのデータ。

### 3.4.5 画像セグメント

説明:

画像セグメントは、画像データを表すセグメントであり、TAD文章データ、図形データの中に埋め込まれる。画像セグメントは、それ自体で独立した座標単位およびカラー情報を持ち、BTRON仕様[ディスプレイ・プリミティブ](#)で規定されている[圧縮ビットマップ形式](#)で表現される。

データ形式:

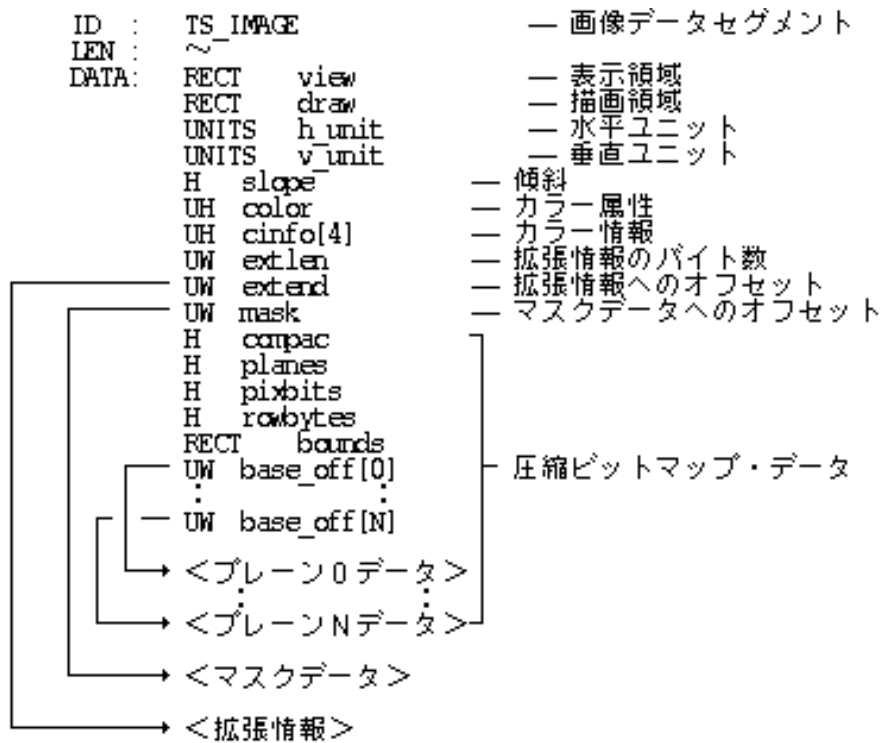


図 10 : 画像セグメントのデータ形式

データ説明:

view:

外側の座標系で定義される画像セグメントの表示領域を示す。文章データ中に埋め込まれた画像データの場合、view が空の場合は、文字サイズ比例という特別な意味を持つ。

draw:

描画領域を示し、画像セグメントの領域の大きさと座標系を定義する。画像データの圧縮ビットマップの bounds で示される領域内の draw で指定される長方形領域が描画対象となる。空の場合は、圧縮ビットマップの bounds と同一と見なされる。

h\_unit:

描画領域の水平の座標単位を示す UNITS タイプのデータ。

v\_unit:

描画領域の垂直の座標単位を示す UNITS タイプのデータ。座標系単位は、画像データの実際のサイズを示すためのものであるが、実際の描画では、draw 領域を view 領域にマッピングして描画されるため使用されない。

slope:

画像の傾きを示す。画像上の水平軸の座標系の水平軸に対する反時計周りの角度 ( 0 ) で示される ( 360 度の剰余が有効 )。slope < 0 は、未定義を意味する。

color:

カラー方式を示す。

xxxx xxxx lxxx PRRR

R:

- 0 -- 白黒
- 1 -- RGB カラー
- 2 -- CMY カラー
- 3 ~ -- その他 ( 予約 )

P:

- 0 -- 直接方式 ( カラーマップ無し )
- 1 -- カラーマップ方式 ( パレット方式 )

l:

- 0 -- 通常表現 ( ピクセル値 0 が白、最大値が黒に対応 )
- 1 -- 反転表現 ( ピクセル値 0 が黒、最大値が白に対応 )

x:

- 予約 ( 0 )

cinfo:

カラー情報を示す。

<カラーマップ方式の場合>

カラーマップ ( ピクセル値に対応するカラー表現を示した配列 ) のバイト数、およびオフセットを示す。カラーマップのバイト数 ( cinfo[0] ) = 0 の場合は、カラーマップは未定義であることを意味する。

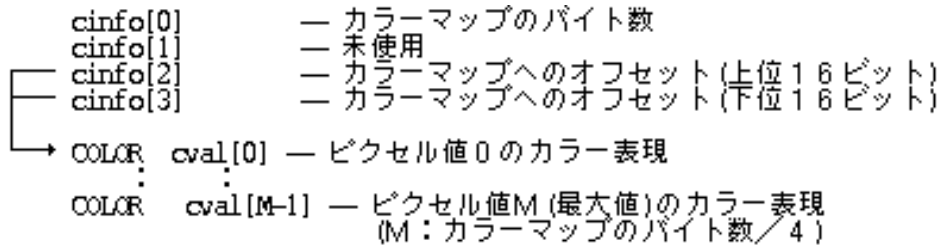


図 11 : カラーマップ

<直接方式の場合>

カラー表現データ ( ピクセル値 ) がどのようにカラー表現に対応するかを示す。cinfo[0] = 0 の場合は、カラー表現データは未定義であることを意味する。

R = 0 ( 白黒 ) の時:

- cinfo[0] -- 白黒の有効階調ビット数とその位置
- cinfo[1] -- 未使用
- cinfo[2] -- 未使用
- cinfo[3] -- 未使用

R = 1 ( RGB ) の時:

- cinfo[0] -- 赤の有効階調ビット数とその位置
- cinfo[1] -- 緑の有効階調ビット数とその位置
- cinfo[2] -- 青の有効階調ビット数とその位置
- cinfo[3] -- 未使用

R = 2 ( CMY ) の時:

- cinfo[0] -- シアンの有効階調ビット数とその位置
- cinfo[1] -- マゼンタの有効階調ビット数とその位置
- cinfo[2] -- 黄の有効階調ビット数とその位置
- cinfo[3] -- 黒の有効階調ビット数とその位置

各データの下位バイトは階調ビット数を示し、上位バイトはピクセル値上の LSB を 0 としたビット位置を示す。例として、ピクセル値の表現が、....BBBBGGGGRRRRR の場合は、

- cinfo[0] 0x0006
- cinfo[1] 0x0606
- cinfo[2] 0x0c04

となる。

extlen:

拡張情報のバイト数を示す。= 0 の場合は拡張情報が存在しないことを意味する。

extend:

拡張情報へのオフセットを示し、extlen = 0 の場合のみ有効である。extlen = 0 の場合の値は無視されるがフィールドとしては必要となる。拡張情報は、以下のいずれかの形式であり、ext\_id により、その意味が規定される。ext\_id の MSB は形式の区別に使用され、下位の 15 ビットの値が ID としての意味を持つ。

直接形式:

```
UH  ext_id  -- 拡張情報のタイプ ( ext_id の MSB = 0 )
UH  len     -- 拡張情報のバイト数
UB  data[]  -- 拡張情報本体 ( len バイト )
```

間接形式:

```
UH  ext_id  -- 拡張情報のタイプ( ext_id の MSB = 1 )
UH  len     -- 拡張情報のバイト数
UW  offset  -- 拡張情報本体へのオフセット
```

現在、規定されている拡張情報は、以下のものである。

ext\_id = 0:

画像の背景色指定

```
len : 4
data: COLOR bgcol -- 背景色の指定
```

1 ~ 16383:

(予約)

16384 ~:

未定義 (アプリケーション依存)

mask:

マスク用の圧縮ビットマップへのオフセットを示す。マスクが存在しない場合は 0 となる。マスクは、1 プレーンの圧縮ビットマップデータであり、マスクの "1" に対応するピクセルは不透明であり、"0" に対応するピクセルは透明であることを意味する。マスクの圧縮ビットマップの構造はイメージデータのビットマップの構造と圧縮方式も含めて同一である。

compac:

これ以降は、ディスプレイ・プリミティブで規定されている圧縮ビットマップ形式のデータであり、各プレーンのデータへのアドレスがオフセットになっている点が異なっている。[\(圧縮ビットマップ形式のデータに関しては BTRON 仕様ディスプレイプリミティブの仕様を参照のこと\)](#) なお、圧縮ビットマップ形式には非圧縮 (compac = 0) の場合も含まれる。

- オフセットは、すべてセグメントデータの先頭 (view) からのバイトオフセットである。

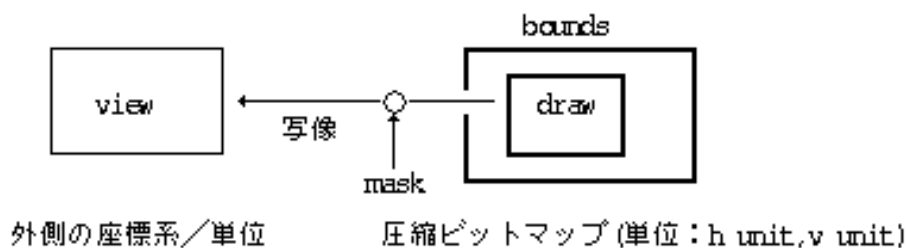


図 12: 画像セグメント

## 3.4.6 仮身セグメント

説明:

仮身セグメントは、仮身を示すセグメントであり、TAD 文章データ、図形データの中に埋め込まれて、他の実身の参照を意味する。仮身セグメントには、仮身を表示するための各種の情報、およびデフォルト・アプリケーション実行のための固有データ情報が入る。仮身自体の内容は、リンク(仮身)レコードとして別に存在し、実身中の出現順により1対1に仮身セグメントとの対応が取られる。リンク・レコードにはBTRON仕様[ファイル管理機能](#)で使用するファイルを示す情報の他に、以下の情報が含まれている。

- 仮身タイプ/属性
- 続柄インデックス
- デフォルト・アプリケーション ID

仮身セグメントおよびリンクレコードは、BTRON仕様[実身/仮身マネージャ機能](#)によりサポートされるため、通常、アプリケーションは直接アクセスしない。

データ形式:

|       |        |            |    |                  |
|-------|--------|------------|----|------------------|
| ID    | :      | TS_VOBJ    | -- | 仮身セグメント          |
| LEN   | :      | ~          |    |                  |
| DATA: | RECT   | view       | -- | 表示領域             |
|       | H      | height     | -- | 開いた場合の仮身高さ       |
|       | CHSIZE | chsz       | -- | 文字サイズ            |
|       | COLOR  | frcol      | -- | 枠の色              |
|       | COLOR  | chcol      | -- | 文字の色             |
|       | COLOR  | tbcoll     | -- | タイトル背景色          |
|       | COLOR  | bgcol      | -- | 開いた場合の背景色        |
|       | UH     | dlen       | -- | 固有データのバイト長       |
|       | UB     | data[dlen] | -- | 固有データ( dlen バイト) |

データ説明:

view:  
外側の座標系で定義される仮身セグメントの表示領域を示す。

height:  
開いた場合の仮身高さを view と同じ座標系で示す。-1 は開けないことを意味し、0 はデフォルトの高さで開くことを意味する。

chsz:  
仮身タイトルの文字サイズを示す、CHSIZE タイプのデータ。

frcol:  
仮身表示枠の色を示す。

chcol:  
仮身タイトルの文字の色を示す。

tbcoll:  
仮身タイトルの背景色を示す。

bgcol:  
開いた仮身の表示領域の背景色を示す。

data:  
デフォルト・アプリケーションの機能付箋の固有データであり、dlen はそのバイト長を示す。

固有データの内容は対応するデフォルトアプリケーションにより定義される。

### 3.4.7 機能付箋セグメント

説明:

機能付箋セグメントは、アプリケーションを起動するための各種のパラメータを保持するもので、他に付箋自身の表示用の情報も保持している。通常、機能付箋セグメントは、独立した機能付箋レコードまたは実行機能付箋レコードとして存在する。機能付箋セグメントが独立した機能付箋レコードまたは実行機能付箋レコードとして存在する場合には、セグメント先頭のセグメント ID、0xFF、およびデータ長のフィールドは省略されレコードの内容は DATA 部のみとなる。セグメント ID とデータ長は、レコードタイプとレコード長で示される。

データ形式:

|       |        |            |    |                    |
|-------|--------|------------|----|--------------------|
| ID    | :      | TS_FFUSEN  | -- | 機能付箋セグメント          |
| LEN   | :      | ~          |    |                    |
| DATA: | RECT   | view       | -- | 表示領域               |
|       | CHSIZE | chsz       | -- | 文字サイズ              |
|       | COLOR  | frcol      | -- | 枠の色                |
|       | COLOR  | chcol      | -- | 文字の色               |
|       | COLOR  | tbcoll     | -- | タイトル背景色            |
|       | UH     | pict       | -- | ピクトグラム / タイプ       |
|       | UH     | appl[3]    | -- | アプリケーション ID        |
|       | UB     | name[32]   | -- | 付箋名                |
|       | UB     | type[32]   | -- | データタイプ名            |
|       | UH     | dlen       | -- | 固有データのバイト長         |
|       | UB     | data[dlen] | -- | 固有データ ( dlen バイト ) |

データ説明:

view:

外側の座標系で定義される機能付箋セグメントの表示領域を示す。

chsz:

機能付箋タイトルの文字サイズを示す、CHSIZE タイプのデータ。

frcoll:

機能付箋表示枠の色を示す。

chcol:

機能付箋タイトルの文字の色を示す。

tbcoll:

機能付箋タイトルの背景色を示す。

pict:

機能付箋のピクトグラムを示す。0 は付箋のデフォルトを意味する。

appl:

対応するアプリケーション ID を示す。

name:

付箋名を示す。

type:

アプリケーション ID に対応したデータタイプ名を示す。

data:



機能付箋の固有データであり、dlen はそのバイト長を示す。固有データの内容は対応するアプリケーションにより定義される。

### 3.4.8 指定付箋セグメント

説明:

指定付箋セグメントは、TAD データの中に埋め込まれる、アプリケーション固有のデータとして解釈されるデータである。

1 つ以上の指定付箋セグメントを独立した指定付箋レコードとして存在させることができる。指定付箋レコードに1つの指定付箋セグメントしか含まれない場合でも、セグメント先頭のセグメント ID、0xFF、およびデータ長のフィールドは省略することはできない。

固有データのバイト長は、仮身セグメントや、機能付箋セグメントと異なり、UWとなる。

データ形式:

|       |           |           |                       |
|-------|-----------|-----------|-----------------------|
| ID :  | TS_DFUSEN | --        | 指定付箋セグメント             |
| LEN : | ~         |           |                       |
| DATA: | RECT      | view      | -- 表示領域               |
|       | CHSIZE    | chsz      | -- 文字サイズ              |
|       | COLOR     | frcol     | -- 枠の色                |
|       | COLOR     | chcol     | -- 文字の色               |
|       | COLOR     | tbcoll    | -- タイトル背景色            |
|       | UH        | pict      | -- ピクトグラム / タイプ       |
|       | UH        | appl[3]   | -- アプリケーション ID        |
|       | UB        | name[32]  | -- 付箋名                |
|       | UW        | dlen      | -- 固有データのバイト長         |
|       | UB        | dat[dlen] | -- 固有データ ( dlen バイト ) |

データ説明:

view:  
外側の座標系で定義される指定付箋セグメントの表示領域を示す。

chsz:  
指定付箋タイトルの文字サイズを示す、CHSIZE タイプのデータ。

frcol:  
指定付箋表示枠の色を示す。

chcol:  
指定付箋タイトルの文字の色を示す。

tbcoll:  
指定付箋タイトルの背景色を示す。

pict:  
指定付箋のピクトグラムを示す。0 は付箋のデフォルトを意味する。

appl:  
対応するアプリケーション IDを示す。

name:  
付箋名を示す。

data:  
指定付箋の固有データであり、dlen はそのバイト長 固有データの内容は対応するアプリケーション

ンにより定義される。

---

[この章の目次にもどる](#)

[前頁:第2章 TRON コード体系にもどる](#)

[次頁:3.5 文章付箋セグメントにすすむ](#)

## 3.5 文章付箋セグメント

### 3.5.1 概要

文章付箋セグメントは、文章データ中に埋め込まれる指定付箋であり、ページ割付け指定、行書式指定、文字指定、文字修飾指定等の主に文章データを2次元的に表現するための情報を保持している。文章付箋セグメントは文章データ中に埋め込まれるため、位置情報を持たない。また、名称、表示色等の表示情報も持たないため、文章付箋セグメント自体の表示方法はアプリケーションに依存する。標準文章付箋セグメントとして、以下のものが定義されている。

| 標準文章付箋セグメント    | セグメントID             |
|----------------|---------------------|
| 文章ページ割付け指定付箋   | TS_TPAGE (0xA0)     |
| 行書式指定付箋        | TS_TRULER (0xA1)    |
| 文字指定付箋         | TS_TFONT (0xA2)     |
| 特殊文字指定付箋       | TS_TCHAR (0xA3)     |
| 文字割付け指定付箋      | TS_TATTR (0xA4)     |
| 文字修飾指定付箋       | TS_TSTYLE (0xA5)    |
| (予約)           | ----- (0xA6 ~ 0xAC) |
| 変数参照指定付箋       | TS_TVAR (0xAD)      |
| 文章メモ指定付箋       | TS_TMEMO (0xAE)     |
| 文章アプリケーション指定付箋 | TS_TAPPL (0xAF)     |

文章付箋セグメントは以下に示す構造を持ち、サブIDにより詳細な内容が規定される。なお、サブIDは、0 ~ 127 の範囲の値を TAD 標準として使用し、128 ~ 255 は未定義 (アプリケーション依存) とする。アプリケーション依存の文章付箋セグメントには、アプリケーション ID が含まれていないため、異なるアプリケーション間での互換性は保証されない。

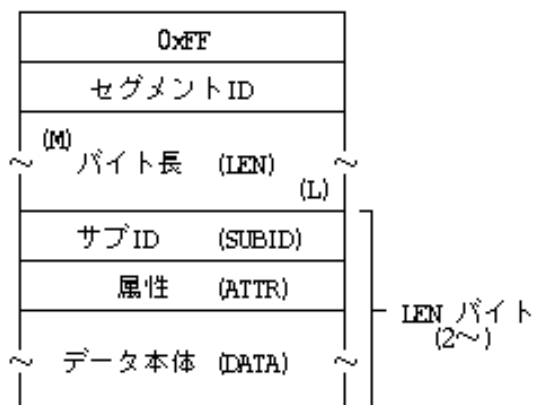


図 13: 文章付箋の構造

文字割付け付箋、および文字修飾付箋は、サブIDにより区別される開始付箋と終了付箋に囲まれた範囲に対してのみ有効となる。他の文章付箋セグメントは、環境条件の指定であるため有効範囲の指定は特になく、再度、環境が指定されるまで有効となる。文章付箋セグメントの機能は非常に広範囲に渡るため、基本レベルと拡張レベルの文章付箋セグメントにレベル分けされており、原則的に文章を取り扱うアプリケーションは、少なくとも基本レベルはサポートしているものとする。

### 3.5.2 文章ページ割付け指定付箋

文章ページ割付け指定付箋は、文章データを指定した用紙上にページ単位でレイアウトを行なうための情報を保持する付箋であり、図形データ中に埋め込まれた文章データの場合は通常無視される。

#### 用紙指定付箋:基本レベル

文章データをレイアウトする用紙の大きさを指定する。ここで指定するのは、レイアウト用紙であり、実際に印刷される時の用紙(印刷用紙)ではない。レイアウト用紙にレイアウトした結果を、実際にどの印刷用紙に、どのように印刷するか(例えば、印刷方向(ポートレイト/ランドスケープ)は、印刷時の指定による。例えばレイアウト用紙がA4の時、印刷時に、印刷用紙をA5とすると、A4用紙にレイアウトされた結果が、A5用紙に縮小されて印刷されることになる。

```
LEN : 14
SUBID: 0
ATTR : UB attr -- 面付け、綴じ方向指定
DATA : UH length -- 用紙の長さ
      UH width -- 用紙の幅
      UH top -- オーバーレイ上マージン (天)
      UH bottom -- オーバーレイ下マージン (地)
      UH left -- オーバーレイ左マージン (ノド)
      UH right -- オーバーレイ右マージン (小口)
```

attr : xxxx xxDP

#### P: 面付け指定

0: 1面付け。

1ページ単位で綴じるようにする。

1: 2面付け。

2ページ単位で見開きで綴じるようにする。この場合、偶数ページのオーバーレイ左右マージン、および本文レイアウト領域の左右マージンは、自動的に左右逆転した値が適用される。

#### D: 綴じ方向指定

0: 左綴じ。

用紙の左側で綴じる。

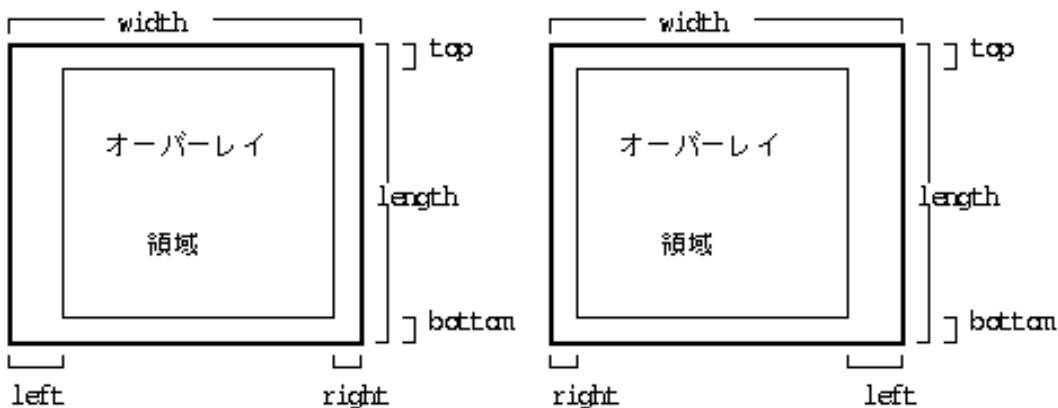
1: 右綴じ。

用紙の右側で綴じる。この場合、オーバーレイ左右マージン、および本文レイアウトの左右マージンは、自動的に左右逆転した値が適用される。

- 左マージンはノド(綴じしろ)指定であるため、綴じ方向、面付けの指定により左右逆転することになる。

オーバーレイ・マージンは、後述するオーバーレイ領域のマージン指定であり、実際に文字をレイアウトする本文領域のマージンではない。用紙の大きさ、オーバーレイ・マージンの指定は、すべて座標系単位で示される。

用紙とオーバーレイ・マージンの関係を以下に示す。



1面付け指定、または  
2面付け指定の奇数ページの時

2面付け指定の偶数ページの時

図 14: レイアウト用紙とオーバーレイマージン

### マージン指定付箋:基本レベル

文章データの本文のレイアウト領域のマージンを指定する。

```

LEN : 10
SUBID: 1
ATTR : UB      -      -- 未使用
DATA : UH      top    -- 上マージン ( 天 )
       UH      bottom -- 下マージン ( 地 )
       UH      left   -- 左マージン ( ノド )
       UH      right  -- 右マージン ( 小口 )
    
```

ここでのマージンは、本文のレイアウト用のマージンを意味する。通常、本文のレイアウト領域はオーバーレイ領域より小さい。用紙指定で2面付け指定を行なった場合は、偶数ページでは、左右マージンは逆転する。また、右綴じの指定を行なった場合も、左右マージンは逆転する。マージンの値が、0xFFFFの場合は、そのマージンは変更せずに現在の値を適用することを意味する。例えば、左マージンのみを変更する場合は、上マージン、下マージン、右マージンの値を0xFFFFとする。マージンの指定は、すべて座標系単位の値で示される。

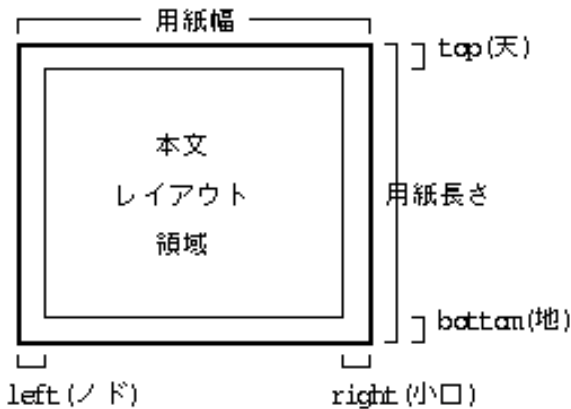


図 15: レイアウト用紙と本文マージン

### コラム指定付箋:拡張レベル

ページのレイアウトを複数コラム(段組み)で行なうための指定であり、コラム数と、コラムマージン(コラム間余白)を指定する。各コラムは均等幅となる。

```

LEN : 4, 6
SUBID: 2
ATTR : UB      column  -- コラム数、およびコラム長均等化指定
DATA : UH      colsp   -- コラムマージン
       [ UH      colline -- コラム間罫線 ]
    
```

column : Kxxx CCCC

CCCC :

コラム数 (0 ~ 15) コラム数が0, 1の場合は、コラム数1となり、段組は行なわれない。

K :

コラム長均等化指定 0: コラム長の均等化を行なわない。 1: コラム長の均等化を行なう。

colsp:

コラムマージンを座標系単位で指定する。 column = 0, 1の時は意味を持たない。

colline :

コラム間に引かれる縦の罫線属性の指定(省略時は罫線なし)。下位バイトで指定した以下の線属性の縦の罫線をコラム間の中心位置に引く(上位バイトは予約)。

DIWW KKKK

D:

線の本数

0: 一本

1: 二本

I:

線の濃さ

0: 100 %

1: 50 %

W:

線の太さ

0: なし (線は引かれない)

1: 細線

2: 中線

3: 太線

K:

線の種類

0: 実線

1: 破線

2: 点線

3: 一点鎖線

4: 二点鎖線

5: 長破線

6: 波線

7: (予約)

8~: 未定義 (アプリケーション依存)

段組みの場合は、改コラムコード (0x0B) により、次 (右) のコラムに移行する。例えば、column = 3 の時は、以下に示す段組みとなる。

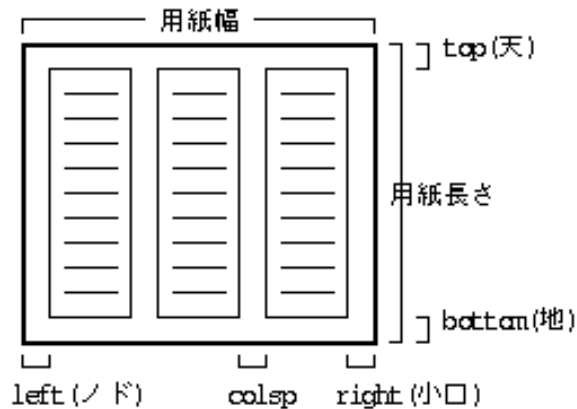


図 16: 段組みレイアウト

コラム長の均等化を指定した場合、次のコラム指定付箋の直前までの文章データを各コラムに均等に振り分け、文章データが格納された各コラムの長さが揃うようにレイアウトされる。直後に段抜き見出しを配置するような場合や文書の最終データの場合に使用される。コラム長の均等化を指定しない場合は、改コラムコードが現れない限り、各コラムに順に文章データをページの最下部まで詰めてレイアウトされる。

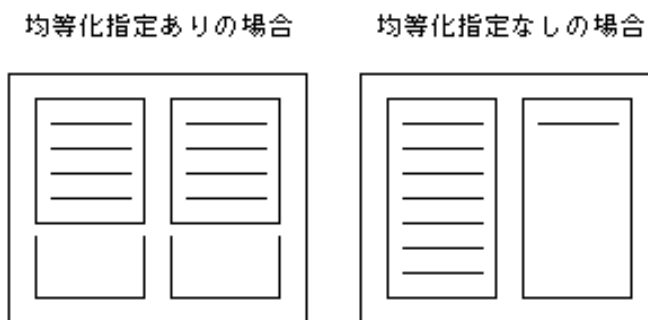


図 17: コラム長の均等化指定

ページ途中でコラム数を変更する場合、前のコラムが均等化指定されていれば、指定位置から新しいコラム数に変化するが、固定長指定の場合は次ページ以降しか変化しない場合がある。例えば、2 段組から段組なしへ変更した場合を以下に示す。

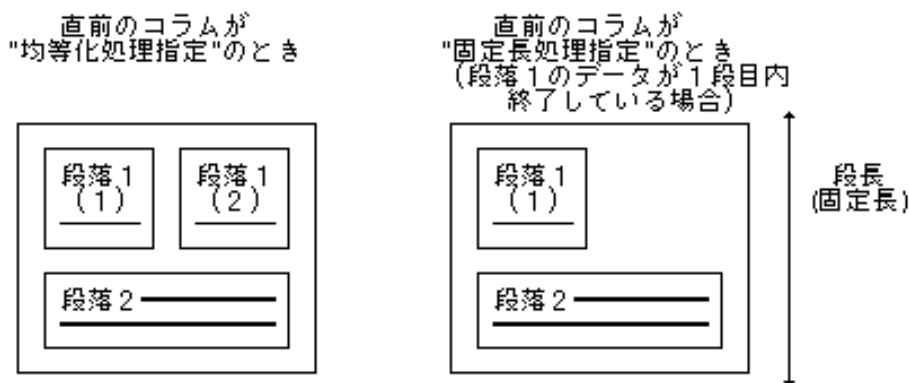


図 18: コラム数を変更した場合 (1)

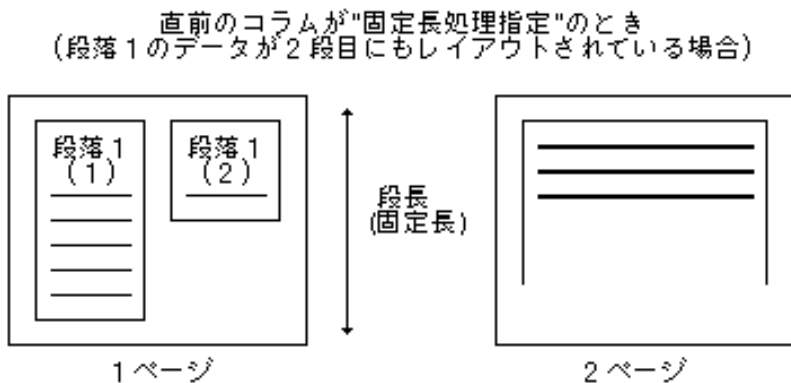


図 19: コラム数を変更した場合 (2)

### 3.4.2.4 用紙オーバーレイ定義付箋: 拡張レベル

用紙上に重ねてレイアウトするオーバーレイ情報を定義する。オーバーレイは最大 16 枚まで重ねることができる。この機能は、通常は柱、ノンブル等の指定に使用されるが、用紙の右マージン(小口)や左マージン(ノド)部分、さらに本文のレイアウト領域にも重なった指定も可能である。この指定付箋はオーバーレイの定義を行なうだけで、実際にそのオーバーレイが有効となるのは、用紙オーバーレイ指定付箋でオーバーレイ番号が指定された場合である。

LEN : ~  
 SUBID: 3  
 ATTR : UB attr -- オーバーレイ番号 / ページ適用属性  
 DATA : UB data[] -- オーバーレイする文章データ  
 attr:

## オーバーレイ番号/ページ適用属性

xxPP NNNN

N:

オーバーレイ番号 (0 ~ 15)

P:

ページ適用属性

0: 奇数 / 偶数ページともに適用。

1: 奇数ページにのみ適用。

2: 偶数ページにのみ適用。

オーバーレイ番号は、オーバーレイの位置を示す番号であり、異なる番号のオーバーレイは重ねて使用することができる。適用ページに対して同一のオーバーレイ番号を持つものが既に定義されていた場合は、以前の定義は無効となる。オーバーレイ番号0のオーバーレイが一番下に重ねられ、オーバーレイ番号15のオーバーレイが一番上に重ねられる。

data:

用紙指定で指定された用紙上のオーバーレイマージンで指定されたオーバーレイ領域内にレイアウトする文章データであり、通常の記事データから、先頭の記事開始セグメント、最後の記事終了セグメントを取り除いたものである。この文章データの中には図形データを含むことができる。この文章データをレイアウトした結果が用紙にオーバーレイされる。

例として、単純な柱、ノンブルは、以下の文章データにより定義される。

【文字サイズ指定等の環境設定】

AAA【充填文字】BBB【充填文字】CCC<改段落>

【充填行】

DDD【充填文字】【変数参照:ページ番号】【充填文字】FFF<改段落>

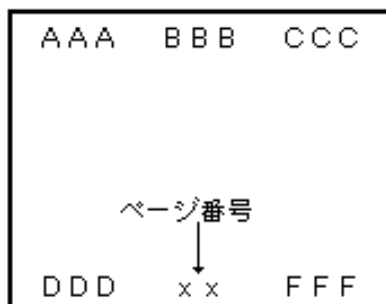


図 20 : 用紙オーバーレイ

用紙オーバーレイ指定付箋 : 拡張レベル

指定した用紙オーバーレイを、この付箋がレイアウトされたページから実際に適用することを指定する。用紙オーバーレイ指定付箋が現れるとそれまでの指定は無効となり、その付箋で指定されるオーバーレイ番号群によって新たな設定がされる。オーバーレイ指定をすべて取り消すためには、オーバーレイ番号を1つも指定しない用紙オーバーレイ指定付箋の指定が必要となる。用紙オーバーレイ処理がされた結果がレイアウト用紙となり、その上に本文がレイアウトされる。

LEN : 4

SUBID: 4

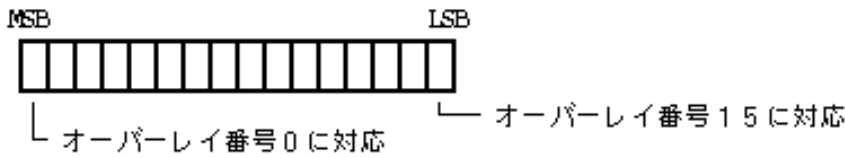
ATTR : UB --- -- 未使用

DATA : UH overlay -- オーバーレイ指定

overlay:

適用するオーバーレイ番号の指定。"1"のビットに対応するオーバーレイ番号を持つオーバーレイを適用する。"1"のビットが複数個ある場合は、複数のオーバーレイをオーバーレイ番号0から順番に重ね書きすることを意味する。指定するオーバーレイ番号は、既に定義済みでなくてはならない。未定義のオーバーレイ番号を指定した場合、その指定は無視される。





例：【定義1】 【定義2】 —— 【指定1】 —— 【指定1,2】 —— 【指定なし】 ——

【定義N】 用紙オーバーレイNの定義  
 【指定N】 用紙オーバーレイNの指定

有効なオーバーレイ：

—— の範囲：オーバーレイなし  
 —— の範囲：オーバーレイ定義1が有効  
 —— の範囲：オーバーレイ定義1, 2が有効

図 21：用紙オーバーレイ指定付箋

### 枠あけ指定付箋：拡張レベル

指定した大きさの枠をあけてレイアウトすることを指定する。この付箋の直後に図形データが存在する場合は、その図形データを枠内の中央に配置する。

LEN : 10  
 SUBID: 5  
 ATTR : UB attr -- 枠あけ属性  
 DATA : RECT area -- 枠あけ領域  
 attr:

APKx HHVV

A:

0 ページ内位置属性指定  
 1 ページ内絶対位置指定

P:

0 付箋の位置以降に割り付ける。  
 1 付箋の位置と同一ページ割り付ける。

K:

0 通常枠(枠の左右に本文データがレイアウトされる)  
 1 行扱い枠(枠の左右に本文データはレイアウトされない)

H:

縦方向の枠位置 (A = 0 の時のみ有効)  
 0: 任意  
 1: 上端  
 2: 中央  
 3: 下端

V:

横方向の枠位置 (A = 0 の時のみ有効)  
 0: 任意  
 1: 左端  
 2: 中央  
 3: 右端

H = 0, V = 0 の場合の枠の配置アルゴリズムはインプメントに依存する。

area:

枠あけ領域の指定。座標系の単位で指定。

ページ内位置属性指定の場合：

area の大きさのみ有効。位置は無視され、attr で指定した条件に適合する位置に配置される。段

組みの場合は、各段の中での位置を意味する。

ページ内絶対位置指定の場合：

area の大きさ、位置ともに有効であり、位置はページの左上を (0,0) とした位置に配置される。段組みの場合でも、段に無関係の絶対位置となる。

attr が P = 0 の時は、枠あけ指定付箋が現れた位置以降で、H, V あるいは area で指定された条件を満たす位置にレイアウトする。従って、付箋が現れたページに配置されないことがありうる。一方、P = 1 の時は、付箋の現れたページ内の条件を満たす位置に配置するため、文章データの再レイアウト処理が発生する。

例: 縦方向上端、横方向左端指定の場合

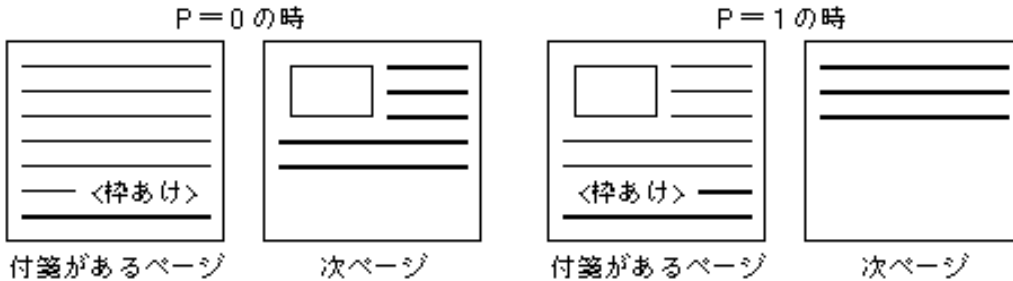


図 22 : 枠あけ指定

ページ番号指定付箋：拡張レベル

現在のページ番号、およびページ番号の増加ステップ数を指定する。この付箋が存在するページから有効となる。

LEN : 4  
SUBID: 6  
ATTR : B step -- ページの増加ステップ  
DATA: UH num -- ページ開始番号

【N1,N2】:

ページ番号指定付箋

N1: ページ開始ページ

N2: ページの増加ステップ

-N3-: ページ番号

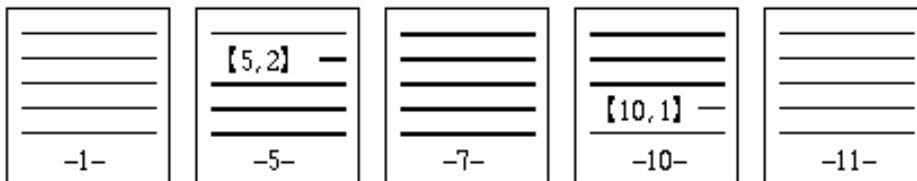


図 23 : ページ番号指定

条件改ページ指定付箋：拡張レベル

指定した条件に適合した場合に、改ページを行なう。

LEN : 2,4  
SUBID: 7  
ATTR : UB cond -- 条件  
DATA : [ SCALE remain -- 残り領域指定 ]

cond :

0: 偶数ページの時、改ページする。(奇数ページから始める)

1: 奇数ページの時、改ページする。(偶数ページから始める)

2: ページの残り領域が、remain で指定した値以下の時、改ページする。

3～：予約

remain：

ページの残り領域を指定する SCALE タイプのデータ。現在位置から、用紙の一番下（下マージンを含む）までの大きさが指定した値以下の場合に改ページする。比率指定の場合の基準値は、用紙の縦サイズ（上下マージンを含む）となる。cond 2 の場合は、意味を持たず、通常は省略される。

例 1：

見だしの直後で改ページされることを防ぐためには、下記のパラメータを持つ付箋を見だしデータの直前に設定する（この機能は、タブ書式指定付箋の属性 P = 1 の場合に相当する）。

cond ( 条件 ) = 2  
remain = ( 見だし行の高さ ) + ( 行間 ) + ( 本文行の高さ )

例 2：

1つの段落を1ページ内に収めるためには、下記のパラメータを持つ付箋をその段落データの直前に設定する（この機能は、タブ書式指定付箋の属性 P = 2 の場合に相当する）。

cond ( 条件 ) = 2  
remain = ( 段落全体の長さ ( 行間も含む ) )

充填行指定付箋：拡張レベル

用紙の空き領域を空行で充填する。1つのページに複数の充填行がある場合は、それぞれの充填行の高さは均等となる。

LEN : 2  
SUBID: 8  
ATTR: UB

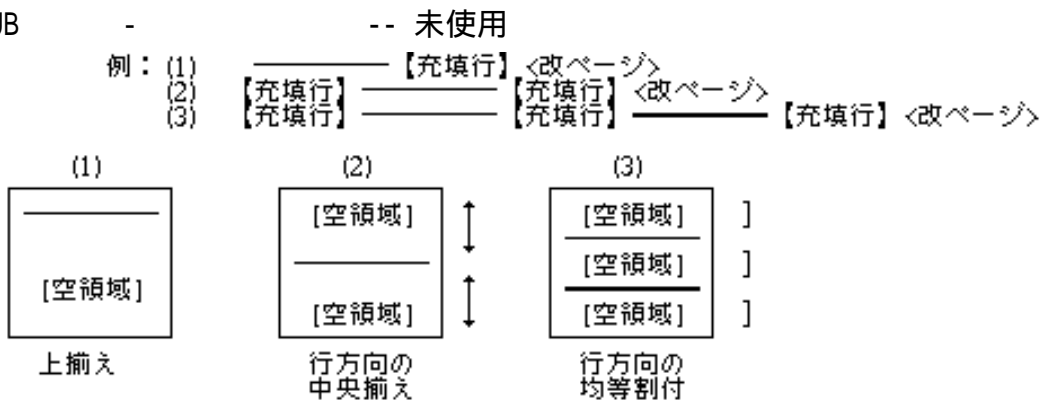


図 24：条件改ページ指定

### 3.5.3 行書式指定付箋

行書式指定付箋は、行書式に関連する指定を行なう付箋であり、必ず行または段落の区切りに存在する。行または段落の区切りでない場合は、この付箋の直前行または段落の区切りがあるものと見なされる。段落の概念を持たない従来のデータを交換する場合、改行機能を現わすためには改段落コード (0x0A) を用い、改行コード (0x0D) はフィールド内の改行でのみ使用する。行ピッチの値としては、改段落コードの場合には段落ピッチが、改行コードの場合には改行ピッチが用いられる。行書式としては、大きく以下の2種類があり、いずれも段落単位で指定することになる。

フィールド書式：

- 論理的なレコード構造を持つデータを表の形で表現するための書式であり、段落を1つのレコードと見なし、段落内のタブコードで区切られたデータを1つのデータ項目と見なす。
- 1つの行を複数の縦割りのフィールドに分割し、各フィールド内にデータ項目を配置する。データ項目がフィールドの幅を超えた場合は、フィールド内で自動的に改行され、次のフィールドにはみ出すことはない。
- フィールドの境界には罫線を引くことができ、全体として罫線付きの表の表現ができる。

○ 各フィールドに対して、フィールド内の揃え指定、小数点揃え位置、インデントマージンの指定ができる。

○ フィールド書式中で使用される制御コードの意味を以下に示す。

タブコード：

次(右)のフィールドへ移動する。

右端のフィールドでのタブコードは一番先頭(左)のフィールドへ移動する。

改行コード：

フィールド内での改行

改段落コード：

段落を終了し、一番先頭(左)のフィールドへの移動する。

タブ書式：

- 論理的なレコード構造を特に持たないデータに対する書式であり、1つの行は唯一の1つのフィールドから構成される。
- 任意個のタブストップ位置を設定可能で、タブコードにより現在の位置以降のタブストップ位置に移動する。
- タブストップとして小数点揃えタブが設定可能であり、タブコードにより移動した位置が小数点揃えタブの場合は、その位置に小数点が揃えられる。

| フィールド                    |            |             |           |
|--------------------------|------------|-------------|-----------|
| #1<br>左揃え                | #2<br>中央揃え | #3<br>小数点揃え | #4<br>右揃え |
| aaaaaa<br>aaaaaa<br>aaaa | bbbbbb     | 123.45      | cccc      |
| xx<br>xxx                | y<br>yyy   | 12.3        | zz        |

データ：

aaaa.a<タブ>bbbbbb<タブ>123.45<タブ>cccc<改段落>  
xx<改行>xxx<タブ>y<改行>yyy<タブ>12.3<タブ>zz<改段落>

図 25：フィールド書式の例

行間隔指定付箋：基本レベル

行の間隔を指定する。行の区切りとなる。

LEN : 4  
SUBID: 0  
ATTR : UB attr -- 属性指定  
DATA : SCALE pitch -- 間隔指定  
attr :

Dxxx xxxG

D:

方向(0:+方向 1:-方向)

G:

行アキ指定(0:行送り指定 1:行アキ指定)

横書きの場合は、以下ようになる。

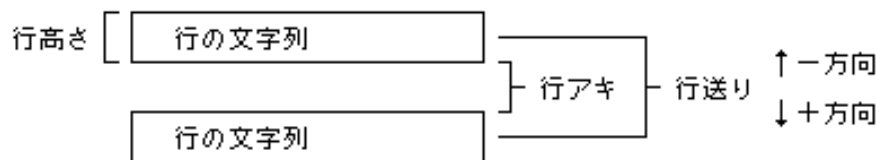


図 26 : 行間隔指定

行高さの定義は、行中の最大の文字の大きさ(拡大縮小された結果を含む)である。但し、行中の文字に付加された添字あるいはルビ文字列は行の高さには含まれない。埋め込み図形および仮身の高さは行高さに含まれるが、行間隔の基準値には含まれない。

pitch :

行間隔を指定する SCALE タイプのデータ。比率指定の場合の基準値は行高さ(ただし、埋め込み図形、仮身の高さを除く)となる。

- 行間隔指定は通常、行間隔指定付箋のある行と直後の行との間の行アキから反映される(行間隔は行の手前(上)に取られる)。
- 行間隔が-(マイナス)方向で、行方向に逆行するような場合(例えば、上 下の場合に次の行が前の行の上側にくるような場合)の描画に関しては、保証しない。  
つまり、行アキの-(マイナス)方向は、前の行の高さまでは保証するが、それ以上の行アキ、および行送りの-(マイナス)方向に関しては保証しない(重なりは保証するが、前に戻ってしまう場合は保証しない)。

行揃え指定付箋 : 基本レベル

行揃えの方法を指定する。行の区切りとなる。

```

LEN : 2
SUBID: 1
ATTR : UB      align      -- 行揃えの方法
align: 行揃えの方法
0: 左揃え
1: 中央揃え
2: 右揃え
3: 両端揃え
4: 均等揃え

```

両端揃えと均等揃えの違いは、揃える対象の文字列が割り付け対象領域の両端に接するか、両端に均等間隔の空白が埋められるかの違いである。



図 27 : 行揃え指定

タブ書式指定付箋 : 基本レベル

タブ書式を指定する。段落の区切りとなる。

```

LEN : ~
SUBID: 2
ATTR : UB      attr      -- 属性
DATA : SCALE  height    -- 書式高さ指定
      SCALE  pargap    -- 段落間隔指定
      H      left      -- 行頭マージン
      H      right     -- 行末マージン

```

H indent -- インデントマージン  
H ntabs -- タブストップ設定数  
UH tabs[] -- タブストップ位置 ( ntabs 個 )

attr :

Rxxx xxPP

R :

基準位置指定

0: 絶対位置 ( 用紙の左マージン、右マージンを基準 )

1: 相対位置 ( 直前の行頭マージン、行末マージンを基準 )

P :

段落のページ拘束の指定

0: ページ拘束なし

1: 見出し同一ページ

( 直後の段落と、続く段落の少なくとも1行を同一ページにレイアウトする ) 2: 全段落同一ページ

( 直後の段落から、次の書式指定のまでの間の全ての段落を同一ページにレイアウトする ) 3: 予約

height :

書式高さを指定する SCALE タイプのデータ。

pargap :

段落間隔を指定する SCALE タイプのデータ。比率指定の場合の基準値は行高さ ( ただし、埋め込み図形、仮身の高さを除く ) となる。height は書式指定付箋自身の高さ、即ち、前の段落と続く段落との間隔を示し、pargap は以降の段落の後の間隔を意味する。

例 :

< 段落 1 >

< 書式指定付箋 A >

< 段落 2 >

< 段落 3 >

< 段落 4 >

< 書式指定付箋 B >

< 段落 5 >

段落 1 と 2 の間隔 : A の height

段落 2 と 3 の間隔 : A の pargap

段落 3 と 4 の間隔 : A の pargap

段落 4 と 5 の間隔 : B の height

left :

行頭マージン ( 基準位置からの座標系単位での位置 )。右方向が正方向となり、基準位置が相対位置の場合は <0 も可。

right :

行末マージン ( 基準位置からの座標系単位での位置 )。左方向が正方向となり、基準位置が相対位置の場合は <0 も可。

indent :

インデント・マージン ( 座標系単位 )。先頭行にのみ適用される行頭マージンであり、行頭位置からの相対位置で指定される。右方向が正方向となり、<0も可。

ntabs :

タブストップの設定数。0の場合はタブストップの設定無しを意味し、<0の場合は変更無しを意味する。

tabs :

タブストップ位置を示す ntabs 個の要素の配列 ( 行頭位置からの座標系単位での位置 )。小数点揃えタブの場合は、タブストップ位置の負数を指定する。右方向が正方向となり、<0は不可。行頭位置は、常にタブストップ位置として定義されているものと見なされる。タブストップ位置の数 ( ntabs ) が 0 の場合でも行頭位置はタブストップ位置となる。従って、最後のタブストップ位置以降のタブコードは次の行の行頭への移動となる。基準位置が絶対位置の場合、以下のようになる。

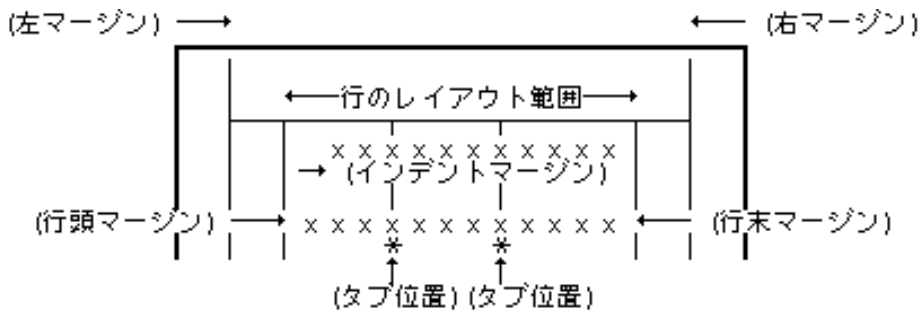


図 28 : タブ書式

### フィールド書式指定付箋 : 拡張レベル

フィールド書式を指定する。段落の区切りとなる。フィールド書式中では、データ列中のタブコードにより次の右側のフィールドへ移動し、改段落コードにより新たな下の段落へ移動する。フィールド書式は主に、タブコードで区切られたデータ1つのフィールドとした表形式のデータ表現に使用される。

```

LEN : ~
SUBID: 3
ATTR : UB      attr    -- 属性
DATA: SCALE   height   -- 書式指定付箋自身の高さ
      SCALE   pargap   -- 段落間隔
      UH      line     -- フィールド段落線属性
      H      nfld     -- フィールド設定数
      -- UH   fld     -- フィールド開始位置
      UH     left    -- フィールド内左マージン
      UH     right   -- フィールド内右マージン
      UH     margin  -- 小数点位置 / インデントマージン
      -- UH   f_attr -- フィールド属性
      :
      :

```

上記のデータをフィールド設定数分繰り返す。

attr:

Rxxx xxPP

R:

基準位置指定

0: 絶対位置 (用紙の左マージン、右マージンを基準)

1: 相対位置 (直前の行頭マージン、行末マージンを基準)

P:

段落のページ拘束の指定

0: ページ拘束なし

1: 見出し同一ページ

(直後の段落と、続く段落の少なくとも1行を同一ページにレイアウトする)

2: 全段落同一ページ

(直後の段落から、次の書式指定のまでの間の全ての段落を同一ページにレイアウトする)

3: 予約

height:

書式指定付箋自身の高さを指定する SCALE タイプのデータ。

pargap:

段落間隔を指定する SCALE タイプのデータ。

比率指定の場合の基準値は行高さ (ただし、埋め込み図形、仮身の高さを除く) となる。height は書式指定付箋自身の高さ、即ち、前の段落と続く段落との間隔を示し、pargap は以降の段落の後の間隔を意味する。

line:

フィールドの段落の境界に引かれる横線の属性。上位バイトは、フィールド書式の先頭に引かれる横線(上線)の属性を示す。右端の最後のフィールドに対して上線は常に引かれない。下位バイトは、フィールド内の各段落の直後に引かれる横線(下線)の属性を示す。実際に線が引かれるか否かは、各フィールド内で指定される。

各バイトは、以下に示す形式を持つ。

DIWW KKKK

D:

線の本数

0: 一本

1: 二本

I:

線の濃さ

0: 100 %

1: 50 %

W:

線の太さ

0: なし(線は引かれない)

1: 細線

2: 中線

3: 太線

K:

線の種類

0: 実線

1: 破線

2: 点線

3: 一点鎖線

4: 二点鎖線

5: 長破線

6: 波線

7: (予約)

8~: 未定義(アプリケーション依存)

nfld:

フィールド設定数。最後のフィールドは右端の指定がないため、最後のフィールドの右端を定義する必要がある場合は、フィールド定義数は必要なフィールド数+1となる。nfld<0とした場合はフィールドの設定を変更しないことを意味するので、各フィールドの設定を変化させることなく、書式高さ(height)、段落間隔(pargap)や段落線属性(line)のみ変更をすることが可能である。

fld:

フィールド開始左側の位置(基準位置からの座標系単位での位置)。フィールドの縦線はこの位置に引かれる。

left:

フィールド内の左マージン(フィールド開始位置からの座標系単位での位置)。

right:

フィールド内の右マージン(次のフィールド開始位置からの座標系単位での位置)。

margin:

小数点位置/インデントマージン(フィールド開始位置からの座標系単位での位置)。揃え指定により意味が異なる。

左揃え -- インデントマージン

中央揃え -- 無視される

右揃え -- 無視される

両端揃え -- インデントマージン

均等揃え -- 無視される



小数点揃え -- 小数点揃え位置

f\_attr :  
 フィールドの属性。  
 上位バイトは、フィールドの区切り線の属性を示し、下位バイトは、フィールド内の揃え指定を示す。

HxWW KKKK xxxx xAAA

H :  
 フィールドの段落の下線の有無。  
 0 : 下線なし  
 1 : 下線あり  
 H=1の時に、lineの下位バイトで定義された属性の下線が引かれる。

W, K :  
 フィールドの左側に引かれる縦線の属性。  
 W, Kの内容は、lineの場合と同じであり、W 0の場合に指定した属性の縦線が引かれる。

A :  
 フィールド内の揃え指定  
 0 : 左揃え  
 1 : 中央揃え  
 2 : 右揃え  
 3 : 両端揃え  
 4 : 均等揃え  
 5 : 小数点揃え

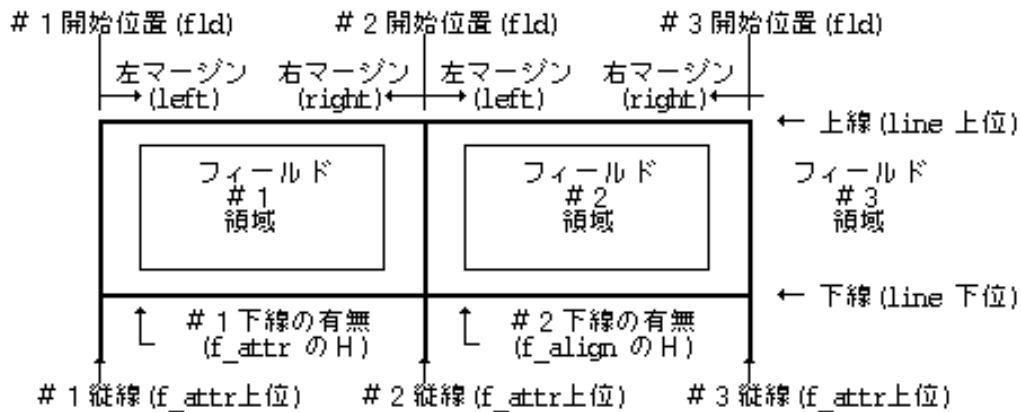


図 29 : フィールド書式(一般)

例 1 : 項目名-説明文の形式

データ列

|    |       |
|----|-------|
| 日本 | _____ |
|    | _____ |
|    | _____ |
| 米国 | ===== |
|    | ===== |
|    | ===== |
| 中国 | _____ |
|    | _____ |
|    | _____ |

【フィールド書式】  
 日本<TAB>——<改段落>  
 米国<TAB>——<改段落>  
 中国<TAB>——

フィールド1 項目名      フィールド2 説明文

図 30 : フィールド書式の例-1

例 2 : 原文-日本語対訳の形式

|          |       |
|----------|-------|
| I'm      | 私は    |
| In 1988, | 1988年 |
| After    | の後    |

フィールド1      フィールド2  
原文          日本語対訳

データ列

【フィールド書式】

I'm<TAB>私は<改段落>  
In 1988<TAB>1988年<改段落>  
After<TAB>の後

図 31 : フィールド書式の例-2

例 3 : 表形式 (1)

|   |    |
|---|----|
| A | 数学 |
| B | 国語 |
| C | 理科 |

5      10                  25  
(フィールド位置)

データ列

【フィールド書式】

A<TAB>数学<改段落>  
B<TAB>国語<改段落>  
C<TAB>理科<改段落>

図 32 : フィールド書式の例-3

【フィールド書式】の内容

- 段落境界の横線の属性 (line)  
線の太さ: 細線 線の種類: 実線 (上線、下線共同じ)
- フィールド設定数 (nFld) --- 3
- 左側の位置 (fld)  
フィールド 1 5  
フィールド 2 10  
フィールド 3 25
- フィールド内の左、右マージン (right, left) --- 0
- フィールド属性  
フィールド 1 中央揃え、下線あり  
フィールド 2 左揃え、下線あり  
フィールド 3 任意

例 4 : 表形式 (2)

|    | 食費     | 娯楽費    | 雑費     |
|----|--------|--------|--------|
| 1月 | 42,000 | 10,000 | 6,200  |
| 2月 | 51,000 | 6,500  | 12,000 |
| 3月 | 44,000 | 40,000 | 10,000 |

5      10      20      30      40      ←【フィールド書式 1】の指定  
(フィールド位置)

←【フィールド書式 2】の指定

←【フィールド書式 3】の指定

図 33 : フィールド書式の例-4

【フィールド書式 1】の内容:

- 段落境界の横線の属性 (line)  
線の太さ 線の種類

上線: 太線 実線  
下線: 任意 任意

- フィールド設定数 (nfld) --- 5
- 左側の位置 (fld)
  - フィールド1 5
  - フィールド2 10
  - フィールド3 20
  - フィールド4 30
  - フィールド5 40
- フィールド内の左、右マージン (right, left) --- 0
- フィールド属性

|        | 揃え   | 下線有無 | 左側の縦線 |
|--------|------|------|-------|
| フィールド1 | 中央揃え | なし   | 太実線   |
| フィールド2 | 中央揃え | なし   | 太実線   |
| フィールド3 | 中央揃え | なし   | 細点線   |
| フィールド4 | 中央揃え | なし   | 細点線   |
| フィールド5 | 任意   | なし   | 太実線   |

#### 【フィールド書式2】の内容:

- 段落境界の横線の属性 (line)
  - 線の太さ 線の種類
  - 上線: 太線 実線
  - 下線: 細線 実線
- フィールド設定数 (nfld) --- 5
- 左側の位置 (fld)
  - フィールド1 5
  - フィールド2 10
  - フィールド3 20
  - フィールド4 30
  - フィールド5 40
- フィールド内の左、右マージン (right, left) --- 0
- フィールド属性

|        | 揃え   | 下線有無 | 左側の縦線 |
|--------|------|------|-------|
| フィールド1 | 中央揃え | あり   | 太実線   |
| フィールド2 | 中央揃え | あり   | 太実線   |
| フィールド3 | 中央揃え | あり   | 細点線   |
| フィールド4 | 中央揃え | あり   | 細点線   |
| フィールド5 | 任意   | なし   | 太実線   |

#### 【フィールド書式】の内容:

- 段落境界の横線の属性 (line)
  - 線の太さ 線の種類
  - 上線: 太線 実線
  - 下線: 任意 任意
- フィールド設定数 (nfld) --- 1

データ列の内容:

#### 【フィールド書式1】

<TAB>食費<TAB>娯楽費<TAB>雑費<TAB><改段落>

### 【フィールド書式 2】

1月<TAB>42,000<TAB>10,000<TAB>6,200<TAB><改段落>

2月<TAB>51,000<TAB>6,500<TAB>12,000<TAB><改段落>

3月<TAB>44,000<TAB>40,000<TAB>10,000<TAB><改段落>

### 【フィールド書式 3】

<改段落>

文字方向指定付箋：拡張レベル

レイアウトする文字の方向を指定する。段落の区切りとなる。

LEN : 2  
SUBID: 4  
ATTR : UB txdir -- 文字列方向  
txdir: で文字列の方向、即ち、縦書きか横書きかを示す。  
0 : 横書き (左 右)  
1 : 横書き (右 左)  
2 : 縦書き

ページ途中で文字方向を変更した場合の解釈は以下ようになる。

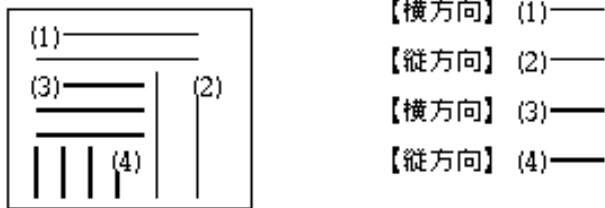


図 34 : 文字方向指定付箋

行頭移動指定付箋：拡張レベル

行頭位置を一時的に移動する。

LEN : 2  
SUBID: 5  
ATTR : UB --- -- 未使用

直後の文字の文字開始位置に行頭位置を一時的に移動する。改行、改段落、改コラムコード、もしくは、改行、改段落を伴う付箋が出現するまで有効となる。

1行内に複数個の「行頭移動指定付箋」が存在した場合は、最初のみ有効となり、2番目以降は無視される。

### 3.5.4 文字指定付箋

文字指定付箋は、文字の表現形態の指定を行なう文章付箋セグメントである。

フォント指定付箋：基本レベル

現在処理中の国語の文字セットでのフォントを、フォントクラスおよびフォント(ファミリー)名により指定する。

LEN : 4~  
SUBID: 0  
ATTR : UB - -- 未使用

DATA : UH class -- フォントクラス  
 [UB name[]] -- フォント (ファミリー) 名

class:  
 フォントクラスは文字体の特徴を表す以下に示すデータである。

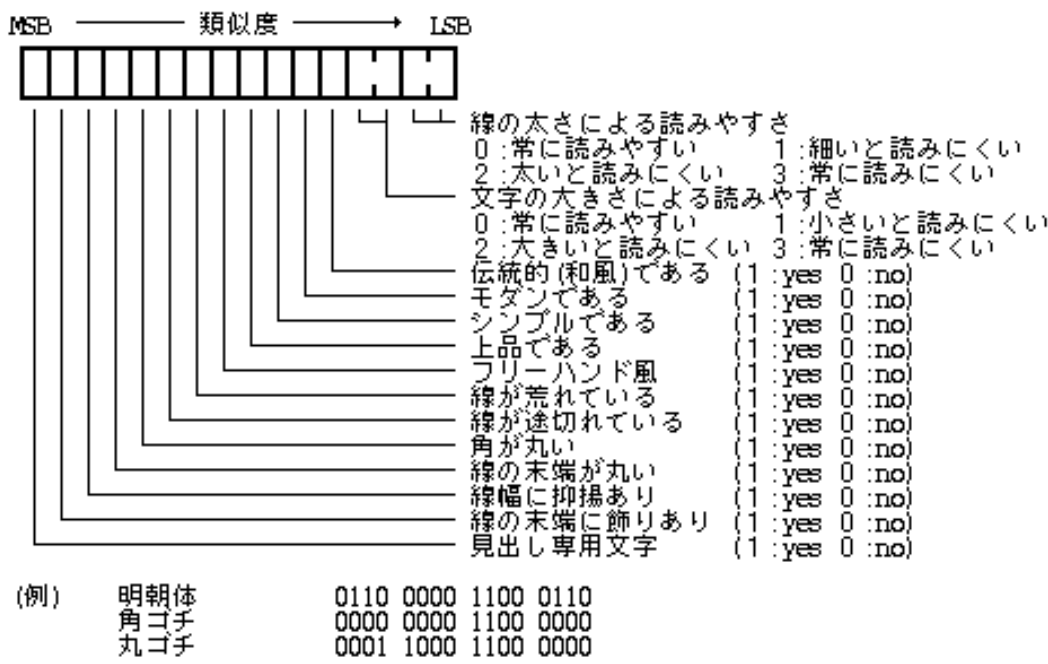


図 35: フォントクラス

name :  
 フォント (ファミリー) 名の指定 (省略可能)。  
 この指定がある場合はフォントクラスの指定は基本的に無視されるが、指定したフォント (ファミリー) 名のフォントが存在しない場合は、フォントクラスで指定したフォントに最も類似度が高い (フォントクラスの最上位ビットから順に一致しているもの) フォントが適用される。

フォント属性指定付箋 : 基本レベル

フォントのバリエーションとしてのフォント属性を指定する。

LEN : 4  
 SUBID: 1  
 ATTR : UB - -- 未使用  
 DATA : UH attr -- フォント属性

attr :  
 以下のフォント属性の指定。  
 複数のビットをセットすることにより、同時に複数の属性設定が可能となる。

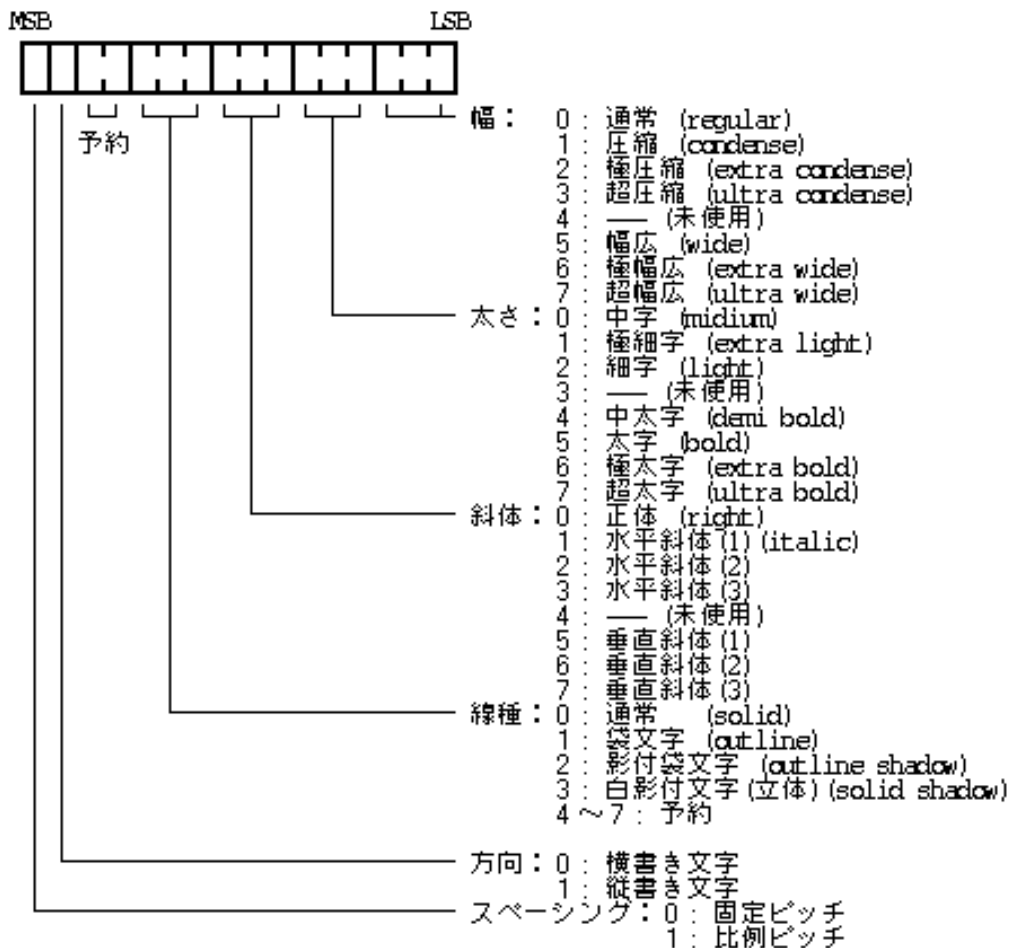


図 36: フォント属性

- 水平斜体 / 垂直斜体 (1) ~ (3) では、(1) より (3) の方が傾きが大きいことを意味する。また、超 (ultra) は 極 (extra) よりも程度が大きいことを意味する。
- 影付袋文字は、袋文字の右側に影が付いた文字である。
- 白影付文字 (立体) は右側に白ぬきの影が付いた文字である。

#### 文字サイズ指定付箋:基本レベル

文字サイズの指定を行なう。

```
LEN : 4
SUBID: 2
ATTR : UB      -          -- 未使用
DATA : CHSIZE size      -- 文字サイズ
size :
```

文字サイズの指定(CHSIZE タイプのデータ)。

この付箋は基準となる文字サイズを指定し、文字拡大/縮小指定付箋、または別の文字サイズ指定付箋が現れない限り、このサイズで表示される。文字サイズ指定付箋が存在しない間は、アプリケーション依存のデフォルトの文字サイズが基準となる。

#### 文字拡大 / 縮小指定付箋 : 基本レベル

文字サイズの拡大 / 縮小の指定を行なう。

```
LEN : 6
SUBID: 3
ATTR : UB      -          -- 未使用
DATA : RATIO  h_ratio -- 文字高さ ( サイズ ) の拡大率
      RATIO  w_ratio -- 文字幅の拡大率
```

h\_ratio :

文字高さ ( サイズ ) の拡大率を示す、RATIO タイプのデータ。

w\_ratio :

文字幅の拡大率を示す、RATIO タイプのデータ。

拡大 / 縮小の基準となる文字サイズは、直前の文字サイズ指定付箋で指定された値であり、拡大/縮小された値ではない ( アプリケーション依存のデフォルトの文字サイズが基準となる )。例えば、通常幅 ==> 1/2幅 ==> 通常幅 の表現をこの付箋で行なう場合には、文字幅を 1/2 倍にする付箋の後の文字列が 1/2 幅となり、次に文字幅を 1 倍 ( 2 倍ではない ) にする付箋の後の文字列が通常幅となる。

文字間隔指定付箋:基本レベル

文字の間隔を指定する。

LEN : 4  
SUBID: 4  
ATTR : UB attr -- 属性指定  
DATA : SCALE pitch -- 間隔指定  
attr:

DSxx xxxG

D:

方向 ( 0 : + 方向 1 : - 方向 )

S:

詰め / kerning 指定 ( 0 : 無し 1 : 有り )

G:

文字アキ指定 ( 0 : 文字送り指定 1 : 文字アキ指定 )

詰め / Kerning 指定の場合は、各文字に定義された最大詰め幅まで文字間隔は詰められ、他の指定は意味を持たない ( 無視される )。横書き ( 左 右 ) の場合は、以下のようなになる。



図 37 : 文字間隔指定

pitch:

文字間隔を指定する SCALE タイプのデータ。比率指定の場合の基準値は直前の文字の文字幅 ( 拡大 / 縮小された結果を含む ) となる。

詰め / Kerning 指定の場合は、この値は意味を持たない ( 無視される )。

- 文字間隔指定は、直後の文字以降に有効となり、直前の文字と直後の文字の間隔には適用されない。
- 文字間隔が - 方向で、文字方向に逆行するような場合 ( 例えば、左 右の場合に次の文字が前の文字の左側にくるような場合 ) の描画に関しては、保証しない。つまり、文字アキの - 方向は、前の文字の幅までは保証するが、それ以上の文字アキ、および文字送りの - 方向に関しては保証しない ( 重なりは保証するが、前に戻ってしまう場合は保証しない )。

文字回転指定付箋 : 拡張レベル

文字の回転を指定する。

LEN : 4  
SUBID: 5

ATTR : UB abs -- 絶対 / 相対指定  
DATA : UH angle -- 文字の横軸の反時計回りの回転角度  
( 360 度の剰余が有効 )

abs :

xxxx xxxA

A

0 : 絶対角度指定。

現在の文字の回転角度に無関係に絶対角度を指定する (水平軸が 0 度となる)。

1 : 相対角度指定。

現在の文字の回転角度に対しての相対的な角度を指定する (現在の文字の横軸が 0 度となる)。

文字カラー指定付箋 : 拡張レベル

文字のカラーを指定する。

LEN : 6  
SUBID: 6  
ATTR : UB - -- 未使用  
DATA : COLOR color -- 文字カラーの指定

文字基準位置移動付箋 : 拡張レベル

文字の基準位置 (ベースライン) の移動量を指定する。

LEN : 4  
SUBID: 7  
ATTR : UB attr -- 属性指定  
DATA : SCALE base -- 基準位置移動量

attr:

Dxxx xxxx

D :

方向 ( 0 : + 方向 1 : - 方向 )

base:

文字の基準位置の移動量を指定する SCALE タイプのデータ。比率指定の場合の基準値は、直前の文字の縦方向の大きさ (拡大縮小された結果を含む) となる。ベースラインの移動起点は現在のベース位置となる。

### 3.5.5 特殊文字指定付箋

特殊文字指定付箋は、1 つの特殊な文字として取り扱われる文章付箋セグメントである。

固定幅空白指定付箋 : 拡張レベル

任意の固定幅の空白を示す。これは、1 つの文字として取り扱われるため、行をまたがることはない。つまり、指定された空白幅に満たないうちに行末に至った場合、次の行の先頭から固定幅空白全体 (残りの空白ではない) が配置される。

LEN : 4  
SUBID: 0  
ATTR : UB - -- 未使用  
DATA : SCALE width -- 空白の幅

width:

空白の幅を指定する SCALE タイプのデータ。比率指定の場合の基準値は、文字サイズとなる。



## 充填文字指定付箋:拡張レベル

1行または1つのフィールドの空き領域を指定した文字列で充填する。1行または1つのフィールド中に複数の充填文字がある場合は、それぞれの充填文字の幅は均等となる。

LEN : ~  
 SUBID: 1  
 ATTR : UB - -- 未使用  
 DATA: UB str[] -- 充填文字列

str:

充填する文字列を示し、この文字列を周期的に繰り返し、充填すべき幅に達するまで1文字単位で充填する。最後に余った部分(1文字未満)は空白とする。通常は、充填文字列として1つの空白を使用することが多い。

充填される領域は行またはフィールドの左端から右端の間であり、以下に示す用途に使用される。

- (1) 【充填文字:空白】AA<改行>
- (2) 【充填文字:空白】AA【充填文字:空白】<改行>
- (3) AA【充填文字:空白】BB<改行>
- (4) AA【充填文字:空白】BB【充填文字:空白】CC<改行>
- (5) 【充填文字:空白】A【充填文字:空白】B【充填文字:空白】<改行>
- (6) 【充填文字:abc】<改行>
- (7) 第1項【充填文字:--】1<改行><TAB>1.1【充填文字:--】2<改行>

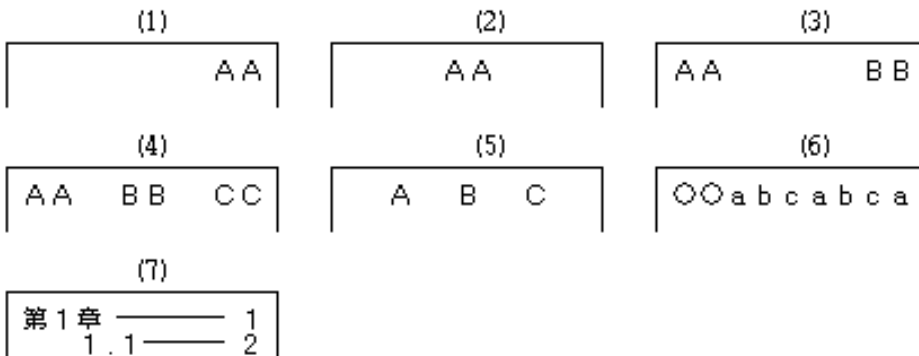


図 38 : 充填文字指定

## 文字罫線指定付箋 : 拡張レベル

罫線列を示す。この付箋は主に既存のデータの変換を目的としている。

LEN : 4 + 罫線列データ数  
 SUBID: 2  
 ATTR : UB type -- 罫線のタイプ  
 DATA : UH count -- 罫線列の繰り返し数  
 UH lines[] -- 罫線指定データ列( LEN で指定されたデータ数 )

type:

UUUU KKKK

K:

- 罫線の種類
- 0: 通常罫線(1文字分の大きさを持つ罫線)
  - 1: 文字間/行間罫線
  - 2: 罫線文字(罫線の扱いをうける文字)
  - 3~:(予約)

U: 未定義 (アプリケーション依存属性)

count: 罫線指定のデータ列で指定された罫線列全体の繰り返し回数。0の場合は1と見なされる。




| 対象罫線  | 繰り返し回数 | 罫線指定データ列            |
|---|--------|---------------------|
|  | 6      | —                   |
|  | 1      | ┌, —, —, —, —, —, ┐ |
|  | 3      | ┌, —                |

図 39: 文字罫線指定

lines: 罫線を指定するデータ列で、罫線の種類により意味が異なる。  
通常罫線の場合 (type 0):

UH vline : 上下の罫線属性 (上位バイトが上)  
UH hline : 左右の罫線属性 (上位バイトが左)  
: : (上記の2ワードのデータ列)

文字間 / 行間罫線の場合 (type 1):

UH tlline : 上及び左の罫線属性 (上位バイトが上)  
: : (上記の1ワードのデータ列)

罫線文字の場合 (type 2):

UH code : 罫線文字の TRON コード  
: : (上記の1ワードのデータ列)

vqline, hline:

(通常罫線)

現在の文字サイズの文字枠を下図のように4つの部分に分け、それぞれに部分に対応する罫線の属性を示したもの。

上 -- vline の上位バイト  
下 -- vline の下位バイト  
左 -- hline の上位バイト  
右 -- hline の下位バイト

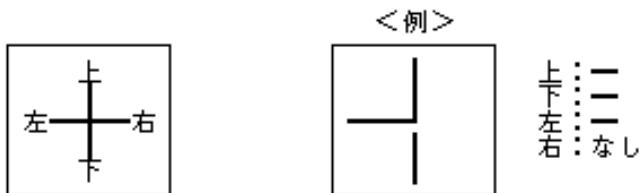


図 40: 文字罫線指定 (通常罫線)

tlline:

(文字間 / 行間罫線)

文字間 / 行間罫線を各文字の上部に接する罫線素片 (行間罫線のパターン) を上位バイトで、左部に接する罫線素片 (文字間罫線のパターン) を下位バイトで表現する。改行コードには含まれた文字間 / 行間罫線群によって、直前と直後の行間の行間罫線と直後の行の文字間罫線を表現する。

- 縦の罫線は段落をまたがらないものとする。

例1： 

|   |   |
|---|---|
| A | B |
|---|---|

 文字列A Bの上部に行間罫線あり  
Aの直前と、A B間に文字間罫線あり

上の場合データ列はつぎのようになる。

|  |  |   |  |   |     |    |    |    |
|--|--|---|--|---|-----|----|----|----|
|  |  | , |  | , | CR, | A, | B, | CR |
|--|--|---|--|---|-----|----|----|----|

例2： 

|         |     |
|---------|-----|
| 6 / 2 0 | 晴れ  |
| 6 / 2 1 | くもり |

 行間、文字間罫線で構成された表

上の場合データ列はつぎのようになる。

一行目の文字間/行間罫線表現：

|  |  |   |  |  |   |     |
|--|--|---|--|--|---|-----|
|  |  | , |  |  | , | CR, |
|--|--|---|--|--|---|-----|

一行目のデータ : 6 , / , 2 , 0 , , 晴 , れ , CR ,

二行目の文字間/行間罫線表現：

|  |  |   |  |  |   |     |
|--|--|---|--|--|---|-----|
|  |  | , |  |  | , | CR, |
|--|--|---|--|--|---|-----|

二行目のデータ : 6 , / , 2 , 1 , < , も , り , CR ,

三行目の文字間/行間罫線表現：

|  |  |   |  |  |   |     |
|--|--|---|--|--|---|-----|
|  |  | , |  |  | , | CR, |
|--|--|---|--|--|---|-----|

図 41：文字罫線指定 (行間 / 文間罫線)

code :

(文字罫線)

罫線として取り扱う文字の TRON 仕様コードを指定する。vline, hline, tlline で指定される罫線の属性は、以下のデータであり、線幅 = 0 の場合は、その部分の罫線がないことを意味する。

DIWW KKKK

D :

線の本数

0 : 一本

1 : 二本

I :

線の濃さ

0 : 100 %

1 : 50 %

W : 線の太さ

0 : なし (線は引かれない)

1 : 細線

2 : 中線

3 : 太線

K : 線の種類

0 : 実線

1 : 破線

2 : 点線

3 : 一点鎖線

4 : 二点鎖線

5 : 長破線

6 : 波線

7 : (予約)

8 ~ : 未定義 (アプリケーション依存)

### 3.5.6 文字割付け指定付箋


文字割付け指定付箋は、文字のレイアウトに関連する指定を行なうもので、開始付箋と終了付箋に囲まれた範囲の文字データに対して有効となる。

結合指定付箋：拡張レベル

文字列の途中での改行を禁止することを指定する付箋であり、結合開始と結合終了で囲まれた、文字列の途中での改行を禁止する。結合文字列が1行の幅より大きい場合の処理はインプリメントに依存する。

```
LEN : 2
SUBID: 0          -- 結合開始指定付箋
ATTR : UB        - -- 未使用
LEN : 2
SUBID: 1          -- 結合終了指定付箋
ATTR : UB        - -- 未使用
```

例：(1) `———TAD仕様———`  
(2) `———【結合開始】TAD仕様【結合終了】———`



(1) (2)

図 42：結合指定付箋

文字割付け指定付箋：拡張レベル

指定した幅の領域に文字を指定した方法により割付けることを指定する。この付箋は、変数参照指定付箋と組み合わせて、不定長の文字列を固定の枠に割付ける場合に有効となる。

```
LEN : 4
SUBID: 2          -- 文字割付け開始指定付箋
ATTR : UB        kind -- 種類
DATA : SCALE    width -- 割り付け幅
LEN : 2
SUBID: 3          -- 文字割付け終了指定付箋
ATTR : UB        - -- 未使用
```

kind:

割り付け方法

- 0: 左揃え
- 1: 右揃え
- 2: 中央揃え
- 3: 両端揃え
- 4: 均等割付け

width:

割り付け幅を指定する SCALE タイプのデータ。比率指定の場合の基準値は、文字サイズとなる。

- 割付けの対象となる文字列は、割付け開始付箋の直後の有効文字から割付け終了付箋の直前の有効文字までとなる。

【割付け開始】対象文字・・・【割付け終了】

- 割付け文字列が割付け幅より長い場合は、割付け幅は自動的に広がる。

添字指定付箋：拡張レベル

文字の前または後ろに付く、上付き / 下付きの添字を指定する。前付きの時は対象文字の直前に、後ろ付き

の時は対象文字の直後に付箋をおく。

LEN : 6  
SUBID: 4 -- 添字開始指定付箋  
ATTR : UB type -- タイプ  
DATA : SCALE pos -- 文字位置の移動量  
RATIO size -- 文字サイズの比率  
LEN : 2  
SUBID: 5 -- 添字終了指定付箋  
ATTR : UB - -- 未使用

Fxxx xxDU

type :

U :

0: 下付き  
1: 上付き

F :

0: 前付き  
1: 後付き

D

0: 添字のベースラインを上方向に移動  
1: 添字のベースラインを下方向に移動

pos :

添字のベースラインの移動量を指定する SCALE タイプのデータ。  
比率指定の場合の基準値は添字対象の文字サイズ(拡大縮小された結果を含む)となる。移動方向は、type の D で指定された方向となる。

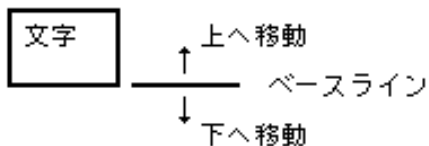


図 43: 添字指定(1)

ベースラインの移動量であるため、下付き文字でも下に移動するとは限らない。また、type の上付き / 下付きの指定と、この移動量は独立であり、下付きの指定として実際には上部に位置させてもよい。ベースラインの移動は、この添字の配置に対してのみ有効なローカルな移動であり、以降の文字基準位置には影響を与えない。

size :

添字の文字サイズを現在の文字サイズの比率で示す、RATIO タイプのデータ。添字の文字サイズの基準となる文字サイズは、添字対象の文字の縦方向の大きさ(拡大縮小された結果を含む)である。

- 添字開始付箋の直後の有効文字から、添字修飾終了付箋の直前の有効文字までが、添字となる。

【添字開始】添字・・・【添字終了】対象文字 -- 前付き  
対象文字【添字開始】添字・・・【添字終了】 -- 後付き

- 添字指定付箋を【添字終了】なしに連続した場合は、同一タイプ(上/下付き)の場合は、添字は連続し、別タイプの場合は、添字は重複して付けられる。

例1: 前付き

- (1) 【上付き】AA【下付き】BB【終了】
- (2) 【上付き】AA【上付き】BB【終了】
- (3) 【上付き】AA【終了】【下付き】BB【終了】
- (4) 【上付き】AA【終了】【上付き】BB【終了】

(2) では2つの添字のベース位置指定が異なるとする

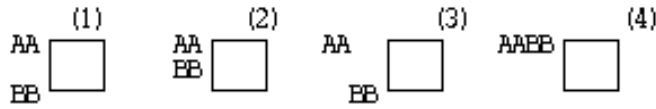


図 44 : 添字指定(2)

例2: 後ろ付き

- (1) 【上付き】AA【下付き】BB【終了】
- (2) 【上付き】AA【上付き】BB【終了】
- (3) 【上付き】AA【終了】【下付き】BB【終了】
- (4) 【上付き】AA【終了】【上付き】BB【終了】

(2) では2つの添字のベース位置指定が異なるとする

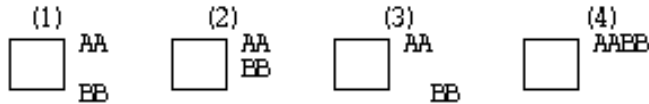


図 45 : 添字指定(3)

ルビ指定付箋 : 拡張レベル

ルビを付けるための指定である。

LEN : ~  
SUBID: 6 -- ルビ開始指定付箋  
ATTR : UB attr -- ルビ属性  
DATA : UB rubi[] -- ルビ文字列  
LEN : 2  
SUBID: 7 -- ルビ終了指定付箋  
ATTR : UB - -- 未使用

UUUU xxxP

attr :

P :

ルビ位置

0: 上ルビ (縦書きの時は右ルビ)

1: 下ルビ (縦書きの時は左ルビ)

U :

未定義 (アプリケーション依存属性)

ルビ終了付箋の場合は意味を持たず無視される。

rubi :

ルビの文字列自体であり、この文字列の長さは付箋のサイズにより決定される。奇数バイトの場合は最後に0がパッドされる。ルビ用のフォント、サイズは、ルビ対象文字列のフォント、サイズに対応した適当なものが使用される。

- ルビの付加対象となる文字列は、ルビ開始付箋の直後の有効文字からルビ終了付箋の直前の有効文字となる。

【ルビ開始(ルビ文字)】対象文字・・・【ルビ終了】

- ルビの対象文字列に対する整列の方法に関しては、特に規定しない。(インプリメントに依存する)。

禁則指定付箋 : 拡張レベル

行頭、行末の禁則処理の指定である。

```
LEN : ~
SUBID: 8 -- 行頭禁則指定
ATTR: UB kind -- 行頭禁則タイプ
[DATA: UB ch[]] -- 行頭禁則対象文字
LEN : ~
SUBID: 9 -- 行末禁則指定
ATTR: UB kind -- 行末禁則タイプ
[DATA: UB ch[]] -- 行末禁則対象文字
```

XXXL KKKK

kind:

L:

レベル

0: 1重禁則

1: 多重禁則

K:

方式

0: 禁則なし

1: 追い出し禁則

2: 追い込み禁則

3: ぶら下がり禁則

4~: 予約

15: その他(方式の規定なし)

追い出し:

行末禁則の場合、禁則文字を次の行に追い出す。

行頭禁則の場合、前の行末文字を行頭に取り込む。

追い込み:

行頭禁則の場合、禁則文字を前の行に追い込む。

行末禁則の場合、その次の文字を取り込む。

ぶら下がり:

各行の行末に1文字分の禁則文字領域を設け、行頭禁則の場合、禁則文字を前の行の禁則文字領域に追い込む。

行末禁則の場合、その次の文字を禁則文字領域に取り込む。

ch[]:

禁則対象文字の列、省略可能

禁則ありで省略された場合の禁則対象文字は処理系に依存する。指定された場合は、指定文字のみ禁則対象とする。

- 禁則指定付箋が存在しない場合は禁則処理なしとする。

### 3.5.7 文字修飾指定付箋

文字修飾指定付箋は、文字の修飾を行なうもので、開始付箋と終了付箋に囲まれた範囲の文字データに対して修飾が行なわれる。

文字修飾指定付箋: 基本レベル

文字列の修飾を指定する。

```
LEN : 2,6
SUBID: type -- 文字修飾開始付箋
ATTR: UB attr -- 属性
DATA: [COLOR color] -- 修飾カラー指定(省略可能)
```

LEN : 2  
SUBID: type + 1 -- 文字修飾終了付箋  
ATTR : UB - -- 未使用  
SUBID: 文字修飾のタイプ ( 終了付箋の場合の SUBID は対応する値 + 1 となる。 )  
0: 下線開始 1: 下線終了  
2: 上線開始 3: 上線終了  
4: 打ち消し線開始 5: 打ち消し線終了  
6: 枠囲み線開始 7: 枠囲み線終了  
8: 上(右)傍点開始 9: 上(右)傍点終了  
10: 下(下)傍点開始 11: 下(下)傍点終了  
12: 反転開始 13: 反転終了  
14: 網掛開始 15: 網掛終了  
18: 無印字開始 ( 終了付箋までのデータを印字しない )  
19: 無印字終了  
20~127 : ( 予約 )  
128~ : 未定義 ( アプリケーション依存 )

attr:

下線、上線、打ち消し線、枠囲み線 の場合 : 以下の線属性

DIWW KKKK

D:

線の本数

0: 一本

1: 二本

I:

線の濃さ

0: 100 %

1: 50 %

W:

線の太さ

0: なし(線は引かれない)

1: 細線

2: 中線

3: 太線

K:

線の種類

0: 実線

1: 破線

2: 点線

3: 一点鎖線

4: 二点鎖線

5: 長破線

6: 波線

7: ( 予約 )

8~ : 未定義 ( アプリケーション依存 )

上、下傍点の場合 : 以下の傍点属性

xxxx KKKK

K:

傍点の種類

0: 「・」



- 1: 「、」
- 2~7:(予約)
- 8~:未定義(アプリケーション依存)

反転の場合: 以下の反転属性

Axxx xxxx

A:

領域指定

- 0: 文字領域のみ反転
- 1: 全領域反転(行間も反転)
  - 文字間は常に反転する。

網掛/背景の場合: 以下のパターン属性

AIDD KKKK

A:

領域指定

- 0: 文字領域のみ網掛
- 1: 全領域網掛(行間も網掛)
  - 文字間は常に網かけされる。

D:

濃さ

- 0: なし
- 1: 薄
- 2: 中
- 3: 濃

I:

粗さ

- 0: 標準
- 1: 粗い

K:

種類

- 0: 均等
- 1: 縦縞
- 2: 横縞
- 3: 右上り斜線
- 4: 左上り斜線
- 5: 黒塗り
- 6~7:(予約)
- 8~:未定義(アプリケーション依存)

無印字の場合: 意味を持たない。

color:

飾りの色の指定であり、省略された時は文字と同じカラーとなる。反転の場合は意味を持たない(通常省略される)。

修飾終了付箋の場合は意味を持たない(通常省略される)。

- 網掛は、文字描画の上にパターンを重ねることを意味し、背景は、パターンの上に文字描画を重ねることを意味する。
- 文字修飾開始付箋の直後の有効文字から、対応するタイプの文字修飾終了付箋の直前の有効文字までが、修飾の対象となる。

【文字修飾開始】対象文字・・・【文字修飾終了】

従って、異なる種類の文字修飾(例えば、網掛と下線)は重複して指定することができる。

【網掛開始】AA【下線開始】BB【網掛終了】CC【下線終了】

```
          下線
          |-----|
AABBCC
|-----|
 網掛
```

### 3.5.8 変数参照指定付箋

変数参照指定付箋は、変数を参照する付箋であり、指定した変数の現在の値を表現する文字列に置き換わることになる。

変数参照指定付箋：拡張レベル

指定した変数の現在値を表現する文字列に置き換える。変数の参照方法として、ID 番号を指定する場合と、変数名を指定する場合がある。

```
LEN   : 4
SUBID : 0          -- 変数参照
ATTR  : UB -      -- 未使用
DATA  : H var_id  -- 変数ID
LEN   : ~
SUBID : 1          -- 変数参照
ATTR  : UB -      -- 未使用
DATA  : UB name[] -- 変数名
```

参照する変数は、特定の ID または名前ですみユーザの実行環境に登録されている必要があり、この登録は基本的に設定付箋により行なわれる。正の変数 ID は自動的に登録されるシステム変数に使用され、以下のものが定義されている。

```
0 : 自分自身の実身名
100 : 年 ( 西暦下 2 桁 )
101 : 年 ( 元号 )
110 : 月 ( 数字 )
111 : 月 ( 数字 2 桁 )
112 : 月 ( 英小文字 3 文字 )
113 : 月 ( 英大文字 3 文字 )
120 : 日 ( 数値 )
121 : 日 ( 数値 2 桁 )
200 : 現在ページ番号 ( 数字表現 )
201 : 現在ページ番号 ( 小文字ローマ数字表現 )
202 : 現在ページ番号 ( 大文字ローマ数字表現 )
250 : 全体ページ番号 ( 数字表現 )
251 : 全体ページ番号 ( 小文字ローマ数字表現 )
252 : 全体ページ番号 ( 大文字ローマ数字表現 )
```

負の変数 ID は、アプリケーションにより定義されるユーザ変数として使用される。

### 3.5.9 文章メモ指定付箋

文章メモ指定付箋は、ユーザが任意に設定したメモを保持する付箋である。この内容は単なるメモとしての文字列であるが、アプリケーションによっては、メモ文字列の内容が所定の文字列と一致した場合には、何らかの効果を持つ場合もある。

文章メモ指定付箋：基本レベル

ユーザが設定したメモを保持する。

```
LEN : ~
SUBID: 0
ATTR : UB - -- 未使用
DATA : UB memo[] -- メモ文字列
```

### 3.5.10 文章アプリケーション指定付箋

文章アプリケーション指定付箋は、アプリケーションが独自に定義して使用する付箋であり、その内容はアプリケーションにより規定される。内容を定義しているアプリケーション ID が内部に含まれており、同一データタイプ ID を持つアプリケーションのみ、その内容を理解できることになる。

文章アプリケーション指定付箋：基本レベル

アプリケーション固有のデータを保持する。

```
LEN : ~
SUBID: - -- アプリケーション定義
ATTR : UB - -- アプリケーション定義
DATA : UH appl[3] -- アプリケーションID
      UB param[] -- アプリケーション・パラメータ
appl: 付箋の内容を定義しているアプリケーション ID。
      アプリケーション ID 中のデータ ID 毎にこの付箋の内容が定義される。
```

---

[この章の目次にもどる](#)

[前頁:第3章 TAD詳細仕様書にもどる](#)

[次頁:3.6 図形描画セグメントにすすむ](#)

## 3.6 図形描画セグメント

### 3.6.1 概要

図形描画セグメントは、図形を表現するために図形データ中に埋め込まれるセグメントである。標準図形描画セグメントとして、以下のものが定義されている。

| 図形描画セグメント      | セグメントID             |
|----------------|---------------------|
| 図形要素セグメント      | TS_FPRIM (0xB0)     |
| データ定義セグメント     | TS_FDEF (0xB1)      |
| グループ定義セグメント    | TS_FGRP (0xB2)      |
| マクロ定義/参照セグメント  | TS_FMAC (0xB3)      |
| 図形修飾セグメント      | TS_FATTR (0xB4)     |
| 図形ページ割付け指定付箋   | TS_FPAGE (0xB5)     |
| (予約)           | ----- (0xB6 ~ 0xBD) |
| 図形メモ指定付箋       | TS_FMEMO (0xBE)     |
| 図形アプリケーション指定付箋 | TS_FAPPL (0xBF)     |

図形描画セグメントは以下に示す構造を持ち、サブIDにより詳細な内容が規定される。なお、サブIDは、0~127の範囲の値をTAD標準として使用し、128~255は未定義(アプリケーション依存)とする。アプリケーション依存の図形描画セグメントには、アプリケーションIDが含まれていないため、異なるアプリケーション間での互換性は保証されない。

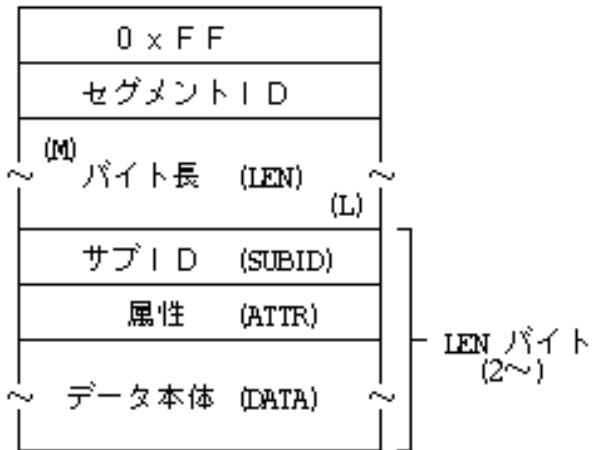


図 46 : 図形描画セグメントの構造

図形描画セグメントも文章付箋セグメントと同様に、基本レベルと拡張レベルにレベル分けされており、原則的に図形を取り扱うアプリケーションは、少なくとも基本レベルはサポートしているものとする。

## 3.6.2 図形要素セグメント

### 概要

図形要素セグメントは、図形データとして表現される実際の図形を示すセグメントであり、標準の図形要素として、以下のものがある。これらの図形要素セグメントの基本的な考え方(ハーフオープンプロパティ等)は、ディスプレイプリミティブでの規定に従う。

#### <閉じた図形要素>

|       |                              |
|-------|------------------------------|
| 長方形   | -- 長方形                       |
| 角丸長方形 | -- 角の丸い長方形                   |
| 楕円    | -- 楕円および正円                   |
| 扇形    | -- (楕)円弧と、その両端と中心を結ぶ線で囲まれた図形 |
| 弓形    | -- (楕)円弧と、その両端を結ぶ線で囲まれた図形    |
| 多角形   | -- 多角形、および角の丸い多角形            |
| 曲線    | -- 点列による曲線で囲まれた図形            |

#### <開いた図形要素>

|     |             |
|-----|-------------|
| 直線  | -- 直線       |
| 楕円弧 | -- 楕円弧および円弧 |
| 折れ線 | -- 連続した直線列  |
| 曲線  | -- 点列による曲線  |

#### <点図形要素>

|       |                |
|-------|----------------|
| マーカー列 | -- 任意位置のマーカーの列 |
|-------|----------------|

#### <任意図形要素>

|      |                 |
|------|-----------------|
| 任意図形 | -- 外周点の列で指定した図形 |
|------|-----------------|

図形要素セグメントには、以下の共通的なパラメータが存在する。これらのパラメータは、図形要素セグメント毎に指定される。

#### <描画モード>

描画モードは、描画の対象とするピクセルに、書き込むべきピクセル値(src)と、既に現在のピクセル値(dest)と、描画後のピクセル値(result)との関係を規定するものであり、以下に示すものが用意されている。

|    |       |   |                        |          |
|----|-------|---|------------------------|----------|
| 0  | STORE | : | (src)                  | (result) |
| 1  | XOR   | : | (src) XOR (dest)       | (result) |
| 2  | OR    | : | (src) OR (dest)        | (result) |
| 3  | AND   | : | (src) AND (dest)       | (result) |
| 4  | CPYN  | : | (NOT (src))            | (result) |
| 5  | XORN  | : | (NOT (src)) XOR (dest) | (result) |
| 6  | ORN   | : | (NOT (src)) OR (dest)  | (result) |
| 7  | ANDN  | : | (NOT (src)) AND (dest) | (result) |
| 8~ | 予約    |   |                        |          |

#### <線属性>

開いた図形、および閉じた図形の外周線に使用される線の属性を示すUHの値。任意図形には適用されない。

TTTT TTTT WWWW WWWW

W: 線幅 (下位8ビット符号無し) (0 ~ 255)

線の幅(太さ)を座標系単位の値で指定したもの。水平と垂直の座標系単位が異なる場合は、平均した単位を使用するものとする。0の場合は、線は描画されない。

T: 線種ID (上位8ビット符号無し) (0 ~ 255)

「線種定義セグメント」により定義された線種ID。

<線パターン>

開いた図形、および閉じた図形の外周線に使用される描画パターンを示す UH の値。  
線属性の線幅が0の場合、および任意図形には適用されない。

0 : 透明

0以外: 「[パターン定義セグメント](#)」により定義されたID

<塗り潰しパターン>

閉じた図形、および任意図形の内部の塗り潰しに使用される描画パターンを示す UH の値。  
開いた図形には適用されない。

0 : 透明

0以外: 「[パターン定義セグメント](#)」により定義されたID

<回転角>

長方形、楕円、楕円弧等の図形要素は、その外接長方形により図形が定義される。この外接長方形は、水平軸(横辺)が水平の長方形を、その左上の点を中心として、反時計周りに回転した図形として表現される。

回転角は、この回転角度を示す。UH の値で、360の剰余の値が有効となる。  
多角形、直線、折れ線、曲線、および任意図形には適用されない。

長方形セグメント:基本レベル

長方形を示す。

```
LEN : 18
SUBID: 0
ATTR : UB      mode      -- 描画モード
DATA : UH      l_attr    -- 線属性
      UH      l_pat      -- 線パターンID
      UH      f_pat      -- 塗り潰しパターンID
      UH      angle      -- 回転角
      RECT    frame      -- 外接長方形
```

角丸長方形セグメント:基本レベル

角丸長方形を示す。

```
LEN : 22
SUBID: 1
ATTR : UB      mode      -- 描画モード
DATA : UH      l_attr    -- 線属性
      UH      l_pat      -- 線パターンID
```

|      |       |               |
|------|-------|---------------|
| UH   | f_pat | -- 塗り潰しパターンID |
| UH   | angle | -- 回転角度       |
| UH   | rh    | -- 丸めの水平直径    |
| UH   | rv    | -- 丸めの垂直直径    |
| RECT | frame | -- 外接長方形      |

rh, rv:

長方形の四隅の丸めのために使用される楕円弧の水平軸の直径、および垂直軸の直径を示す。

### 楕円セグメント:基本レベル

楕円、または正円を示す。

|        |   |      |                     |
|--------|---|------|---------------------|
| LEN    | : | 18   |                     |
| SUBID: |   | 2    |                     |
| ATTR   | : | UB   | mode -- 描画モード       |
| DATA   | : | UH   | l_atr -- 線属性        |
|        |   | UH   | l_pat -- 線パターンID    |
|        |   | UH   | f_pat -- 塗り潰しパターンID |
|        |   | UH   | angle -- 回転角度       |
|        |   | RECT | frame -- 外接長方形      |

### 扇形セグメント:基本レベル

扇形を示す。

扇形は(楕)円弧と、その両端と中心を結ぶ線で囲まれた図形である。

|        |   |      |                     |
|--------|---|------|---------------------|
| LEN    | : | 26   |                     |
| SUBID: |   | 3    |                     |
| ATTR   | : | UB   | mode -- 描画モード       |
| DATA   | : | UH   | l_atr -- 線属性        |
|        |   | UH   | l_pat -- 線パターンID    |
|        |   | UH   | f_pat -- 塗り潰しパターンID |
|        |   | UH   | angle -- 回転角度       |
|        |   | RECT | frame -- 外接長方形      |
|        |   | PNT  | start -- 開始点        |
|        |   | PNT  | end -- 終了点          |

start, end:

frame を外接長方形とする(楕)円において、中心と start を結ぶ直線と(楕)円の交点から時計周りに、中心と end を結ぶ直線と(楕)円の交点までの範囲が対象とする(楕)円弧となる。(start, end は(楕)円上に存在する必要はない)。

### 弓形セグメント:基本レベル

弓形を示す。

弓形は(楕)円弧と、その両端結ぶ線で囲まれた図形である。

|        |   |    |               |
|--------|---|----|---------------|
| LEN    | : | 26 |               |
| SUBID: |   | 4  |               |
| ATTR   | : | UB | mode -- 描画モード |
| DATA   | : | UH | l_atr -- 線属性  |

|      |       |               |
|------|-------|---------------|
| UH   | l_pat | -- 線パターンID    |
| UH   | f_pat | -- 塗り潰しパターンID |
| UH   | angle | -- 回転角度       |
| RECT | frame | -- 外接長方形      |
| PNT  | start | -- 開始点        |
| PNT  | end   | -- 終了点        |

start, end:

frame を外接長方形とする(楕)円において、中心と start を結ぶ直線と(楕)円の交点から時計周りに、中心と end を結ぶ直線と(楕)円の交点までの範囲が対象とする(楕)円弧となる。(start, end は(楕)円上に存在する必要はない)。

## 多角形セグメント:基本レベル

多角形を示す。

|       |   |     |                          |
|-------|---|-----|--------------------------|
| LEN   | : | ~   |                          |
| SUBID | : | 5   |                          |
| ATTR  | : | UB  | mode -- 描画モード            |
| DATA  | : | UH  | l_atr -- 線属性             |
|       |   | UH  | l_pat -- 線パターンID         |
|       |   | UH  | f_pat -- 塗り潰しパターンID      |
|       |   | UH  | round -- 角の丸めの円の直径:拡張レベル |
|       |   | UH  | np -- 点数(>2)             |
|       |   | PNT | pt[np] -- 点の配列(np 個の要素)  |

round:

多角形の各頂点を円弧により丸める場合の円の直径を示す。0の場合 は丸めない。

pt:

多角形の頂点の配列であり、pt[0] pt[1] pt[2]... pt[np-1] pt[0] の順番に頂点を結んだ多角形を意味する。

## 直線セグメント:基本レベル

直線を示す。

|       |   |     |                  |
|-------|---|-----|------------------|
| LEN   | : | 14  |                  |
| SUBID | : | 6   |                  |
| ATTR  | : | UB  | mode -- 描画モード    |
| DATA  | : | UH  | l_atr -- 線属性     |
|       |   | UH  | l_pat -- 線パターンID |
|       |   | PNT | start -- 開始点     |
|       |   | PNT | end -- 終了点       |

## 楕円弧セグメント:基本レベル

楕円弧、または円弧を示す。

|       |   |    |               |
|-------|---|----|---------------|
| LEN   | : | 24 |               |
| SUBID | : | 7  |               |
| ATTR  | : | UB | mode -- 描画モード |
| DATA  | : | UH | l_atr -- 線属性  |



|      |       |            |
|------|-------|------------|
| UH   | l_pat | -- 線パターンID |
| UH   | angle | -- 回転角度    |
| RECT | frame | -- 外接長方形   |
| PNT  | start | -- 開始点     |
| PNT  | end   | -- 終了点     |

start, end:

frame を外接長方形とする(楕)円において、中心と start を結ぶ直線と(楕)円の交点から時計周りに、中心と end を結ぶ直線と(楕)円の交点までの範囲が対象とする(楕)円弧となる。(start, end は(楕)円上に存在する必要はない)。

折れ線セグメント:基本レベル

折れ線(連続した直線列)を示す。

|       |   |     |                         |
|-------|---|-----|-------------------------|
| LEN   | : | ~   |                         |
| SUBID | : | 8   |                         |
| ATTR  | : | UB  | mode -- 描画モード           |
| DATA  | : | UH  | l_atr -- 線属性            |
|       |   | UH  | l_pat -- 線パターンID        |
|       |   | UH  | round -- 角の丸め:拡張レベル     |
|       |   | UH  | np -- 点数 (> 1)          |
|       |   | PNT | pt[np] -- 点の配列(np 個の要素) |

round:

直線列の各折れ点を円弧により丸める場合の円の直径を示す。0の場合は丸めない。

pt:

直線列の頂点の配列であり、pt[0] pt[1] pt[2]... pt[np-1] の順番に頂点を結んだ直線列を意味する。

曲線セグメント:拡張レベル

点列による曲線を示す。

|       |   |     |                         |
|-------|---|-----|-------------------------|
| LEN   | : | ~   |                         |
| SUBID | : | 9   |                         |
| ATTR  | : | UB  | mode -- 描画モード           |
| DATA  | : | UH  | l_atr -- 線属性            |
|       |   | UH  | l_pat -- 線パターンID        |
|       |   | UH  | f_pat -- 塗り潰しパターンID     |
|       |   | H   | type -- 曲線のタイプ          |
|       |   | UH  | np -- 点数 (> 0)          |
|       |   | PNT | pt[np] -- 点の配列(np 個の要素) |

type:

曲線のタイプの指定。

0: 折れ線

1: 3次B - スプライン曲線

2~: 予約

< 0: 未定義(インプリメント依存)

f\_pat:

閉じた曲線の場合に塗り潰すパターンを示す。開いた曲線の場合は無視される。

pt:

曲線のパラメータとしての点の配列であり、pt[0] pt[1] pt[2]... pt[np-1] の順番となる。  
pt[0] pt[np-1] の場合は開曲線であり、pt[0]=pt[np-1] の場合は、pt[0] ~ pt[np-2] の点を結ぶ閉曲線となる。  
曲線のタイプによっては、指定点を通るとは限らない。  
点の数が不足して曲線の描画できない場合は、点または直線が描画される。

マーカー列セグメント:拡張レベル

マーカー列を示す。

LEN : ~  
SUBID: 10  
ATTR : UB mode -- 描画モード  
DATA : UH marker -- マーカーID  
UH np -- 点数 (> 0)  
PNT pt[np] -- 点の配列(np 個の要素)  
pt[] で指定した np 個の位置に marker で指定したマーカーを描画する。  
marker: 「マーカー定義セグメント」により定義されたマーカーID。

任意図形セグメント:基本レベル

外周点の列で指定した任意図形を示す。任意図形は、各垂直座標毎に水平座標の複数の点を指定することにより定義され、2つの隣接した水平座標値で囲まれた範囲が、任意図形の領域となる。

LEN : ~  
SUBID: 11  
ATTR : UB mode -- 描画モード  
DATA : UH f\_pat -- 塗り潰しパターンID  
UH sy -- 垂直座標の開始値(最小値)  
UH nr -- 垂直座標数  
H bx -- 水平座標値のバイアス  
< 垂直座標値 sr ~ sr+nr-1 にそれぞれ対応した、以下に示す  
水平座標値列の要素が nr 個連続する >  
UH nh -- 水平座標の数  
UH h[nh] -- 水平座標の値 (nh 個の要素)  
:  
:

sy:

垂直座標の開始値(最小値)

nr:

垂直座標数 (垂直座標の終了値(最大値) - 開始値(最小値) + 1)

bx:

水平座標値のバイアス。

nh:

水平座標の数。0は存在しない事を意味し、続く h[] は実際には存在しない。

h[]:

水平座標の値で、nh 個の要素からなる配列。実際の水平座標値は、この値に bx を加えたものとなる。

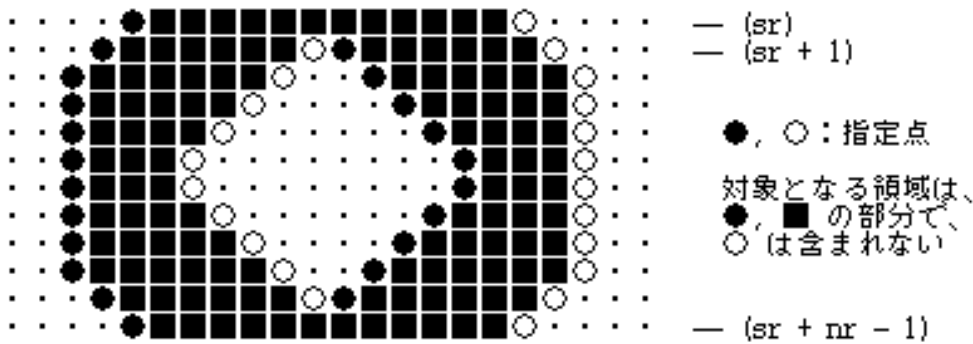


図 47 : 任意図形セグメントの例

### 3.6.3 データ定義セグメント

#### 概要

データ定義セグメントでは、図形要素セグメントから参照される以下のデータの定義を行なう。

- カラーマップ
- マスクデータ
- パターン
- 線種
- マーカー

定義したデータは、定義以降に有効となり、定義時に指定したIDにより参照されることになる。同一のタイプのデータの同一のIDが既に定義されていた場合は、新しい定義が有効となる。

データのタイプによっては、デフォルトとして定義済みのいくつかのIDが存在するが、そのIDを別の定義とすることも可能である。

定義したIDは、その図形データの中でのみ有効であり、埋め込み図形データを含む場合は、埋め込み図形データ内にも有効となる。即ち、図形データがネスティングしている場合は、以下のようになる。

現在(一番内側)の図形データ内で定義されているIDが有効であるが、現在の埋め込み図形データ内で定義されていない場合は、一番近い外側の図形データ内で定義されているIDが有効となる。

#### カラーマップ定義セグメント:基本レベル

図形データ内で使用するカラーマップを定義する。この定義以降に、インデックスによるカラー表現が使用された場合に、このカラーマップが有効となる。

```
LEN : ~
SUBID: 0
ATTR : UB      -      -- 未使用
DATA : UH      nent   -- エントリ数
      COLOR   col[nent] -- カラーマップ (nent 個の要素)
```

nent:

カラーマップのエントリ数を示す。

col[]:

各エントリに対応する絶対カラー表現を示す。

## マスクデータ定義セグメント:基本レベル

パターン定義や、マーカー定義に使用される、マスクデータを定義する。

LEN : ~  
SUBID: 1  
ATTR : UB type -- マスク定義タイプ  
DATA : UH id -- 定義するマスクID  
< type = 0の場合 >  
UH hsize -- 横のサイズ  
UH vsize -- 縦のサイズ  
UB mask[] -- マスクビットマップ

type:

マスクデータタイプ  
0: ビットマップ形式  
1~ 予約

id:

定義するマスクデータのID(0~)  
以下のマスクIDは、デフォルトで定義されている。

|    |      |   |          |
|----|------|---|----------|
| 1  | 0    | % | メッシュマスク  |
| 2  | 12.5 | % | メッシュマスク  |
| 3  | 25   | % | メッシュマスク  |
| 4  | 50   | % | メッシュマスク  |
| 5  | 75   | % | メッシュマスク  |
| 6  | 87.5 | % | メッシュマスク  |
| 7  | 100  | % | メッシュマスク  |
| 8  |      |   | 縦線       |
| 9  |      |   | 横線       |
| 10 |      |   | 右上がり斜線   |
| 11 |      |   | 右下がり斜線   |
| 12 |      |   | 縦横クロスハッチ |
| 13 |      |   | 斜線クロスハッチ |

hsize:

vsize:

マスクデータの横、縦の大きさを示す値。

mask:

マスクを示す hsize × vsize の大きさのビットマップデータ。ビットマップデータは、以下に示すバイト配列であり、"1"のビットがマスクとして有効となる。横のバイト数は常に偶数である。

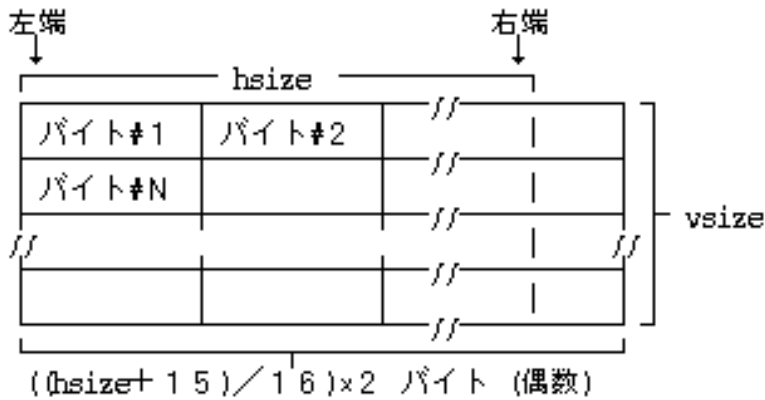


図 48: マスクデータの構成

## パターン定義セグメント:基本レベル

線パターン、または塗り潰しパターンを定義する。

```

LEN : ~
SUBID: 2
ATTR : UB type -- パターン定義タイプ
DATA : UH id -- 定義するパターンID
      < type = 0の場合 >
      UH hsize -- 横のサイズ
      UH vsize -- 縦のサイズ
      UH ncol -- 前景カラー数
      COLOR fgcol[ncol] -- 前景色 (ncol 個の要素)
      COLOR bgcol -- 背景色
      UH mask[ncol] -- マスクID(ncol 個の要素)
  
```

type:

パターン定義タイプ  
 0: 色別ビットマップ形式  
 1~ 予約

id:

定義するパターンのID(1~)  
 デフォルトで定義されているパターンデータは特に存在しないが、パターンID = 0は、完全に透明なパターンを意味するため、定義できない。

hsize:

vsize:

パターンの横、縦の大きさを示す座標系単位の値。

ncol:

前景色の数。(> 0)

fgcol[]:

bgcol:

パターンデータを構成する前景色、背景色を示す。fgcol, bgcol < 0の場合は、それぞれ透明を示す。

mask:

マスクを示すマスクIDである。0はマスクが存在しないことを意味する。bgcol で塗り潰さ

れた、hsize × vsize の領域に対して、mask[i] で指定したマスクデータの "1" のビットに対応する部分を、fgcol[i] で指定した色で塗るという操作を、i = 0 ~ ncol-1 まで繰り返して得られたものが実際のパターンとなる。fgcol[i] < 0 の場合は特別に、対応する mask[i] の "0" のビットを透明にすることを意味する。  
 通常は、fgcol[i] < 0 にする必要はない。  
 bgcol < 0 の場合は、fgcol[i]、mask[i]、i = 0 ~ ncol-1 で塗られなかった部分が透明となる。

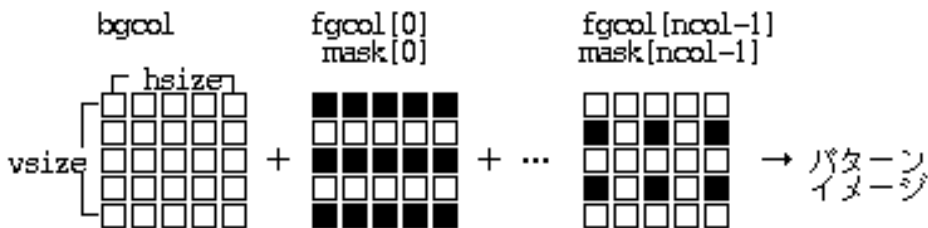


図 49: 実際のパターンの生成

マスクデータのサイズがパターンのサイズよりも大きい場合は、パターンのサイズの部分のみが有効となる。逆に小さい場合は、マスクデータをパターンのサイズを覆うように敷きつめたものとして取り扱う。

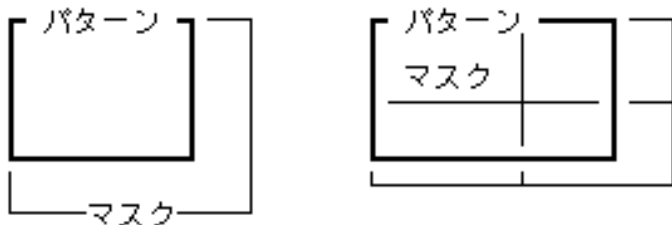


図 50: パターンサイズとマスクサイズ

線種定義セグメント: 拡張レベル

```

LEN : ~
SUBID: 3
ATTR : UB      type          -- 線種定義タイプ
DATA : UH id    -- 定義する線種ID
      < type = 0 の場合 >
      UH nb     -- 線種定義バイト数
      UB mask[nb] -- 線種定義バイト列
    
```

type:  
 線種定義タイプ  
 0: ビットパターン形式  
 1~ 予約

id:  
 定義する線種ID(0 ~ 255)  
 以下のIDはデフォルトとして定義されている。

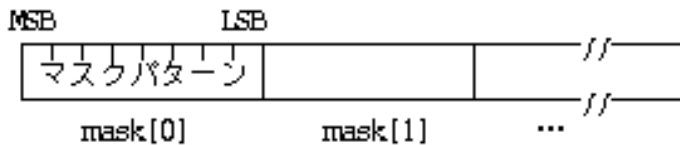
- 0 実線
- 1 破線
- 2 点線
- 3 一点鎖線

- 4 二点鎖線
- 5 長破線

nb:

mask:

線種を定義する nb 個のバイト列である。線幅 = 1 の場合の線種を、(nb × 8) 周期のビット列パターンとして指定する。即ち、mask[] の "1" のビットが描画され、"0" のビットは描画されない。



※ 例えば、nb = 1 で、その内容が 1 1 1 0 0 0 1 0 の場合は、以下に示すパターンの線種となる。



図 51 : 線種定義データ

### マーカー定義セグメント: 拡張レベル

```

LEN : ~
SUBID: 4
ATTR : UB      type      -- マーカー定義タイプ
DATA : UH      id        -- 定義するマーカーID
      < type = 0 の場合 >
      UH      size      -- マーカーサイズ
      COLOR   fgcol     -- マーカー色
      [ UH    mask      -- マスクデータID ]

```

type:

マーカー定義タイプ

0: 色、ビットマップ指定形式

1~ 予約

id:

定義するマーカーのID(0~)

以下のマーカーIDの場合は、マスクパターンがデフォルトとして定義されているため、マスクデータIDは無効となり、省略することができる。

- |   |     |   |
|---|-----|---|
| 0 | 点   | ・ |
| 1 | プラス | + |
| 2 | 星印  | * |
| 3 | 丸   | ○ |
| 4 | バツ  | × |

size:

マーカーの縦、横の大きさを示す座標系単位の値。マーカーは必ず正方形であり、その中心点がマーカーの描画位置にその中心を合わせて描画される。

fgcol:

マーカーの色を示す。マーカーの背景は常に透明となる。

mask:

マーカーのパターンを示すマスクデータのID。 マーカーID = 0 ~ 4の場合は省略できる。

### 3.6.4 グループ定義セグメント

グループ定義セグメント:拡張レベル

グループ定義開始セグメントと終了セグメントで囲んだセグメント群が1つのグループとしてまとまっていることを示す。グループは任意のネスティングが可能である。

このグループ定義は、座標系、データ定義の有効範囲等には一切影響を与えないため、主にアプリケーションによるデータの意味付けに利用される。座標系、データ定義が異なる場合は、埋め込み図形データとして定義する必要がある。

```
LEN : 4
SUBID: 0          -- グループ開始
ATTR : UB        -  -- 未使用
DATA : UH id     -- グループID
LEN : 2
SUBID: 1          -- グループ終了
ATTR : UB        -  -- 未使用
```

id:

グループに任意のIDを付けることができる。0はID無しを意味する。  
IDを付けた場合は、グループのネスティングの範囲内では一意的でなくてはならない。

### 3.6.5 マクロ定義 / 参照セグメント

マクロ定義セグメント:拡張レベル

マクロ定義開始セグメントと終了セグメントで囲んだセグメント群を1つのマクロとして定義する。定義したマクロは後にマクロ参照付箋により参照することができる。

マクロは任意のネスティングが可能である。

定義したマクロIDのスコープは、データ定義セグメントの場合と同様である。

```
LEN : 4
SUBID: 0          -- マクロ定義開始
ATTR : UB        -  -- 未使用
DATA : UH id     -- 定義するマクロID
LEN : 2
SUBID: 1          -- マクロ定義終了
ATTR : UB        -  -- 未使用
```

id: 定義するマクロのID。(0 ~ )

マクロ参照セグメント:拡張レベル

マクロを参照する。パラメータを持たないため、マクロの参照は単なる置換を意味する。

```
LEN : 4
SUBID: 2
ATTR : UB        -  -- 未使用
DATA : UH id     -- 参照するマクロID
```



### 3.6.6 図形修飾セグメント

図形修飾セグメントは、直後に続く有効セグメントを修飾する機能を持つ。有効セグメントとは、実際に図形描画を行なうセグメントを意味し、その有効範囲は以下に示す通りとなる。

|            |              |
|------------|--------------|
| 通常の図形セグメント | -- そのセグメントのみ |
| グループセグメント  | -- グループ内全体   |
| マクロ参照セグメント | -- マクロ内全体    |
| 埋め込み図形     | -- 埋め込み図形全体  |
| 埋め込み文章     | -- 埋め込み文章全体  |
| 画像セグメント    | -- そのセグメントのみ |

#### 図形要素修飾セグメント:拡張レベル

LEN : ~  
SUBID: 0  
ATTR : UB           type                   -- 修飾タイプ  
DATA : < type = 0の場合 >  
      UH   arrow                   -- 矢印属性

type:

修飾タイプ

0: 直線の矢印属性

1~ 予約

arrow: 矢印属性:

連続した直後のセグメントが開いた図形の場合にのみ適用される。

xxxx xxxx xxxx xxES

S: 開始点に矢印

E: 終了点に矢印

#### 座標変換セグメント:拡張レベル

直後に続く有効セグメントの一時的な座標変換(傾斜、回転、移動)を行なう。

LEN : 6, 8, 10  
SUBID: 1  
ATTR : UB           -                   -- 未使用  
DATA : H           dh                   -- 水平座標移動量  
      H           dv                   -- 垂直座標移動量  
      [ UH       hangle               -- 水平軸回転角度 ]  
      [ H        vangle               -- 垂直軸回転角度 ]

dh,dv:

それぞれ、水平、垂直の座標値の移動量を示す。座標系はディスプレイプリミティブの座標系、即ち、水平軸は右方向の増大、垂直軸は下方向に増大する座標系である。

hangle:

水平座標軸の水平を0度とした反時計周りの回転角度を示す。回転の中心は、座標原点(0,0)である。省略可能。

vangle:

水平軸に直交する垂直座標系軸を0度とした反時計周りの角度 (-90 ~ +90) を示す。回転の中心は、座標原点(0,0)である。この値が0でない場合は、直交座標系でないことを意味する。省略可能。

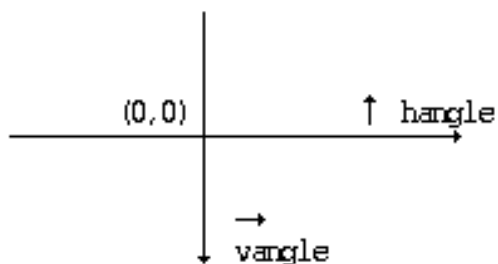


図 52 : 回転角度

この付箋の直後に続く有効セグメントの座標値 (x, y) を以下の一連の座標変換 処理によって変換した結果 (x''', y''') の図形を意味する。

1. 垂直軸回転 (水平方向の傾斜)

$$\begin{aligned} x' &= x + y \cdot \tan(\text{vangle}) \\ y' &= y \end{aligned}$$

2. 水平軸回転 (一般的な意味での回転)

$$\begin{aligned} x'' &= x' \cdot \cos(\text{hangle}) + y' \cdot \sin(\text{hangle}) \\ y'' &= -x' \cdot \sin(\text{hangle}) + y' \cdot \cos(\text{hangle}) \end{aligned}$$

3. 移動

$$\begin{aligned} x''' &= x'' + dh \\ y''' &= y'' + dv \end{aligned}$$

直後の有効セグメントが、埋め込み図形、埋め込み文章、および画像セグメントの場合は、view 座標系に対する変換を意味する。即ち、draw から view への変換がなされたのち、上記の座標変換が行なわれる。

### 3.6.7 図形ページ割り付け指定付箋

図形ページ割り付け指定付箋は、図形データを指定した用紙上にページ単位でレイアウトを行なうための情報を保持する付箋である。図形ページ割り付け指定付箋は、文章ページ割り付け指定付箋と全く同一の内容を持ち、以下のように対応づけられる。

文章ページ割り付け指定付箋 図形ページ割り付け指定付箋

|                   |                   |
|-------------------|-------------------|
| (TS_TPAGE : 0xA0) | (TS_FPAGE : 0xB5) |
| 0:用紙指定付箋          | 0:用紙指定付箋          |
| 1:マージン指定付箋        | 1:マージン指定付箋        |
| 2:コラム指定付箋         | 2: - (未定義)        |
| 3:用紙オーバーレイ定義付箋    | 3:用紙オーバーレイ定義付箋    |
| 4:用紙オーバーレイ指定付箋    | 4:用紙オーバーレイ指定付箋    |
| 5:枠あけ指定付箋         | 5: - (未定義)        |
| 6:ページ番号指定付箋       | 6:ページ番号指定付箋       |
| 7:条件改ページ指定付箋      | 7: - (未定義)        |
| 8:充填行指定付箋         | 8: - (未定義)        |

図形ページ割り付け指定付箋が図形要素セグメントの後に存在している場合、および、複数個存在した場合の処理は処理系に依存する。

### 3.6.8 図形メモ指定付箋

図形メモ指定付箋は、ユーザが任意に設定したメモを保持する付箋である。この内容は単なるメモとしての文字列であるが、アプリケーションによっては、メモ文字列の内容が所定の文字列と一致した場合には、何らかの効果を持つ場合もある。図形メモ指定付箋は表示に関する情報を保持していないため、表示方法はアプリケーションに依存する。

図形メモ指定付箋:基本レベル

ユーザが設定したメモを保持する。

```
LEN : ~
SUBID: 0
ATTR : UB - -- 未使用
DATA : UB memo[] -- メモ文字列
```

### 3.6.9 図形アプリケーション指定付箋

図形アプリケーション指定付箋は、アプリケーションが独自に定義して使用する付箋であり、その内容はアプリケーションにより規定される。内容を定義しているアプリケーションIDが内部に含まれており、同一データタイプIDを持つアプリケーションのみ、その内容を理解できることになる。図形アプリケーション指定付箋は表示に関する情報を保持していないため、表示方法はアプリケーションに依存する。

図形アプリケーション指定付箋:基本レベル

アプリケーション固有のデータを保持する。

```
LEN : ~
SUBID: - -- アプリケーション定義
ATTR : UB - -- アプリケーション定義
DATA : UH appl[3] -- アプリケーションID
      UB param[] -- アプリケーション・パラメータ
```

appl:

付箋の内容を定義しているアプリケーションID。アプリケーションID中のデータID毎にこの付箋の内容が定義される。

## 付録 A TADセグメント一覧

セグメントID:

|           |            |        |
|-----------|------------|--------|
| 管理情報セグメント | TS_INFO    | (0xE0) |
| 文章開始セグメント | TS_TEXT    | (0xE1) |
| 文章終了セグメント | TS_TEXTEND | (0xE2) |
| 図形開始セグメント | TS_FIG     | (0xE3) |
| 図形終了セグメント | TS_FIGEND  | (0xE4) |

|                |           |                     |
|----------------|-----------|---------------------|
| 画像セグメント        | TS_IMAGE  | (0xE5)              |
| 仮身セグメント        | TS_VOBJ   | (0xE6)              |
| 指定付箋セグメント      | TS_DFUSEN | (0xE7)              |
| 機能付箋セグメント      | TS_FFUSEN | (0xE8)              |
| 設定付箋セグメント      | TS_SFUSEN | (0xE9)              |
| 文章ページ割付け指定付箋   | TS_TPAGE  | (0xA0)              |
| 行書式指定付箋        | TS_TRULER | (0xA1)              |
| 文字指定付箋         | TS_TFONT  | (0xA2)              |
| 特殊文字指定付箋       | TS_TUB    | (0xA3)              |
| 文字割付け指定付箋      | TS_TATTR  | (0xA4)              |
| 文字修飾指定付箋       | TS_TSTYLE | (0xA5)              |
|                | (予約)      | ----- (0xA6 ~ 0xAC) |
| 変数参照指定付箋       | TS_TVAR   | (0xAD)              |
| 文章メモ指定付箋       | TS_TMEMO  | (0xAE)              |
| 文章アプリケーション指定付箋 | TS_TAPPL  | (0xAF)              |
| 図形要素セグメント      | TS_FPRIM  | (0xB0)              |
| データ定義セグメント     | TS_FDEF   | (0xB1)              |
| グループ定義セグメント    | TS_FGRP   | (0xB2)              |
| マクロ定義/参照セグメント  | TS_FMAC   | (0xB3)              |
| 図形修飾セグメント      | TS_FATTR  | (0xB4)              |
| 図形ページ割付け指定付箋   | TS_FPAGE  | (0xB5)              |
|                | (予約)      | ----- (0xB6 ~ 0xBD) |
| 図形メモ指定付箋       | TS_FMEMO  | (0xBE)              |
| 図形アプリケーション指定付箋 | TS_FAPPL  | (0xBF)              |

#### 管理情報セグメント:

```

ID : TS_INFO          -- 管理情報セグメント
LEN : ~
DATA: UH subid        -- 項目ID
      UH sublen       -- 項目のバイト数
      UH data[]       -- 項目データ本体 (sublen バイト)
      :               :

```

<上記の繰り返し>

#### 文章開始セグメント:

```

ID : TS_TEXT
LEN : 24
DATA: RECT view      -- 表示領域
      RECT draw      -- 描画領域
      UNITS h_unit    -- 水平ユニット
      UNITS v_unit    -- 垂直ユニット
      UH lang         -- デフォルト言語
      UH bgpat        -- 背景パターンID

```

文章終了セグメント:

ID : TS\_TEXTEND  
LEN : 0  
DATA: なし

図形開始セグメント:

ID : TS\_FIG  
LEN : 24  
DATA: RECT view -- 表示領域  
RECT draw -- 描画領域  
UNITS h\_unit -- 水平ユニット  
UNITS v\_unit -- 垂直ユニット  
W ratio -- 倍率

図形終了セグメント:

ID : TS\_FIGEND  
LEN : 0  
DATA: なし

画像データセグメント:

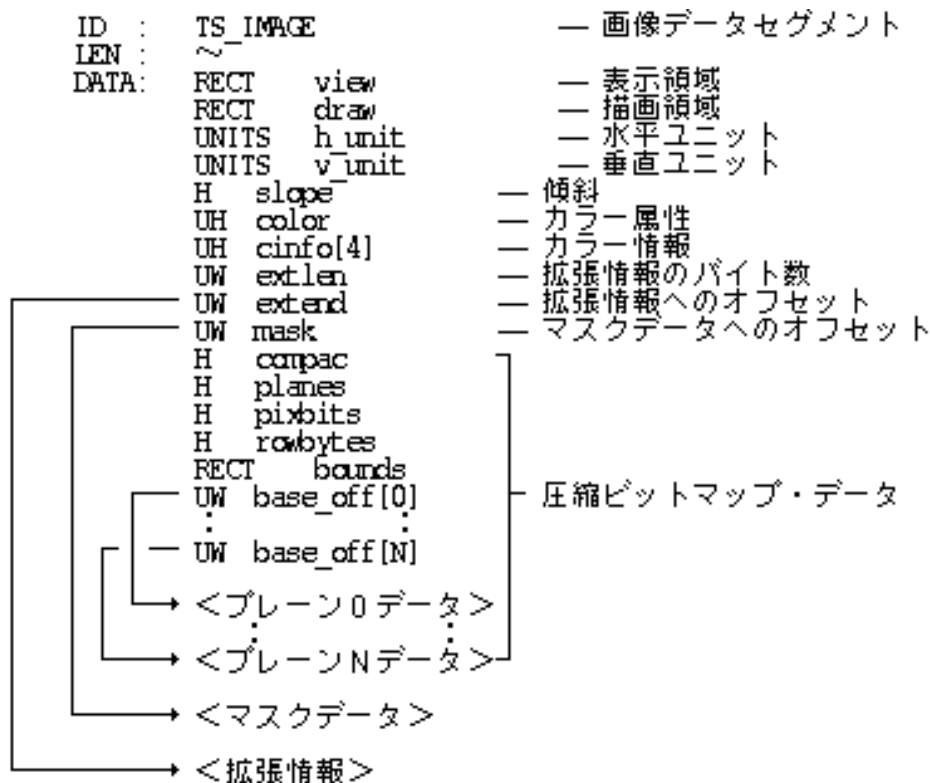


図 53 : 画像セグメントのデータ形式

仮身セグメント:

ID : TS\_VOBJ  
 LEN : ~  
 DATA: RECT view -- 表示領域  
 H height -- 開いた場合の仮身高さ  
 CHSIZE chsz -- 文字サイズ  
 COLOR frcol -- 枠の色  
 COLOR chcol -- 文字の色  
 COLOR tbcoll -- タイトル背景色  
 COLOR bgcol -- 開いた場合の背景色  
 UH dlen -- 固有データのバイト長  
 UB data[dlen] -- 固有データ(dlen バイト)

機能付箋セグメント:

ID : TS\_FFUSEN  
 LEN : ~  
 DATA: RECT view -- 表示領域  
 CHSIZE chsz -- 文字サイズ  
 COLOR frcol -- 枠の色  
 COLOR chcol -- 文字の色  
 COLOR tbcoll -- タイトル背景色  
 UH pict -- ピクトグラム/タイプ  
 UH appl[3] -- アプリケーションID  
 UB name[32] -- 付箋名  
 UB type[32] -- データタイプ名  
 UH dlen -- 固有データのバイト長  
 UB data[dlen] -- 固有データ(dlen バイト)

指定付箋セグメント:

ID : TS\_DFUSEN  
 LEN : ~  
 DATA: RECT view -- 表示領域  
 CHSIZE chsz -- 文字サイズ  
 COLOR frcol -- 枠の色  
 COLOR chcol -- 文字の色  
 COLOR tbcoll -- タイトル背景色  
 UH pict -- ピクトグラム/タイプ  
 UH appl[3] -- アプリケーションID  
 UB name[32] -- 付箋名  
 UW dlen -- 固有データのバイト長  
 UB dat[dlen] -- 固有データ(dlen バイト)

文章ページ割付け指定付箋セグメント:

用紙指定付箋:

ID : TS\_TPAGE  
 LEN : 14  
 SUBID: 0

|        |    |        |    |                 |
|--------|----|--------|----|-----------------|
| ATTR : | UB | attr   | -- | 面付け、綴じ方向指定      |
| DATA : | UH | length | -- | 用紙の長さ           |
|        | UH | width  | -- | 用紙の幅            |
|        | UH | top    | -- | オーバーレイ上マージン(天)  |
|        | UH | bottom | -- | オーバーレイ下マージン(地)  |
|        | UH | left   | -- | オーバーレイ左マージン(ノド) |
|        | UH | right  | -- | オーバーレイ右マージン(小口) |

マージン指定付箋:

|        |          |        |              |
|--------|----------|--------|--------------|
| ID :   | TS_TPAGE |        |              |
| LEN :  | 10       |        |              |
| SUBID: | 1        |        |              |
| ATTR : | UB       | -      | -- 未使用       |
| DATA : | UH       | top    | -- 上マージン(天)  |
|        | UH       | bottom | -- 下マージン(地)  |
|        | UH       | left   | -- 左マージン(ノド) |
|        | UH       | right  | -- 右マージン(小口) |

コラム指定付箋:

|        |          |         |                      |
|--------|----------|---------|----------------------|
| ID :   | TS_TPAGE |         |                      |
| LEN :  | 4,6      |         |                      |
| SUBID: | 2        |         |                      |
| ATTR : | UB       | column  | -- コラム数、およびコラム長均等化指定 |
| DATA : | UH       | colsp   | -- コラムマージン           |
|        | [UH      | colline | -- コラム罫線]            |

用紙オーバーレイ定義付箋:

|        |          |        |                     |
|--------|----------|--------|---------------------|
| ID :   | TS_TPAGE |        |                     |
| LEN :  | ~        |        |                     |
| SUBID: | 3        |        |                     |
| ATTR : | UB       | attr   | -- オーバーレイ番号/ページ適用属性 |
| DATA : | UB       | data[] | -- オーバーレイする文章データ    |

用紙オーバーレイ指定付箋:

|        |          |         |             |
|--------|----------|---------|-------------|
| ID :   | TS_TPAGE |         |             |
| LEN :  | 4        |         |             |
| SUBID: | 4        |         |             |
| ATTR : | UB       | ---     | -- 未使用      |
| DATA : | UH       | overlay | -- オーバーレイ指定 |

枠あけ指定付箋:

|        |          |      |          |
|--------|----------|------|----------|
| ID :   | TS_TPAGE |      |          |
| LEN :  | 10       |      |          |
| SUBID: | 5        |      |          |
| ATTR : | UB       | attr | -- 枠あけ属性 |
| DATA : | RECT     | area | -- 枠あけ領域 |

ページ番号指定付箋:

|      |          |  |  |
|------|----------|--|--|
| ID : | TS_TPAGE |  |  |
|------|----------|--|--|

LEN : 4  
SUBID: 6  
ATTR : B step -- ページの増加ステップ  
DATA: UH num -- ページ開始番号

条件改ページ指定付箋:

ID : TS\_TPAGE  
LEN : 2,4  
SUBID: 7  
ATTR : UB cond -- 条件  
DATA : [SCALE remain -- 残り領域指定]

充填行指定付箋:

ID : TS\_TPAGE  
LEN : 2  
SUBID: 8  
ATTR : UB - -- 未使用

行書式指定付箋:

行間隔指定付箋:

ID : TS\_TRULER  
LEN : 4  
SUBID: 0  
ATTR : UB attr -- 属性指定  
DATA : SCALE pitch -- 間隔指定

行揃え指定付箋:

ID : TS\_TRULER  
LEN : 2  
SUBID: 1  
ATTR : UB align -- 行揃えの方法

タブ書式指定付箋:

ID : TS\_TRULER  
LEN : ~  
SUBID: 2  
ATTR : UB attr -- 属性  
DATA: SCALE height -- 書式高さ指定  
SCALE pargap -- 段落間隔指定  
H left -- 行頭マージン  
H right -- 行末マージン  
H indent -- インデントマージン  
H ntabs -- タブストップ設定数  
UH tabs[] -- タブストップ位置(ntabs個)

フィールド書式指定付箋:

ID : TS\_TRULER



```

LEN : ~
SUBID: 3
ATTR : UB      attr      -- 属性
DATA:  SCALE   height    -- 書式高さ指定
      SCALE   paragap    -- 段落間隔指定
      UH      line      -- フィールド段落線属性
      H      nfld      -- フィールド設定数
+- UH      fld        -- フィールド開始位置
| UH      left       -- フィールド内左マージン
| UH      right      -- フィールド内右マージン
| UH      margin     -- 小数点位置 / インデントマージン
+- UH      f_attr    -- フィールド属性
      :
      :
```

文字方向指定付箋:

```

ID : TS_TRULER
LEN : 2
SUBID: 4
ATTR : UB      txdir    -- 文字列方向
```

行頭移動指定付箋:

```

ID : TS_TRULER
LEN : 2
SUBID: 5
ATTR : UB      ---     -- 未使用
```

文字指定付箋:

フォント指定付箋:

```

ID : TS_TFONT
LEN : 4~
SUBID: 0
ATTR : UB      -       -- 未使用
DATA : UH      class   -- フォントクラス
      [UB      name[] ] -- フォント(ファミリー)名
```

フォント属性指定付箋:

```

ID : TS_TFONT
LEN : 4
SUBID: 1
ATTR : UB      -       -- 未使用
DATA : UH      attr    -- フォント属性
```

文字サイズ指定付箋:

```

ID : TS_TFONT
LEN : 4
SUBID: 2
ATTR : UB      -       -- 未使用
DATA : CHSIZE  size    -- 文字サイズ
```

文字拡大 / 縮小指定付箋:

ID : TS\_TFONT  
LEN : 6  
SUBID: 3  
ATTR : UB - -- 未使用  
DATA : RATIO h\_ratio -- 文字高さ(サイズ)の拡大率  
RATIO w\_ratio -- 文字幅の拡大率

文字間隔指定付箋:

ID : TS\_TFONT  
LEN : 4  
SUBID: 4  
ATTR : UB attr -- 属性指定  
DATA : SCALE pitch -- 間隔指定

文字回転指定付箋:

ID : TS\_TFONT  
LEN : 4  
SUBID: 5  
ATTR : UB abs -- 絶対 / 相対指定  
DATA : UH angle -- 文字の横軸の反時計回りの回転角度

文字カラー指定付箋:

ID : TS\_TFONT  
LEN : 6  
SUBID: 6  
ATTR : UB - -- 未使用  
DATA : COLOR color -- 文字カラーの指定

文字基準位置移動付箋:

ID : TS\_TFONT  
LEN : 4  
SUBID: 7  
ATTR : UB attr -- 属性指定  
DATA : SCALE base -- 基準位置移動量

特殊文字指定付箋:

固定幅空白付箋:

ID : TS\_TUB  
LEN : 4  
SUBID: 0  
ATTR : UB - -- 未使用  
DATA : SCALE width -- 空白の幅

充填文字指定付箋:

ID : TS\_TUB  
LEN : ~  
SUBID: 1

ATTR : UB - -- 未使用  
DATA: UB str[] -- 充填文字列

#### 文字罫線指定付箋:

ID : TS\_TUB  
LEN : 4 + 罫線列データ数  
SUBID: 2  
ATTR : UB type -- 罫線のタイプ  
DATA : UH count -- 罫線列の繰り返し数  
UH lines[] -- 罫線指定データ列( LEN で指定されたデータ数)

#### 文字割付け指定付箋:

##### 結合指定付箋:

ID : TS\_TATTR  
LEN : 2  
SUBID: 0 -- 結合開始指定付箋  
ATTR : UB - -- 未使用  
ID : TS\_TATTR  
LEN : 2  
SUBID: 1 -- 結合終了指定付箋  
ATTR : UB - -- 未使用

##### 文字割付け指定付箋:

ID : TS\_TATTR  
LEN : 4  
SUBID: 2 -- 文字割付け開始指定付箋  
ATTR : UB kind -- 種類  
DATA : SCALE width -- 割り付け幅  
ID : TS\_TATTR  
LEN : 2  
SUBID: 3 -- 文字割付け終了指定付箋  
ATTR : UB - -- 未使用

##### 添字指定付箋:

ID : TS\_TATTR  
LEN : 6  
SUBID: 4 -- 添字開始指定付箋  
ATTR : UB type -- タイプ  
DATA : SCALE pos -- 文字位置の移動量  
RATIO size -- 文字サイズの比率  
ID : TS\_TATTR  
LEN : 2  
SUBID: 5 -- 添字終了指定付箋  
ATTR : UB - -- 未使用

##### ルビ指定付箋:

ID : TS\_TATTR  
LEN : ~

SUBID: 6 -- ルビ開始指定付箋  
ATTR : UB attr -- ルビ属性  
DATA : UB rubi[] -- ルビ文字列  
ID : TS\_TATTR  
LEN : 2  
SUBID: 7 -- ルビ終了指定付箋  
ATTR : UB - -- 未使用

禁則指定付箋:

ID : TS\_TATTR  
LEN : ~  
SUBID: 8 -- 行頭禁則指定  
ATTR : UB kind -- 行頭禁則タイプ  
[DATA : UB ch[]] -- 行頭禁則対象文字  
ID : TS\_TATTR  
LEN : ~  
SUBID: 9 -- 行末禁則指定  
ATTR : UB kind -- 行末禁則タイプ  
[DATA : UB ch[]] -- 行末禁則対象文字

文字修飾指定付箋:

文字修飾指定付箋:

ID : TS\_TSTYLE  
LEN : 2,6  
SUBID: type -- 文字修飾開始付箋  
ATTR : UB attr -- 属性  
DATA : [COLOR color ] -- 修飾カラー指定(省略可能)  
ID : TS\_TSTYLE  
LEN : 2  
SUBID: type + 1 -- 文字修飾終了付箋  
ATTR : UB - -- 未使用  
SUBID: 0: 下線開始 1: 下線終了  
2: 上線開始 3: 上線終了  
4: 打ち消し線開始 5: 打ち消し線終了  
6: 枠囲み線開始 7: 枠囲み線終了  
8: 上(右)傍点開始 9: 上(右)傍点終了  
10: 下(下)傍点開始 11: 下(下)傍点終了  
12: 反転開始 13: 反転終了  
14: 網掛開始 15: 網掛終了  
16: 背景開始 17: 背景終了  
18: 無印字開始(終了付箋までのデータを印字しない)  
19: 無印字終了  
20~127: (予約)  
128~ : 未定義(アプリケーション依存)

変数参照指定付箋:

変数参照指定付箋:

```

ID   : TS_TVAR
LEN  : 4
SUBID: 0          -- 変数参照
ATTR : UB        -  -- 未使用
DATA : H   var_id -- 変数ID
ID   : TS_TVAR
LEN  : ~
SUBID: 1          -- 変数参照
ATTR : UB        -  -- 未使用
DATA : UB      name[] -- 変数名

```

#### 文章メモ指定付箋:

##### 文章メモ指定付箋:

```

ID   : TS_TMEMO
LEN  : ~
SUBID: 0
ATTR : UB        -  -- 未使用
DATA : UB      memo[] -- メモ文字列

```

#### 文章アプリケーション指定付箋:

##### 文章アプリケーション指定付箋:

```

ID   : TS_TAPPL
LEN  : ~
SUBID: -          -- アプリケーション定義
ATTR : UB        -  -- アプリケーション定義
DATA : UH      appl[3] -- アプリケーションID
      UB      param[] -- アプリケーション・パラメータ

```

#### 図形要素セグメント:

##### 長方形セグメント:

```

ID   : TS_FPRIM
LEN  : 18
SUBID: 0
ATTR : UB      mode -- 描画モード
DATA : UH      l_atr -- 線属性
      UH      l_pat  -- 線パターンID
      UH      f_pat  -- 塗り潰しパターンID
      UH      angle  -- 回転角
      RECT    frame  -- 外接長方形

```

##### 角丸長方形セグメント:

```

ID   : TS_FPRIM
LEN  : 22
SUBID: 1
ATTR : UB      mode -- 描画モード

```

|        |      |        |    |            |
|--------|------|--------|----|------------|
| DATA : | UH   | l_attr | -- | 線属性        |
|        | UH   | l_pat  | -- | 線パターンID    |
|        | UH   | f_pat  | -- | 塗り潰しパターンID |
|        | UH   | angle  | -- | 回転角度       |
|        | UH   | rh     | -- | 丸めの水平直径    |
|        | UH   | rv     | -- | 丸めの垂直直径    |
|        | RECT | frame  | -- | 外接長方形      |

楕円セグメント:

|        |          |        |    |            |
|--------|----------|--------|----|------------|
| ID :   | TS_FPRIM |        |    |            |
| LEN :  | 18       |        |    |            |
| SUBID: | 2        |        |    |            |
| ATTR : | UB       | mode   | -- | 描画モード      |
| DATA : | UH       | l_attr | -- | 線属性        |
|        | UH       | l_pat  | -- | 線パターンID    |
|        | UH       | f_pat  | -- | 塗り潰しパターンID |
|        | UH       | angle  | -- | 回転角度       |
|        | RECT     | frame  | -- | 外接長方形      |

扇形セグメント:

|        |          |        |    |            |
|--------|----------|--------|----|------------|
| ID :   | TS_FPRIM |        |    |            |
| LEN :  | 26       |        |    |            |
| SUBID: | 3        |        |    |            |
| ATTR : | UB       | mode   | -- | 描画モード      |
| DATA : | UH       | l_attr | -- | 線属性        |
|        | UH       | l_pat  | -- | 線パターンID    |
|        | UH       | f_pat  | -- | 塗り潰しパターンID |
|        | UH       | angle  | -- | 回転角度       |
|        | RECT     | frame  | -- | 外接長方形      |
|        | PNT      | start  | -- | 開始点        |
|        | PNT      | end    | -- | 終了点        |

弓形セグメント:

|        |          |        |    |            |
|--------|----------|--------|----|------------|
| ID :   | TS_FPRIM |        |    |            |
| LEN :  | 26       |        |    |            |
| SUBID: | 4        |        |    |            |
| ATTR : | UB       | mode   | -- | 描画モード      |
| DATA : | UH       | l_attr | -- | 線属性        |
|        | UH       | l_pat  | -- | 線パターンID    |
|        | UH       | f_pat  | -- | 塗り潰しパターンID |
|        | UH       | angle  | -- | 回転角度       |
|        | RECT     | frame  | -- | 外接長方形      |
|        | PNT      | start  | -- | 開始点        |
|        | PNT      | end    | -- | 終了点        |

多角形セグメント:

|        |          |      |    |       |
|--------|----------|------|----|-------|
| ID :   | TS_FPRIM |      |    |       |
| LEN :  | ~        |      |    |       |
| SUBID: | 5        |      |    |       |
| ATTR : | UB       | mode | -- | 描画モード |

|        |     |        |                    |
|--------|-----|--------|--------------------|
| DATA : | UH  | l_atr  | -- 線属性             |
|        | UH  | l_pat  | -- 線パターンID         |
|        | UH  | f_pat  | -- 塗り潰しパターンID      |
|        | UH  | round  | -- 角の丸めの円の直径:拡張レベル |
|        | UH  | np     | -- 点数(>2)          |
|        | PNT | pt[np] | -- 点の配列(np 個の要素)   |

#### 直線セグメント:

|        |          |       |            |
|--------|----------|-------|------------|
| ID :   | TS_FPRIM |       |            |
| LEN :  | 14       |       |            |
| SUBID: | 6        |       |            |
| ATTR : | UB       | mode  | -- 描画モード   |
| DATA : | UH       | l_atr | -- 線属性     |
|        | UH       | l_pat | -- 線パターンID |
|        | PNT      | start | -- 開始点     |
|        | PNT      | end   | -- 終了点     |

#### 楕円弧セグメント:

|        |          |       |            |
|--------|----------|-------|------------|
| ID :   | TS_FPRIM |       |            |
| LEN :  | 24       |       |            |
| SUBID: | 7        |       |            |
| ATTR : | UB       | mode  | -- 描画モード   |
| DATA : | UH       | l_atr | -- 線属性     |
|        | UH       | l_pat | -- 線パターンID |
|        | UH       | angle | -- 回転角度    |
|        | RECT     | frame | -- 外接長方形   |
|        | PNT      | start | -- 開始点     |
|        | PNT      | end   | -- 終了点     |

#### 折れ線セグメント:

|        |          |        |                  |
|--------|----------|--------|------------------|
| ID :   | TS_FPRIM |        |                  |
| LEN :  | ~        |        |                  |
| SUBID: | 8        |        |                  |
| ATTR : | UB       | mode   | -- 描画モード         |
| DATA : | UH       | l_atr  | -- 線属性           |
|        | UH       | l_pat  | -- 線パターンID       |
|        | UH       | round  | -- 角の丸め:拡張レベル    |
|        | UH       | np     | -- 点数 (>1)       |
|        | PNT      | pt[np] | -- 点の配列(np 個の要素) |

#### 曲線セグメント:

|        |          |       |               |
|--------|----------|-------|---------------|
| ID :   | TS_FPRIM |       |               |
| LEN :  | ~        |       |               |
| SUBID: | 9        |       |               |
| ATTR : | UB       | mode  | -- 描画モード      |
| DATA : | UH       | l_atr | -- 線属性        |
|        | UH       | l_pat | -- 線パターンID    |
|        | UH       | f_pat | -- 塗り潰しパターンID |
|        | H        | type  | -- 曲線のタイプ     |
|        | UH       | np    | -- 点数 (>0)    |

PNT pt[np] -- 点の配列(np 個の要素)  
マーカー列セグメント:

ID : TS\_FPRIM  
LEN : ~  
SUBID: 10  
ATTR : UB mode -- 描画モード  
DATA : UH market -- マーカーID  
UH np -- 点数 (>0)  
PNT pt[np] -- 点の配列(np 個の要素)

任意図形セグメント:

ID : TS\_FPRIM  
LEN : ~  
SUBID: 11  
ATTR : UB mode -- 描画モード  
DATA : UH f\_pat -- 塗り潰しパターンID  
UH sy -- 垂直座標の開始値(最小値)  
UH nr -- 垂直座標数  
H bx -- 水平座標値のバイアス  
< 垂直座標値 sy ~ sy+nr-1 にそれぞれ対応した、以下に示す  
水平座標値列の要素が nr 個連続する >  
UH nh -- 水平座標の数  
UH h[nh] -- 水平座標の値 (nh 個の要素)  
:  
:

データ定義セグメント:

カラーマップ定義セグメント:

ID : TS\_FDEF  
LEN : ~  
SUBID: 0  
ATTR : UB - -- 未使用  
DATA : UH nent -- エントリ数  
COLOR col[nent] -- カラーマップ (nent 個の要素)

マスクデータ定義セグメント:

ID : TS\_FDEF  
LEN : ~  
SUBID: 1  
ATTR : UB type -- マスク定義タイプ  
DATA : UH id -- 定義するマスクID  
< type = 0の場合 >  
UH hsize -- 横のサイズ  
UH vsize -- 縦のサイズ  
UB mask[] -- マスクビットマップ

パターン定義セグメント:

ID : TS\_FDEF  
LEN : ~



SUBID: 2  
 ATTR : UB type -- パターン定義タイプ  
 DATA : UH id -- 定義するパターンID  
 < type = 0の場合 >  
 UH hsize -- 横のサイズ  
 UH vsize -- 縦のサイズ  
 UH ncol -- 前景カラー数  
 COLOR fgcol[ncol] -- 前景色 (ncol 個の要素)  
 COLOR bgcol -- 背景色  
 UH mask[ncol] -- マスクID(ncol 個の要素)

線種定義セグメント:

ID : TS\_FDEF  
 LEN : ~  
 SUBID: 3  
 ATTR : UB type -- 線種定義タイプ  
 DATA : UH id -- 定義する線種ID  
 < type = 0 の場合 >  
 UH n -- 線種定義バイト数  
 UB mask[nb] -- 線種定義バイト列

マーカー定義セグメント:

ID : TS\_FDEF  
 LEN : ~  
 SUBID: 4  
 ATTR : UB type -- マーカー定義タイプ  
 DATA : UH id -- 定義するマーカーID  
 < type = 0 の場合 >  
 UH size -- マーカーサイズ  
 COLOR fgcol -- マーカー色  
 [UH mask -- マスクデータID]

グループ定義セグメント:

グループ定義セグメント:

ID : TS\_FGRP  
 LEN : 4  
 SUBID: 0 -- グループ開始  
 ATTR : UB - -- 未使用  
 DATA : UH id -- グループID  
 ID : TS\_FGRP  
 LEN : 2  
 SUBID: 1 -- グループ終了  
 ATTR : UB - -- 未使用

マクロ定義 / 参照セグメント:

マクロ定義セグメント:

ID : TS\_FMAC

```

LEN : 4
SUBID: 0 -- マクロ定義開始
ATTR : UB - -- 未使用
DATA : UH id -- 定義するマクロID
ID : TS_FMAC
LEN : 2
SUBID: 1 -- マクロ定義終了
ATTR : UB - -- 未使用

```

マクロ参照セグメント:

```

LEN : 4
SUBID: 2
ATTR : UB - -- 未使用
DATA : UH id -- 参照するマクロID

```

図形修飾セグメント:

図形要素修飾セグメント:

```

ID : TS_FATTR
LEN : ~
SUBID: 0
ATTR : UB type -- 修飾タイプ
DATA : < type = 0の場合 >
      UH arrow -- 矢印属性

```

座標変換セグメント:

```

ID : TS_FATTR
LEN : 6, 8, 10
SUBID: 1
ATTR : UB - -- 未使用
DATA : H dh -- 水平座標移動量
      H dv -- 垂直座標移動量
      [UH hangle -- 水平軸回転角度]
      [H vangle -- 垂直軸回転角度]

```

図形ページ割付け指定付箋セグメント:

用紙指定付箋:

```

ID : TS_FPAGE
LEN : 14
SUBID: 0
ATTR : UB attr -- 面付け、綴じ方向指定
DATA : UH length -- 用紙の長さ
      UH width -- 用紙の幅
      UH top -- オーバーレイ上マージン(天)
      UH bottom -- オーバーレイ下マージン(地)
      UH left -- オーバーレイ左マージン(ノド)
      UH right -- オーバーレイ右マージン(小口)

```

マージン指定付箋:

ID : TS\_FPAGE  
LEN : 10  
SUBID: 1  
ATTR : UB - -- 未使用  
DATA : UH top -- 上マージン(天)  
UH bottom -- 下マージン(地)  
UH left -- 左マージン(ノド)  
UH right -- 右マージン(小口)

用紙オーバーレイ定義付箋:

ID : TS\_FPAGE  
LEN : ~  
SUBID: 3  
ATTR : UB attr -- オーバーレイ番号/ページ適用属性  
DATA : UB data[] -- オーバーレイする文章データ

用紙オーバーレイ指定付箋:

ID : TS\_FPAGE  
LEN : 4  
SUBID: 4  
ATTR : UB --- -- 未使用  
DATA : UH overlay -- オーバーレイ指定

ページ番号指定付箋:

ID : TS\_FPAGE  
LEN : 4  
SUBID: 6  
ATTR : B step -- ページの増加ステップ  
DATA: UH num -- ページ開始番号

図形メモ指定付箋:

図形メモ指定付箋:

ID : TS\_FMEMO  
LEN : ~  
SUBID: 0  
ATTR : UB - -- 未使用  
DATA : UB memo[] -- メモ文字列

図形アプリケーション指定付箋:

図形アプリケーション指定付箋:

ID : TS\_FAPPL  
LEN : ~  
SUBID: - -- アプリケーション定義  
ATTR : UB - -- アプリケーション定義  
DATA : UH appl[3] -- アプリケーションID

## 付録 B 準TAD規格

TADデータ構成では、異なるプロセッサ間を含めたデータの互換性を完全に保証するために8ビット以上のビット長を持つデータのバイトオーダーは、上位バイトが先にくるビッグエンディアン形式に統一している。

しかしながら、この形式はリトルエンディアン形式のCPU上で動作する既存のアプリケーションを移植する場合等には負担となる場合もあるため、これを救済するために、リトルエンディアンのデータ形式を持つTADデータ構成を、準TAD規格として規定する。

準TAD規格は、TAD規格に対して以下に示す変更 / 制限を行なったものである。

1. 使用する文字コードは2バイトの固定長とし、かつ第2バイトが先にくる形式とする。1バイトの制御コードの直後には、常に値 "0" のバイトを詰めるものとする。すなわち、文字コードはリトルエンディアンの16ビットデータとみなす。
2. 各セグメントのデータ構造として記述されている、H, UH, W, UW, COLOR, UNITS, CHSIZE, SCALE, RATIO のデータタイプのバイトオーダーは、下位バイトが先にくるリトルエンディアン形式とする。UBタイプのデータには、1.で記述した2バイト固定長の文字コードのみを使用する。
3. 可変長セグメントの形式は、以下示すバイトオーダーが逆転した形式となる。

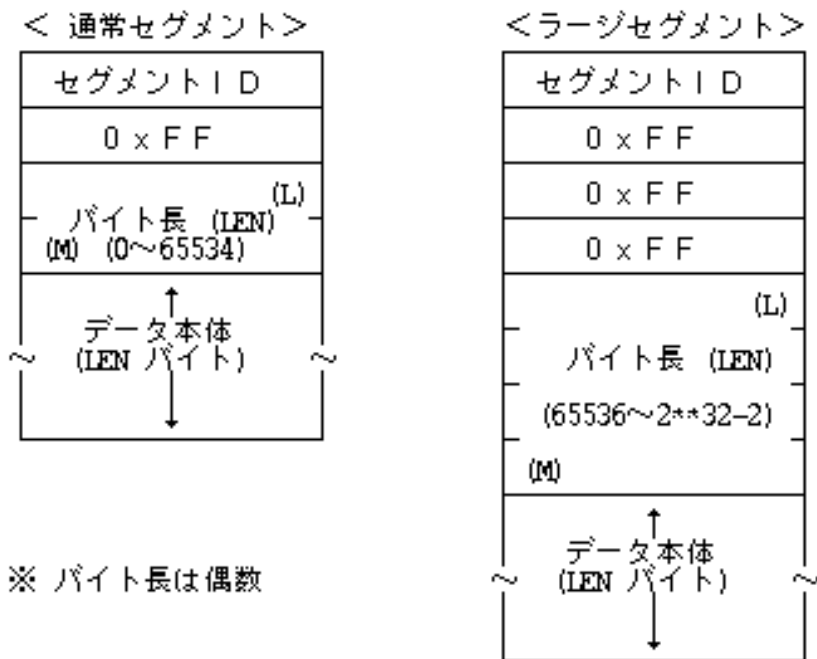


図 54 : 準 TAD 規格での可変長セグメントの構造

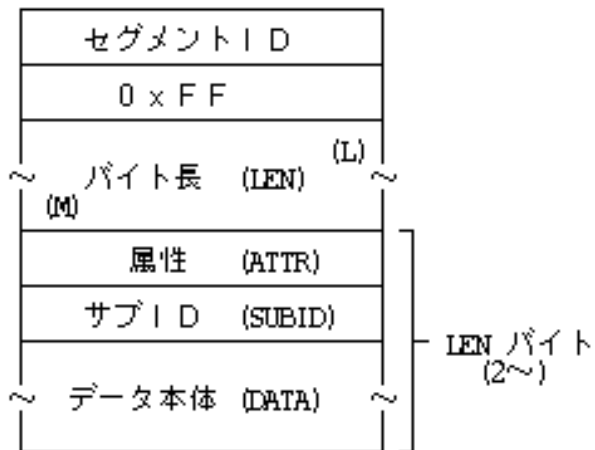


図 55 : 準 TAD 規格での文章 / 図形付箋の構造

準TAD規格に基づいたシステム / アプリケーションは、以下の点を遵守しなくてはならない。

- 準TAD規格のデータは、リトルエンディアン形式の準標準FD形式の交換媒体内、またはシステム内部でのみ使用すること。
- 何らかの方法により、TAD規格データとの双方向のデータ交換機能を有していること。  
すなわち:
  - TAD規格の交換用データを出力できること。
  - TAD規格の交換用データを入力できること。
  - 準TAD規格である旨を明記すること。
  - TAD規格データとの交換方法を明記すること。

[この章の目次にもどる](#)

[前頁:3.5 文章付箋セグメントにもどる](#)

[次頁:第4章 フロッピーディスク形式にすすむ](#)

# 第4章 フロッピーディスク形式

## 4.1 概要

TRON仕様では、データの交換媒体であるフロッピーディスクの標準形式を規定している。これは、基本的にアプリケーションプログラム、およびアプリケーションデータの交換用フロッピーディスクに対してのみ適用されるが、マシンに依存したシステムフロッピーディスク等の場合も同一形式であることが望ましい。

フロッピーディスクのセクタは物理ブロックとも呼び、物理的なアクセスの単位となる。また、論理的なアクセス/アドレッシングの単位を論理ブロックと呼び、連続したn個の物理ブロック(セクタ)から構成される。論理ブロックの大きさは標準フロッピーディスクでは1024バイトとする。

単にブロックといった場合は論理ブロックを示し、論理ブロック単位のアドレスをブロックアドレスと呼ぶ。

最大論理ブロック数、最大ファイル参照カウンタの制限を緩和した拡張形式がある。拡張形式も基本的には標準形式と同一である。拡張形式に関しては、標準形式と異なる部分のみ記述する。

## 4.2 ファイルシステム構成

### 4.2.1 全体構成

標準フロッピーディスクは以下に示す全体の構成を持つ。

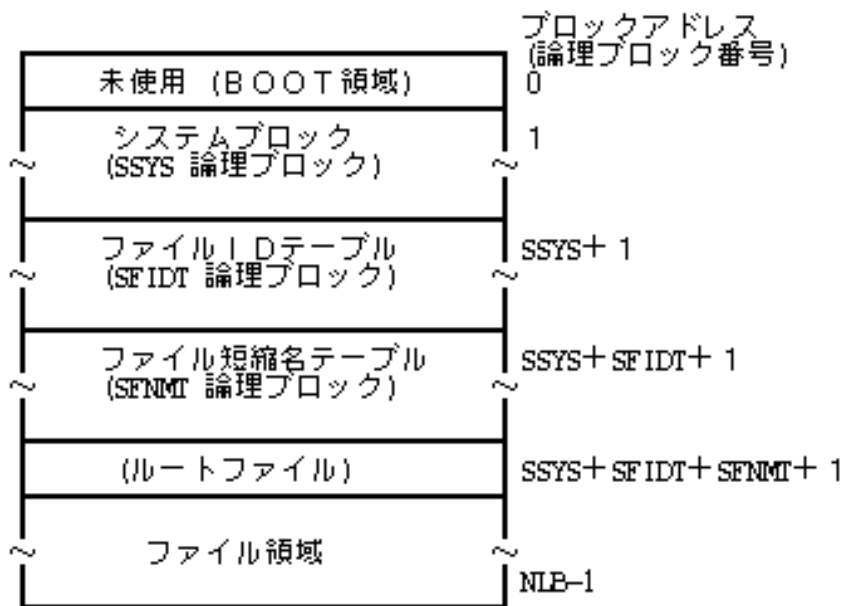


図 56 : 全体構成

## 4.2.2 システムブロック構成

システムブロックはファイルシステム全体の情報を保持するブロックであり、以下に示す構成を持つ。

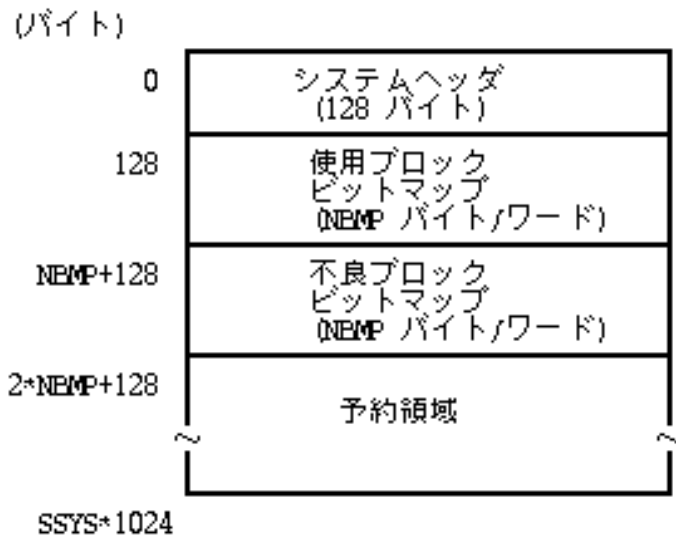


図 57: システムブロック構成

システムヘッダは、128 バイトから構成され、以下の構成を持つ。この部分はファイルシステムの生成時に作成され、以後、“\*”を付けたデータ以外は変更されない。

|     | 0                             | 1 | 2 | 3      |
|-----|-------------------------------|---|---|--------|
| 0   | TRON ディスク ID (=0x42FE/0x52FE) |   |   |        |
| 2   | ディスク形式 ID (=0x6400/0x6401)    |   |   |        |
| 4   | システムブロックのサイズ (=SSYS)          |   |   |        |
| 6   | ファイル ID テーブルサイズ (=SFIDT)      |   |   |        |
| 8   | ファイル短縮名テーブルサイズ (=SFNMT)       |   |   |        |
| 10  | ビットマップのサイズ (=NBMP)            |   |   |        |
| 12  | ~ (予約) - 12 バイト ~             |   |   |        |
| 22  |                               |   |   |        |
| 24  | 論理ブロックのバイト数 (=1024)           |   |   |        |
| 26  | 最大ファイル数 (=NFMAX)              |   |   |        |
| 28  | 使用言語 (コード体系)                  |   |   |        |
| 30  | アクセス管理レベル                     |   |   |        |
| 32  | 論理ブロックの総数                     |   |   | (=NLB) |
| 36  | 空き論理ブロック数                     |   |   | *      |
| 40  | 最新のシステムブロックの更新日時              |   |   | *      |
| 44  | ファイルシステムの生成日時                 |   |   |        |
| 48  | ファイルシステム名称 (40 バイト)           |   |   | ~ } *  |
| 88  | デバイス所在名 (40 バイト)              |   |   | ~ } *  |
| 128 |                               |   |   |        |

図 58 : システムヘッダの構成

各データの内容を以下に示す。各データは上位バイトが先(若いアドレス)にくるビッグエンディアン形式である。

TRONディスクID:

TRON仕様OSを示す下記の固定データである。

標準形式の場合 0x42FE

拡張形式の場合 0x52FE

ディスク形式ID:

ディスク形式を示すIDであり、下記の固定データである。

標準形式の場合 0x6400

拡張形式の場合 0x6401

システムブロックサイズ:

システムブロックの論理ブロック数である。

ファイルIDテーブルサイズ:

ファイルIDテーブルの論理ブロック数であり、以下の計算式により求まる値を整数に切り上げた値である。

$$\begin{aligned} \text{SFIDT} &= (\text{最大ファイル数} \times \text{ファイルIDテーブルの1エントリバイト数}) / \text{論理ブロックのバイト数} \\ &= (\text{NFMAX} \times 4) / 1024 \end{aligned}$$

ファイル短縮名テーブルサイズ:

ファイル短縮名テーブルの論理ブロック数であり、以下の計算式により求まる値を整数に切り上げた値である。

$$\begin{aligned} \text{SFNMT} &= (\text{最大ファイル数} \times \text{ファイル短縮名テーブルの1エントリバイト数}) / \text{論理ブロックのバイト数} \\ &= (\text{NFMAX} \times 4) / 1024 \end{aligned}$$

ビットマップのサイズ:

ビットマップのサイズであり、標準形式の場合はバイト数、拡張形式の場合はワード数(1ワード = 4バイト)で表わす。

以下の計算式により求まる値を整数に切り上げた以上の値となる。

標準形式の場合

NBMP 論理ブロックの総数/8

NLB/8

拡張形式の場合

NBMP 論理ブロックの総数/32

NLB/32

論理ブロックのバイト数:

1論理ブロックのバイト数で、標準フロッピーディスクでは1024となる。

最大ファイル数:

1ファイルシステム内で生成できるファイルの最大数である。

使用言語:

ファイルシステムでの使用言語、即ち使用文字コード体系を示すコードであり、日本語



コード(2バイトコード体系)の場合は0x0021となる。

#### アクセス管理レベル:

ファイルシステム内のファイルのアクセス管理情報の設定方法を示す。標準フロッピーの場合は、通常0とする。

0(レベル0) アクセス管理は行なわない。

1(レベル1) アクセス管理を部分的に行なう。

2(レベル2) アクセス管理を完全に行なう。

#### 論理ブロックの総数:

ファイルシステム内の論理ブロックの総数である。

#### 空き論理ブロック数:

未使用の論理ブロック数を示し、使用ブロックビットマップ上の"0"のビットの総数に等しい。

#### 最新のシステムブロックの更新日時:

システムブロックの最新の更新日時であり、1985年1月1日0時0分0秒(GMT)からの秒数で示される。

#### ファイルシステムの生成日時:

ファイルシステムの生成日時であり、1985年1月1日0時0分0秒(GMT)からの秒数で示される。

#### ファイルシステムの名称:

40バイトのファイルシステムの名称であり、ファイルシステムの生成時に使用言語で示される文字コード体系により設定される。40バイトに満たない部分には、0が詰められる。ファイルシステムは、この名称により一意的に識別される。

#### デバイス所在名:

40バイトのデバイス所在名であり、ファイルシステムの生成時に使用言語で示される文字コード体系により設定される。40バイトに満たない部分には、0が詰められる。標準フロッピーの場合は、"フロッピーディスク"となる。

ビットマップは論理ブロックの管理に使用され、使用ブロックビットマップと不良ブロックビットマップの2つがある。

1つの論理ブロックが1つのビットに一対一対応し、使用ブロックビットマップでは、"0"で未使用、"1"で使用中または不良を示し、不良ブロックビットマップでは、"0"で正常、"1"で不良を示す。不良ブロックに対応する使用ブロックビットマップ上のビットは、必ず"1"に設定される。

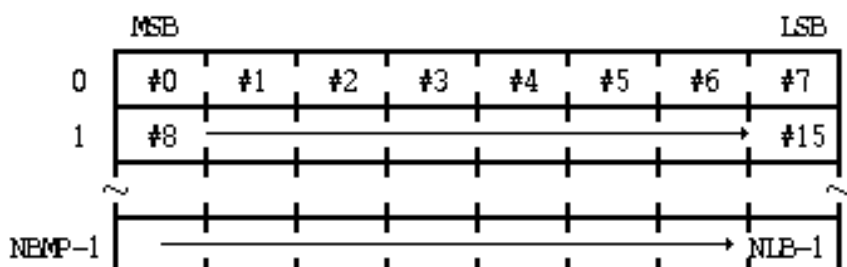


図 59: ビットマップの構成例

### 4.2.3 ファイルIDテーブル

ファイルIDテーブルは、そのファイルシステム内に存在するファイルの開始ブロックアドレス (3 バイト) とファイルの参照カウント (1バイト) をファイル ID 順に並べたテーブルであり、システムヘッダ内にある最大ファイル数分のエントリから構成される。

ファイルの参照カウントは、そのファイルを参照している同一ファイルシステム内の固定リンク数を示す。参照カウント0は、どこからも参照されておらずファイルが削除可能であることを示す。

標準形式では、最大ファイル参照カウントは 255 となる。拡張形式では、ファイル参照カウントが 255 以上になった場合は、ファイルIDテーブルのファイル参照カウント値を 255 とし、実際の参照カウント値 (255 以上) をファイルヘッダへ記録する。

標準フロッピーでは、最大ファイル数は NFMAX であり、ファイル ID テーブル全体は (NFMAX × 4) バイトから構成される。

ID に対応する 4 バイトのエントリが 0 の場合は、未使用エントリを意味する。

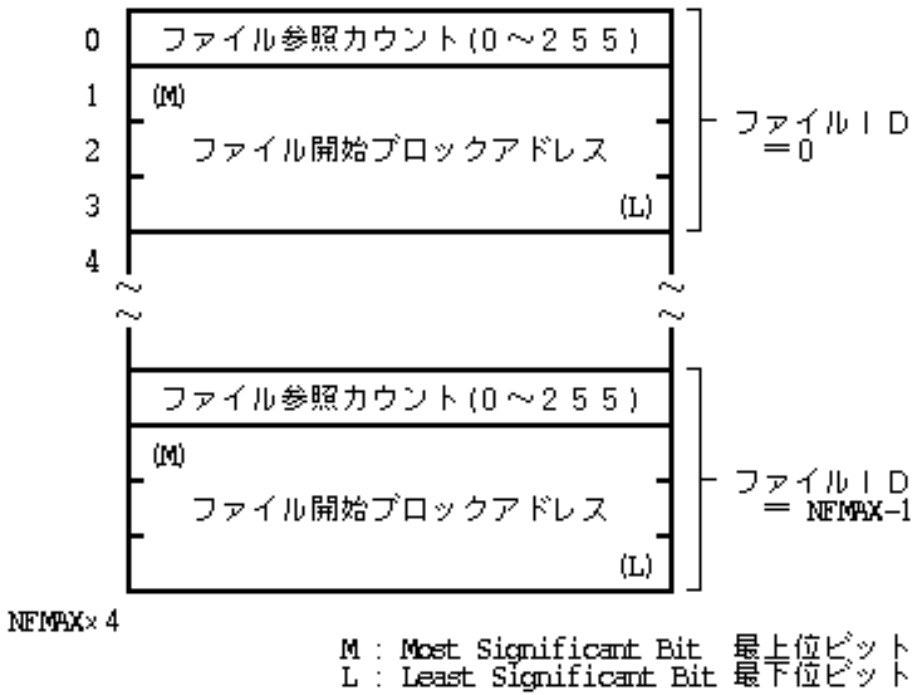


図 60 : ファイルIDテーブルの構成

### 4.2.4 ファイル短縮名テーブル

ファイル短縮名テーブルは、そのファイルシステム内に存在するファイルのファイル名を短縮化 (ハッシュ化) した4バイトの値をファイル ID 順に並べたテーブルであり、システムヘッダ内にある最大ファイル数分のエントリから構成される。

標準フロッピーでは、最大ファイル数は NFMAX であり、ファイル短縮名テーブル全体は (NFMAX × 4) バイトから構成される。

40 バイトのファイル名全体は各ファイルのヘッダ部分に入るが、このテーブルはファイル名によるファイルの検索を高速に行なうために使用される。

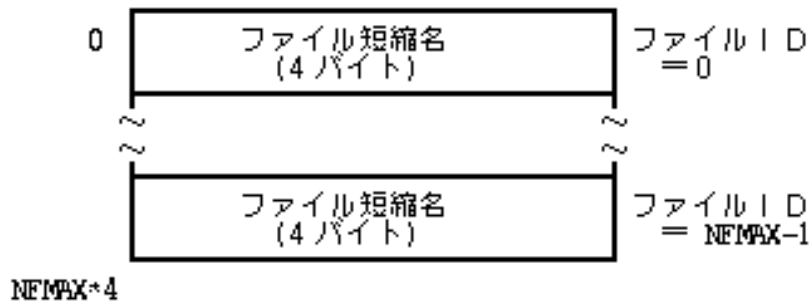


図 61 : ファイル短縮名テーブルの構成

ファイル短縮名は以下の方法により生成される。

1. ファイル名を4バイトずつパックした以下のような32ビットデータの配列をNAME[10]とする。ファイル名が40バイトに満たない場合は0が詰められる。

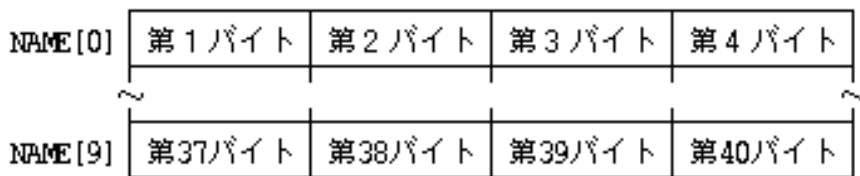


図 62 : NAME[10]

2. 初期値を0とし、NAME[]の各要素と順番に排他的論理和を取りながら、右に1ビット回転シフトすることにより得られたデータをファイル短縮名とする。即ち、短縮名は以下の手順で得られる。

```
短縮名 = 0;
for (i = 0; i < 10; i++) {
    短縮名 = 短縮名 ^ NAME[i];
    短縮名 を右に1ビット回転シフトする。
}
```

## 4.2.5 ファイル領域

ファイル領域にはそのファイルシステム内のすべてのファイルが入り、その先頭にはルートファイルが必ず存在する。ルートファイルはファイルシステムの生成時(フォーマット時)に自動的に生成されるファイルで以下の特徴を持つ。

- ファイルID = 0
- ファイル名はシステムブロックのファイルシステム名と同一
- 参照カウント = 1

## 4.3 ファイル構成

### 4.3.1 全体構成

1つのファイルは、順序付けられたレコード列から構成され、1つのレコードは可変長のバイト列から構成される。1つのファイルは、ヘッダブロック、データブロック、インデックスブロック、および間接ブロックから構成され、その構成はインデックスレベルと呼ぶ値により異なる。インデックスレベルはレコードインデックスの多重度を示す値で、レコードの数に応じて動的に

変化する。インデックスレベルが 0, 1, 2 の場合のファイルの構成を以下に示す。  
 なお、リンクファイルはヘッダブロックからのみ構成される。

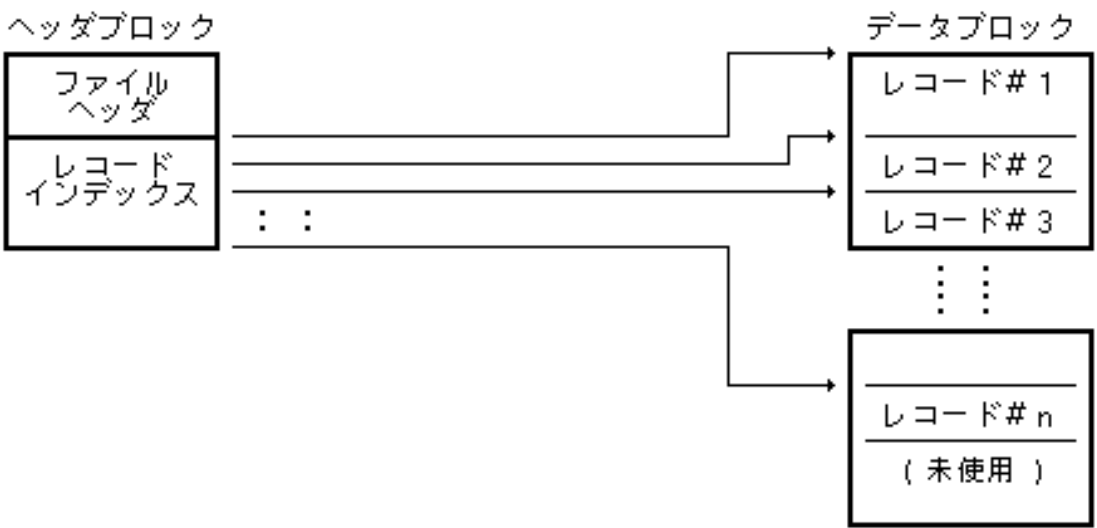


図 63 : ファイルの構成(インデックスレベル=0)

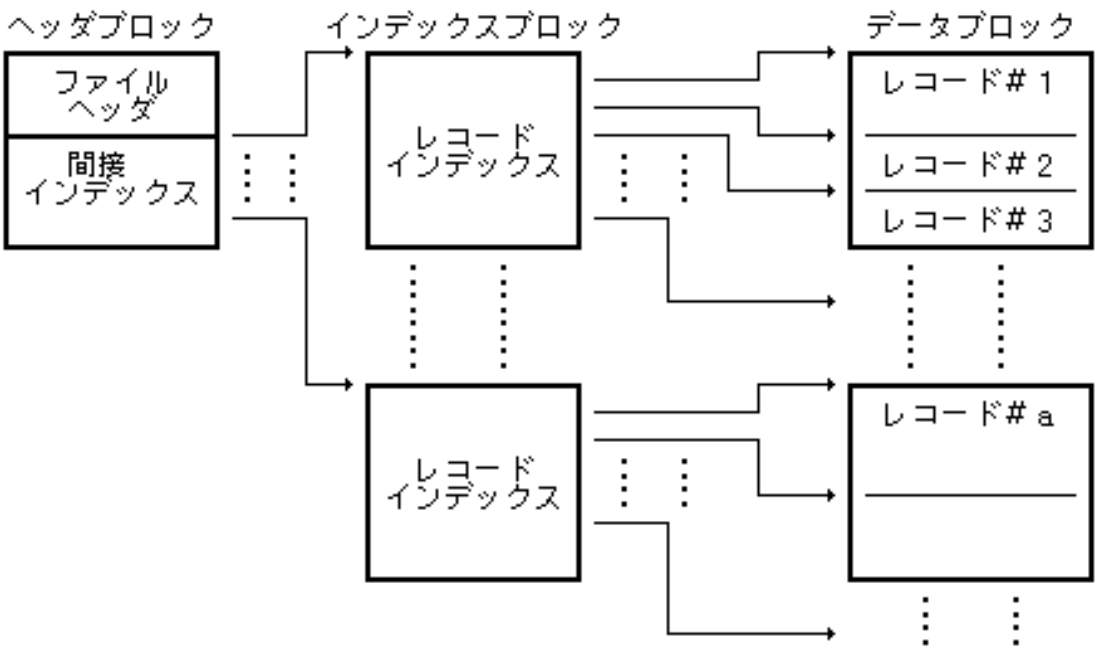


図 64 : ファイルの構成(インデックスレベル=1)

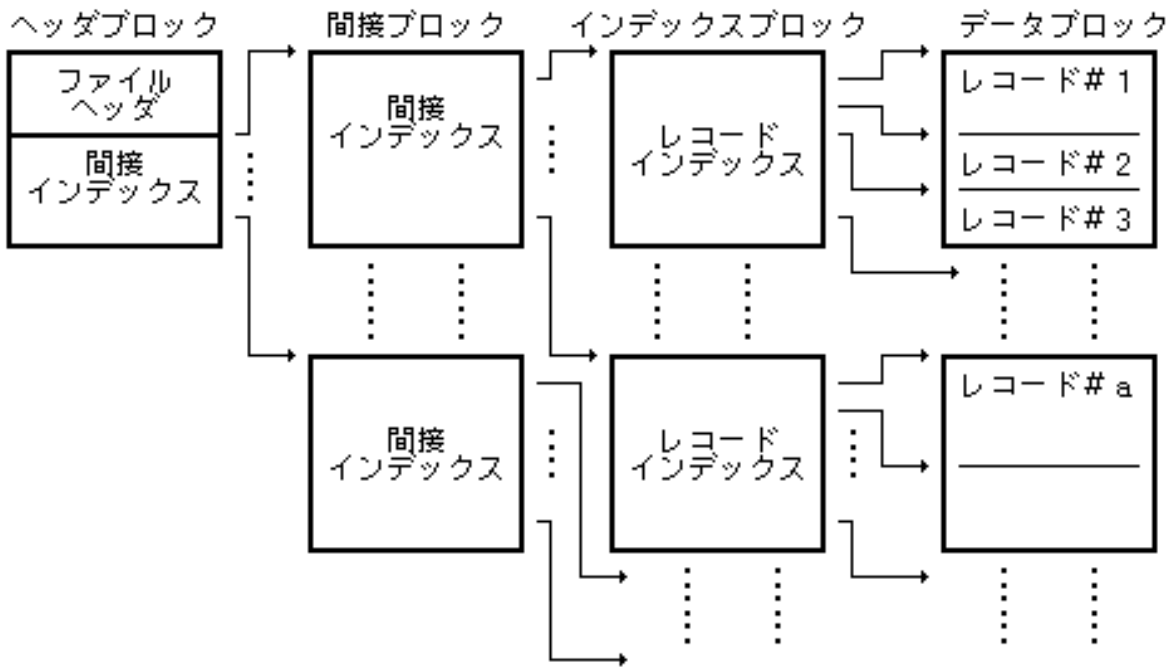


図 65 : ファイルの構成(インデックスレベル=2)

### 4.3.2 ヘッダブロックの構成

ファイルのヘッダブロックは、以下に示す構成となる。

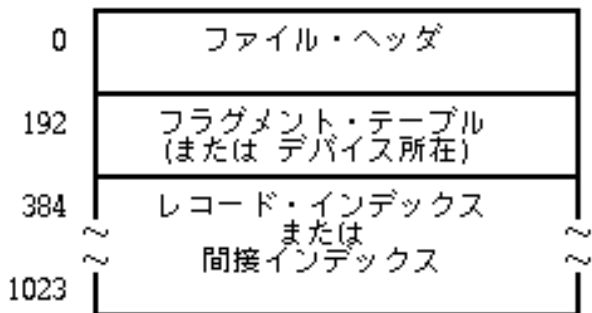


図 66 : ヘッダブロックの構成

### 4.3.3 ファイルヘッダの構成

ファイルヘッダは 192 バイトで構成され、ファイルの各種の管理情報が保持されている。以下にファイルヘッダの内容を示す。

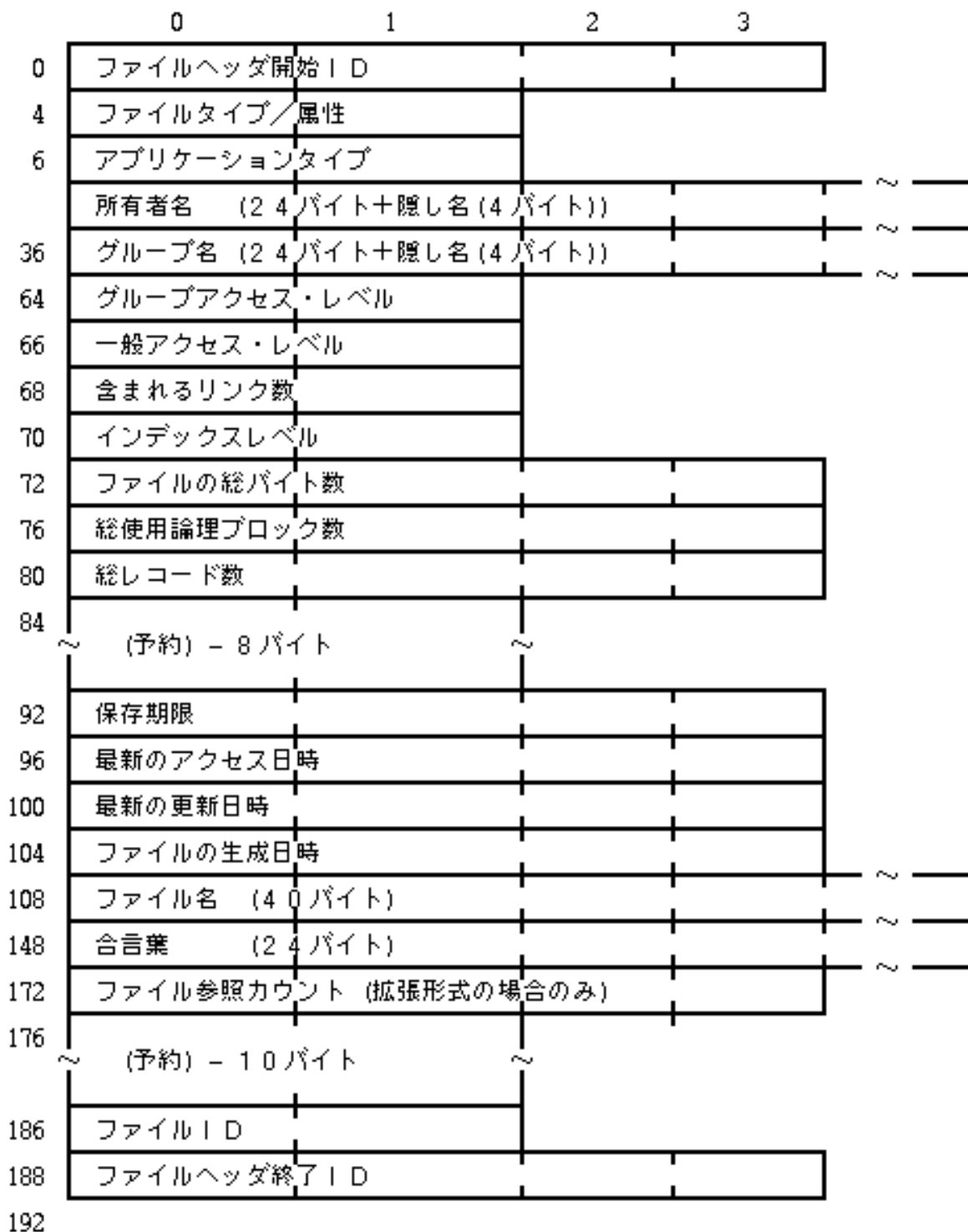


図 67: ファイルヘッダの構成

各データの内容を以下に示す。各データは上位バイトが先(若いアドレス)にくるビッグエンディアン形式である。

ファイルヘッダ開始ID:

0x54726f6e の固定パターン

ファイルタイプ/属性:

ファイルのタイプ、属性、および所有者アクセスモードを示す以下の形式のデータである。

TTTT xxxx BAPO xRWE

T: ファイルタイプ 0 リンクファイル

1 通常ファイル

## 2~ 予約

P: 削除不可属性

O: 変更(書込)不可属性

A: アプリケーション属性1

B: アプリケーション属性2

RWE: ファイル所有者のアクセスモード

x: 予約 (0)

リンクファイルの場合はファイルタイプ以外は意味を持たず0に固定される。

アプリケーションタイプ:

アプリケーションが設定するファイルのタイプ。

リンクファイルの場合、対象とするファイルのアプリケーションタイプが入る。

所有者名:

グループ名:

グループアクセスレベル:

一般アクセスレベル:

ファイルのアクセス管理情報であり、ファイルシステムのアクセス管理のレベルに従って設定される。リンクファイルの場合は意味を持たず、すべて0に固定される。

含まれるリンク数:

ファイルに含まれているリンクレコードの数を示す。リンクファイルの場合は意味を持たず、0に固定される。

インデックスレベル:

レコードインデックスの多重レベルを示す。リンクファイルの場合は意味を持たず、0に固定される。

ファイルの総バイト数:

ファイルに存在するレコードのレコード長の合計バイト数を示す。リンクレコードのサイズは含まない。リンクファイルの場合は意味を持たず、0に固定される。

総使用論理ブロック数:

ヘッダブロックを含んだ、ファイルに割り当てられている論理ブロックの総数を示す。リンクファイルの場合は1に固定される。

総レコード数:

ファイルに存在するレコードの総数を示す。リンクファイルの場合は意味を持たず、0に固定される。

保存期限:

ファイルの保存期限であり、アプリケーションにより使用される。ファイルの生成時には-1が設定される。リンクファイルの場合は-1のままで更新されることはない。

最新のアクセス日時:

ファイル内のレコードデータを参照した、あるいはファイル管理情報を更新した最新日時であり、1985年1月1日0時0分0秒 (GMT) からの秒数で示される。

ファイルの生成時には初期値として生成日時が入る。リンクファイルの場合は生成時の初期値のままで更新されることはない。

最新の更新日時:

ファイル内のレコードデータの最新更新日時であり、1985年1月1日0時0分0秒 (GMT) からの秒数で示される。

ファイルの生成時には初期値として生成日時が入る。

リンクファイルの場合は生成時の初期値のままで更新されることはない。

ファイルの生成日時:

ファイルの生成日時であり、1985年1月1日0時0分0秒 (GMT) からの秒数で示される。  
リンクファイルの場合はリンクファイル自身が生成された日時が入る。

ファイル名:

40 バイトのファイル名であり、40 バイトに満たない場合は0が詰められる。  
リンクファイルの場合はリンクファイル自身が生成された時点での対象とするファイル名が入る。

合言葉:

暗号化済みの24 バイトの合言葉であり、存在しない場合は0が詰められる。存在するか否かは実際には、先頭の1バイトが0か否かで判断される。ファイルの生成時には存在しないため0が詰められる。

ファイル参照カウント: 拡張形式の場合のみ使用される。

ファイル参照カウントが255以上になったとき、その参照カウント数を記録する。ファイル参照カウントが255未満のときは0を設定する。つまり、このフィールドは、0または255以上の値が設定され、1~254が設定されることはない。

ファイルID:

ファイルのファイルIDである。これは、ファイルシステムの故障等が起きた場合に、ファイルシステムを復旧するためにファイルヘッダ内に保持される。

ファイルヘッダ終了ID:

0x82dde96b の固定パターン

リンクファイルの場合は、ファイルヘッダに続くフラグメントテーブルの部分に以下に示す90 バイト対象ファイルの所在を示すファイル所在データが入る。各データは上位バイトが先 (若いアドレス) にくるビッグエンディアン形式である。

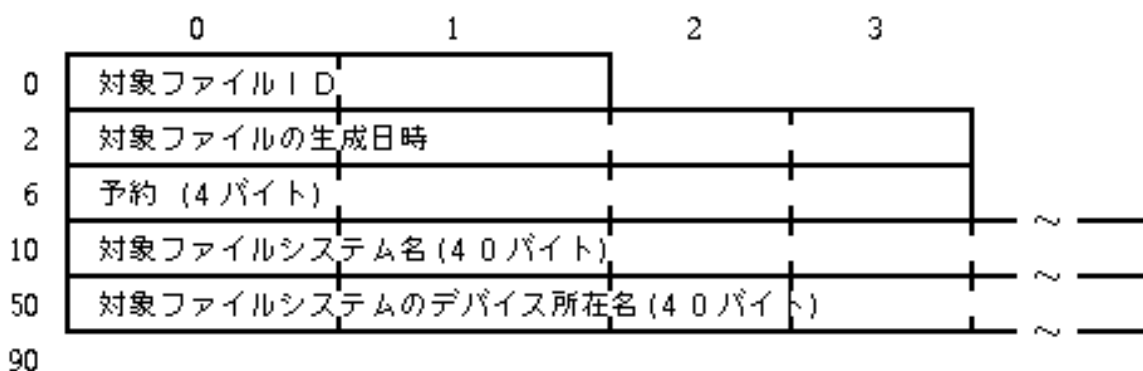


図 68 : ファイル所在データの構成

対象ファイルID:

対象とするファイルが存在するファイルシステムでのファイルID。

対象ファイルの生成日時:

対象とするファイルの生成日時。

対象ファイルシステム名:

リンクファイルが生成された時点で、対象ファイルが存在していたファイルシステムのファイルシステム名。

対象ファイルシステムのデバイス所在名:

リンクファイルが生成された時点で、対象ファイルが存在していたファイルシステムのデバイス所在名。



### 4.3.4 フラグメントテーブルの構造

フラグメントテーブルは、レコードの削除/サイズ縮小等によって発生した1つの論理ブロック内の断片的な空き領域(フラグメント)を登録するテーブルであり、レコードの追加/サイズ拡張で新たな領域が必要となった場合参照されるテーブルである。

このテーブルには現在そのファイルに対して割り当てられている論理ブロック中の空き領域(フラグメント)が最大32個まで登録される。フラグメントが32個以上となった場合は、もっともサイズの小さいフラグメントはテーブルから取り除かれる。

レコードの削除/サイズ縮小等が行なわれていない場合でも、通常は、最後に書き込んだブロックの残り領域が1つのフラグメントとして登録されることになる。

フラグメントテーブルは以下に示す構成をとり、各エントリは6バイトから構成される。未使用エントリは、フラグメントサイズ=0で示される。エントリは、必ずサイズの大きい順に並び、途中で未使用エントリが入ることはない。即ちフラグメントサイズ=0のエントリがあった場合は、それ以後のエントリはないものと見なされる。

リンクファイルの場合は、フラグメントテーブルの代わりにファイル所在データが入る。

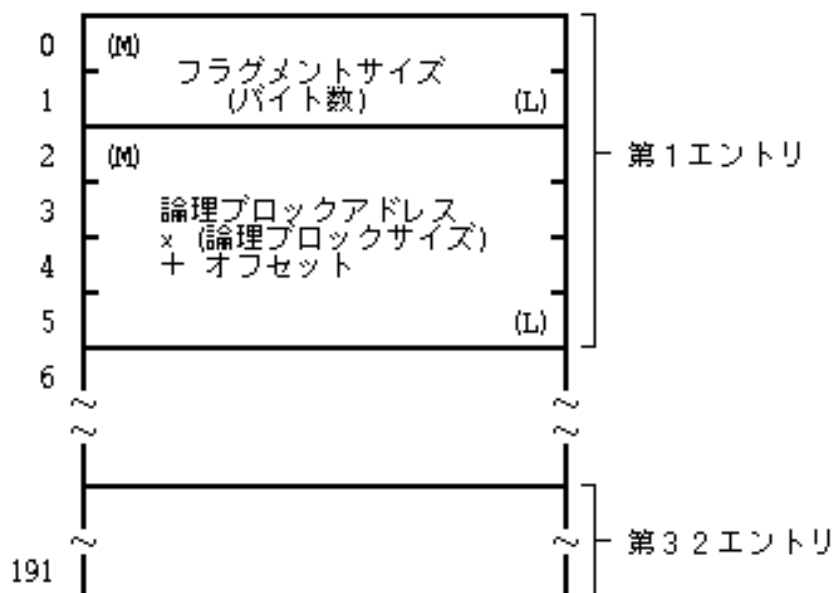


図 69 : フラグメントテーブルの構成

### 4.3.5 レコードインデックスの構成

レコードインデックスは各レコードの管理情報を保持している16バイトのインデックスである。インデックスレベル=0の場合は、ヘッダブロックに最大40個のレコードインデックスが入り、インデックスレベル<0の場合は、インデックスブロックに入る。

1つのレコードが複数の論理ブロックにまたがる場合は、通常のレコードインデックスの直後に接続インデックスと呼ぶ論理ブロックの割り当て情報を示すインデックスエントリが続く。この接続インデックスは内部的なものであり、レコードの数としてはカウントされない。

以下に各レコードインデックスの構成を示す。

|    |                      |             |                   |     |
|----|----------------------|-------------|-------------------|-----|
| 0  | 0 0 0 0 0 0 0 0      | 連続カウント (>0) | 0 0 0 0 0 0 0 0   | 0   |
| 1  | レコードタイプ              | (M)         | 1 0 0 0 0 0 0 0   | 0   |
| 2  | (M)                  | 論理ブロック      | (M)               | 未使用 |
| 3  | レコード<br>サブタイプ (L)    | アドレス# 1 (L) | レコード<br>サブタイプ (L) |     |
| 4  | (予約)                 | 連続カウント      | (M)               |     |
| 6  | (M)                  | 論理ブロック      | (M)               |     |
| 7  | 開始オフセット<br>(バイト) (L) | アドレス# 2 (L) | 属性データ 1 (L)       |     |
| 8  | (M)                  | 連続カウント      | (M)               |     |
| 9  | レコードサイズ<br>(バイト数)    | (M)         | 属性データ 2 (L)       |     |
| 10 | (L)                  | 論理ブロック      | (M)               |     |
| 11 | (L)                  | アドレス# 3 (L) | 属性データ 3 (L)       |     |
| 12 | 連続カウント (>0)          | 連続カウント      | (M)               |     |
| 13 | (M)                  | (M)         | 属性データ 4 (L)       |     |
| 14 | 論理ブロック               | 論理ブロック      | (M)               |     |
| 15 | アドレス (L)             | アドレス# 4 (L) | 属性データ 5 (L)       |     |

通常インデックス      接続インデックス      リンクインデックス      未使用インデックス

図 70 : レコードインデックスの構成

各データの内容を以下に示す。各データは上位バイトが先(若いアドレス)にくるビッグエンディアン形式である。

レコードタイプ:

レコードのタイプ(0 ~ 31)を示すデータで、以下の形式を持つ。

100T TTTT

T:レコードタイプ0 リンクレコード  
1~ その他のレコード

リンクレコードの場合のレコードインデックスの形式は特殊なものとなる。

レコードサブタイプ:

レコードのサブタイプを示すデータであり、その内容はレコードタイプにより規定される。

開始オフセット:

レコードが論理ブロックの先頭から始まっていない場合に、論理ブロックの先頭からのバイト数を示す、0~(論理ブロックサイズ-1)の値。

レコードサイズ:

レコードの長さをバイト数で表わしたもの。

連続カウント:

論理ブロックアドレスから連続して何ブロック割り当てられているかを示す0 ~ 255の値で、0は論理ブロックが割り当てられていないことを意味する。

論理ブロックアドレスを X とした場合、以下の論理ブロックが割り当てられていることを示す。

割り当てられている論理ブロック

連続カウント 1 X

連続カウント 2 X, X+1

連続カウント n X, X+1, ... X+n-1

論理ブロックアドレス:

レコードに割り当てられている 3 バイトの論理ブロックアドレスを示す。

ファイルID:

リンクが示すファイルのファイルID。

属性データ:

リンクレコードの属性データ(1~5)。

レコードインデックスの形式は、先頭の2バイトの値により決められる。

第1バイト 第2バイト

接続インデックス 0 any

通常インデックス =0 0

リンクインデックス =0 =0x80

未使用インデックス =0 =0

### 4.3.6 間接インデックスの構成

間接インデックスは、インデックスレベル >0 の場合に使用され、レコードインデックスを含む論理ブロックを示す以下の 8 バイトのエントリである (1 論理ブロックに 128 個、ヘッダブロック内は 80 個)。

最初の 4 バイトのデータは、間接インデックスで示される間接ブロックまたはインデックスブロックが含む有効なレコード数 (接続インデックス、未使用インデックスを除いたレコードインデックスの数) を示す。これは、レコードの位置によるシークを行なう場合に使用される。

論理ブロックアドレス =0 の間接インデックスは未使用エントリと見なされ、未使用インデックスは、論理ブロック内の任意の位置に存在してよい。なお、間接ブロック中で使用されていない部分は、すべて未使用エントリで埋められていなければならない。

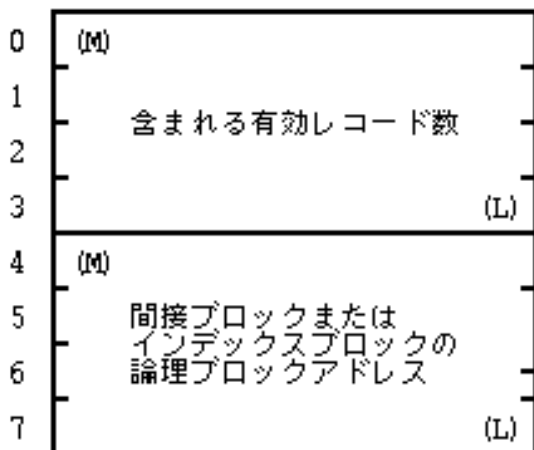


図 71 : 間接インデックスの構成

レコードインデックスの最大数はインデックスレベルの値により以下のようになる。

インデックスレベル 最大レコードインデックス

|   |        |                 |
|---|--------|-----------------|
| 0 | 40     |                 |
| 1 | 5120   | (64 × 80)       |
| 2 | 655360 | (64 × 128 × 80) |

### 4.3.7 レコードインデックスの実際

レコードインデックスとデータブロックとの対応の例を以下に示す。

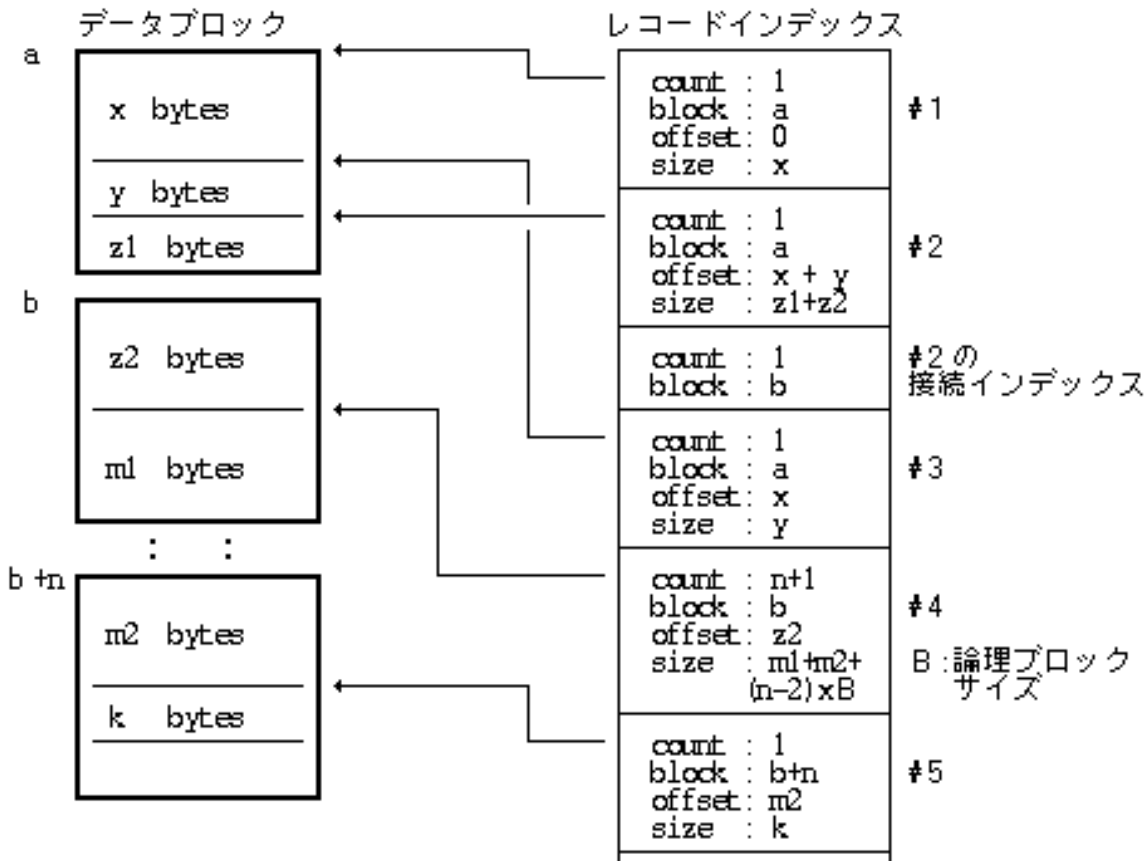


図 72 : レコードインデックスの例

レコードインデックスは、ヘッダブロックまたはインデックスブロックの中で、レコード順に並べられているが途中(最初、最後も含む)に任意個の未使用インデックスが入っていてもよい。ただし、接続インデックスは、通常インデックスの直後から隙間無く連続して入っていないといけない。接続インデックスが複数の論理ブロックにまたがる場合でも、論理ブロックの切れ目(最後、最初)に未使用エントリが入ってはいけない。

なお、インデックスブロック中で使用されていない部分は、すべて未使用インデックスで埋められていなければならない。

インデックスレベル=1で、以下の状態の場合を考える。

間接インデックス インデックスブロック

|     |                          |
|-----|--------------------------|
| 53個 | A:64個のエントリ(内、接続インデックス11) |
| 42個 | B:50個のエントリ(内、接続インデックス8)  |
| 20個 | C:20個のエントリ(内、接続インデックス0)  |
| 未使用 |                          |

この例で、110番目のレコードをアクセスしたい場合は、 $110 - 53 - 42 = 15$ から、インデックスブロックCの15番目のレコードインデックスを使用することになる。

また、10番目のレコードの直前にレコードを追加した場合は、インデックスレコードAを2つに分割（通常は2等分）しインデックスブロックA'とA''を生成して、間接インデックスに登録することになる。レコードの追加後の状態は以下に示す通りとなる。

|                     |                           |
|---------------------|---------------------------|
| 間接インデックス インデックスブロック |                           |
| 26個                 | A':33個のエントリ(内、接続インデックス7)  |
| 28個                 | A'':32個のエントリ(内、接続インデックス4) |
| 42個                 | B:50個のエントリ(内、接続インデックス8)   |
| 20個                 | C:20個のエントリ(内、接続インデックス0)   |
| 未使用                 |                           |

### 4.3.8 データブロックの構成

データブロックにはレコードの実際のデータが、レコードインデックスで示されたブロック、オフセットに入るが、1つの論理ブロック内のフラグメントを管理するために、1つの論理ブロック内のレコードとレコードの境界には空き領域を示す2つの区切りコードを格納する必要がある。

区切りコードは空き領域のサイズをバイト数で示した2バイトのコードであり、空き領域の先頭と最後にそれぞれ入る。レコードとレコードの間が隙間無く詰まっている場合は値0の区切りコードが連続して2つ入ることになる。論理ブロックの先頭、および最後の区切りレコードは省略される。

以下に1つの論理ブロック内の区切りコードを示す。図で、 $\rightarrow$  はレコードインデックス内のオフセットを意味し、 $\Rightarrow$  はフラグメントテーブル内のオフセットを示す。いずれの場合も区切りコードの直後の位置を示す。

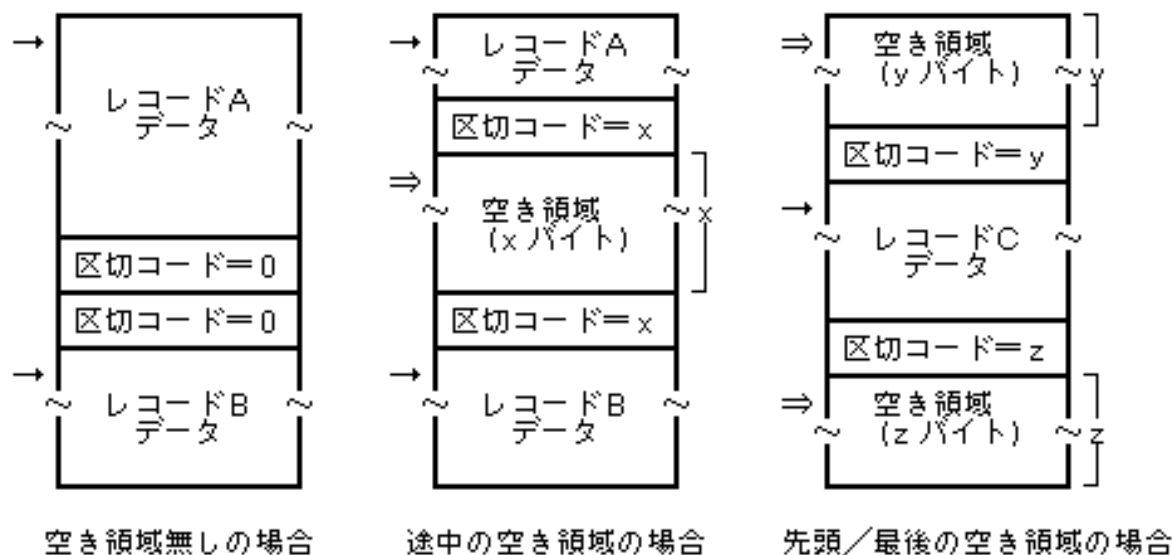


図 73: レコード間の区切りコード

1つの論理ブロック内の隣接する空き領域は常にマージされ1つの空き領域にまとめられるため、2つ以上の空き領域が連続して存在することはない。

レコードの生成、追加、削除、サイズ変更、再配置等の操作では、区切りコードを正しく設定し直す必要があり、その結果、論理ブロック全体が空き領域となった場合はそのブロックを解放することになる。

# 付録 A 準標準フロッピーディスク形式

TRON 標準フロッピーディスク形式では、異なるプロセッサ間を含めたフロッピーディスクの交換を完全に保証するために 8 ビット以上のビット長を持つデータのバイトオーダーは、上位バイトが先にくるビッグエンディアン形式に統一している。

しかしながら、この形式はリトルエンディアン形式の CPU 上で動作する BTRON では負担となる場合もあるため、これを救済するために、リトルエンディアンのデータ形式を持つフロッピーディスク形式を、準標準フロッピーディスク形式として規定する。

準標準フロッピーディスク形式は、TRON 標準フロッピーディスク形式に対して以下に示す変更/制限を行なったものである。

1. 使用する文字コードは 2 バイトの固定長とし、かつ第 2 バイトが先にくる形式とする。すなわち、文字コードはリトルエンディアンの 16 ビットデータとみなす。
2. 2 バイト以上のデータのバイトオーダーは、下位バイトが先にくるリトルエンディアン形式とする。文字列データには、1. で記述した 2 バイト固定長の文字コードのみを使用する。

システムヘッダの TRON ディスク ID、およびディスク形式 ID のバイトオーダーにより、TRON 標準フロッピーディスク形式か、準標準フロッピーディスク形式かの識別が行なわれる。すなわち、BTRON 標準の場合は、0x42, 0xFE, 0x64, 0x00 であり準標準の場合は、0xFE, 0x42, 0x00, 0x64 となる。

準標準フロッピーディスク形式に基づいたシステムは、以下の点を遵守しなくてはならない。

- 何らかの方法により、標準フロッピーディスク形式との双方向の変換機能を有していること。
- 準標準フロッピーディスク形式である旨を明記すること。
- 標準フロッピーディスク形式との変換方法を明記すること。

本仕様書の中で、ビッグエンディアン形式によって記述された図は、準標準フロッピーディスクにおいては、M(Most Significant Bit) と L(Least Significant Bit) の表記が逆になる。ここではそのすべてを掲げないが、その一例として、準標準フロッピーディスク形式によるレコードインデックスの構成を示す。

|    |                 |             |                 |       |
|----|-----------------|-------------|-----------------|-------|
| 0  | 0 0 0 0 0 0 0 0 | 連続カウンタ (>0) | 0 0 0 0 0 0 0 0 | 0     |
| 1  | レコードタイプ         | (L)         | 1 0 0 0 0 0 0 0 | 0     |
| 2  | レコード (L)        | 論理ブロック      | レコード (L)        | 未 使 用 |
| 3  | (M) サブタイプ       | アドレス # 1    | (M) サブタイプ       |       |
| 4  | (予約)            | 連続カウンタ      | ファイル ID (L)     |       |
|    |                 | (L)         | (M)             |       |
| 6  | 開始オフセット (L)     | 論理ブロック      | 属性データ 1 (L)     |       |
| 7  | (M) (バイト)       | アドレス # 2    | (M)             |       |
| 8  | (L)             | 連続カウンタ      | 属性データ 2 (L)     |       |
|    |                 | (L)         | (M)             |       |
| 10 | レコードサイズ (バイト数)  | 論理ブロック      | 属性データ 3 (L)     |       |
| 11 | (M)             | アドレス # 3    | (M)             |       |
| 12 | 連続カウンタ (>0)     | 連続カウンタ      | 属性データ 4 (L)     |       |
| 13 | (L)             | (L)         | (M)             |       |
| 14 | 論理ブロック          | 論理ブロック      | 属性データ 5 (L)     |       |
| 15 | アドレス (M)        | アドレス # 4    | (M)             |       |

通常インデックス    接続インデックス    リンクインデックス    未使用インデックス

図 74: 準標準フロッピーディスク形式によるレコードインデックスの構成

また、システムブロックにおけるビットマップ構成は、以下のようなになる。

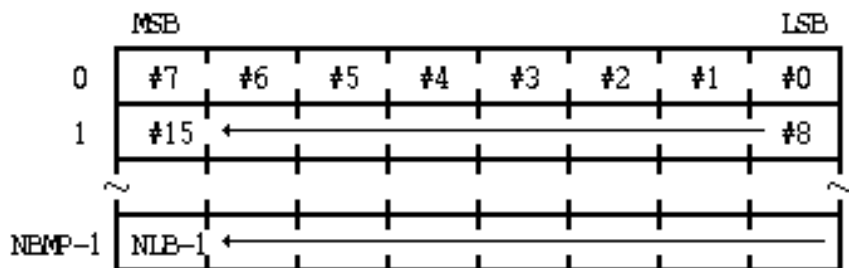


図 75: 準標準フロッピーディスク形式におけるビットマップの構成

## 付録 B フロッピーディスク形式の実現例

TRON標準フロッピーディスク形式の仕様に基づいて実現されたフロッピーディスクのいくつかの例をあげる。

| 名称                       | 2HD<br>(big-endian) | 2HD<br>(little-endian) | 2DD        | 2HC        |
|--------------------------|---------------------|------------------------|------------|------------|
| [物理フォーマット]               |                     |                        |            |            |
| トラック/サイド数                | 77                  | 77                     | 80         | 80         |
| セクタ/トラック数                | 8                   | 8                      | 9          | 15         |
| 物理ブロック数                  | 1232                | 1232                   | 1440       | 2400       |
| 物理ブロックサイズ<br>(バイト)       | 1024                | 1024                   | 512        | 512        |
| [ブート領域]                  |                     |                        |            |            |
| 物理ブロック数                  | 1                   | 2                      | 2          | 2          |
| 論理ブロック数                  | 1                   | 1                      | 1          | 1          |
| ブート領域総バイト数               | 1024                | 2048                   | 1024       | 1024       |
| [物理ブロック#Nに対応する論理ブロック]    |                     |                        |            |            |
| 物理ブロック#0                 | 論理ブロック#0            | 論理ブロック#0               | 論理ブロック#0   | 論理ブロック#0   |
| 物理ブロック#1                 | 論理ブロック#1            | 論理ブロック#0               | 論理ブロック#0   | 論理ブロック#0   |
| 物理ブロック#2                 | 論理ブロック#2            | 論理ブロック#1               | 論理ブロック#1   | 論理ブロック#1   |
| 物理ブロック#3                 | 論理ブロック#3            | 論理ブロック#2               | 論理ブロック#1   | 論理ブロック#1   |
| 物理ブロック#4                 | 論理ブロック#4            | 論理ブロック#3               | 論理ブロック#2   | 論理ブロック#2   |
| 物理ブロック#N                 | 論理ブロック#N            | 論理ブロック#N-1             | 論理ブロック#N/2 | 論理ブロック#N/2 |
| 論理ブロックサイズ<br>(バイト)       | 1024                | 1024                   | 1024       | 1024       |
| 論理ブロック当りの物理ブロック数         | 1                   | 1                      | 2          | 2          |
| [論理フォーマット]               |                     |                        |            |            |
| 論理ブロック数                  | 1232                | 1231                   | 720        | 1200       |
| システムブロック数 (SSYS)         | 1                   | 1                      | 1          | 1          |
| ファイルIDテーブルブロック数 (SFIDT)  | 2                   | 3                      | 2          | 2          |
| ファイル短縮名テーブルブロック数 (SFNMT) | 2                   | 3                      | 2          | 2          |
| 使用ビットマップのバイト数            | 154                 | 154                    | 90         | 150        |
| 最大ファイル数                  | 512                 | 525                    | 307        | 512        |

図 76: フロッピーディスク形式の実現例

[この章の目次にもどる](#)

[前頁:3.6 図形描画セグメントにもどる](#)



[この章の目次にもどる](#)

[前頁:1.10 システム管理機能にもどる](#)

[次頁:2.1 はじめににすすむ](#)

---

# 第2章 ディスプレイプリミティブ

---

## [第2章 ディスプレイプリミティブ](#)

### [2.1 はじめに](#)

#### [2.1.1 ディスプレイプリミティブの概要](#)

### [2.2 基本概念](#)

#### [2.2.1 座標平面 / 点 / 長方形](#)

#### [2.2.2 ビットマップ](#)

#### [2.2.3 カラー表現](#)

#### [2.2.4 描画環境](#)

### [2.3 基本関数](#)

#### [2.3.1 描画環境の生成 / 削除](#)

#### [2.3.2 カラーマップの操作](#)

#### [2.3.3 描画環境レコードのアクセス](#)

#### [2.3.4 座標変換](#)

### [2.4 図形描画関数](#)

#### [2.4.1 直線 / 点の描画](#)

#### [2.4.2 長方形の描画](#)

#### [2.4.3 角丸長方形の描画](#)

#### [2.4.4 楕円\(円\)の描画](#)

#### [2.4.5 弧 / 扇形 / 弓形の描画](#)

#### [2.4.6 多角形\(直線列\)の描画](#)

#### [2.4.7 任意図形の描画](#)

#### [2.4.8 曲線の描画](#)

#### [2.4.9 ビットマップの操作](#)

#### [2.4.10 ピッキング処理](#)

### [2.5 文字\(列\)描画関数](#)

#### [2.5.1 描画環境レコードアクセス](#)

#### [2.5.2 文字\(列\)チェック関数](#)

#### [2.5.3 文字\(列\)の描画関数](#)

### [2.6 ポインタ制御関数](#)

#### [2.6.1 ポインタ](#)

#### [2.6.2 ポインタの状態 / 形状](#)

#### [2.6.3 ポインタの制御関数](#)

#### [標準ポインタ形状\(参考\)](#)

---

[この章の目次にもどる](#)

[前頁:1.10 システム管理機能にもどる](#)

[次頁:2.1 はじめににすすむ](#)

# 第2編 OS 仕様 目次

---

## [第1章 周辺核](#)

### [1.1 プロセス / タスク管理](#)

#### [1.1.1 プロセス / タスク](#)

#### [1.1.2 プロセス / タスクの状態](#)

#### [1.1.3 プロセス / タスクの優先度とスケジューリング](#)

#### [1.1.4 プロセスの実行環境](#)

#### [1.1.5 ユーザ情報](#)

#### [1.1.6 プロセスの生成](#)

#### [1.1.7 プロセス / タスクの実行](#)

#### [1.1.8 データ / 定数の定義](#)

#### [1.1.9 システムコール](#)

### [1.2 メッセージ管理](#)

#### [1.2.1 プロセス間メッセージ](#)

#### [1.2.2 メッセージの種類](#)

#### [1.2.3 メッセージハンドラ](#)

#### [1.2.4 データ / 定数の定義](#)

#### [1.2.5 システムコール](#)

### [1.3 プロセス / タスク間同期通信管理](#)

#### [1.3.1 データ / 定数の定義](#)

#### [1.3.2 システムコール](#)

### [1.4 グローバル名管理](#)

#### [1.4.1 グローバル名データ](#)

#### [1.4.2 データ / 定数の定義](#)

#### [1.4.3 システムコール](#)

### [1.5 メモリ管理](#)

#### [1.5.1 メモリ管理機能の概要](#)

#### [1.5.2 データ / 定数の定義](#)

#### [1.5.3 システムコール](#)

#### [1.5.4 ライブラリ](#)

### [1.6 ファイル管理](#)

#### [1.6.1 ファイル管理機能の概要](#)

#### [1.6.2 ファイルシステム構成](#)

#### [1.6.3 ファイルの構成](#)

#### [1.6.4 ファイルのアクセス管理](#)

- [1.6.5 ファイルの操作](#)
- [1.6.6 ファイルの詳細情報](#)
- [1.6.7 データ / 定数の定義](#)
- [1.6.8 システムコール](#)

## [1.7 イベント管理](#)

- [1.7.1 イベント管理機能の概要](#)
- [1.7.2 イベント](#)
- [1.7.3 キーボード](#)
- [1.7.4 ポインティングデバイス](#)
- [1.7.5 データ/定数の定義](#)
- [1.7.6 システムコール](#)

## [1.8 デバイス管理](#)

- [1.8.1 デバイス管理機能の概要](#)
- [1.8.2 デバイス](#)
- [1.8.3 データ / 定数の定義](#)
- [1.8.4 システムコール](#)

## [1.9 時計管理](#)

- [1.9.1 時計管理機能の概要](#)
- [1.9.2 データ / 定数の定義](#)
- [1.9.3 システムコール](#)

## [1.10 システム管理機能](#)

- [1.10.1 データ / 定数の定義](#)
- [1.10.2 システムコール](#)

# [第2章 ディスプレイプリミティブ](#)

## [2.1 はじめに](#)

- [2.1.1 ディスプレイプリミティブの概要](#)

## [2.2 基本概念](#)

- [2.2.1 座標平面 / 点 / 長方形](#)
- [2.2.2 ビットマップ](#)
- [2.2.3 カラー表現](#)
- [2.2.4 描画環境](#)

## [2.3 基本関数](#)

- [2.3.1 描画環境の生成 / 削除](#)
- [2.3.2 カラーマップの操作](#)
- [2.3.3 描画環境レコードのアクセス](#)
- [2.3.4 座標変換](#)

## [2.4 図形描画関数](#)

- [2.4.1 直線 / 点の描画](#)
- [2.4.2 長方形の描画](#)

- [2.4.3 角丸長方形の描画](#)
- [2.4.4 楕円\(円\)の描画](#)
- [2.4.5 弧 / 扇形 / 弓形の描画](#)
- [2.4.6 多角形\(直線列\)の描画](#)
- [2.4.7 任意図形の描画](#)
- [2.4.8 曲線の描画](#)
- [2.4.9 ビットマップの操作](#)
- [2.4.10 ピッキング処理](#)

## [2.5 文字\(列\)描画関数](#)

- [2.5.1 描画環境レコードアクセス](#)
- [2.5.2 文字\(列\)チェック関数](#)
- [2.5.3 文字\(列\)の描画関数](#)

## [2.6 ポインタ制御関数](#)

- [2.6.1 ポインタ](#)
- [2.6.2 ポインタの状態 / 形状](#)
- [2.6.3 ポインタの制御関数](#)

### [標準ポインタ形状\(参考\)](#)

## [第3章 外殻](#)

### [3.1 ウィンドウマネージャ](#)

- [3.1.1 ウィンドウマネージャの機能](#)
  - [3.1.1.1 概要](#)
  - [3.1.1.2 ウィンドウ](#)
  - [3.1.1.3 ウィンドウの表示](#)
  - [3.1.1.4 ウィンドウの操作](#)
- [3.1.2 ウィンドウの管理](#)
  - [3.1.2.1 ウィンドウとプロセス](#)
  - [3.1.2.2 ウィンドウイベント](#)
  - [3.1.2.3 ウィンドウの再表示処理](#)
  - [3.1.2.4 ウィンドウの表示管理](#)
  - [3.1.2.5 ウィンドウの描画環境](#)
  - [3.1.2.6 ウィンドウ情報レコード](#)
- [3.1.3 データ/定数の定義](#)
- [3.1.4 ウィンドウマネージャの関数](#)

### [3.2 メニューマネージャ](#)

- [3.2.1 メニューマネージャの機能](#)
  - [3.2.1.1 概要](#)
  - [3.2.1.2 標準メニュー](#)
  - [3.2.1.3 標準メニューの操作](#)
  - [3.2.1.4 標準メニューのデータ構造](#)
  - [3.2.1.5 汎用メニュー](#)
  - [3.2.1.6 汎用メニューの操作](#)

[3.2.1.7 汎用メニューのデータ構造](#)

[3.2.2 データ / 定数の定義](#)

[3.2.3 メニューマネージャの関数](#)

### [3.3 パーツマネージャ](#)

[3.3.1 パーツマネージャの機能](#)

[3.3.1.1 概要](#)

[3.3.1.2 パーツの種類](#)

[3.3.1.3 パーツのデータ構造](#)

[3.3.1.4 パーツの状態](#)

[3.3.2 パーツの種類](#)

[3.3.2.1 テキストボックス](#)

[3.3.2.2 シークレットテキストボックス](#)

[3.3.2.3 数値ボックス](#)

[3.3.2.4 シリアルボックス](#)

[3.3.2.5 テキストオルタネートスイッチ](#)

[3.3.2.6 テキストモーメンタリスイッチ](#)

[3.3.2.7 ピクトグラムオルタネートスイッチ](#)

[3.3.2.8 ピクトグラムモーメンタリスイッチ](#)

[3.3.2.9 スイッチセレクタ](#)

[3.3.2.10 スクロールセレクタ](#)

[3.3.2.11 ボリューム](#)

[3.3.3 データ / 定数の定義](#)

[3.3.4 パーツマネージャの関数](#)

### [3.4 パネルマネージャ](#)

[3.4.1 パネルマネージャの機能](#)

[3.4.1.1 概要](#)

[3.4.1.2 パネル](#)

[3.4.1.3 システムメッセージパネル](#)

[3.4.1.4 パネルの表示形状](#)

[3.4.1.5 パネルのデータ構造](#)

[3.4.1.6 パネルの描画環境](#)

[3.4.1.7 パネルの動作](#)

[3.4.2 データ / 定数の定義](#)

[3.4.3 パネルマネージャの関数](#)

### [3.5 トレーマネージャ](#)

[3.5.1 トレーマネージャの機能](#)

[3.5.1.1 概要](#)

[3.5.1.2 トレー](#)

[3.5.1.3 トレーの格納データ](#)

[3.5.1.4 トレーの操作](#)

[3.5.2 データ / 定数の定義](#)

[3.5.3 トレーマネージャの関数](#)

## 3.6 データマネージャ

### 3.6.1 データマネージャの機能

#### 3.6.1.1 概要

#### 3.6.1.2 データタイプID / データ番号

#### 3.6.1.3 データマネージャの動作

#### 3.6.1.4 データボックスの構造

### 3.6.2 データ / 定数の定義

#### 3.6.2.1 データ / 定数の定義

#### 3.6.2.2 標準データ形式

### 3.6.3 データマネージャの関数

## 3.7 テキスト入力プリミティブ

### 3.7.1 テキスト入力プリミティブの機能

#### 3.7.1.1 概要

#### 3.7.1.2 テキスト入力プリミティブ

#### 3.7.1.3 テキスト入力ポート

#### 3.7.1.4 カレット表示

### 3.7.2 データタイプ/定数の定義

### 3.7.3 テキスト入力プリミティブの関数

## 3.8 実身 / 仮身マネージャ

### 3.8.1 実身 / 仮身マネージャの機能

#### 3.8.1.1 概要

#### 3.8.1.2 仮身 / 実身 / 虚身

#### 3.8.1.3 仮身の表示

#### 3.8.1.4 仮身の操作

### 3.8.2 仮身の移動 / 複写

#### 3.8.2.1 付箋

### 3.8.3 実身 / 仮身マネージャの詳細

#### 3.8.3.1 仮身

#### 3.8.3.2 付箋

#### 3.8.3.3 アプリケーションプログラム管理

#### 3.8.3.4 アプリケーションプログラムの起動

#### 3.8.3.5 メニュー管理

### 3.8.4 データ / 定数の定義

### 3.8.5 実身 / 仮身マネージャの関数

### 3.8.6 アプリケーション支援関数

## 3.9 フォントマネージャ

### 3.9.1 フォントマネージャの機能

#### 3.9.1.1 概要

#### 3.9.1.2 フォント

#### 3.9.1.3 フォントデータ

#### 3.9.1.4 フォントのアクセス

### 3.9.2 データ / 定数の定義

[3.9.3 フォントマネージャの関数](#)

[3.9.4 標準フォントデータ形式](#)

[3.10 TCP/IPマネージャ](#)

[3.10.1 構成](#)

[3.10.2 TCP/IP](#)

[3.10.3 エラーコード](#)

[3.10.4 システムコール](#)

[3.11 印刷マネージャ](#)

[3.11.1 印刷マネージャの機能](#)

[3.11.2 印刷管理](#)

[3.11.2.1 レイアウト用紙と印刷用紙](#)

[3.11.2.2 印刷管理の機能](#)

[3.11.2.3 印刷パラメータ](#)

[3.11.3 スプール管理](#)

[3.11.3.1 スプール管理の機能](#)

[3.11.3.2 印刷登録状態](#)

[3.11.3.3 印刷終了メッセージ](#)

[3.11.4 プリンタ管理](#)

[3.11.4.1 プリンタ管理の機能](#)

[3.11.4.2 プリンタ構成情報](#)

[3.11.5 データ / 定数の定義](#)

[3.11.6 印刷マネージャの関数](#)

---

[全目次にもどる](#)

[次頁: 図版目次にすすむ](#)



## 第2編 OS 仕様 図版目次

---

- [図 1 プロセスの基本メモリモデル](#)
- [図 2 プロセス/タスクの基本状態遷移](#)
- [図 3 プロセスの状態ワード](#)
- [図 4 main\(\)での引数の構造](#)
- [図 5 グローバル名データの操作](#)
- [図 6 ファイルとリンク](#)
- [図 7 ファイルシステムの構成](#)
- [図 8 ファイルシステムの接続](#)
- [図 9 間接リンク/リンクファイル](#)
- [図 10 パス名の出現順](#)
- [図 11 レコード番号/現在レコード](#)
- [図 12 ファイルのオープンモード](#)
- [図 13 キー状態 \(KeyMap\)](#)
- [図 14 デバイス管理機能の位置付け](#)
- [図 15 論理デバイス名](#)
- [図 16 デバイスのアクセスモード](#)
- [図 17 座標系とピクセル](#)
- [図 18 ビットマップ上の長方形](#)
- [図 19 ビットマップの構造](#)
- [図 20 ビットマップの座標定義](#)
- [図 21 ビットマップのピクセル値](#)
- [図 22 ビットマップの座標定義](#)
- [図 23 カラー指定](#)
- [図 24 任意領域の例 - 1](#)
- [図 25 任意領域の例 - 2](#)
- [図 26 任意領域の例 - 3](#)
- [図 27 クリッピング領域](#)
- [図 28 描画パターンの配置](#)
- [図 29 パターンマスクの構成](#)
- [図 30 ビットマップでのパターン指定](#)
- [図 31 線マスクパターン](#)
- [図 32 文字描画カラー](#)
- [図 33 文字描画方向/文字間隔](#)
- [図 34 ビットマップ境界の移動](#)
- [図 35 座標系の移動](#)
- [図 36 直線の描画](#)
- [図 37 長方形の描画](#)

- [図 38 楕円\(円\)の描画](#)
- [図 39 多角形の描画](#)
- [図 40 ビットマップの操作関数](#)
- [図 41 ビットマップの圧縮](#)
- [図 42 文字イメージの描画](#)
- [図 43 ポインタイメージの例](#)
- [図 44 標準ポインタ形状](#)
- [図 45 入力受付状態の遷移](#)
- [図 46 ウィンドウの親子関係](#)
- [図 47 ウィンドウの標準的な境界線](#)
- [図 48 変形不可のウィンドウの標準的な形状](#)
- [図 49 変形可能なウィンドウの標準的な形状](#)
- [図 50 変形可能で左スクロールバーのないウィンドウの標準的な形状](#)
- [図 51 タイトルバーのないウィンドウの標準的な形状](#)
- [図 52 入力受付状態の標準的なタイトルバー表示](#)
- [図 53 ウィンドウの属性](#)
- [図 54 frameの内容](#)
- [図 55 アクティブプロセスの遷移](#)
- [図 56 ウィンドウイベント一覧](#)
- [図 57 ウィンドウイベントの処理](#)
- [図 58 フロントエンド機能の基本構造](#)
- [図 59 フロントエンドプロセスへのイベント流れ\(1\)](#)
- [図 60 フロントエンドプロセスへのイベント流れ\(2\)](#)
- [図 61 前面ウィンドウの再表示処理](#)
- [図 62 表示管理メニュー](#)
- [図 63 ウィンドウ拡大時の作業領域の座標](#)
- [図 64 W\\_MOVE](#)
- [図 65 W\\_MOVEC](#)
- [図 66 W\\_HOLD](#)
- [図 67 標準メニューの例](#)
- [図 68 標準メニューのフレーム属性](#)
- [図 69 属性コード](#)
- [図 70 メニュー項目の例](#)
- [図 71 不能項目 / 選択フラグ](#)
- [図 72 項目の選択番号](#)
- [図 73 汎用メニューの例](#)
- [図 74 汎用メニューのフレーム属性](#)
- [図 75 汎用メニューの不能項目 / 選択フラグ](#)
- [図 76 汎用メニューの項目領域](#)
- [図 77 mchg\\_atr](#)
- [図 78 mchg\\_gat](#)
- [図 79 パーツのタイプ / 属性 / 状態](#)
- [図 80 属性コード](#)

- [図 81 テキストボックス](#)
- [図 82 シークレットテキストボックス](#)
- [図 83 数値ボックス](#)
- [図 84 数値の形式](#)
- [図 85 シリアルボックス](#)
- [図 86 テキストオルタネートスイッチ](#)
- [図 87 インジケータの反転表示](#)
- [図 88 テキストモーメンタリスイッチ](#)
- [図 89 ピクトグラムオルタネートスイッチ](#)
- [図 90 ピクトグラムモーメンタリスイッチ](#)
- [図 91 スイッチセレクト](#)
- [図 92 スクロールセレクト](#)
- [図 93 ノブの最小幅](#)
- [図 94 ボリューム](#)
- [図 95 ボリュームの値](#)
- [図 96 システムメッセージパネル](#)
- [図 97 パネルの一般的な境界線](#)
- [図 98 パネルの一般的な形状](#)
- [図 99 パネルの境界線の属性](#)
- [図 100 パネルの内枠の属性](#)
- [図 101 パネルの描画環境](#)
- [図 102 トレー](#)
- [図 103 トレーレコードの構造](#)
- [図 104 トレー / 一時トレーに格納するデータ指定](#)
- [図 105 トレー / 一時トレーから取り出すデータ形式\(\(1\)全体取り出し\)](#)
- [図 106 トレー / 一時トレーから取り出すデータ形式\(\(2\)ヘッダ取り出し\)](#)
- [図 107 トレー / 一時トレーから取り出すデータ形式\(\(3\)指定レコードデータ取り出し\)](#)
- [図 108 全体構造](#)
- [図 109 パターンイメージ](#)
- [図 110 図形ビットマップイメージ](#)
- [図 111 テキスト入力プリミティブのモデル](#)
- [図 112 テキスト入力レコードの内容](#)
- [図 113 通常カレットの標準形状](#)
- [図 114 仮身 / 実身 / 虚身](#)
- [図 115 仮身の表示](#)
- [図 116 タイトルバー](#)
- [図 117 タイトルバーの表示例](#)
- [図 118 開いた虚身の表示](#)
- [図 119 仮身内でのハンドルの位置](#)
- [図 120 ハンドルによる変形](#)
- [図 121 新版作成](#)
- [図 122 付箋の表示](#)
- [図 123 仮身の属性 / 状態](#)

- [図 124 実身のアプリケーションタイプ](#)
  - [図 125 フォントマネージャとディスプレイプリミティブ](#)
  - [図 126 フォントクラス](#)
  - [図 127 フォント属性](#)
  - [図 128 フォントデータの全体構造](#)
  - [図 129 フォントの文字イメージ - 1](#)
  - [図 130 文字高さ と 文字幅](#)
  - [図 131 フォントの文字イメージ2](#)
  - [図 132 文字の描画位置\(横書き\)](#)
  - [図 133 文字の描画位置\(縦書き\)](#)
  - [図 134 イメージビットマップ上の文字イメージ](#)
  - [図 135 連続文字イメージ](#)
  - [図 136 フォントデータの全体構造](#)
  - [図 137 標準ドット形式\(固定イメージサイズ\)](#)
  - [図 138 標準文字幅データの形式](#)
  - [図 139 イメージ幅指定付き文字幅データの形式](#)
  - [図 140 文字幅インデックスデータ](#)
  - [図 141 間接インデックス](#)
  - [図 142 印刷の全体構成](#)
  - [図 143 レイアウト用紙と印刷用紙](#)
- 

[全目次にもどる](#)

[前頁:目次にもどる](#)

## 1.1 プロセス / タスク管理

プロセス / タスク管理機能では、シングルユーザ / マルチプロセスのシステムを実現するために必要な各種の機能を提供している。

### 1.1.1 プロセス / タスク

プロセスとはプログラムを OS が管理する単位であり、1つのマシン上に複数のプロセスが存在する。

タスクとはプログラムの実行単位であり、1つのプロセスには1つまたは複数のタスクが存在し、優先度順および時分割のスケジューリングにより同時に実行される。

1つのプロセスは基本的に下図に示すような独立したアドレス空間を持つ。

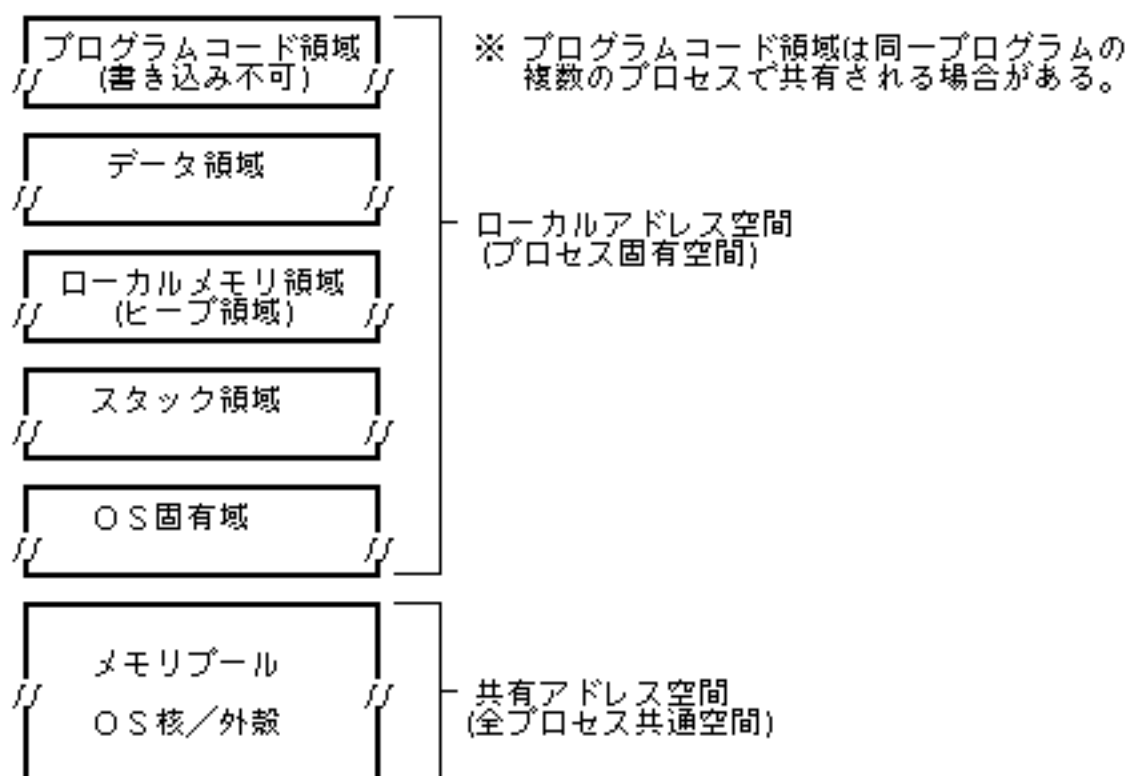


図 1: プロセスの基本メモリモデル

プロセスは、対象となる実行プログラムファイルを指定して生成され、生成時に割り付けられるプロセス ID により識別される。プロセス ID は、正の整数値であり、その割り付け方法は若い番号から順番に割り付けられる。

プロセスが生成されると、メインタスクが実行を開始する。メインタスクによってサブタスクを起動することで、1つのプロセス内に複数のタスクが存在する状態になる。つまり、プロセスには1つのメインタスクと0個以上のサブタスクが同時に存在する。

メインタスクとサブタスクを総称してタスクと呼ぶ。タスクはタスクIDにより識別され

る。タスクIDは正の整数となる。

システムの立ち上げ時に、まず初期プロセスが生成され、初期プロセスが順次必要なプロセスを生成していくことになる。自分が生成したプロセスを子プロセスと呼び、自分を生成してくれたプロセスを親プロセスと呼ぶ(初期プロセスの親プロセスは存在しない)。従って、システム全体としては、初期プロセスをルートとする木構造のプロセス構造をとる。

あるプロセスが終了した場合には、その子プロセスの親プロセスは終了したプロセスの親プロセスに入れ換え、全体として木構造は保たれることになるが、例外的に初期プロセスが終了した場合には、その子プロセスの親プロセスは存在しない状態となる。

### 1.1.2 プロセス / タスクの状態

プロセスの状態として以下の4つの基本状態が定義される。  
プロセスの状態は、メインタスクの状態である。

- 未生成状態 (Non-Existent) -- プロセスは生成されていない状態
- 実行可能状態 (Ready) -- 実行可能でディスパッチ待ち状態
- 実行状態 (Run) -- 実行中状態
- 待ち状態 (Wait) -- メッセージ、時間、入出力等の待ち状態

プロセスの各状態は各種のシステムコールや、スケジューリング(ディスパッチ、プリエンプト)により以下のように遷移する。

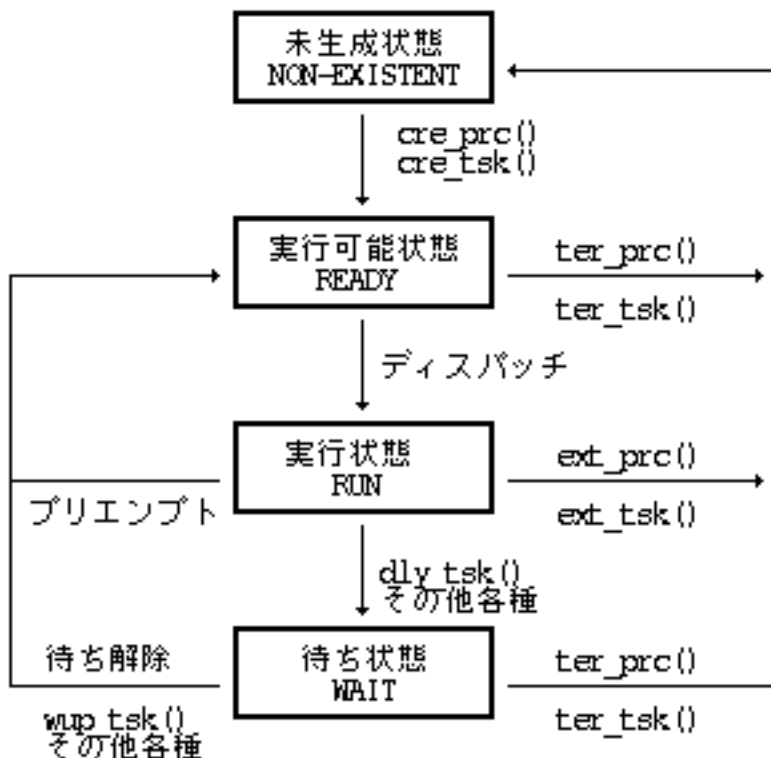


図2: プロセス/タスクの基本状態遷移

プロセスの状態は、以下に示すデータとして取り出すことが可能である。

```
typedef struct {
    UW state; /* プロセス状態 */
```

```

W priority; /* 現在のプロセス優先度( 0 ~ 255 ) */
W parpid; /* 親プロセスのプロセス ID */
} P_STATE;

```

プロセスの状態 (state) は以下の内容であり、それぞれ "1" でその状態にあることを示す。

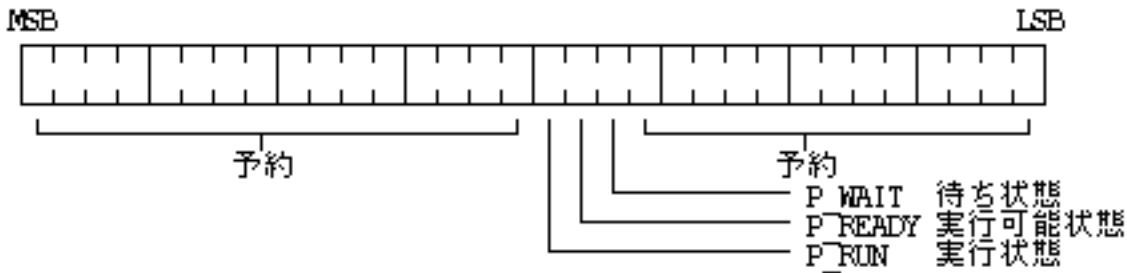


図 3: プロセスの状態ワード

サブタスクも同等の状態を持ち、同様の状態遷移が行われる。

プロセスの終了により、すべてのタスクが終了する。また、メインタスクの終了により、すべてのサブタスクが強制終了され、プロセスの終了となる。サブタスクの終了は、プロセスの終了にはならない。

### 1.1.3 プロセス / タスクの優先度とスケジューリング

プロセスには、生成時に 0 ~ 255 (0 が最高優先度) の優先度が与えられる。この優先度はメインタスクの優先度となる。プロセス優先度と言った場合、それはメインタスクの優先度と等価である。各プロセスはその優先度により、下記の3つのグループに分類される。

また、サブタスクにもタスク生成時に優先度が与えられる。

各タスクは、その優先度に従ってスケジューリングが行なわれる。

#### A. 絶対優先度グループ (優先度: 0 ~ 127)

厳密に優先度順にスケジューリングされ(0が最高優先度)、同一優先度の場合は、ラウンドロビン方式で平等にスケジューリングされる。

#### B. ラウンドロビングループ1 (優先度: 128 ~ 191)

このグループ内全体でラウンドロビンスケジューリングが行なわれ、優先度は相対的なスケジューリングの頻度を示す (128が最高優先度)。

従って、優先度が低いタスクでも必ず実行されることが保証される。

#### C. ラウンドロビングループ2 (優先度: 192 ~ 255)

このグループ内全体でラウンドロビンスケジューリングが行なわれ、優先度は相対的なスケジューリングの頻度を示す (192が最高優先度)。

従って、優先度が低いタスクでも必ず実行されることが保証される。

実際のスケジューリングは、全体として以下のように行なわれる。

1. 絶対優先度グループに属する実行可能状態のタスクがあれば、その中の最高優先度のタスクを実行状態とし、実行する。なければ、2. へ進む。

2. ラウンドロビングループ 1 に属する実行可能状態のタスクがあれば、その中の相対優先度に従って、選択されたタスクを実行状態とし、実行する(最高優先度とは限らない)。なければ、2. へ進む。
3. ラウンドロビングループ 2 に属する実行可能状態のタスクがあれば、その中の相対優先度に従って、選択されたタスクを実行状態とし、実行する(最高優先度とは限らない)。なければ、スケジューリングを始めからやり直す。

通常、絶対優先度グループは、システムプロセスやリアルタイムプロセスに使用され、一般のアプリケーションプロセスはラウンドロビングループ 1、または 2 を使用する。

一度生成されたプロセス / タスクの優先度の変更は、同一グループ内でのみ可能であり、グループが異なる優先度への変更はできない。

### 1.1.4 プロセスの実行環境

プロセスには、その実行環境として以下のような情報が保持されている。

- 自分 / 親 / 子のプロセス ID
- プロセス / タスクの優先度
- ユーザ情報
- 現在の作業ファイル
- オープン中のファイル
- メッセージキュー

プロセスを生成した場合(cre\_prc)は、生成された子プロセスの実行環境は以下のように設定される。

|                    |                  |
|--------------------|------------------|
| 自分 / 親 / 子のプロセス ID | 自分 / 親のプロセス ID   |
| プロセスの優先度           | 生成時に指定した優先度      |
| ユーザ情報              | 生成時の親プロセスのユーザ情報  |
| 現在の作業ファイル          | 生成時の親プロセスの作業ファイル |
| オープン中のファイル         | 無し               |
| メッセージキュー           | 空                |

### 1.1.5 ユーザ情報

ユーザ情報は、そのプロセスを生成して使用しているユーザに関する情報であり、以下に示す内容を持つ。

ユーザ名:

ユーザの名称であり、12 文字の名前の後に 2 文字分 (32 ビット) の隠し名を付けたものである。12 文字より少ない名前の場合は 0 がパッドされ、名前が存在しない場合は、14 文字がすべて 0 となる。

所属グループ名 (1 ~ 4):

ユーザが所属する 4 つのグループの名称であり、それぞれ 12 文字の名前の後に 2 文字分 (32 ビット) の隠し名を付けたものである。12 文字より少ない名前の場合は 0 がパッドされ、名前が存在しない場合は、14 文字がすべて 0 となる。



ユーザレベル:

ユーザの特権レベルであり、0 ~ 15 の範囲の値を取り、0 が最高レベルとなる。

ネットワークユーザレベル:

ネットワーク経由で他のマシンをアクセスする場合のユーザレベルで、1 ~ 15 の範囲の値を取り、1 が最高レベルである。この場合、ネットワークユーザレベルの0は適用されない。

ファイルのデフォルトアクセスモード:

ファイルを生成する場合のデフォルトのアクセスモード。

ユーザ情報は、以下に示す構造体で定義され、システムコールにより取出し/設定が可能である。ただし、隠し名を取り出すことはできず、隠し名の部分には常に0が得られることになる。

```
typedef struct {
    TC  usr_name[14];    /*ユーザ名(12文字 + 隠し名2文字)*/
    TC  grp_name1[14];  /*グループ名1(12文字 + 隠し名2文字)*/
    TC  grp_name2[14];  /*グループ名2(12文字 + 隠し名2文字)*/
    TC  grp_name3[14];  /*グループ名3(12文字 + 隠し名2文字)*/
    TC  grp_name4[14];  /*グループ名4(12文字 + 隠し名2文字)*/
    W   level;          /*ユーザレベル(0 ~ 15)*/
    W   net_level;      /*ネットワークユーザレベル(1 ~ 15)*/
} P_USER;
```

初期プロセスのユーザ情報は、当初はシステムで定義された固定的な内容を持つが、その後、ユーザーが確定した時点で、実際のユーザ情報が設定されることになる。

## 1.1.6 プロセスの生成

プロセスの生成は、対象とするファイル、プロセスの優先度、およびプロセスに渡すメッセージを指定することにより行なわれる。

指定したファイル内の最初の実行プログラムレコードの内容が生成するプロセスのプログラムコードとなる。実行プログラムレコードは、先頭レコードでなければならない。

プロセスに渡すメッセージは以下のメッセージ構造体へのポインタで指定される。これはプロセス間メッセージと同一の構造を持つ。

```
typedef struct {
    W   msg_type;       /*メッセージタイプ*/
    W   msg_size;       /*メッセージサイズ(バイト数)*/
    UB  msg_body[n];    /*メッセージ本体(msg_size バイト)*/
} MESSAGE;
```

生成されたプロセスは以下に示す2つのどちらかの形式により、メッセージを受け取ることができる(ただし、形式-Bは、実際にはライブラリとして実現される)。

MAIN() および main() からのリターンはプロセスの終了となり、ext\_prc() による終了と同等である。終了コードは、正常終了メッセージに設定され、親プロセスに通知される。

## 形式-A

```
W MAIN (MESSAGE *msg)
  /* MESSAGE *msg;      メッセージへのポインタ*/
{
  < プログラムの実行コード >
  return 終了コード;
}
```

- 生成時に指定したメッセージを直接的に得る基本形式。  
msg\_type に無関係にすべてのメッセージを受け取ることが可能。

## 形式-B

```
W main (W ac, TC **argv)
  /* W      ac;      文字列項目数*/
  /* TC     **argv;  文字列項目のポインタ配列へのポインタ*/
{
  < プログラムの実行コード >
  return 終了コード;
}
```

- 生成時に指定したメッセージの msg\_type = 0 の場合にのみ、この形式が適用可能で、この場合は msg\_body[] は TNULL で終了する 1 つの文字列と見なされる。文字列は空白で区切られ、複数の項目に分解される。プログラムへは、各項目の文字列へのポインタの配列として渡されることになる。
- msg\_body[] が TNULL で終了していない場合は、最後の文字は TNULL と見なされる。
- この形式の場合は、生成時に指定されたメッセージの msg\_type = 0 の時は、ac= 0, \*argv = NULL となり、メッセージを受け取ることができない。

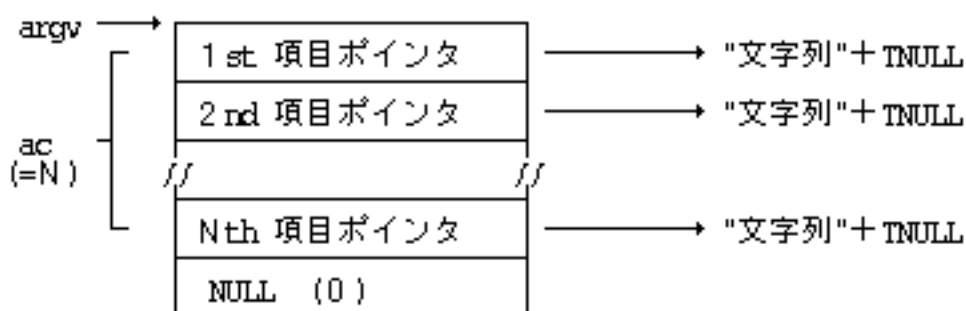


図 4 : main() での引数の構造

### 1.1.7 プロセス / タスクの実行

プロセスは複数のタスクで構成されるため、各タスクが同時並行的に動作することができる。タスクは、プログラムの流れに沿って実行されるが、例外的に以下のハンドラ(関数)は非同期に割り込んで実行される。

## メッセージハンドラ

メッセージ受信により非同期に実行される関数。

## 例外処理ハンドラ

自プロセス内の例外発生により非同期に実行される関数。

これらの関数はいずれもプロセスの一部として、そのプロセスのメモリ空間、および環境の下で実行され、使用可能なシステムコール等の制限は特にない。実行終了後は、割り込まれた位置に戻るか、または任意の位置に直接ジャンプすることができる。

メッセージハンドラの詳細に関しては「[1.2.3 メッセージハンドラ](#)」を、例外処理ハンドラの詳細に関しては「[1.10 システム管理機能](#)」を参照のこと。

また、プロセスはその実行時に以下のようなプロセス統計情報を得ることができる。

```
typedef struct {
    UW  etime;      /* 累計経過時間 (秒単位) */
    UW  utime;     /* プロセスで費やした累計CPU時間 */
    UW  stime;     /* システムで費やした累計CPU時間 */
    W   tmem;      /* 実行に必要なとする全体のメモリサイズ */
    W   wmem;      /* 現在割り付けられている実メモリサイズ */
    W   resv[11];  /* 予約 */
} P_INFO;
```

utime と stime は、プロセスに含まれる、その時点で存在するすべてのタスクの合計時間となる。したがって、それ以前に終了したタスクが費やした時間は含まれない。utime と stime を合計したものが、そのプロセスが費やした累積 CPU 時間となる。

## 1.1.8 データ / 定数の定義

### プロセス状態

```
typedef struct {
    UW  state;     /* プロセス状態 */
    W   priority; /* 現在のプロセス優先度( 0 ~ 255 ) */
    W   parpid;   /* 親プロセスのプロセス ID */
} P_STATE;
```

### プロセスユーザ情報

```
typedef struct {
    TC  usr_name[14]; /* ユーザ名(12文字 + 隠し名2文字) */
    TC  grp_name1[14]; /* グループ名1(12文字 + 隠し名2文字) */
    TC  grp_name2[14]; /* グループ名2(12文字 + 隠し名2文字) */
    TC  grp_name3[14]; /* グループ名3(12文字 + 隠し名2文字) */
    TC  grp_name4[14]; /* グループ名4(12文字 + 隠し名2文字) */
    W   level;      /* ユーザレベル(0 ~ 15) */
    W   net_level;  /* ネットワークユーザレベル(1 ~ 15) */
}
```

```
} P_USER;
```

## プロセス統計情報

```
typedef struct {  
    UW  etime;    /* 累計経過時間 (秒単位) */  
    UW  utime;    /* プロセスで費した累計CPU時間 */  
                /* (OS内での処理時間は除外する) */  
                /* (ミリ秒単位) */  
    UW  stime;    /* システムで費した累計CPU時間 */  
                /* (ミリ秒単位) */  
    W   tmem;     /* 実行に必要とする全体のメモリサイズ */  
                /* (バイト単位) */  
    W   wmem;     /* 現在割り付けられている実メモリサイズ */  
                /* (バイト単位) */  
    W   resv[11]; /* 予約 */  
} P_INFO;
```

## プロセス操作関連の定数

```
#define TERM_NRM    0x0000    /* 指定プロセスのみの強制終了 */  
#define TERM_ALL   0x0001    /* 子プロセスまで含めた強制終了 */  
  
#define P_ABS      0x0000    /* 優先度の絶対指定 ( 0) */  
#define P_REL      0x0001    /* 優先度の相対指定 */  
#define P_TASK     0x0002    /* タスクを対象とする */  
  
#define P_WAIT     0x2000    /* 待ち状態 */  
#define P_READY    0x4000    /* 実行可能状態 */  
#define P_RUN      0x8000    /* 実行状態 */
```

## 1.1.9 システムコール

### cre\_prc

プロセス生成 / 実行

#### 【形式】

```
WERR cre_prc(LINK* lnk, W pri, MESSAGE* msg)
```

#### 【パラメータ】

LINK\*    Ink    対象実身  
W        pri    プロセス優先度  
          0    pri    255    任意の優先度  
          = -1        自プロセスと同じ優先度  
MESSAGE\* msg    プロセス起動時メッセージ

### 【リターン値】

> 0    正常(生成したプロセス ID)  
< 0    エラー(エラーコード)

### 【解説】

指定した実身内の実行プログラムレコードを、新しいプロセスとして生成する。同時にメインタスクが生成され実行を開始する。

[「1.1.6 プロセスの生成」](#)を参照のこと。

### 【エラーコード】

ER\_ACCES    : ファイル(Ink)のアクセス権(E)がない。  
ER\_ADR      : アドレス(Ink,msg)のアクセスは許されていない。  
ER\_BUSY     : ファイル(Ink)は既に排他的にオープンされている為、同時にファイルをオープンすることができなかった。  
ER\_IO        : 入出力エラーが発生した。  
ER\_NOEXS    : ファイル(Ink)は存在していない。  
ER\_NOFS     : ファイル(Ink)の属するファイルシステムは接続されていない。  
ER\_NOMEM    : メモリ領域が不足した。  
ER\_NOSPC    : システムのメモリ領域が不足した(プロセス数が多すぎる)。  
ER\_PPRI     : 優先度の値が範囲外である(-1, -0 ~ 255 以外)。  
ER\_REC      : ファイルにプログラムレコードは存在しない、またはプログラムレコードの内容が異常である。  
ER\_SZOVR    : プロセス起動メッセージのサイズがシステムの制限を越えた。

**ext\_prc**

プロセス終了

### 【形式】

VOID    ext\_prc(W code)

### 【パラメータ】

W code プロセス終了コード

### 【リターン値】

リターンしない

### 【解説】

自プロセスを正常終了し、指定した code を含むプロセス正常終了メッセージを親プロセスに送信する。

自プロセスで使用中のファイル等のリソースは一部のリソース ( cre\_sem 等オプションに DELEXIT 指定がないもの)を除き、すべて自動的に解放される。

### 【エラーコード】

発生しない。

ter\_prc

他プロセス強制終了

### 【形式】

ERR ter\_prc(W pid, W code, W opt)

### 【パラメータ】

W pid 対象プロセス ID  
> 0 任意のプロセス  
= 0 自プロセス(指定不可:エラー)  
= -1 親プロセス

W code 強制終了コード

W opt 強制終了属性  
( TERM\_NRM TERM\_ALL )  
TERM\_NRM 指定したプロセスのみの強制終了。  
TERM\_ALL 指定したプロセスおよびその全ての子孫のプロセスの強制終了。親プロセスまたはさらにその親のプロセスを指定した場合には、自プロセスも強制終了させられる。  
この場合、指定したプロセスの子孫の強制終了メッセージは送信されない。

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

指定したプロセスを強制終了し、指定した code を含むプロセス強制終了メッセージを指定したプロセスの親プロセスに送信する。

自プロセスより高いユーザレベルのプロセスは強制終了できない。

### 【エラーコード】

ER\_SELF : 自プロセスを指定した(pid = 0または自プロセスのPID)。  
ER\_NOPRC : プロセス(pid)は存在していない。  
ER\_LEVEL : 自プロセスより高いユーザレベルのプロセスを指定した。  
ER\_PAR : パラメータが不正である(opt=TERM\_NRM,TERM\_ALL以外を指定した)。

## chg\_pri

プロセス / タスク優先度変更

### 【形式】

WERR chg\_pri(W id, W pri, W opt)

### 【パラメータ】

W id 対象プロセス ID またはタスク ID

W pri 変更する優先度

W opt 優先度属性

( P\_ABS P\_REL ) | [ P\_TASK ]

P\_ABS 絶対指定：指定した優先度に変更。

P\_REL 相対指定：(現在の優先度) + pri に変更。

T\_TASK タスクを対象とする。

### 【リターン値】

0 正常(変更後の優先度 0 ~ 255)

P\_TASK 指定あり id のタスクの優先度

P\_TASK 指定なし id のプロセスのメインタスクの優先度

< 0 エラー(エラーコード)

### 【解説】

指定したプロセス / タスクの優先度を変更する。

P\_TASK の指定がない場合 :

- id = 0 自プロセス内の全タスクの優先度を変更する。
- id = -1 親プロセス内の全タスクの優先度を変更する。
- id > 0 id で指定したプロセス ID のプロセス内の全タスクの優先度を変更する。

P\_TASK が指定された場合

- id = 0 自タスクの優先度を変更する。
- id > 0 id で指定したタスク ID のタスクの優先度を変更する。  
指定できるタスクは、自プロセス内のタスクのみである。

指定できる優先度の範囲は、対象タスクが属するプロセスの現在の優先度グループ内となる。

### 【エラーコード】

- ER\_NOPRC : プロセス(id)は存在していない。
- ER\_ID : タスク(id)は存在していない。または、自プロセス内のタスクではない。
- ER\_PPRI : 優先度の値が範囲外である(新優先度は現在優先度のグループ外)。
- ER\_PAR : パラメータが不正である(opt=P\_ABS,P\_REL,P\_TASK以外を指定した)。

## prc\_sts

プロセス状態取得

### 【形式】

WERR prc\_sts(W pid, P\_STATE\* buff, TC\* name)

### 【パラメータ】

- W pid 対象プロセス ID
  - > 0 任意のプロセス
  - = 0 自プロセス
  - = -1 親プロセス
- P\_STATE\* buff プロセス状態の格納領域  
NULL 格納しない
- TC\* name プロセス名(実身名)の格納領域(最大実身名+1文字分の領域)  
NULL 格納しない

### 【リターン値】



>0 正常(指定プロセス ID)  
<0 エラー(エラーコード)

### 【解説】

指定したプロセスの状態を取得する。

pid = 0, buf = NULL, name = NULL として自プロセス ID の取得にも使用する。

### 【エラーコード】

ER\_ADR : アドレス(buff,path)のアクセスは許されていない。  
ER\_NOPRC : プロセス(pid)は存在していない。

## chg\_usr

自プロセスユーザ情報変更

### 【形式】

ERR chg\_usr(P\_USER\* buff)

### 【パラメータ】

P\_USER\* buff 変更するユーザ情報

### 【リターン値】

=0 正常  
<0 エラー(エラーコード)

### 【解説】

自プロセスのユーザ情報を指定した内容に変更する。変更した内容は、それ以後生成した子プロセスに継承される。

本システムコールはシステムプロセスでのみ実行可能で、一般のアプリケーションプロセスでは実行できない。

### 【エラーコード】

ER\_ADR : アドレス(buff)のアクセスは許されていない。  
ER\_PAR : パラメータが不正である(ユーザ情報が不正)。

# get\_usr

ユーザ情報取得

## 【形式】

WERR get\_usr(W pid, P\_USER\* buff)

## 【パラメータ】

W pid 対象プロセス ID  
> 0 任意のプロセス  
= 0 自プロセス  
= -1 親プロセス

P\_USER\* buff ユーザ情報の格納領域  
NULL 格納しない

## 【リターン値】

> 0 正常(指定プロセス ID)  
< 0 エラー(エラーコード)

## 【解説】

指定したプロセスのユーザ情報を取得する。取り出したユーザー名およびグループ名 1 ~ 4 の隠し名は常に 0 となり、取り出すことはできない。

pid = 0, buf = NULL として自プロセス ID の取得にも使用する。

## 【エラーコード】

ER\_ADR : アドレス(buff)のアクセスは許されていない。  
ER\_NOPRC : プロセス(pid)は存在していない。

# get\_inf

プロセス統計情報取得

## 【形式】

ERR get\_inf(W pid, P\_INFO\* buff)

## 【パラメータ】

W pid 対象プロセス ID  
> 0 任意のプロセス  
= 0 自プロセス  
= -1 親プロセス

P\_INFO\* buff 統計情報の格納領域

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

指定したプロセスの統計情報を取得する。

## 【エラーコード】

ER\_ADR : アドレス(buff)のアクセスは許されていない。  
ER\_NOPRC : プロセス(pid)は存在していない。

# cre\_tsk

サブタスク生成 / 実行

## 【形式】

WERR cre\_tsk(FP entry, W pri, W arg)

## 【パラメータ】

FP entry サブタスク開始アドレス  
W pri サブタスク優先度  
W arg サブタスク起動パラメータ

## 【リターン値】

> 0 正常(生成したサブタスク ID)  
< 0 エラー(エラーコード)

## 【解説】

自プロセス内のサブタスクを生成して実行する。

サブタスクの優先度は、自プロセスの優先度グループの範囲内でのみ指定できる。サブタスクは以下の形式の関数として定義される。

```
VOID    subtask(W arg)
{
    /* arg は cre_tsk() で指定したパラメータ */
    < サブタスクコード >
    /* タスク終了 */
    ext_tsk();
}
```

サブタスクを return で終了することはできない。

### 【エラーコード】

ER\_ADR : アドレス(entry)が不正である。  
ER\_PPRI : 優先度(pri)が自プロセスの優先度グループの範囲を超えている。  
ER\_LIMIT : サブタスク数の制限を超えた。  
ER\_NOMEM : メモリー領域が不足した。

## ext\_tsk

自タスク終了

### 【形式】

```
VOID    ext_tsk()
```

### 【パラメータ】

なし

### 【リターン値】

リターンしない。

### 【解説】

自タスクを終了する。

メインタスクが終了した場合はプロセスも終了する。

### 【エラーコード】

発生しない。

# ter\_tsk

他タスク強制終了

## 【形式】

ERR ter\_tsk(W tskid)

## 【パラメータ】

W tskid 対象タスク ID

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

指定タスクを強制終了する。

指定できるタスクは、自プロセス内のサブタスクのみである。自タスクおよびメインタスクを指定することはできない。

## 【エラーコード】

ER\_ID : タスクID(tskid)が不正である。

# slp\_tsk

タスク起床待ち

## 【形式】

ERR slp\_tsk(W time)

## 【パラメータ】

W time タイムアウト時間  
> 0 指定時間 ( ミリ秒 ) 待つ  
= -1 無限に待つ

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

自タスクを起床待ち状態にする。

time で指定したタイムアウト時間だけ経過するか、他タスクからタスク起床された場合に待ち状態が解除される。

## 【エラーコード】

ER\_NONE : タイムアウト時間が経過したが、起床されなかった。  
ER\_MINTR : メッセージハンドラが起動されたため、待ちが中断された。  
ER\_PAR : パラメータ(time)が不正である。

wup\_tsk

タスク起床

## 【形式】

ERR wup\_tsk(W tskid)

## 【パラメータ】

W tskid 対象タスク ID

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

tskid で指定したタスクが起床待ち状態であった時に、その待ちを解除する。指定したタスクが起床待ち状態でない場合は、起床要求はキューイングされる。

指定するタスクは自プロセス内のタスクでなくてはならない。他プロセスのタスクを起床することはできない。また、自タスクを指定することはできない。

## 【エラーコード】

ER\_ID : タスクID(tskid)が不正である。  
ER\_LIMIT : 起床要求のキューイング数が制限を超えた。

## can\_wup

タスク起床要求のキャンセル

### 【形式】

WERR can\_wup(W tskid)

### 【パラメータ】

W tskid 対象タスク ID  
> 0 任意のタスク  
= 0 自タスク

### 【リターン値】

0 正常(起床要求がキューイングされていた数)  
<0 エラー(エラーコード)

### 【解説】

tskid で指定したタスクにキューイングされている起床要求をすべてキャンセルする。  
指定できるタスクは、自プロセス内のタスクのみである。

### 【エラーコード】

ER\_ID : タスクID(tskid)が不正である。

## dly\_tsk

タスク遅延

### 【形式】

ERR dly\_tsk(W time)

### 【パラメータ】

W time 遅延時間( ミリ秒 0 )

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

自タスクを指定した時間だけ待ち状態にする。

time = 0 を指定した場合は、実行を中断して再スケジュールを行うことを意味する。つまり、タスクの状態を実行状態から、実行可能状態へ移行させることになる。

### 【エラーコード】

ER\_MINTR : メッセージハンドラが起動されたため、待ちが中断された。  
ER\_PAR : パラメータ(time)が不正である。

## get\_tid

自タスク ID の取得

### 【形式】

W get\_tid()

### 【パラメータ】

なし

### 【リターン値】

> 0 正常(自タスク ID)

### 【解説】

自タスクのタスク ID を取り出す。

### 【エラーコード】

発生しない。

---

[この章の目次にもどる](#)

[前頁:第1章 周辺核にもどる](#)

[次頁:1.2 メッセージ管理にすすむ](#)



# 第1章 周辺核

---

## [第1章 周辺核](#)

### [1.1 プロセス/タスク管理](#)

#### [1.1.1 プロセス/タスク](#)

#### [1.1.2 プロセス/タスクの状態](#)

#### [1.1.3 プロセス/タスクの優先度とスケジューリング](#)

#### [1.1.4 プロセスの実行環境](#)

#### [1.1.5 ユーザ情報](#)

#### [1.1.6 プロセスの生成](#)

#### [1.1.7 プロセス/タスクの実行](#)

#### [1.1.8 データ/定数の定義](#)

#### [1.1.9 システムコール](#)

### [1.2 メッセージ管理](#)

#### [1.2.1 プロセス間メッセージ](#)

#### [1.2.2 メッセージの種類](#)

#### [1.2.3 メッセージハンドラ](#)

#### [1.2.4 データ/定数の定義](#)

#### [1.2.5 システムコール](#)

### [1.3 プロセス/タスク間同期通信管理](#)

#### [1.3.1 データ/定数の定義](#)

#### [1.3.2 システムコール](#)

### [1.4 グローバル名管理](#)

#### [1.4.1 グローバル名データ](#)

#### [1.4.2 データ/定数の定義](#)

#### [1.4.3 システムコール](#)

### [1.5 メモリ管理](#)

#### [1.5.1 メモリ管理機能の概要](#)

#### [1.5.2 データ/定数の定義](#)

#### [1.5.3 システムコール](#)

#### [1.5.4 ライブラリ](#)

### [1.6 ファイル管理](#)

#### [1.6.1 ファイル管理機能の概要](#)

#### [1.6.2 ファイルシステム構成](#)

#### [1.6.3 ファイルの構成](#)

#### [1.6.4 ファイルのアクセス管理](#)

- [1.6.5 ファイルの操作](#)
- [1.6.6 ファイルの詳細情報](#)
- [1.6.7 データ / 定数の定義](#)
- [1.6.8 システムコール](#)

## [1.7 イベント管理](#)

- [1.7.1 イベント管理機能の概要](#)
- [1.7.2 イベント](#)
- [1.7.3 キーボード](#)
- [1.7.4 ポインティングデバイス](#)
- [1.7.5 データ/定数の定義](#)
- [1.7.6 システムコール](#)

## [1.8 デバイス管理](#)

- [1.8.1 デバイス管理機能の概要](#)
- [1.8.2 デバイス](#)
- [1.8.3 データ / 定数の定義](#)
- [1.8.4 システムコール](#)

## [1.9 時計管理](#)

- [1.9.1 時計管理機能の概要](#)
- [1.9.2 データ / 定数の定義](#)
- [1.9.3 システムコール](#)

## [1.10 システム管理機能](#)

- [1.10.1 データ / 定数の定義](#)
- [1.10.2 システムコール](#)

---

[この章の目次にもどる](#)

[次頁:1.1 プロセス / タスク管理にすすむ](#)

## 1.2 メッセージ管理

### 1.2.1 プロセス間メッセージ

プロセスにはメッセージキューが付属しており、このメッセージキューを経由してプロセス間メッセージ通信が行なわれる。この機能は、プロセス間でのアプリケーションメッセージの通信だけではなく、子プロセスの終了メッセージ等のシステムでの用途にも使用される。このため、いくつかのメッセージの形式はシステムで定義されている。

メッセージは、プロセスに付属しているメッセージキューに送られ、送信先はそのプロセス ID により指定される。また送信元も送信プロセスのプロセス ID により識別される。

1つのメッセージは以下の構造を持ち、msg\_type により 1 ~ 31 の 31 種のタイプに分けられている。さらに各タイプにビット対応したタイプマスクにより、メッセージの受信/クリアにおいて、対象とするメッセージのタイプ(複数)を指定できるようになっている。

```
typedef union {
    struct {
        W    pid;
        W    code;
    } ABORT;      /* プロセスの異常終了メッセージ */
    struct {
        W    pid;
        W    code;
    } EXIT;      /* プロセスの正常終了メッセージ */
    struct {
        W    pid;
        W    code;
    } TERM;     /* プロセスの強制終了メッセージ */
    struct {
        W    code;
    } TMOU;     /* タイムアウトメッセージ */
    struct {
        W    code;
    } SYSEVT;   /* システムイベントメッセージ */
    struct {
        UB   msg_str[32];
    } ANYMSG;   /* その他一般 */
} MSGBODY;
```

```
typedef struct message {
    W    msg_type; /* メッセージタイプ */
    W    msg_size; /* メッセージ本体サイズ(バイト) */
    MSGBODY msg_body; /* メッセージ本体 */
} MESSAGE;
```

プロセスのメッセージキューには他プロセスからのメッセージ以外にも自プロセスから送信したメッセージも(送信先が自プロセスの場合)入り、この種のメッセージとしてはタイムアウトメッセージ

がある。なお、子プロセスが終了した場合の終了メッセージもこのメッセージキューに入る。

メッセージキューはFIFOであり、必ず送信順に入れられる。送信時に相手のメッセージキューがフルの場合は、キューに空きができるまで待つか、エラーリターンするかを指定できる。

通常は、受信したメッセージはメッセージキューに蓄えられ、メッセージの受信要求により取り出す。メッセージハンドラを定義することにより、指定したタイプのメッセージを受信した場合に、非同期な処理を行なうことが可能である。これは一種の割り込み処理として使用される。

## 1.2.2 メッセージの種類

メッセージは1 ~ 31のタイプ番号により31種のタイプに分類され、そのうち1 ~ はシステムでその用途が規定されており、6 ~ 23がシステム予約で、システム及び外殻により用途が規定される。残りの24 ~ 31がアプリケーションで使用可能となっている。

メッセージタイプは以下のように定義される。

```
#define MS_ABORT      ( 1)    /* プロセス異常終了 */
#define MS_EXIT      ( 2)    /* プロセス正常終了 */
#define MS_TERM      ( 3)    /* プロセス強制終了 */
#define MS_TMOUT     ( 4)    /* タイムアウト */
#define MS_SYSEVT    ( 5)    /* システムイベント(強制終了) */

#define MS_SYS1      ( 6)    /* システム予約 */
#define MS_SYS2      ( 7)    /* システム予約 */
#define MS_SYS3      ( 8)    /* システム予約 */
#define MS_SYS4      ( 9)    /* システム予約 */
#define MS_SYS5      (10)    /* システム予約 */
#define MS_MNG0      (11)    /* 外殻予約 */
#define MS_MNG1      (12)    /* 外殻予約 */
#define MS_MNG2      (13)    /* 外殻予約 */
#define MS_MNG3      (14)    /* 外殻予約 */
#define MS_MNG4      (15)    /* 外殻予約 */
#define MS_MNG5      (16)    /* 外殻予約 */
#define MS_MNG6      (17)    /* 外殻予約 */
#define MS_MNG7      (18)    /* 外殻予約 */
#define MS_MNG8      (19)    /* 外殻予約 */
#define MS_MNG9      (20)    /* 外殻予約 */
#define MS_MNG10     (21)    /* 外殻予約 */
#define MS_MNG11     (22)    /* 外殻予約 */
#define MS_MNG12     (23)    /* 外殻予約 */

#define MS_TYPE0     (24)    /* アプリケーションメッセージ */
#define MS_TYPE1     (25)
#define MS_TYPE2     (26)
#define MS_TYPE3     (27)
#define MS_TYPE4     (28)
#define MS_TYPE5     (29)
#define MS_TYPE6     (30)
#define MS_TYPE7     (31)

#define MS_MIN       ( 1)    /* 最小メッセージタイプ */
#define MS_MAX       (31)    /* 最大メッセージタイプ */
```

各メッセージタイプにはビット対応のタイプマスクが対応づけられており、タイプマスクのORの

パターンにより、対象とするメッセージのタイプを複数指定することができる。

メッセージタイプマスクは以下のように定義される。

```
#define MSGMASK(msgtype)    (1 << ((msgtype) - 1))

#define MM_ABORT    MSGMASK(MS_ABORT)    /* プロセス異常終了 */
#define MM_EXIT     MSGMASK(MS_EXIT)     /* プロセス正常終了 */
#define MM_TERM     MSGMASK(MS_TERM)     /* プロセスの強制終了 */
#define MM_TMOU     MSGMASK(MS_TMOU)     /* タイムアウト */
#define MM_SYSEVT   MSGMASK(MS_SYSEVT)   /* システムイベント(強制終了) */
#define MM_SYS1     MSGMASK(MS_SYS1)     /* システム予約 */
#define MM_SYS2     MSGMASK(MS_SYS2)     /* システム予約 */
#define MM_SYS3     MSGMASK(MS_SYS3)     /* システム予約 */
#define MM_SYS4     MSGMASK(MS_SYS4)     /* システム予約 */
#define MM_SYS5     MSGMASK(MS_SYS5)     /* システム予約 */
#define MM_MNG0     MSGMASK(MS_MNG0)     /* 外殻予約 */
#define MM_MNG1     MSGMASK(MS_MNG1)     /* 外殻予約 */
#define MM_MNG2     MSGMASK(MS_MNG2)     /* 外殻予約 */
#define MM_MNG3     MSGMASK(MS_MNG3)     /* 外殻予約 */
#define MM_MNG4     MSGMASK(MS_MNG4)     /* 外殻予約 */
#define MM_MNG5     MSGMASK(MS_MNG5)     /* 外殻予約 */
#define MM_MNG6     MSGMASK(MS_MNG6)     /* 外殻予約 */
#define MM_MNG7     MSGMASK(MS_MNG7)     /* 外殻予約 */
#define MM_MNG8     MSGMASK(MS_MNG8)     /* 外殻予約 */
#define MM_MNG9     MSGMASK(MS_MNG9)     /* 外殻予約 */
#define MM_MNG10    MSGMASK(MS_MNG10)    /* 外殻予約 */
#define MM_MNG11    MSGMASK(MS_MNG11)    /* 外殻予約 */
#define MM_MNG12    MSGMASK(MS_MNG12)    /* 外殻予約 */

#define MM_TYPE0    MSGMASK(MS_TYPE0)    /* アプリケーションメッセージ */
#define MM_TYPE1    MSGMASK(MS_TYPE1)
#define MM_TYPE2    MSGMASK(MS_TYPE2)
#define MM_TYPE3    MSGMASK(MS_TYPE3)
#define MM_TYPE4    MSGMASK(MS_TYPE4)
#define MM_TYPE5    MSGMASK(MS_TYPE5)
#define MM_TYPE6    MSGMASK(MS_TYPE6)
#define MM_TYPE7    MSGMASK(MS_TYPE7)

#define MM_ALL      (0x7fffffff)         /* 全マスク */
#define MM_NULL     (0)                  /* 空マスク */
```

システムメッセージは、システムで自動的に発生するメッセージであり、以下に示す内容となる。これらは、基本的にメッセージキューのオーバーフローの影響を受けず、捨てられることはないようになっている。なお、1～7のメッセージタイプを持つメッセージをアプリケーションが送信することは特に禁止されていない。

MS\_ABORT -- 子プロセスの異常終了メッセージ

プロセスがシステムエラーにより異常終了させられた場合に、親プロセスに対して子プロセスより自動的に送信される以下の内容のメッセージである。

```
W    1    /* タイプ = MS_ABORT */
W    8    /* 本体のバイト数 */
W    pid  /* 終了した子プロセスのプロセス ID */
```

```
W code /* システムエラーコード */
```

code は発生したシステムエラーのコードであり、MH\_TERM メッセージハンドラによる異常終了の場合は、0 となる。

#### MS\_EXIT -- 子プロセスの正常終了メッセージ

プロセスが `ext_prc()` システムコールにより正常終了した場合に、親プロセスに対して子プロセスより自動的に送信される以下の内容のメッセージである。

```
W 2 /* タイプ = MS_EXIT */
W 8 /* 本体のバイト数 */
W pid /* 終了した子プロセスのプロセス ID */
W code /* ext_prc () で指定した終了コード */
```

#### MS\_TERM -- 子プロセスの強制終了メッセージ

プロセスが `ter_prc()` システムコールにより他から強制終了させられた場合に、親プロセスに対して子プロセスより自動的に送信される以下の内容のメッセージである。

```
W 3 /* タイプ = MS_TERM */
W 8 /* 本体のバイト数 */
W pid /* 終了した子プロセスのプロセス ID */
W code /* ter_prc() で指定した終了コード */
```

#### MS\_TMOU -- 自プロセスのタイムアウトメッセージ

`req_tmng()` システムコールにより要求したタイムアウトメッセージであり、指定した時間後に、自プロセスに対して自動的に送信される以下の内容のメッセージである。

```
W 4 /* タイプ = MS_TMOU */
W 4 /* 本体のバイト数 */
W code /* req_tmng () で指定したコード */
```

## 1.2.3 メッセージハンドラ

メッセージハンドラとは、受信したメッセージを現在実行中の処理と非同期に処理するための関数であり、メッセージのタイプと、対応する関数を指定することにより定義される。従って、メッセージの各タイプに対応した最大 31 種のメッセージハンドラを同時に定義可能である。

メッセージハンドラは以下に示す形で起動されて実行される。

- 対応するタイプのメッセージを受信した場合、メッセージハンドラは、メインタスクの現在の処理に割り込む形で実行される。
- 何らかのシステムコールによる待ち状態の時にメッセージが送信された場合にもメッセージハンドラは起動され、そのプロセスのメインタスクが実行状態となる。  
この場合、メインタスクが実行中の長い待ちを伴ったシステムコールは無条件に中断され、その結果は保証されない。即ち、メッセージハンドラが終了した時点で、中断されたシステムコールは `ER_MINTR` のエラーとして戻る。
- メッセージハンドラは通常のプロセスコードの一部として動作し、実行できるシステムコール等の制限はない。
- メッセージハンドラが終了した場合は、通常、割り込まれた位置からそのメインタスクの実行が再開されるが、メッセージハンドラから直接 `longjmp ()` により、そのメインタスクの任意の位置 (`setjmp` で指定された位置) に実行を移行することも可能である。
- メッセージハンドラはネスティングしない。即ち、新たなメッセージハンドラの起動 (別の

メッセージタイプのものも含む)は、現在処理中のメッセージハンドラの終了まで待たされる。

- メッセージハンドラが起動された場合、その起動の原因となったメッセージはメッセージキューから取り出され、取り出されたメッセージのポインタがハンドラのパラメータとして渡される。
- メッセージハンドラは、必ず `ret_msg ()` システムコールによって終了しなくてはならない。

メッセージハンドラは、以下に示す形の関数として定義される。

```
VOID msg_hdr(W pid, MESSAGE *r_msg)
{
/* pid は 送信したプロセスの ID 。(自プロセスの場合は 0) */
/* r_msg は受信したメッセージへのポインタである。 */

受信メッセージの処理

ret_msg (0);          /* 終了 (割り込んだ位置に戻る場合) */

または

ret_msg (1);          /* 終了 (任意の位置に移行する場合) */
longjmp (reent, code); /* reent へジャンプする */
}
```

特殊なメッセージハンドラとして以下のシステム定義のものが存在し、使用することができる。

MH\_NONE :

何もせずに、メッセージは無視される。この場合は、例外的に待ちを含むシステムコールは中断されない。これは、単にある特定のタイプのメッセージをキューにも入れずに完全に無視するために使用される。

MH\_BREAK :

何もせずに、メッセージは無視される。この場合は、待ちを含むシステムコールは中断され、`ER_MINTR` エラーが戻る。これは、タイムアウト等の処理に使用される。

MH\_TERM :

自プロセスを異常終了し、`MS_ABORT` メッセージ (エラーコードは 0) を親プロセスに送信する。

## 1.2.4 データ / 定数の定義

### プロセスメッセージ定義

```
typedef union {
struct {          /* MS_ABORT */
W pid;
W code;
} ABORT;
struct {          /* MS_EXIT */
W pid;
W code;
} EXIT;
```

```

struct {
    W pid;
    W code;
} TERM;
struct {
    W code;
} TMOUT;
struct {
    W code;
} SYSEVT;
struct {
    UB msg_str[32];
} ANYMSG;
} MSGBODY;

typedef struct message {
    W msg_type; /* メッセージタイプ */
    W msg_size; /* メッセージ本体サイズ(バイト) */
    MSGBODY msg_body; /* メッセージ本体 */
} MESSAGE;

#define MSGSIZE(msgsz) (offsetof(MESSAGE,msg_body) + msgsz)

```

## メッセージタイプ

```

#define MS_ABORT ( 1) /* プロセス異常終了 */
#define MS_EXIT ( 2) /* プロセス正常終了 */
#define MS_TERM ( 3) /* プロセス強制終了 */
#define MS_TMOUT ( 4) /* タイムアウト */
#define MS_SYSEVT ( 5) /* システムイベント ( 強制終了 ) */
#define MS_SYS1 ( 6) /* システム予約 */
#define MS_SYS2 ( 7) /* システム予約 */
#define MS_SYS3 ( 8) /* システム予約 */
#define MS_SYS4 ( 9) /* システム予約 */
#define MS_SYS5 (10) /* システム予約 */
#define MS_MNG0 (11) /* 外殻予約 */
#define MS_MNG1 (12) /* 外殻予約 */
#define MS_MNG2 (13) /* 外殻予約 */
#define MS_MNG3 (14) /* 外殻予約 */
#define MS_MNG4 (15) /* 外殻予約 */
#define MS_MNG5 (16) /* 外殻予約 */
#define MS_MNG6 (17) /* 外殻予約 */
#define MS_MNG7 (18) /* 外殻予約 */
#define MS_MNG8 (19) /* 外殻予約 */
#define MS_MNG9 (20) /* 外殻予約 */
#define MS_MNG10 (21) /* 外殻予約 */
#define MS_MNG11 (22) /* 外殻予約 */
#define MS_MNG12 (23) /* 外殻予約 */

#define MS_TYPE0 (24) /* アプリケーションメッセージ */
#define MS_TYPE1 (25)
#define MS_TYPE2 (26)
#define MS_TYPE3 (27)

```



```
#define MS_TYPE4      (28)
#define MS_TYPE5      (29)
#define MS_TYPE6      (30)
#define MS_TYPE7      (31)

#define MS_MIN        ( 1)      /* 最小メッセージタイプ */
#define MS_MAX        (31)      /* 最大メッセージタイプ */
```

## メッセージタイプマスク

```
#define MSGMASK(msgtype)  (1 << ((msgtype) - 1))
```

```
#define MM_ABORT      MSGMASK(MS_ABORT)
#define MM_EXIT       MSGMASK(MS_EXIT)
#define MM_TERM       MSGMASK(MS_TERM)
#define MM_TMOU       MSGMASK(MS_TMOU)
#define MM_SYSEVT     MSGMASK(MS_SYSEVT)
#define MM_SYS1       MSGMASK(MS_SYS1)
#define MM_SYS2       MSGMASK(MS_SYS2)
#define MM_SYS3       MSGMASK(MS_SYS3)
#define MM_SYS4       MSGMASK(MS_SYS4)
#define MM_SYS5       MSGMASK(MS_SYS5)
#define MM_MNG0       MSGMASK(MS_MNG0)
#define MM_MNG1       MSGMASK(MS_MNG1)
#define MM_MNG2       MSGMASK(MS_MNG2)
#define MM_MNG3       MSGMASK(MS_MNG3)
#define MM_MNG4       MSGMASK(MS_MNG4)
#define MM_MNG5       MSGMASK(MS_MNG5)
#define MM_MNG6       MSGMASK(MS_MNG6)
#define MM_MNG7       MSGMASK(MS_MNG7)
#define MM_MNG8       MSGMASK(MS_MNG8)
#define MM_MNG9       MSGMASK(MS_MNG9)
#define MM_MNG10      MSGMASK(MS_MNG10)
#define MM_MNG11      MSGMASK(MS_MNG11)
#define MM_MNG12      MSGMASK(MS_MNG12)
```

```
#define MM_TYPE0      MSGMASK(MS_TYPE0)
#define MM_TYPE1      MSGMASK(MS_TYPE1)
#define MM_TYPE2      MSGMASK(MS_TYPE2)
#define MM_TYPE3      MSGMASK(MS_TYPE3)
#define MM_TYPE4      MSGMASK(MS_TYPE4)
#define MM_TYPE5      MSGMASK(MS_TYPE5)
#define MM_TYPE6      MSGMASK(MS_TYPE6)
#define MM_TYPE7      MSGMASK(MS_TYPE7)
```

```
#define MM_ALL        (0x7fffffff) /* 全マスク */
#define MM_NULL       (0)          /* 空マスク */
```

## メッセージオプション

```
#define WAIT          0x0000      /* 待つ */
#define NOWAIT        0x0001      /* 待たない */
```

```
#define CONFM      0x0002      /* 受信(確認)待ち */
#define CHECK      0x0002      /* メッセージチェック */
#define WAIEVT     0x0004      /* メッセージおよびイベント待ち */
```

## デフォルトメッセージハンドラ

```
#define MH_NONE     ((FUNCP)1) /* 無視 */
#define MH_BREAK    ((FUNCP)2) /* 処理中断 */
#define MH_TERM     ((FUNCP)3) /* プロセス終了 */
```

## 1.2.5 システムコール

### snd\_msg

メッセージ送信

#### 【形式】

```
ERR snd_msg(W pid, MESSAGE* msg, W opt)
```

#### 【パラメータ】

|          |     |                                    |
|----------|-----|------------------------------------|
| W        | pid | 送信先プロセス ID                         |
|          |     | > 0 任意のプロセス                        |
|          |     | = 0 自プロセス                          |
|          |     | = -1 親プロセス                         |
| MESSAGE* | msg | 送信メッセージ                            |
| W        | opt | 送信待ち属性 ( NOWAIT    WAIT    CONFM ) |

NOWAIT :

メッセージを送信先プロセスのメッセージキューに入れた時点で正常終了する。送信先のメッセージキューに空きがない時はエラー終了する。

WAIT :

メッセージを送信先プロセスのメッセージキューに入れた時点で正常終了する。メッセージキューに空きがない時は空きができるまで待つ。待っている間に送信先プロセスが終了した場合はエラー終了する。

CONFM :

メッセージを送信先プロセスのメッセージキューに入れ、送信先プロセスが送信したメッセージを受信、またはメッセージキューからクリアした時点で正常終了する。それまで待つ。

rcv\_msg() で CHECK 指定によりヘッダ部分のみを得た場合には、受信したとみなされない。NOCLR 指定により受信した場合は、キューに残っていても受信したとみなされる。待っている間に送信先プロセスが終了した場合はエラー終了する。送信先が自プロセスの場合、CONFM 指定はエラーとなる。

#### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

pid で指定したプロセスにメッセージを送信する。

### 【エラーコード】

ER\_ADR : アドレス(msg)のアクセスは許されていない。  
ER\_MINTR : メッセージハンドラが起動されたため待ち処理が中断された。  
ER\_NOPRC : プロセス(pid)は存在していない。  
メッセージキューの空き待ち、または確認待ちの間に送信先のプロセスが終了した。  
ER\_NOSPC : システムのメモリ領域が不足した  
(送信先のメッセージキューがフル(NOWAIT指定時))。  
ER\_PAR : パラメータが不正である(オプション,メッセージタイプが不正)。  
ER\_SELF : 自プロセスを指定した(pid = 0 または自プロセスのPID)(CONFM指定時)。  
ER\_SZOVr : メッセージ本体のサイズがシステムの制限を越えた、または0以下である。

## rcv\_msg

メッセージ受信

### 【形式】

WERR rcv\_msg(W t\_mask, MESSAGE\* msg, W msgsz, W opt)

### 【パラメータ】

W t\_mask 受信対象メッセージタイプマスク  
MESSAGE\* msg 受信メッセージ格納領域  
W msgsz 受信メッセージ格納領域全体のバイトサイズ  
メッセージのヘッダ部分も含むため msgsz 8でなくてはならない。  
W opt 受信属性 (WAIT NOWAIT WAIEVT) | (CLR NOCLR) | (CHECK)

WAIT :

指定タイプのメッセージを受信していない時は、メッセージが来るまで待つ。

NOWAIT :

指定タイプのメッセージを受信していない時は、エラー終了する。

WAIEVT :

指定タイプのメッセージを受信していない時は、指定タイプのメッセージが来るか、イベント発生まで待つ。

CLR :

メッセージを受信した後はそのメッセージをキューから取り除く。

NOCLR :

メッセージを受信した後もそのメッセージをキューに残しておく。

CHECK

メッセージの有無をチェックする。

## 【リターン値】

- >0 正常(受信メッセージの送信元プロセス ID)
- =0 正常(受信メッセージの送信元は自プロセス)
- <0 エラー(エラーコード)

## 【解説】

自プロセス宛の指定したタイプのメッセージを受信する。

受信したメッセージが指定した領域に入り切らない場合は、msgsz の範囲内のみ msg の領域に格納されてエラー終了する。この時は、CLR 指定があってもメッセージはバッファに残る。ただし、msgsz < 8 の時は msg の領域には何も格納されない。メッセージの全体が格納できなかった場合は、格納されたメッセージのヘッダ部分から実際のメッセージサイズが判断できるので、そのメッセージが格納可能な十分な大きさの領域を用意して再度本システムコールを実行することになる。

CHECK 指定の時は、次のような動作になる。

1. メッセージが何も存在しない場合、WAIT または WAIEVT 指定時は待ち、NOWAIT 指定時はエラー終了する。
2. 指定したタイプのメッセージが存在している場合は、msg にメッセージを格納し正常終了する。CLR 指定であればキューから取り除き、NOCLR 指定であればキューに残す。
3. 指定したタイプのメッセージが存在せず、それ以外のメッセージが存在している場合は、メッセージの先頭 8 バイト (msg\_type と msg\_size) のみを msg に格納し正常終了する。この場合、CLR 指定があってもメッセージはキューに残す。

CHECK 指定の時は、指定したタイプ以外のメッセージが得られる可能性があるため、受信したメッセージのタイプを必ずチェックする必要がある。

WAIEVT を指定は基本的な動作は WAIT 指定と同じであるが、メッセージが受信されていなくても、brk\_msg() によりイベント発生が通知されると待ちが解除される。この場合、エラー (ER\_NONE) 終了する。

WAIEVT を指定した場合は、t\_mask = 0 が指定できる。この場合、メッセージの受信は行われず、イベント発生が通知されるまで待つ。

WAIEVT を指定できるのは同時にはシステム全体で 1 つのタスクのみで、複数のタスクが WAIEVT 指定で rcv\_msg() を呼び出したときは、最後に呼び出したタスクの WAIEVT 指定のみが有効となる。他のタスクは WAIT と同じ扱いとなる。

WAIEVT は外殻 (ウインドウマネージャ) の wget\_evt() の実現のために用意されたオプションである。アプリケーションは WAIEVT を指定してはいけない。

## 【エラーコード】

- ER\_ADR : アドレス(msg)のアクセスは許されていない。
- ER\_MINTR : メッセージハンドラが起動されたため待ち処理が中断された。
- ER\_NONE : 指定したタイプ(t\_mask)のメッセージは存在しない(NOWAIT指定時)。
- ER\_PAR : パラメータが不正である  
(msgsz が小さすぎる、  
WAIEVT指定時以外 t\_mask 0、WAIEVT指定時 t\_mask<0)。

# clr\_msg

メッセージクリア

## 【形式】

ERR clr\_msg(W t\_mask, W last\_mask)

## 【パラメータ】

W t\_mask      クリア対象メッセージタイプマスク  
              = MM\_ALL      全メッセージタイプ

W last\_mask    クリア終了メッセージタイプマスク  
              = MM\_NULL    対象はメッセージキューの最後まで  
              = MM\_ALL      メッセージを1つだけクリア

## 【リターン値】

= 0      正常  
< 0      エラー(エラーコード)

## 【解説】

自プロセス宛の指定したタイプの受信済みメッセージをクリアする。

自プロセスのメッセージキューに受信済みのメッセージのうち、t\_maskで指定したタイプのメッセージをlast\_maskで指定したタイプのメッセージの直前までクリアする。last\_maskで指定したタイプのメッセージはクリアされないが、last\_mask = MM\_ALLの時は、特別にメッセージを1つだけクリアすることを意味する。

t\_mask と last\_mask の指定例を以下に示す。

| t_mask | last_mask | 動作                       |
|--------|-----------|--------------------------|
| MM_ALL | MM_NULL   | 受信済みの全メッセージをクリア          |
| -      | MM_ALL    | t_maskで指定したメッセージを1つだけクリア |
| MM_ALL | MM_ALL    | 先頭のメッセージを1つだけクリア         |

## 【エラーコード】

ER\_PAR : パラメータが不正である(t\_mask 0、last\_mask<0)。

# req\_tmng

タイムアウトメッセージ要求

## 【形式】

ERR req\_tmg(W time, W code)

### 【パラメータ】

W time     メッセージ送信時間(ミリ秒)  
W code     タイムアウトメッセージのコード

### 【リターン値】

=0     正常  
<0     エラー(エラーコード)

### 【解説】

自プロセス宛に指定した時間後にタイムアウトメッセージ (MS\_TMOUT) を送信することを要求する。この機能は、メッセージハンドラと組み合わせて特定の処理のタイムアウト監視などに使用される。

### 【エラーコード】

ER\_NOSPC    : システムのメモリ領域が不足した。  
ER\_PAR       : パラメータが不正である(time 0)。

## can\_tmg

タイムアウトメッセージ取り消し

### 【形式】

ERR can\_tmg()

### 【パラメータ】

なし

### 【リターン値】

=0     正常

### 【解説】

自プロセスのタイムアウトメッセージ要求をすべて取り消す。タイムアウトメッセージ要求がない場合は何もしない。

すでに送信されてメッセージキューに入っているタイムアウトメッセージはクリアされない。

### 【エラーコード】

発生しない。

## def\_msg

メッセージハンドラ定義

### 【形式】

```
ERR def_msg(W t_mask, FP msghdr)
```

### 【パラメータ】

W t\_mask 対象メッセージタイプマスク

FP msghdr メッセージハンドラ開始アドレス  
= NULL       メッセージハンドラの定義解除  
= MH\_NONE    システム定義ハンドラ(無視)  
= MH\_BREAK   システム定義ハンドラ(処理中断)  
= MH\_TERM    システム定義ハンドラ(プロセス終了)

### 【リターン値】

= 0     正常  
< 0    エラー(エラーコード)

### 【解説】

指定したタイプのメッセージに対応するメッセージハンドラを定義する。

同じタイプのメッセージに対するメッセージハンドラがすでに定義されている場合には、後から定義したメッセージハンドラが有効となる。

### 【エラーコード】

ER\_ADR : アドレス(msg\_hdr)のアクセスは許されていない。

ER\_PAR : パラメータが不正である(t\_mask 0)。

## ret\_msg

メッセージハンドラ終了

### 【形式】

```
ERR ret_msg(W ret)
```

### 【パラメータ】

W ret リターン指定

=0:

メッセージハンドラが割り込んだ位置から実行を再開し、本システムコールからは戻らない。待ちを含むシステムコールの実行中に割り込んだ場合、そのシステムコールの実行は保証されずに、メッセージハンドラが起動されたことを示すエラーコードがそのシステムコールから戻る。

0:

メッセージハンドラが割り込んだ位置からは再開せずに、本システムコールから戻って、そのまま実行を続ける。この場合、通常はハンドラの最後で longjmp () により制御を他へ移行することになる。

### 【リターン値】

リターンしない(ret = 0 の時)

=0 正常(ret 0 の時)

### 【解説】

メッセージハンドラの実行を終了する。

メッセージハンドラの最後には必ず本システムコールを実行しなくてはならない。また、本システムコールはメッセージハンドラ内でのみ発行することができ、それ以外の部分で発行された時はシステムエラーとなる。

メッセージハンドラの起動要求が多重に発生していた場合、本システムコールの実行後に要求のあったメッセージハンドラが起動する。

### 【エラーコード】

発生しない。

brk\_msg

イベント発生通知

### 【形式】

ERR brk\_msg()

### 【パラメータ】

なし

### 【リターン値】

=0 正常

### 【解説】

WAIEVT 指定による rcv\_msg() の待ちを解除する。brk\_msg() を呼び出したとき WAIEVT 指定で待ちに入っているタスクが無ければ、待ち解除の要求を記録する。ただし、待ち解除の要求回数は記録さ



れない。

【エラーコード】

発生しない。

---

[この章の目次にもどる](#)

[前頁:1.1 プロセス/タスク管理にもどる](#)

[次頁:1.3 プロセス/タスク間同期通信管理にすすむ](#)

## 1.3 プロセス/タスク間同期通信管理

プロセス/タスク間の同期や通信を行うための機構として、下記の同期通信機能を提供する。

- セマフォ
- イベントフラグ
- メッセージバッファ
- ランデブ

これらの同期・通信オブジェクトは動的に生成され、生成時に割り当てられたID (> 0) により識別される。

メッセージによる通信と一番大きく異なる点は、メッセージがプロセス間の通信であるのに対し、タスク間の通信であることである。したがって、主な用途はプロセス内のタスク間の同期・通信である。生成したオブジェクトはすべてのプロセスから使用可能となるので、プロセス間の同期・通信にも使用できる。

### 1.3.1 データ / 定数の定義

#### タイムアウト定義

```
typedef W    TMOU;           /* タイムアウト指定 (ミリ秒) */

#define T_NOWAIT    ( 0)    /* 待たない */
#define T_FOREVER  (-1)    /* 永久待ち */
```

#### セマフォ定数定義

```
#define SEM_SYNC    0x0000 /* 同期用セマフォ */
#define SEM_EXCL    0x4000 /* 排他制御用セマフォ */
```

#### イベントフラグ定数定義

```
#define WF_AND      0x0000 /* イベントフラグ AND 待ち */
#define WF_OR       0x0002 /* イベントフラグ OR 待ち */
```

## 1.3.2 システムコール

### cre\_sem

セマフォの生成

#### 【形式】

WERR cre\_sem(W cnt, UW opt)

#### 【パラメータ】

W cnt セマフォ初期値  
同期用セマフォの場合には cnt = 0 でなければならない。  
排他制御用セマフォの場合には cnt = 1 でなければならない。

UW opt セマフォ属性  
( SEM\_SYNC SEM\_EXCL ) | [ DELEXIT ]  
SEM\_SYNC 同期用セマフォ  
SEM\_EXCL 排他制御用セマフォ  
DELEXIT 生成プロセス終了時に自動削除

#### 【リターン値】

> 0 正常(生成したセマフォID)  
< 0 エラー(エラーコード)

#### 【解説】

セマフォを生成する。生成したセマフォは、すべてのプロセスから使用可能である。

SEM\_EXCL -- 排他制御用セマフォ

- 必ず、獲得 (wai\_sem) ~ 開放 (sig\_sem) の順で使用しなくてはならない。
- セマフォを獲得したまま解放せずにタスクが終了した場合には、自動的に解放される。
- 初期値は 1 でなければならない。

SEM\_SYNC -- 同期用セマフォ

- セマフォの獲得 (wai\_sem)、開放 (sig\_sem) の順序に対する制限はない。
- タスクが終了した場合の自動解放は行われない。
- 生成時の初期値に制限はない。

#### 【エラーコード】

ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_LIMIT : セマフォの数がシステムの制限を超えた。  
ER\_PAR : パラメータが不正である。

## del\_sem

セマフォの削除

### 【形式】

ERR del\_sem(ID id)

### 【パラメータ】

ID id セマフォID

### 【リターン値】

=0 正常  
<0 エラー(エラーコード)

### 【解説】

セマフォを削除する。  
削除したセマフォで待ち状態にあったタスクは待ちが解除され、そのタスクへ ER\_DLT が返される。

### 【エラーコード】

ER\_ID : セマフォIDが不正である。  
ER\_NOEXS : セマフォは存在していない。

## sig\_sem

セマフォへ資源の返却

### 【形式】

ERR sig\_sem(ID id)

### 【パラメータ】

ID id セマフォID

### 【リターン値】

=0 正常  
<0 エラー(エラーコード)

### 【解説】

セマフォへ資源を1つ返却する。

セマフォの資源待ちのタスクがあれば、待ち行列の先頭のタスクの待ちを解除する。資源待ちのタスクがなければ、セマフォの資源を1増やす。

排他制御用セマフォの場合には、対象セマフォの資源を獲得していない状態で返却しようとした場合はER\_OBJになる。

### 【エラーコード】

ER\_ID : セマフォIDが不正である。  
ER\_LIMIT : セマフォのカウント値が制限を越えた。  
ER\_NOEXS : セマフォは存在していない。  
ER\_OBJ : 排他制御用セマフォに対しての不正操作である。

## wai\_sem

セマフォから資源の獲得

### 【形式】

ERR wai\_sem(ID id, TMOUT tmo)

### 【パラメータ】

ID id セマフォID

TMOUT tmo タイムアウト指定 (ミリ秒)  
T\_NOWAIT 資源を獲得できなくても待たない  
T\_FOREVER 資源を獲得できるまで待つ  
>0 資源を獲得できるまで最長 tmo ミリ秒待つ

### 【リターン値】

=0 正常  
<0 エラー(エラーコード)

## 【解説】

セマフォから資源を1つ獲得する。セマフォに1つ以上の資源がある場合には、その資源を1つ獲得する(減らす)。資源が0の場合には、sig\_sem()により資源が返却されるのを待つ。指定された待ち時間が経過しても資源が獲得できない場合には、ER\_NONEが返される。

排他制御用セマフォの場合：

- 対象セマフォの資源をすでに獲得している状態で、再度獲得しようとした場合にはER\_OBJになる。
- 資源を獲得しているタスクが終了した場合には、自動的に資源は返却される。

## 【エラーコード】

ER\_DLT : セマフォは獲得待ちの間に削除された。  
ER\_ID : セマフォIDが不正である。  
ER\_MINTR : メッセージハンドラが起動されたため待ち処理が中断された。  
ER\_NOEXS : セマフォは存在していない。  
ER\_NONE : セマフォを獲得できずにタイムアウトした。  
ER\_PAR : パラメータが不正である。  
ER\_OBJ : 排他制御用セマフォに対しての不正操作である。

## cre\_flg

イベントフラグの生成

## 【形式】

WERR cre\_flg(W ptn, UW opt)

## 【パラメータ】

W ptn イベントフラグ初期パターン ( 0)

UW opt イベントフラグ属性  
[ DELEXIT ]  
DELEXIT 生成プロセス終了時に自動削除

## 【リターン値】

>0 正常(イベントフラグID)  
<0 エラー(エラーコード)

## 【解説】

イベントフラグを生成する。生成したイベントフラグは、すべてのプロセスから使用可能である。

## 【エラーコード】

ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_LIMIT : イベントフラグの数がシステムの制限を超えた。  
ER\_PAR : パラメータが不正である。

# del\_flg

イベントフラグの削除

## 【形式】

ERR del\_flg(ID id)

## 【パラメータ】

ID id イベントフラグID

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

イベントフラグを削除する。削除したイベントフラグで待ち状態にあったタスクは待ちが解除され、そのタスクへ ER\_DLT が返される。

## 【エラーコード】

ER\_ID : イベントフラグIDが不正である。  
ER\_NOEXS : イベントフラグは存在していない。

# set\_flg

イベントフラグをセット

## 【形式】

ERR set\_flg(ID id, W ptn)

### 【パラメータ】

|    |     |                    |
|----|-----|--------------------|
| ID | id  | イベントフラグID          |
| W  | ptn | イベントフラグ設定パターン ( 0) |

### 【リターン値】

=0 正常  
<0 エラー(エラーコード)

### 【解説】

イベントフラグの現在の値に ptn が OR (論理和)される。  
その結果、待ち状態にあるタスクの待ち解除条件が満たされれば、そのタスクの待ちが解除される。

### 【エラーコード】

ER\_ID : イベントフラグIDが不正である。  
ER\_NOEXS : イベントフラグは存在していない。  
ER\_PAR : パラメータが不正である。

## clr\_flg

イベントフラグをクリア

### 【形式】

ERR clr\_flg(ID id, W ptn)

### 【パラメータ】

|    |     |               |
|----|-----|---------------|
| ID | id  | イベントフラグID     |
| W  | ptn | イベントフラグ消去パターン |

### 【リターン値】

=0 正常  
<0 エラー(エラーコード)



## 【解説】

イベントフラグの現在の値に ptn が AND (論理積)される。  
イベントフラグのクリアで、タスクの待ちが解除されることはない。

## 【エラーコード】

ER\_ID : イベントフラグIDが不正である。  
ER\_NOEXS : イベントフラグは存在していない。

# wai\_flg

イベントフラグ待ち

## 【形式】

WERR wai\_flg(ID id, W ptn, UW mode, TMOUT tmo)

## 【パラメータ】

ID id イベントフラグID

W ptn 待ちパターン (>0)

UW mode 待ち解除条件

( WF\_AND WF\_OR ) | [ NOCLR ]

WF\_AND 0x0000 ptn で1のビットがすべてセットされるまで待つ

WF\_OR 0x0002 ptn で1のビットのいずれかがセットされるまで待つ

NOCLR 0x0008 待ち解除条件が満たされた時、イベントフラグの値を  
クリアしない

TMOUT tmo タイムアウト指定 (ミリ秒)

T\_NOWAIT 待ち解除条件が満たされなくても待たない

T\_FOREVER 待ち解除条件が満たされるまで待つ

>0 待ち解除条件が満たされるまで最長 tmo ミリ秒待つ

## 【リターン値】

> 0 正常(待ち状態が解除されたときのイベントフラグの値)

< 0 エラー(エラーコード)

## 【解説】

mode で指定された条件にしたがって、イベントフラグがセットされるのを待つ。

次の条件が満たされたとき待ち解除となる。

WF\_AND の時 ( flg & ptn ) == ptn  
WF\_OR の時 ( flg & ptn ) != 0  
flg はイベントフラグの値

NOCLR が指定された時には、待ち解除条件が満たされたときイベントフラグの値は変化しない。NOCLR の指定がない場合には、待ち解除条件が満たされたときイベントフラグの値 ( 全ビット ) が 0 クリアされる。

指定された待ち時間が経過しても待ち解除条件が満たされない場合には、ER\_NONE が返される。

### 【エラーコード】

ER\_DLT : イベントフラグが待ちの間に削除された。  
ER\_ID : イベントフラグIDが不正である。  
ER\_MINTR : メッセージハンドラが起動されたため待ち処理が中断された。  
ER\_NOEXS : イベントフラグは存在していない。  
ER\_NONE : イベントフラグの条件が満たされずにタイムアウトした。  
ER\_PAR : パラメータが不正である。

## cre\_mbf

メッセージバッファの生成

### 【形式】

WERR cre\_mbf(W bufksz, W max, UW opt)

### 【パラメータ】

W bufksz メッセージバッファの全体のサイズ (バイト数)  
W max メッセージ1つ当たりの最大サイズ (バイト数)  
UW opt メッセージバッファ属性  
[ DELEXIT ]  
DELEXIT 生成プロセス終了時に自動削除

### 【リターン値】

> 0 正常(メッセージバッファID)  
< 0 エラー(エラーコード)

### 【解説】

メッセージバッファを生成する。生成したメッセージバッファは、すべてのプロセスから使用可能である。

bufsz / max 個のメッセージがメッセージバッファに入れられる (保存できる) ことは保証されない。

bufsz に 0 を指定することができる。この場合、メッセージはメッセージバッファに入らず (保存されず)、送信側のタスクと受信側のタスクが同時に送信 / 受信を行うことになる (同期通信)。

### 【エラーコード】

ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_LIMIT : メッセージバッファの数がシステムの制限を超えた。  
ER\_PAR : パラメータが不正である。

## del\_mbf

メッセージバッファの削除

### 【形式】

ERR del\_mbf (ID id)

### 【パラメータ】

ID id メッセージバッファID

### 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

### 【解説】

メッセージバッファを削除する。  
削除したメッセージバッファで待ち状態にあったタスクは待ちが解除され、そのタスクへ ER\_DLT が返される。

### 【エラーコード】

ER\_ID : メッセージバッファIDが不正である。  
ER\_NOEXS : メッセージバッファは存在していない。

# snd\_mbf

メッセージバッファへ送信

## 【形式】

```
ERR snd_mbf(ID id, VP msg, W sz, TMOUT tmo)
```

## 【パラメータ】

|       |           |                     |
|-------|-----------|---------------------|
| ID    | id        | メッセージバッファID         |
| VP    | msg       | 送信メッセージ             |
| W     | sz        | 送信メッセージサイズ (バイト数)   |
| TMOUT | tmo       | タイムアウト指定 (ミリ秒)      |
|       | T_NOWAIT  | 送信できなくても待たない        |
|       | T_FOREVER | 送信できるまで待つ           |
|       | >0        | 送信できるまで最長 tmo ミリ秒待つ |

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

メッセージバッファへメッセージを送信する。メッセージバッファに空きが足りない場合には、空きができるまで待って送信する。指定された待ち時間が経過しても送信できない場合には、ER\_NONE が返される。

メッセージバッファの生成時に指定したメッセージの最大サイズ (max) より大きなメッセージを送信することはできない。

## 【エラーコード】

|          |                                |
|----------|--------------------------------|
| ER_DLT   | : メッセージバッファが待ちの間に削除された。        |
| ER_ID    | : メッセージバッファIDが不正である。           |
| ER_MINTR | : メッセージハンドラが起動されたため待ち処理が中断された。 |
| ER_NOEXS | : メッセージバッファは存在していない。           |
| ER_NONE  | : メッセージを送信できずにタイムアウトした。        |
| ER_PAR   | : パラメータが不正である。                 |

# rcv\_mbf

メッセージバッファから受信

## 【形式】

WERR rcv\_mbf(ID id, VP msg, TMOUT tmo)

## 【パラメータ】

|       |           |                     |
|-------|-----------|---------------------|
| ID    | id        | メッセージバッファID         |
| VP    | msg       | 受信メッセージ             |
| TMOUT | tmo       | タイムアウト指定 (ミリ秒)      |
|       | T_NOWAIT  | 受信できなくても待たない        |
|       | T_FOREVER | 受信できるまで待つ           |
|       | >0        | 受信できるまで最長 tmo ミリ秒待つ |

## 【リターン値】

>0 正常(受信メッセージのバイト数)  
<0 エラー(エラーコード)

## 【解説】

メッセージバッファからメッセージを受信し、msgの領域に書き込む。メッセージバッファにある先頭のメッセージが受信される。メッセージバッファにメッセージがない場合には、メッセージが送信されるまで待って受信する。指定された待ち時間が経過しても受信できない場合には、ER\_NONEが返される。msgには、メッセージバッファの生成時に指定したメッセージ1つ当たりの最大サイズ(max)以上の領域がなくてはならない。

## 【エラーコード】

|          |   |                              |
|----------|---|------------------------------|
| ER_DLT   | : | メッセージバッファが待ちの間に削除された。        |
| ER_ID    | : | メッセージバッファIDが不正である。           |
| ER_MINTR | : | メッセージハンドラが起動されたため待ち処理が中断された。 |
| ER_NOEXS | : | メッセージバッファは存在していない。           |
| ER_NONE  | : | メッセージを受信できずにタイムアウトした。        |
| ER_PAR   | : | パラメータが不正である。                 |

# cre\_por

ランデブポートの生成

## 【形式】

WERR cre\_por(W maxcmsz, W maxrmsz, UW opt)

## 【パラメータ】

W maxcmsz 呼出メッセージ最大サイズ (バイト数)  
W maxrmsz 返答メッセージ最大サイズ (バイト数)  
UW opt ランデブポート属性  
[ DELEXIT ]  
DELEXIT 生成プロセス終了時に自動削除

## 【リターン値】

>0 正常(ランデブポートID)  
<0 エラー(エラーコード)

## 【解説】

ランデブポートを生成する。生成したランデブポートは、すべてのプロセスから使用可能である。

## 【エラーコード】

ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_LIMIT : ランデブポートの数がシステムの制限を超えた。  
ER\_PAR : パラメータが不正である。

# del\_por

ランデブポートの削除

## 【形式】

ERR del\_por(ID id)

## 【パラメータ】

ID id ランデブポートID

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

ランデブポートを削除する。

削除したランデブポートで受け付け待ちまたは呼び出し待ちの状態にあったタスクは待ちが解除され、そのタスクへ ER\_DLT が返される。ランデブ成立後にランデブポートが削除されても、ランデブは成立したままであり、正常にランデブの返答ができる。

### 【エラーコード】

ER\_ID : ランデブポートIDが不正である。  
ER\_NOEXS : ランデブポートは存在していない。

# cal\_por

ランデブ呼出

### 【形式】

WERR cal\_por(ID id, UW calptn, VP msg, W cmsz, TMOUT tmo)

### 【パラメータ】

|       |           |                         |
|-------|-----------|-------------------------|
| ID    | id        | ランデブポートID               |
| UW    | calptn    | 呼出側選択条件 (>0)            |
| VP    | msg       | 呼出 / 返答メッセージ            |
| W     | cmsz      | 呼出メッセージサイズ (バイト数)       |
| TMOUT | tmo       | タイムアウト指定 (ミリ秒)          |
|       | T_NOWAIT  | ランデブが成立しなくても待たない        |
|       | T_FOREVER | ランデブが成立するまで待つ           |
|       | >0        | ランデブが成立するまで最長 tmo ミリ秒待つ |

### 【リターン値】

> 0 正常(返答メッセージのバイト数)  
< 0 エラー(エラーコード)

### 【解説】

ランデブ呼出を行う。ランデブ受け付けタスクが存在し、選択条件が満たされるとランデブ成立となる。指定された待ち時間が経過してもランデブが成立しない場合には、ER\_NONE が返される。ランデブ成立後は、ランデブの返答があるまで待つ。

ランデブは、次の選択条件が満たされると成立する。

( calptn & acpptn ) != 0          acpptn は、ランデブ受付側の選択条件

ランデブが成立すると、msg の内容がランデブ受付タスクへ送られる。ランデブの返答が msg に格納され、そのサイズが戻値に返される。

ランデブポートの生成時に指定した、呼出メッセージの最大サイズ ( maxcmsz ) より大きなメッセージを送信することはできない。また、サイズ 0 のメッセージを送信することもできない。

msg には、ランデブポートの生成時に指定した返答メッセージの最大サイズ ( maxrmsz ) 以上の領域がなくてはならない。

### 【エラーコード】

ER\_DLT       : ランデブポートがランデブ成立待ちの間に削除された。  
ER\_ID        : ランデブポートIDが不正である。  
ER\_MINTR     : メッセージハンドラが起動されたため待ち処理が中断された。  
ER\_NOEXS     : ランデブポートは存在していない。  
ER\_NONE      : ランデブが成立せずにタイムアウトした。  
ER\_PAR       : パラメータが不正である。

## acp\_por

ランデブ受付

### 【形式】

WERR      acp\_por(ID id, UW acpptn, W \*rdv, VP msg, TMOUT tmo)

### 【パラメータ】

|       |           |                         |
|-------|-----------|-------------------------|
| ID    | id        | ランデブポートID               |
| UW    | acpptn    | 受付側選択条件 (>0)            |
| W*    | rdv       | ランデブ番号                  |
| VP    | msg       | 呼出メッセージ                 |
| TMOUT | tmo       | タイムアウト指定 (ミリ秒)          |
|       | T_NOWAIT  | ランデブが成立しなくても待たない        |
|       | T_FOREVER | ランデブが成立するまで待つ           |
|       | >0        | ランデブが成立するまで最長 tmo ミリ秒待つ |

### 【リターン値】

> 0    正常(呼出メッセージのバイト数)



<0 エラー(エラーコード)

### 【解説】

ランデブを受け付ける。

ランデブ呼び出しタスクが存在し、選択条件が満たされるとランデブ成立となる。指定された待ち時間が経過してもランデブが成立しない場合には、ER\_NONE が返される。

ランデブは、次の選択条件が満たされると成立する。

( calptn & acpptn ) != 0          calptn は、ランデブ呼出側の選択条件

ランデブが成立すると、そのランデブの識別番号が rdv に返される。そして、呼び出しメッセージが msg に格納され、そのサイズが戻値に返される。

msg には、ランデブポートの生成時に指定した呼出メッセージの最大サイズ ( maxcmsz)以上の領域がなくてはならない。

### 【エラーコード】

- ER\_DLT       : ランデブポートがランデブ成立待ちの間に削除された。
- ER\_ID        : ランデブポートIDが不正である。
- ER\_MINTR     : メッセージハンドラが起動されたため待ち処理が中断された。
- ER\_NOEXS     : ランデブポートは存在していない。
- ER\_NONE      : ランデブが成立せずにタイムアウトした。
- ER\_PAR       : パラメータが不正である。

fwd\_por

ランデブ回送

### 【形式】

ERR fwd\_por(ID id, UW calptn, W rdv, VP msg, W cmsz)

### 【パラメータ】

- ID id        ランデブポートID
- UW calptn   呼出側選択条件 (>0)
- W rdv        ランデブ番号
- VP msg       呼出メッセージ
- W cmsz       呼出メッセージサイズ (バイト数)

### 【リターン値】

=0 正常  
<0 エラー(エラーコード)

## 【解説】

一旦受け付けたランデブを回送する。

現在のランデブ相手(呼出タスク)と自タスクの間で成立しているランデブを解除し、呼出タスクが新たに回送先のポートに対してランデブ呼出を行う。回送先のポートに対するランデブ呼出が成立しない場合には、呼出タスクがランデブ成立待ちの状態になる。呼出タスクの待ちは、`cal_por()`で指定したタイムアウト指定に関係なく無限待ち(`T_FOREVER`)となる。自タスクは待ちに入らない。

呼出タスクのランデブは、`fwd_por()`で指定された条件で行われる。つまり、`fwd_por()`で指定された`calptn`により選択条件が満たされるとランデブが成立する。

回送先のランデブポートの生成時に指定した、呼出メッセージの最大サイズ(`maxcmsz`)より大きなメッセージを送信することはできない。また、サイズ0のメッセージを送信することもできない。

回送先のランデブポートの返答メッセージの最大サイズ(`maxrmsz`)は、回送前の成立していたランデブポートの返答メッセージの最大サイズより大きくてはいけない(`ER_OBJ`)。また、回送の呼出メッセージサイズ(`cmsz`)も回送前のランデブポートの返答メッセージの最大サイズより大きくてはいけない(`ER_PAR`)。

## 【エラーコード】

ER\_ID : ランデブポートIDが不正である。  
ER\_NOEXS : ランデブポートは存在していない。  
ER\_OBJ : 回送先のランデブポートの返答メッセージの最大サイズより、  
回送前のランデブポートの返答メッセージの最大サイズの方が小さい。  
ER\_PAR : ランデブ番号が不正である。  
ER\_PAR : パラメータが不正である。

# rpl\_rdv

## ランデブ返答

## 【形式】

ERR rpl\_rdv(W rdv, VP msg, W rmsz)

## 【パラメータ】

W rdv      ランデブ番号  
VP msg     返答メッセージ  
W rmsz     返答メッセージサイズ (バイト数)

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

ランデブ相手のタスクに返答し、ランデブを終了する。

ランデブを受け付けたランデブポートを生成したときに指定した、応答メッセージの最大サイズ (maxrmsz) より大きなメッセージを返答することはできない。また、サイズ 0 のメッセージを返答することもできない。

## 【エラーコード】

ER\_OBJ : ランデブ番号が不正である。  
ER\_PAR : パラメータが不正である。

---

[この章の目次にもどる](#)

[前頁:1.2 メッセージ管理にもどる](#)

[次頁:1.4 グローバル名管理にすすむ](#)

## 1.4 グローバル名管理

### 1.4.1 グローバル名データ

グローバル名データ管理機能は、プロセス間で共有的に使用されるデータを任意の名前により生成する機能、および生成された名前によるデータの参照機能を提供している。このように名前付けられて生成された共有データをグローバル名データと呼ぶ。

生成されるデータは1つの32ビットデータ(W)であり、データに付けられる名前は最大8文字である。名前としては通常、文字コードとして意味のあるものが使用されるが特に制限はなく、TNULL(0)までの最大8文字分(16バイト)の任意のデータが名前とされる。

生成されたグローバル名データは、すべてのプロセスから参照/変更可能である。

グローバル名データとしては、生成/変更したプロセスの存在とは無関係に存在し続けるものと、生成/変更したプロセスが終了した場合に自動的に削除されるものの2種類があり、グローバル名データの生成/変更時に指定できる。前者の場合はグローバル名データが不要となった時点で明示的に削除する必要がある。

この機能は、主にある種のデータを不特定多数のプロセスから参照可能とするために使用され、例えば、以下に示すようなデータに対応する固定的な名前により生成し、参照することが考えられる。

- サーバ機能を持ったプロセスのプロセスID
- 特定に資源に対応したセマフォID
- 共通的に使用される環境パラメータ

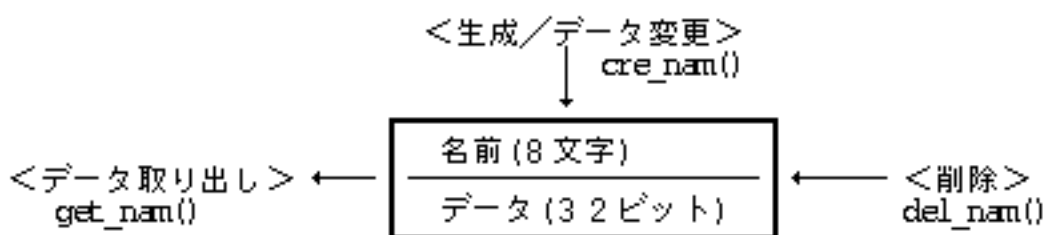


図5: グローバル名データの操作

### 1.4.2 データ / 定数の定義

#### グローバル名管理

```
#define N_CREATE          0x0000      /* グローバル名データの新規生成 */
#define N_MODIFY         0x0001      /* グローバル名データの変更      */
#define N_FORCE          0x0002      /* グローバル名データの強制生成  */
```

## 1.4.3 システムコール

### cre\_nam

グローバル名データ生成

#### 【形式】

WERR cre\_nam(TC\* name, W data, W opt)

#### 【パラメータ】

TC\* name 対象グローバル名(先頭8文字(16バイト)までが有効)  
W data 登録するデータ  
W opt グローバル名属性  
( N\_CREATE N\_MODIFY N\_FORCE ) | [ DELEXIT ]

N\_CREATE :

指定した名前のグローバル名データが存在していない場合に生成する (すでに存在している場合はエラー)。

N\_MODIFY :

指定した名前のグローバル名データが存在している場合にそのデータを変更する (存在していない場合はエラー)。

N\_FORCE :

指定した名前のグローバル名データが存在していない場合は新規に生成し、すでに存在している場合はそのデータを変更する。

DELEXIT :

生成したプロセス、または最後にデータを変更したプロセスの終了時にグローバル名データを自動削除する。既に他のプロセスから DELEXIT 指定が行なわれていた場合でも、DELEXIT 指定により本システムコールを最後に実行したプロセスが対象となる。

#### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

#### 【解説】

name で指定した名前のグローバル名データを生成または変更する。

#### 【エラーコード】

ER\_ADR : アドレス(name)のアクセスは許されていない。  
ER\_EXS : 名前(name)は既に存在している(N\_CREATEの時)。  
ER\_NOEXS : 名前(name)は存在していない(N\_MODIFYの時)。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_PAR : パラメータが不正である(optが不正, nameが空)。

## del\_nam

グローバル名データ削除

### 【形式】

ERR del\_nam(TC\* name)

### 【パラメータ】

TC\* name 対象グローバル名(先頭8文字(16バイト)までが有効)

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

name で指定したグローバル名データを削除する。

### 【エラーコード】

ER\_ADR : アドレス(name)のアクセスは許されていない。  
ER\_NOEXS : 名前(name)は存在していない。  
ER\_PAR : パラメータが不正である(nameが空)。

## get\_nam

グローバル名データ取得

### 【形式】

ERR get\_nam(TC\* name, W\* data)

### 【パラメータ】

TC\* name 対象グローバル名(先頭8文字(16バイト)までが有効)  
W\* data 取得データ格納領域

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

name で指定したグローバル名データを取得する。

### 【エラーコード】

ER\_ADR : アドレス(name, data)のアクセスは許されていない。  
ER\_NOEXS : 名前(name)は存在していない。  
ER\_PAR : パラメータが不正である(nameが空)。

---

[この章の目次にもどる](#)

[前頁:1.3 プロセス/タスク間同期通信管理にもどる](#)

[次頁:1.5 メモリ管理にすすむ](#)

[この章の目次にもどる](#)

[前頁:1.4 グローバル名管理にもどる](#)

[次頁:1.6 ファイル管理にすすむ](#)

---

## 1.5 メモリ管理

### 1.5.1 メモリ管理機能の概要

データメモリ領域には以下に示す 3 種類があり、それぞれの領域に対して指定した大きさのメモリブロックの獲得 / 解放等の機能を提供している。

ローカルメモリ領域：

1 つのプロセス内でのみ使用可能なメモリ領域であり、他のプロセスからはアクセスできない。

ローカルメモリとして確保した領域は、プロセス終了時に自動的に解放される。

共有メモリ領域：

すべてのプロセスから使用可能なメモリ領域である。

獲得したプロセスの存在とは無関係に存在するが、獲得したプロセスの終了時に自動的に解放することもできる。

システムメモリ領域：

システム ( OS やデバイスドライバ等 ) からのみ使用可能なメモリ領域である。アプリケーションプロセスからは使用できない。

プロセスとは無関係に存在する。

基本的にはブロック単位 ( 例えば 4KB 単位 ) でメモリを割り当てる機能のみを OS は提供する。さらに細かく内部を分割するのはライブラリ、またはアプリケーション自身で行う。通常は、システムコールを直接使用せずにライブラリを利用する。ライブラリの機能では不十分な場合のみシステムコールを直接利用する。

獲得したメモリブロックは連続した ( 論理 ) アドレスを持ち、その先頭 ( 論理 ) アドレスがアプリケーションに戻される。一度獲得したメモリブロックの ( 論理 ) アドレスは変更されることはないため、得られたアドレスにより直接メモリブロックをアクセスすることができる。獲得したメモリブロックへのデータの書込み / 読込みは自由にできるが、原則として実行することはできない。

メモリ管理機能のある CPU の場合には、原則として、共有メモリ、システムメモリは負のアドレスとなり、ローカルメモリは正のアドレスとなる。CPU の種類によってはこの原則通りにならない場合がある。また、メモリ管理機能のない CPU の場合は、この限りではない。

メモリに対する排他アクセス制御の機能はないので、必要であればセマフォなどを利用してアプリケーションで行う。



## 1.5.2 データ / 定数の定義

### メモリ属性

```
#define M_COMMON      0x00000001    /* 共有 */
#define M_SYSTEM     0x00000003    /* システムのみ */
#define M_RESIDENT   0x00004000    /* 常駐 */
```

### メモリ状態

```
typedef struct m_state {
    W    blksize;    /* ブロックサイズ */
    W    total;     /* 全ブロック数 */
    W    free;      /* 残りブロック数 */
} M_STATE;
```

## 1.5.3 システムコール

### get\_mbk

メモリブロックの獲得

#### 【形式】

```
ERR get_mbk( VP *adr, W nblk, UW atr )
```

#### 【パラメータ】

VP \*adr 獲得したメモリブロックの先頭アドレスを返す領域  
W nblk 獲得するメモリブロック数( > 0)  
UW atr メモリブロックの属性  
[ (M\_COMMON M\_SYSTEM) ] | [M\_RESIDENT] | [DELEXIT]

M\_COMMON :

共有メモリを指定する。  
共有メモリは、すべてのプロセスからアクセス可能。

M\_SYSTEM

システムメモリを指定する。  
システムメモリは、システム ( OS やデバイスドライバ等 ) からのみアクセス可能。  
この指定は、アプリケーションプロセスからは使用してはいけない。

M\_COMMON, M\_SYSTEM のいずれの指定もなければ、ローカルメモリとなる。ローカルメモリは、メモリブロックを獲得したプロセスからのみアクセス可能。

共有 / システム / ローカルメモリの指定は、メモリ管理機能がないCPUの場合には特に意味を持たない。いずれも同じ扱いとなってしまう、保護されない。

M\_RESIDENT :

常駐を指定する。指定がなければ非常駐となる。

常駐指定されたメモリブロックは、ディスクなどにスワップアウトされることなく常に主メモリ上に存在する。

仮想記憶を行っていないシステムの場合には、特に意味を持たない(常駐と同等になる)。

DELEXIT :

プロセス終了時削除を指定する。

この指定があると、メモリブロックを獲得したプロセスが終了すると自動的にメモリブロックが解放される。ただし、ローカルメモリの場合は、この指定の有無に関係なく、プロセスの終了時に解放される。

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

nbblk 個の連続したメモリブロックを獲得し、その先頭アドレスを \*adr に返す。

### 【エラーコード】

er\_adr : アドレス(adr)のアクセスは許されていない。  
er\_nomem : メモリ領域が不足した。  
er\_nospc : システムのメモリ領域が不足した。  
er\_par : パラメータが不正である。  
er\_ctx : 不正なコンテキストから呼び出した。

## rel\_mbk

メモリブロックの解放

### 【形式】

ERR rel\_mbk( VP adr )

### 【パラメータ】

VP adr 解放するメモリブロックのアドレス

### 【リターン値】

= 0 正常

< 0 エラー(エラーコード)

### 【解説】

adr で示すメモリブロックを解放する。adr は get\_mbk() で得たアドレスでなければならない。

### 【エラーコード】

ER\_MPTR : メモリブロックのアドレスが不正である。

ER\_CTX : 不正なコンテキストから呼び出した。

## mbk\_sts

メモリ状態参照

### 【形式】

ERR mbk\_sts( M\_STATE \*sts )

### 【パラメータ】

M\_STATE sts メモリ状態を返す領域

```
typedef struct m_state {
    W blkksz; /* ブロックサイズ */
    W total; /* 全ブロック数 */
    W free; /* 残りブロック数 */
} M_STATE;
```

### 【リターン値】

= 0 正常

< 0 エラー(エラーコード)

### 【解説】

現在のメモリ使用状況などを取得する。

blkksz:

メモリ割り当ての単位(1ブロック)のバイト数。  
一般にはCPUのページサイズとなる。標準的には4KB。

total :

システム全体の総ブロック数。

free :

システム全体の未使用ブロック数。

仮想記憶を行っているシステムでは、全ブロック数、残りブロック数が一意に決定できない場合がある。したがって、具体的な意味はインプリメントに依存する。ただし、 $free \div total$  が残りメモリの割合の参考値となるような値とする。

インプリメントにより、具体的な値が設定できない場合は、全ブロック数、残りブロック数ともに -1 を設定するものとする。

### 【エラーコード】

ER\_ADR : アドレス(sts)のアクセスは許されていない。

## 1.5.4 ライブラリ

### malloc

非常駐ローカルメモリの獲得(ライブラリ)  
メモリ属性 : DELEXIT

### 【形式】

void\* malloc( size\_t size )

### 【パラメータ】

size\_t size 獲得したいメモリのバイト数(> 0)

### 【リターン値】

NULL 正常(獲得したメモリアドレス)  
= NULL エラー

### 【解説】

非常駐ローカルメモリから指定したサイズのメモリを割り当て、その先頭アドレスを戻す。

# calloc

非常駐ローカルメモリの獲得(ライブラリ)  
メモリ属性 : DELEXIT

## 【形式】

```
void*  calloc( size_t nelem, size_t elsize )
```

## 【パラメータ】

size\_t nelem 獲得したい要素数( > 0 )  
size\_t elsize 1要素のバイト数( > 0 )

## 【リターン値】

NULL 正常(獲得したメモリアドレス)  
= NULL エラー

## 【解説】

非常駐ローカルメモリから elsize の大きさの nelem 個の要素を格納する領域を割り当て、その先頭アドレスを戻す。 割り当てた領域は 0 で初期化される。

# realloc

非常駐ローカルメモリの再獲得(ライブラリ)  
メモリ属性 : DELEXIT

## 【形式】

```
void*  realloc( void *ptr, size_t size )
```

## 【パラメータ】

void \*ptr サイズ変更したい領域のアドレス  
 NULL を指定した場合は新規獲得  
size\_t size 獲得したいメモリのバイト数( 0 )  
 0 を指定した場合は解放

## 【リターン値】

NULL 正常(獲得したメモリアドレス)  
= NULL エラー

### 【解説】

非常駐ローカルメモリ中の ptr で指定される領域のサイズを size に変更して再獲得し、その先頭アドレスを戻す。

ptr は、malloc(), calloc(), realloc() で返されたアドレスでなければならない。

## free

非常駐ローカルメモリの解放(ライブラリ)

### 【形式】

```
void free( void *ptr )
```

### 【パラメータ】

void \*ptr 解放したい領域のアドレス

### 【解説】

ptr で指定された非常駐ローカルメモリ内の領域を解放する。

ptr は、malloc(), calloc(), realloc() で返されたアドレスでなければならない。

## Smalloc

非常駐共有メモリの獲得(ライブラリ)

メモリ属性 : M\_COMMON

### 【形式】

```
void* Smalloc( size_t size )
```

### 【パラメータ】

size\_t size 獲得したいメモリのバイト数(> 0)

### 【リターン値】

NULL 正常(獲得したメモリアドレス)

= NULL エラー

### 【解説】

非常駐共有メモリから指定したサイズのメモリを割り当て、その先頭アドレスを戻す。

## Scalloc

非常駐共有メモリの獲得(ライブラリ)

メモリ属性: M\_COMMON

### 【形式】

```
void* Scalloc( size_t nelem, size_t elsize )
```

### 【パラメータ】

size\_t nelem 獲得したい要素数(> 0)

size\_t elsize 1 要素のバイト数(> 0)

### 【リターン値】

NULL 正常(獲得したメモリアドレス)

= NULL エラー

### 【解説】

非常駐共有メモリから elsize の大きさの nelem 個の要素を格納する領域を割り当て、その先頭アドレスを戻す。割り当てた領域は0で初期化される。

## Srealloc

非常駐共有メモリの再獲得(ライブラリ)

メモリ属性: M\_COMMON

### 【形式】

```
void* Srealloc( void *ptr, size_t size )
```

### 【パラメータ】

void \*ptr サイズ変更したい領域のアドレス

size\_t size      NULL を指定した場合は新規獲得  
再獲得したいメモリのバイト数( 0)  
0 を指定した場合は解放

### 【リターン値】

NULL 正常(獲得したメモリアドレス)  
= NULL エラー

### 【解説】

非常駐共有メモリ中の ptr で指定される領域のサイズを size に変更して再獲得し、その先頭アドレスを戻す。

ptr は、同じプロセス内で獲得したメモリ領域で、Smalloc(), Scalloc(), Srealloc() で返されたアドレスでなければならない。

## Sfree

非常駐共有メモリの解放(ライブラリ)

### 【形式】

void Sfree( void \*ptr )

### 【パラメータ】

void \*ptr      解放したい領域のアドレス

### 【解説】

ptr で指定された非常駐共有メモリ内の領域を解放する。

ptr は、同じプロセス内で獲得したメモリ領域で、Smalloc(), Scalloc(), Srealloc() で返されたアドレスでなければならない。

## Kmalloc

常駐システムメモリの獲得(ライブラリ)

メモリ属性 : M\_SYSTEM | M\_RESIDENT

### 【形式】

void\* Kmalloc( size\_t size )



## 【パラメータ】

size\_t size 獲得したいメモリのバイト数(> 0)

## 【リターン値】

NULL 正常(獲得したメモリアドレス)  
= NULL エラー

## 【解説】

常駐システムメモリから指定したサイズのメモリを割り当て、その先頭アドレスを戻す。  
アプリケーションプロセスからは使用できない。

# Kcalloc

常駐システムメモリの獲得(ライブラリ)  
メモリ属性: M\_SYSTEM | M\_RESIDENT

## 【形式】

```
void* Kcalloc( size_t nelem, size_t elsize )
```

## 【パラメータ】

size\_t nelem 獲得したい要素数(> 0)  
size\_t elsize 1 要素のバイト数(> 0)

## 【リターン値】

NULL 正常(獲得したメモリアドレス)  
= NULL エラー

## 【解説】

常駐システムメモリから elsize の大きさの nelem 個の要素を格納する領域を割り当て、その先頭アドレスを戻す。割り当てた領域は 0 で初期化される。

アプリケーションプロセスからは使用できない。

# Krealloc

常駐システムメモリの再獲得(ライブラリ)  
メモリ属性 : M\_SYSTEM | M\_RESIDENT

## 【形式】

```
void*   Krealloc( void *ptr, size_t size )
```

## 【パラメータ】

|        |      |                                      |
|--------|------|--------------------------------------|
| void   | *ptr | サイズ変更したい領域のアドレス<br>NULL を指定した場合は新規獲得 |
| size_t | size | 再獲得したいメモリのバイト数( 0)<br>0 を指定した場合は解放   |

## 【リターン値】

NULL 正常(獲得したメモリアドレス)  
= NULL エラー

## 【解説】

常駐システムメモリ中の ptr で指定される領域のサイズを size に変更して再獲得し、その先頭アドレスを戻す。

ptr は、Kmalloc(), Kcalloc(), Krealloc() で返されたアドレスでなければならない。

アプリケーションプロセスからは使用できない。

# Kfree

常駐システムメモリの解放(ライブラリ)

## 【形式】

```
void   Kfree( void *ptr )
```

## 【パラメータ】

|      |      |              |
|------|------|--------------|
| void | *ptr | 解放したい領域のアドレス |
|------|------|--------------|

## 【解説】

ptr で指定された常駐システムメモリ内の領域を解放する。  
ptr は、Kmalloc(), Kcalloc(), Krealloc() で返されたアドレスでなければならない。

アプリケーションプロセスからは使用できない。

## Vmalloc

非常駐システムメモリの獲得(ライブラリ)

メモリ属性 : M\_SYSTEM

### 【形式】

```
void* Vmalloc( size_t size )
```

### 【パラメータ】

size\_t size 獲得したいメモリのバイト数(> 0)

### 【リターン値】

NULL 正常(獲得したメモリアドレス)  
= NULL エラー

### 【解説】

非常駐システムメモリから指定したサイズのメモリを割り当て、その先頭アドレスを戻す。

アプリケーションプロセスからは使用できない。

## Vcalloc

非常駐システムメモリの獲得(ライブラリ)

メモリ属性 : M\_SYSTEM

### 【形式】

```
void* Vcalloc( size_t nelem, size_t elsize )
```

### 【パラメータ】

size\_t nelem 獲得したい要素数(> 0)  
size\_t elsize 1 要素のバイト数(> 0)

## 【リターン値】

NULL 正常(獲得したメモリアドレス)  
= NULL エラー

## 【解説】

非常駐システムメモリから `elsize` の大きさの `nelem` 個の要素を格納する領域を割り当て、その先頭アドレスを戻す。割り当てた領域は 0 で初期化される。

アプリケーションプロセスからは使用できない。

# Vrealloc

非常駐システムメモリの再獲得(ライブラリ)  
メモリ属性: M\_SYSTEM

## 【形式】

```
void* Vrealloc( void *ptr, size_t size )
```

## 【パラメータ】

|        |      |                                      |
|--------|------|--------------------------------------|
| void   | *ptr | サイズ変更したい領域のアドレス<br>NULL を指定した場合は新規獲得 |
| size_t | size | 再獲得したいメモリのバイト数( 0 )<br>0 を指定した場合は解放  |

## 【リターン値】

NULL 正常(獲得したメモリアドレス)  
= NULL エラー

## 【解説】

非常駐システムメモリ中の `ptr` で指定される領域のサイズを `size` に変更して再獲得し、その先頭アドレスを戻す。

`ptr` は、`Vmalloc()`、`Vcalloc()`、`Vrealloc()` で返されたアドレスでなければならない。

アプリケーションプロセスからは使用できない。

# Vfree

非常駐システムメモリの解放(ライブラリ)

## 【形式】

```
void    Vfree( void *ptr )
```

## 【パラメータ】

```
void    *ptr    解放したい領域の先頭アドレス
```

## 【解説】

ptr で指定された非常駐システムメモリ内の領域を解放する。  
ptr は、Vmalloc(), Vcalloc(), Vrealloc() で返されたアドレスでなければならない。

アプリケーションプロセスからは使用できない。

---

[この章の目次にもどる](#)

[前頁:1.4 グローバル名管理にもどる](#)

[次頁:1.6 ファイル管理にすすむ](#)

## 1.6 ファイル管理

### 1.6.1 ファイル管理機能の概要

#### 概要

ファイル管理機能ではフロッピーディスク (FD)、ハードディスク (HD) 等の上に作成されるファイルシステムの論理構造、およびファイルを取り扱うためのシステムコール群を規定している。

ファイルシステムを実現する上での各種の物理構造やデータ構造は、インプリメントに依存するが、交換用媒体としてのフロッピーディスクに関しては互換性を保つために物理構造の詳細が規定されている。

ファイルシステムは、実身 / 仮身モデルを基本とした構造となっており、以下のような特徴を持つ。

- 可変長レコードの順序列によるファイル編成 (レコードストリーム)
- ファイルに含まれるリンク (仮身) による任意のネットワーク状の参照関係 (従来のファイルシステムにおけるディレクトリは存在しない)
- リンク (仮身) によるファイルの直接的なアクセス
- ユーザレベルの概念を導入したアクセス管理

ファイルは時系列上において複数のユーザにより使用され、さらにネットワーク環境では複数のユーザにより同時に使用されるためファイルのアクセス管理をきめ細かく行ない、高いレベルの保護機構を提供している。

#### ファイルとリンク

ファイルはデータを保持するための容器または保持されたデータ自体であり、1つのファイルは可変長の順序付けられたレコードの列から構成される。

リンクはファイルを参照するための手掛かりとなる一種のポインタであり、参照するファイルを示すデータとリンク独自のいくつかの属性データをひとまとめにしたデータ構造である。

リンクは、1つのレコードとして任意のファイルの中に埋め込まれて存在する。1つのファイルを示すリンクは複数個存在しても良く、これにより全体としてネットワーク状の任意のファイル間の参照関係が定義される。

実身 / 仮身モデルとの対応において、ファイルは実身に、リンクは仮身に1対1に対応することになる。実身 / 仮身モデルでは、ユーザとのビジュアルインタフェースを基本としており、その意味で、特に仮身はリンクのビジュアル表現とみなすことができる。

ファイルの参照は基本的にリンクを通して直接行なわれるため、ファイルの名前は絶対的な意味を持たず、1つの検索キーとして使用される。ファイル名としては最大20文字の任意のファイル名が付けられるが、同一のファイル名が存在しても構わない。

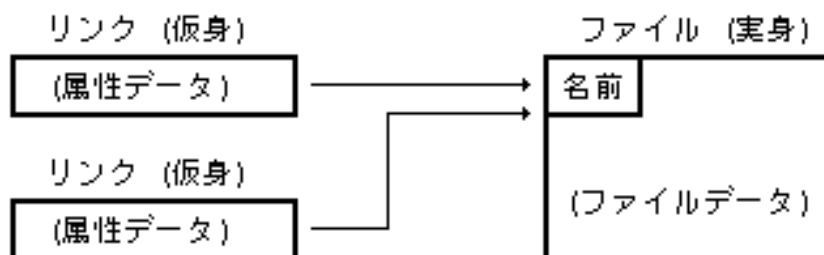


図6: ファイルとリンク

## 1.6.2 ファイルシステム構成

### ファイルシステム

ファイルシステムはファイルを管理するための1つの物理的な単位であり、1つの機器上の以下のような記憶媒体上に構築され、物理的なサイズの上限を持つ。

- フロッピーディスク
- ハードディスク (の1つのパーティション)

ファイルシステムには、必ず1つのルートファイルが存在し、そのルートファイルに含まれるリンクを順次たどっていくことにより、そのファイルシステム内の全てのファイルに基本的に到達可能である。ルートファイルは、実身/仮身モデルとの対応においてデバイス実身に対応する。

ファイル間のリンクによる参照関係は、基本的に1つのファイルシステム内で定義され整合が取られるが、他のファイルシステムに存在するファイルを参照するための間接的なリンクを持つことができ、これを特に間接リンクと呼ぶ。間接リンクによる参照の場合は、参照先の他のファイルシステムの変更に対する整合は取られないため、参照先のファイルの存在は保証されない場合があるため注意が必要である。

ファイルシステムの生成時には、ファイルシステム名、およびデバイス所在名が設定される。

ファイルシステム名はルートファイルの名称としても設定される最大20文字の名前であり、システムおよびユーザがファイルシステムを絶対的に識別するために使用される。ファイルシステム名が同一のファイルシステムは、同一とみなされるため、ファイルシステム名はユニークでなければいけない。

デバイス所在名は、ファイルシステムが格納されている物理デバイスを示す最大20文字の名前であり、ネットワーク経由で他のマシンをアクセスする場合や、フロッピー等の装着を求める場合に使用される。通常、デバイス所在名としてはハードディスク等の機器に固定的に付属しているデバイスの場合にはその媒体の一般名称 (即ち「ハードディスク」等) が使用され、フロッピーディスク等の取り外し可能な媒体の場合にはその媒体の一般名称 (即ち「フロッピーディスク」等) が使用される。

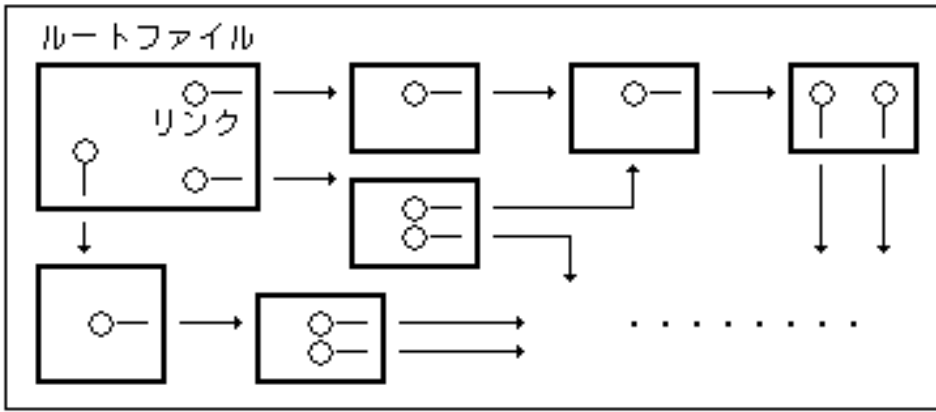


図 7: ファイルシステムの構成

## ファイルシステムの接続

システムのスタートアップ時には、ファイルシステムは1つも存在していない状態であり、接続操作を行なうことにより初めて、ファイルシステムとして使用することが可能となる。従って、通常はシステムの初期化処理として最低限のファイルシステムの接続を行なう必要がある。

ファイルシステムの接続は、接続するファイルシステムが存在する論理デバイス名と、接続名を指定して行なわれる。接続名は接続したファイルシステムを識別するための最大8文字の名前であり、接続したファイルシステムのルートファイルを示す絶対パス名称として使用される。また、接続時には接続したファイルシステムのルートファイルへのリンクが得られる。

従って、接続時に得られたリンク、または接続名を使用することにより、接続したファイルシステム上のルートファイルにアクセスすることが可能となり、ルートファイルから順次リンクをたどることにより、接続したファイルシステム上の任意のファイルにアクセスすることが可能となる。

ファイルシステムの切断は、切断するファイルシステムの論理デバイス名を指定して行なわれ、これにより切断したファイルシステム上のファイルを示すリンクを通したファイルのアクセスはできない状態となり、この状態を切断状態と呼ぶ。

実身/仮身モデルとの対応において、切断状態のリンクは虚身に対応することになる。

ファイルシステムの接続は、単にファイルシステムの存在をファイル管理機能に動的に登録するだけの機能であり、構造を持たない平坦な接続である。従って、複数のファイルシステムをまたがるネットワーク状の静的なファイル参照構造は、異なるファイルシステム上のファイルを参照する間接リンクを利用して構築することになる。



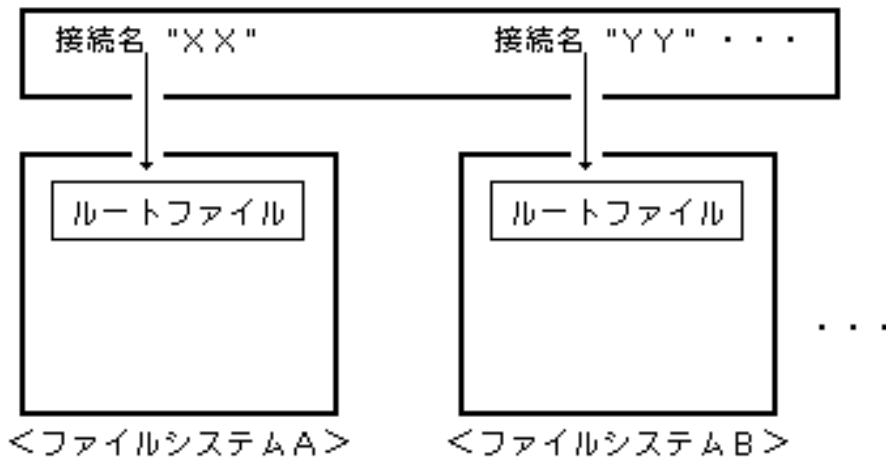


図 8: ファイルシステムの接続

## ファイルID

1つのファイルシステム内の全てのファイルには、生成時にファイルIDと呼ばれるユニークな番号が付けられ、内部的に識別される。ファイルIDは0 ~ (最大ファイルID) の範囲の値であり、最大ファイルID(即ち、最大のファイル数)はファイルシステムの生成時に規定される。ファイルIDは16ビットの数値で表わされるため、最大ファイルIDは65535を越えることはできない。

ファイルシステムのルートファイルは常に0のファイルIDを持つ。

## リンク

リンクは、ファイルにアクセスするための手掛かりとなる一種のポインタであり、参照するファイルが存在するファイルシステム名、ファイルID、およびリンクとしてのいくつかの属性データを保持しているデータ構造である。

リンクは、ビジュアルな操作環境では、仮身として表示される。

リンクは単なるポインタとしての動的なデータであるが、ファイル内に1つのレコードとして格納することにより固定的な存在となる。このように格納されたリンクを特に固定(フィックスド)リンクと呼ぶ。固定リンクはファイルシステム名を持たないため、同一のファイルシステム内のファイルの参照のみ可能であり、ファイルから固定リンクを取り出した時点で、そのファイルの属するファイルシステム名がリンクのデータ構造として設定されることになる。

このため、異なるファイルシステムを参照するリンクを固定リンクとしてファイル内に格納する場合は、あらかじめ、格納するファイルシステム内にリンクファイルと呼ばれる特殊なファイルを生成し、そのリンクファイルを示すリンクを固定リンクとして格納する必要がある。

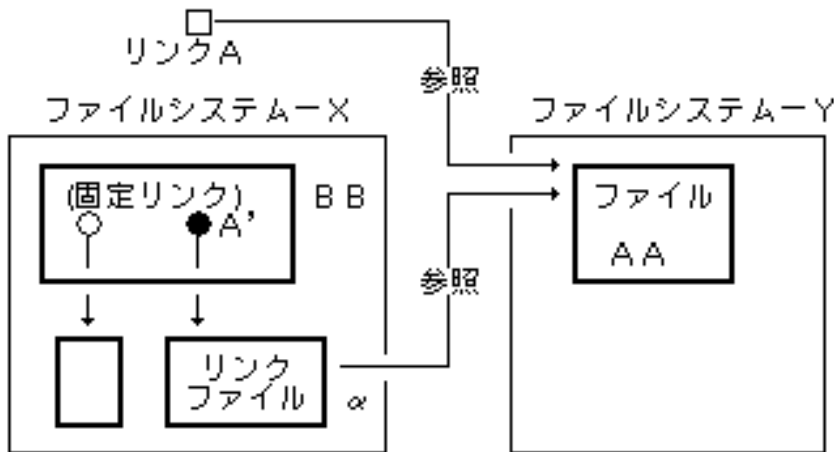
リンクファイルは、参照するファイルの存在するファイルシステム名、ファイルID、ファイル名、および生成日時を保持している特殊なファイルであり、リンクファイルへのアクセスは、参照している異なるファイルシステム内のファイルへのアクセスと自動的に解釈される。リンクファイルを示すリンクを特に間接リンクと呼び、通常ファイルを示すリンクを直接リンクと呼ぶ。

多重の間接リンク、即ち2つ以上のリンクファイルを経由したファイルの参照はサポートさ

れないため、アクセスした時点でファイルは存在しないというエラーになる。

間接リンクによるリンクファイル経由によるファイルの参照は、以下のように行なわれる。

- ファイルシステム名によりファイルシステムを特定する。接続されていない場合は、アクセスは不可となる。なお、接続名は、リンクファイル経由のアクセスには無関係である。
- ファイルIDにより特定したファイルシステム内のファイルをチェックし、ファイル名および生成日時の両方またはどちらか一方が一致しており、かつそのファイルがリンクファイルでない場合に、そのファイルを対象としてアクセスを行なう。それ以外の場合は、参照すべきファイルが存在しなくなったものとみなしてアクセスは不可となる。



リンクAをファイルBBに格納するためには、まずリンクファイルαを生成し、αへのリンクA' (間接リンク)をファイルBBに格納する。

図9：間接リンク / リンクファイル

## 作業ファイル

あるプロセスが現在処理の対象としているファイルをそのプロセスの作業ファイルと呼ぶ。プロセスはシステムコールにより任意のファイルを作業ファイルとすることが可能である。

作業ファイルはプロセスの実行環境として保持されており、生成した子プロセスに継承される。

作業ファイルは未定義とすることも可能であり、システムで最初に生成されたプロセスの作業ファイルは未定義状態となっている。

## パス名

ファイルの参照は基本的にリンクにより直接的に行なわれるが、バッチ的なアプリケーション等では、リンクをインタラクティブに順番にたどることができないため、直接的にたどるべきリンクの列を指定することによりファイルを参照することになる。

このためのリンクの列として、各リンクの参照するファイル名を順番に並べたものを、パス名と呼ぶ。この場合、ファイル名だけでは一意性が保証されないため、その出現順をファイル名に付加して使用する。

出現順は、1つのファイル内に同一のファイル名を持つファイルを参照するリンクがn個存在

する場合に先頭から付けられた 0 ~ n-1 の連続番号である。出現順を省略した場合は 0、即ち最初とみなされる。

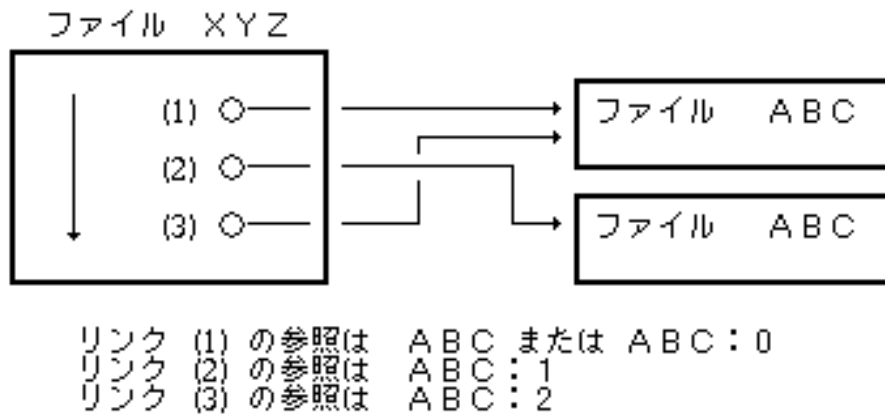


図 10: パス名の出現順

パス名は以下に示す構文を持ち、最大 256 文字までの 1 つの文字列として取り扱われる。

[パス名] ::= [特殊参照] | [特殊参照] / [単純パス名] | [単純パス名]  
 [単純パス名] ::= [単純パス名] / [参照指定] | [参照指定]  
 [参照指定] ::= [ファイル名] | [ファイル名] : [出現順]  
 [特殊参照] ::= / [接続名] |  
 [出現順] ::= 数値  
 [ファイル名] ::= 文字列 (最大20文字)  
 [接続名] ::= 文字列 (最大8文字)

特殊参照は以下の意味を持つ。

/ [接続名] -- 指定した接続名で接続したファイルシステムのルートファイルを示す。  
 -- 作業ファイルを示す。

/ : の記号は、それぞれ以下に示す特殊コードであるため、ファイル名としては空白を含めた表示可能なすべての文字が使用可能である。パス名の文字列の最後に / が存在していた場合は、それは無視されるものとする。

/ TC\_FDLM 0xff21  
 : TC\_FSEP 0xff22  
 TC\_FOWN 0xff23

/ [接続名] で始まるパス名は、そのファイルシステムのルートファイルからのパス名であり、絶対パス名と呼ぶ。それ以外の場合は現在の作業ファイルからの相対的なパス名であり、相対パス名と呼ぶ。

パス名としては以下のようなものがあげられる。

/最新/プロジェクト/ソフトウェア仕様/核仕様/ファイル管理

外部仕様/第10章/例:1

## 1.6.3 ファイルの構成

### ファイルのタイプ

ファイルには、大きく以下の2種類のタイプが存在し、単にファイルと言った場合は、通常ファイルを意味する。

通常ファイル：

データの保存場所としての通常の意味でのファイルである。

リンクファイル：

別のファイルシステム内のファイルを間接的に参照するために使用される特殊なファイルであり、このファイルを示すリンクは間接リンクである。

なお、FD、HD、プリンタ、通信ライン等の物理的なデバイスは、ファイルとしてではなく独立したデバイスとして取り扱う。これらのデバイスはユニークな名前によりシステムに登録され、その名前を使用してアクセスされる。

### 通常ファイルの構成

(通常)ファイルは、任意バイト長のレコードの順序付けられた列、即ちレコードストリームにより構成され、各レコードは、以下の要素により構成される。

- レコードタイプ
- レコードサブタイプ
- レコードサイズ
- レコード本体

レコードタイプは、レコードのタイプを示す 0 ~ 31 の値である。

0 : リンクレコード

他のファイルへのリンクを格納するレコードであり、その内容はファイル管理機能により直接的に取り扱われ、アプリケーションからの直接的な操作は限定される。

1 ~ 31 : データレコード

システムとして規定されるレコードタイプであるが、その内容に関してはファイル管理機能としては特に関知せず、単なるバイト列として取り扱う。

レコードサブタイプは、レコードタイプに応じて使用される補助的なタイプ指定や、キーワードに使用される 16 ビットの符号無し数値である。

レコードサイズは、レコード本体のバイト数を示す32ビットのデータである。リンクレコードの場合はレコードサイズ情報を持たないが、レコードの入出力に必要な領域のサイズである LINK 構造体のサイズ (52バイト) をレコードサイズとする。ただし、このリンクレコードのサイズは、ファイル管理情報としての総バイト数には、カウントされない。

レコード本体は、レコードサイズで示されたバイト数のデータ列であり、その内容はレコードタイプに依存して決められている。リンクレコードの場合はレコード本体は特殊な取り扱いとなる。

### 番号 / 現在レコード

ファイルの各レコードには、先頭のレコードを 0 としたレコードの順番に従った連続番号が付

けられており、これをレコード番号と呼ぶ。レコード番号はレコードの順番を示しているためレコードの挿入 / 削除により動的に変化する。

最後のレコードの次に仮想的にレコードが存在しているものと考え、このレコードを終端レコードと呼ぶ。N個のレコードがある場合、終端レコードはレコード番号Nを持つことになる。

オープンしたファイルに対しては現在、アクセスの対象としているレコードを示す現在レコードが定義され、現在レコードのレコードに対してデータアクセスが行なわれる。現在レコードはレコード番号の指定、レコードタイプによる検索等により移動させることができる。

現在レコードはレコードの挿入 / 削除によっても変化せず、現在レコードに対応するレコード番号が変化することになる。

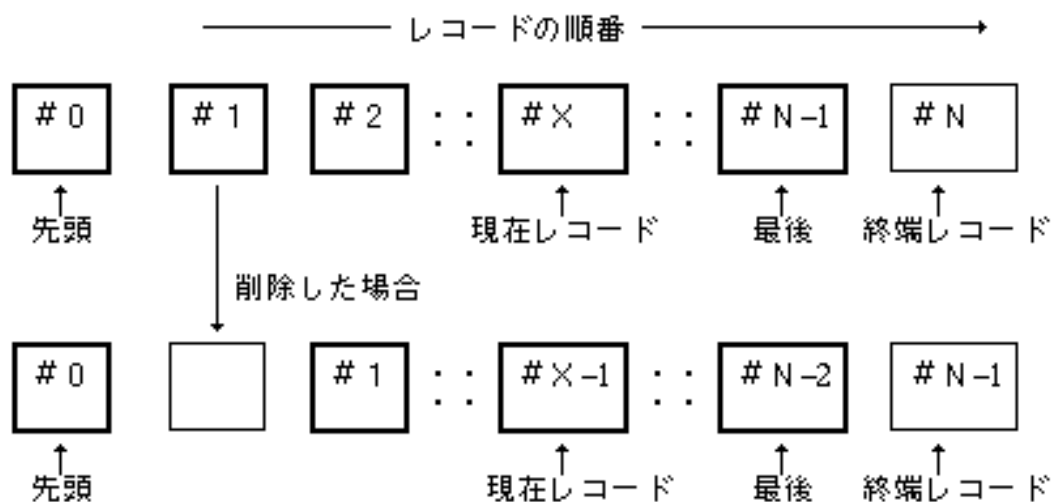


図 11 : レコード番号 / 現在レコード

## リンクファイルの構成

リンクファイルは、異なるファイルシステム内に存在するファイルを間接的に参照するために生成され使用されるファイルであり、アプリケーションデータは存在せず、以下に示す管理用データのみが保存されている。

- 参照するファイルのファイルID
- 参照するファイルのアプリケーションタイプ
- 参照するファイルのファイル名
- 参照するファイルの生成日時
- 参照するファイルが存在したファイルシステム名
- 参照するファイルが存在したファイルシステムのデバイス所在名

## 1.6.4 ファイルのアクセス管理

### ファイルのアクセス管理

ファイルのアクセス管理はファイルにアクセスできるユーザを特定、または制限することにより、ファイルの機密保護や不慮の改変 / 削除から保護するための機構であり、ファイルにアクセスするプロセスが持つユーザ情報と、各ファイルに設定されたアクセス管理情報を照

らし合わせるによりファイルアクセスの可否を決定することになる。

## ユーザ情報

各ユーザには、以下に示すユーザ情報が設定される。

- ユーザ名 (12文字) + 隠しユーザ名 (2文字)
- 所属グループ名1 (12文字) + 隠しグループ名1 (2文字)
- 所属グループ名2 (12文字) + 隠しグループ名2 (2文字)
- 所属グループ名3 (12文字) + 隠しグループ名3 (2文字)
- 所属グループ名4 (12文字) + 隠しグループ名4 (2文字)
- ユーザレベル (0 ~ 15、0が最高レベル)
- ネットワークユーザレベル (1 ~ 15、1が最高レベル)

隠しユーザ名、隠しグループ名は、英数字 4 文字程度を暗号化して、2 文字にコード化したものであり、ユーザ名、グループ名と合わせて、それぞれ 14 文字の名前として使用される。隠し名は同一名の衝突を避けるとともに、暗証コードとしての役割を持っている。

隠し名の暗号化アルゴリズムはインプリメントに依存した非公開のものであり、また個々の機器毎に異なる暗号化がされる必要がある。

これらのユーザ情報はシステム動作開始時に使用者のデフォルト値から読み込まれ、初期プロセスのユーザ情報となる。このユーザ情報は子プロセスに継承され、そのプロセスがファイルをアクセスする際のユーザ情報として使用される。

## ファイルのアクセス管理情報

各ファイルには以下のアクセス管理情報が設定される。

- ファイルの所有者名(12文字) + 隠し名 (2文字)
- ファイルの所有グループ名 (12文字) + 隠し名 (2文字)
- ファイルのアクセスモード
  - 所有者アクセスモード：  
ファイルの所有者に対しての RWE それぞれの可否
  - グループアクセスレベル：  
ファイルの所有グループに所属する各ユーザに対して、RWE それぞれが許可されている最低ユーザレベル (0 ~ 15) の指定
  - 一般アクセスレベル：  
一般のユーザに対して RWE それぞれが許可されている最低ユーザレベル (0 ~ 15) の指定
- ファイルのアクセス属性
  - 書込不可属性
  - 削除不可属性

アクセスモードの、RWE は以下の意味を持つ。

- R -- ファイルの読出しの可否
- W -- ファイルへの書込み / 変更の可否
- E -- ファイルの実行またはファイル内のリンクの検索の可否

ファイルの所有者名は、そのファイルを生成したユーザ名が自動的に設定され、生成した後は一切変更することはできない。また、グループ名、アクセスモードは、ファイルの生成時に指定することができる。グループ名は、そのユーザが所属する最大4つのグループのなかの1つを指定することになる。

即ち、ファイルの生成時には、以下のものを指定することができる。

- 所有者の RWE それぞれの可否
- グループの RWE それぞれの許可最低レベル (0 ~ 15)
- 一般の RWE それぞれの許可最低レベル (0 ~ 15)
- 自分の所属グループの中でどれをファイルの所有グループとするかの指定 (0はグループなし、1~4は自分の所属グループのどれを設定するかを示す)

ユーザ毎にデフォルトアクセスモードが定義されており、ファイルの生成時にデフォルトとして適用させることができる。このデフォルトアクセスモードはユーザー固有の情報として格納されている内容が使用されるが、実行時に変更することも可能である。変更した内容はそのユーザの全プロセスに対して有効となる。

また、ファイルの所有者名、グループ名の最初の1バイトが0の場合は、所有者またはグループが存在しないことを意味し、その場合は所有者アクセスモード、グループアクセスレベルは誰にも適用されないことになる。

ファイルに設定されたアクセス管理情報は、その所有者のみが変更できる(ただし、所有者名は一切変更できない)。所有者がないファイルの場合は、例外的にレベル0のユーザーであればアクセス管理情報を変更することができる。

書込不可属性は、ファイルの内容の変更を禁止する属性であり、この属性がONの状態ではすべての変更操作は所有者を含めた全てのユーザに対して禁止される。

削除不可属性は、ファイルの削除を禁止する属性であり、この属性がONの状態では、ファイルの削除は所有者を含めた全てのユーザに対して禁止される。

削除不可属性が設定されていないファイルの削除は、特にアクセス権を必要としないが、ファイルはどこからも参照されていない場合にのみ削除可能であるため、実際には削除すべきファイルへの固定リンクを含むすべてのファイルの書込みアクセス権がない限り削除できないことになる。

なお、ファイルシステムの接続操作においては、書込不可の接続指定が可能であり、この場合はそのファイルシステムに対するいかなる変更操作も禁止される。同様に物理的に「書き込み不可」となっているフロッピーディスク等の場合も、いかなる変更操作も禁止される。

## ファイルのアクセスチェック

ファイルのアクセスチェックは、そのユーザが所有者であれば、所有者アクセスモード、グループに所属していればグループアクセスレベル、その他の場合は一般アクセスレベルが適用される。

ファイルの所有者名、グループ名の隠し名が0の場合は、隠し名のマッチングは行なわないものとする。即ち、プロセスのユーザ情報にある所有者名、グループ名のそれぞれ12文字のみのマッチングにより同一性を判断することになる。

レベル0のユーザは、基本的にすべてのファイルのRWEが可能(書込不可属性が付けられた場合にはWはできない)となるが、所有者としての権利はないため、アクセスモード、アクセス属性等を変更することはできない。

例：

所有者：佐藤  
所有グループ：プロジェクトA  
所有者アクセスモード：RWE可能  
グループアクセスレベル：Rレベル13, Wレベル10, Eレベル13  
一般アクセスレベル：Rレベル3, Wレベル1, Eレベル5

上記の例の場合、「プロジェクトA」に所属するレベル0～10のユーザは、RWE可能であり、レベル11～13のユーザは、RE可能で、レベル14, 15のユーザは一切のアクセスは禁止される。

また、「プロジェクトA」に属さないユーザに対しては、レベル0, 1はRWE可能であり、レベル2, 3はRE可能、レベル4, 5は3のみ可能、レベル6～15は一切のアクセスが禁止される。

## アクセス管理のレベル

フロッピーディスクは、通常データの交換媒体として使用されることが多く、フロッピーディスク上のファイルに対してはアクセス管理はない方が都合が良い場合が多い。ただし、交換の相手を特定したい場合や、共用的に使用する場合にはアクセス管理が有効な場合もある。

このために、ファイルシステム単位で以下の3種類のアクセス管理のレベルを設定することができるようにする。このアクセス管理のレベルは、そのファイルシステム上のすべてのファイルに対して適用され、ファイルを生成する場合、およびファイルのアクセス管理情報を変更する場合に、以下に示したルールに従ってアクセス管理情報の設定が行なわれる。

従って、ファイルの通常のアクセスマジックに関してはレベルの区別なく行なわれるが、所有者のチェックに関しては、レベル0のファイルシステムは特別に取り扱われ、誰でも所有者としての権利を持つことになる。

### a. レベル0 -- アクセス管理は一切行なわれない

常に以下の内容のアクセス管理情報が設定される。

|             |                    |
|-------------|--------------------|
| 所有者名、隠し名    | : 0 (なし)           |
| グループ名、隠し名   | : 0 (なし)           |
| 所有者アクセス     | : 0 (RWE不可)        |
| グループアクセスレベル | : 0 (RWE不可)        |
| 一般アクセスレベル   | : 0x0fff (誰でもRWE可) |

### b. レベル1 -- アクセス管理は部分的に行なわれる

所有者名/グループ名の隠し名は常に0に設定される。

### c. レベル2 -- アクセス管理は完全に行なわれる

完全なアクセス管理情報が設定される。

一般にフロッピーディスク等の取り外し可能媒体に対しては、レベル0またはレベル1とし、ハードディスク等の機器に固定的に付属するものに関してはレベル2とする。取り外し可能媒体に対してレベル2を設定した場合は、暗号化が機器毎に異なるため事実上1つの機器でしか使用できなくなる。



このアクセス管理レベルはファイルシステムを生成する時点で、ファイルシステムの管理情報として設定される。

## 1.6.5 ファイルの操作

### プロセスの環境

ファイルはプロセスによりアクセスされるが、オープンしたファイルに対しては各プロセス毎で定義されるファイルディスクリプタ ( $fd > 0$ ) という正の整数値が割り当てられ、そのファイルディスクリプタを使用して実際のファイルアクセスを行なう。

プロセスの終了時には、オープンしたファイルはすべて自動的にクローズされる。

また、オープンしたファイルに対しては、現在対象としているレコードを示す現在レコードが定義される。

ファイルディスクリプタ、現在レコード位置は、そのプロセス固有のものとして定義され、子プロセスには特に継承されない。

プロセスの環境として作業ファイル、ユーザ管理情報は子プロセスに継承される。また、デフォルトアクセスモードはユーザ毎に定義され、変更した場合は同一ユーザーのすべてのプロセスに対して即時に有効となる。

### ファイルの参照カウント

ファイルには、そのファイルを参照している同一ファイルシステム内の固定リンクの数を示す参照カウントが存在する。参照カウントはファイルを生成した時点では0であり、ファイルに対する固定リンクを生成した時点、即ち、リンクをファイル内に格納した時点で +1 される。逆に固定リンクが削除された時点で参照カウントは -1 される。

参照カウントは、同一ファイルシステム内での参照を示すため、リンクファイルを経由したファイルの参照は、参照カウントに反映されないことになる。なお、リンクファイル自体にも参照カウントは適用される。

ファイルの削除は参照カウント 0 のファイルに対してのみ可能である。削除したファイルに固定リンクが含まれていた場合は、その固定リンクが参照しているファイルの参照カウントが -1 されるが、その結果 0 となった場合でも、そのファイルは削除されない。なお、固定リンクが含まれているファイルの削除は、削除時に強制削除の指定を行なった時のみ可能となる。

リンクファイルの削除も同様であり、リンクファイル自体の参照カウントが 0 の場合に削除可能となる。なお、リンクファイルの参照先のファイルは、リンクファイル経由で削除することはできない。

ファイルシステムのルートファイルは例外的に参照カウントが最初から 1 となっており、決して削除することができないようになっている。

参照カウント 0 のファイルは、そのファイルを参照する固定リンクを持たないため、動的なリンクが失われてしまうと、通常の方法ではアクセスできないことになるが、ファイルシステム内の全てのファイルに対するリンクを取り出す方法によりアクセスすることは可能である。

### ファイルのアクセス

ファイルは、読み込み(READ) / 書き込み(WRITE) / 更新(UPDATE) のいずれかを指定してオープンするが、オープン時に他からの同一ファイルの同時オープンを制限するための以下のモード指定が可能である。デフォルトは共有モードとなるが、通常は排他書込モードとすることが安全である。

排他モード：

他からのいかなる同時オープンを一切禁止するモード。

排他書込モード：

他からの書き込み / 更新の同時オープンを一切禁止するモード。

共有モード：

他からのいかなる同時オープンも禁止しないモード。

以下に既にオープンされているモードに対して新規に同時オープンできるモードの組み合わせを示す。新規の同時オープンが不可の場合はオープン時にエラーとなる。

| 既オープンモード |   | 新規同時オープンのモード |   |   |         |   |   |       |   |   |
|----------|---|--------------|---|---|---------|---|---|-------|---|---|
|          |   | 排他モード        |   |   | 排他書込モード |   |   | 共有モード |   |   |
|          |   | R            | W | U | R       | W | U | R     | W | U |
| 排他モード    | R | —            | — | — | —       | — | — | —     | — | — |
|          | W | —            | — | — | —       | — | — | —     | — | — |
|          | U | —            | — | — | —       | — | — | —     | — | — |
| 排他書込モード  | R | —            | — | — | ○       | — | — | ○     | — | — |
|          | W | —            | — | — | —       | — | — | ○     | — | — |
|          | U | —            | — | — | —       | — | — | ○     | — | — |
| 共有モード    | R | —            | — | — | ○       | ○ | ○ | ○     | ○ | ○ |
|          | W | —            | — | — | —       | — | — | ○     | ○ | ○ |
|          | U | —            | — | — | —       | — | — | ○     | ○ | ○ |

R：読み込みオープン， W：書き込みオープン， U：更新オープン  
 —：不可， ○：可

図 12：ファイルのオープンモード

また、オープンしたファイルのレコード単位で他からのアクセスを禁止するためのレコードロック機能が用意されている。

ロックしたレコードに対する他からのアクセスは以下ようになる。

- レコードの読み込み、書き込み、置換、サイズ縮小、および削除はエラーとなる。
- サーチの対象、現在レコードとすることは可能。

既にロックされているレコードをロックしようとした場合はエラーとなるか、またはロックが解かれるまで待たされることになる。

### 1.6.6 ファイルの詳細情報

ファイルシステムの管理情報

個々のファイルシステムに対して、以下の内容の管理情報を読み出すことができる。

```

typedef struct {
    H      fs_bsize;          /* 論理ブロックのバイト数 */
    UH     fs_nfile;         /* 最大ファイル数 */
    H      fs_lang;          /* ファイルシステムでの使用言語 */
    H      fs_level;         /* ファイルシステムのアクセス管理レベル */
    W      fs_nblk;          /* 全体のブロック数 */
    W      fs_nfree;         /* 未使用ブロックの総数 */
    STIME  fs_mtime;         /* 最新のシステムブロックの更新日時 */
    STIME  fs_ctime;         /* ファイルシステムの生成日時 */
    TC     fs_name[L_FSNM];  /* ファイルシステム名 */
    TC     fs_locat[L_DLNM]; /* デバイス所在名 */
} FS_STATE;

```

- fs\_bsize は 1 論理ブロックのバイト数であり、2 のべき乗の値とする。
- fs\_nfile はそのファイルシステムに登録可能な最大のファイル数を示す。この値は、最大のファイルID + 1 に等しい。
- fs\_lang はファイルシステムでの使用言語を示し、このファイルシステムで使用している文字コード体系を表わしている。
- fs\_level は、ファイルシステムのアクセス管理のレベルを表わすもので、以下の値をとる。
  - 0:
    - レベル0 -- アクセス管理なし
  - 1:
    - レベル1 -- アクセス管理あり(隠し名なし)
  - 2:
    - レベル2 -- アクセス管理あり(隠し名あり)
- fs\_nblk はファイルシステム全体の論理ブロックの総数であり、この値は論理ブロック番号の最大値 + 1 に等しい。
- fs\_nfree は、現時点の未使用論理ブロックの総数であり、このデータは動的に変動する。
- fs\_mtime, fs\_ctime はそれぞれファイルシステムの最新更新日時と生成された日時で、基準日(1985年1月1日00:00:00 GMT)からの秒数で表わす。
- fs\_name, fs\_locat は、それぞれファイルシステムの生成時に設定される名称であり 20 文字に満たない場合は 0 が詰められている。

ファイルシステムの管理情報はファイルシステムの生成時(フォーマット時)に設定され、以後は未使用ブロックの総数(fs\_nfree)と、最新のシステムブロックの更新日時(fs\_mtime)、ファイルシステム名、およびデバイス所在名を除いて変更されることはない。

## ファイルの管理情報

個々のファイルに対して以下の内容の管理情報を読み出すことができる。但し、リンクファイルの場合はリンクファイルが参照するファイルの管理情報となり、リンクファイル自体の管理情報は読み出すことはできない。

ファイル名:

20文字のファイル名であり変更可能である。

参照カウント：

ファイルを参照している同一ファイルシステム内の固定リンクの数である。

ファイル管理情報：

以下に示す各種の管理情報である。

```
typedef struct {
    UH    f_type;           /* ファイルタイプ / 所有者アクセスモード */
    UH    f_atype;         /* アプリケーションタイプ */
    TC    f_owner[L_USRNM]; /* ファイル所有者名 (隠し名は常に0) */
    TC    f_group[L_USRNM]; /* 所有グループ名 (隠し名は常に0) */
    UH    f_grpacc;        /* グループアクセスレベル */
    UH    f_pubacc;        /* 一般アクセスレベル */
    H     f_nlink;         /* 含まれているリンク数 */
    H     f_index;         /* インデックスレベル */
    W     f_size;          /* ファイルの総バイト数 */
    W     f_nblk;          /* 総使用論理ブロック数 */
    W     f_nrec;          /* 総レコード数 */
    STIME f_ltime;         /* ファイルの保存期限(日時) */
    STIME f_atime;         /* 最新のアクセス日時 */
    STIME f_mtime;         /* 最新の更新日時 */
    STIME f_ctime;         /* ファイルの作成日時 */
} F_STATE;
```

- f\_type はファイルのタイプ、アクセス属性、所有者アクセスモードを示す以下のものである。

TTTT xxxx BAPO xRWE

T: ファイルタイプ

0: リンクファイル

1: 通常ファイル

2~: 予約

P: 削除不可属性

1の場合、このファイルが削除禁止であることを示す。

0: 書込不可属性

1の場合、このファイルが書込み禁止であることを示す。

A: アプリケーション属性1

B: アプリケーション属性2

アプリケーションで規定して使用する属性であり、ファイル管理では、その意味は関知しない。

RWE:

ファイル所有者のアクセスモード(それぞれ、1の時可)

x:

予約(0とする)

- アプリケーションタイプ (f\_atype) はアプリケーションが設定 / 使用するデータで

あり、ファイル管理では使用しない。

- } 所有者名 (f\_owner)、 所有グループ名 (f\_group) はそれぞれ 12 文字であり、 12 文字以下の場合は 0 で埋められる。 続く 2 文字の隠し名は常に 0 として得られる。
- グループアクセスレベル(f\_grpacc)、 一般アクセスレベル(f\_pubacc) は以下の構成となる。

xxxx RRRR WWWW EEEE

RRRR :

読み込み可能な最低のユーザレベル (0 ~ 15)

WWW :

書き込み可能な最低のユーザレベル (0 ~ 15)

EEEE :

実行可能な最低のユーザレベル (0 ~ 15)

XXXX :

未使用 (0)

- 含んでいるリンク数は (f\_nlink) はそのファイルが含んでいるリンクレコードの数を示す。
- インデックスレベルは 0 ~ のレコードインデックスの間接の多重度を示す。
- ファイルの総バイト数 (f\_size) はファイル内に実際に書かれているデータの総バイト数であり、 各レコードのレコードサイズの合計となる。 この場合、 リンクレコードのレコードサイズは 0 としてカウントされる。
- 総使用論理ブロック数は、 そのファイルで使用している論理ブロックの総数を示す。
- 総レコード数は、 そのファイル内に存在するレコードの総数を示す。
- 日時に関しては、 1985年1月1日0 : 00 GMTからの秒数が設定される。 この値が -1 の場合にはそのデータは無効であることを示す。

最新のアクセス日時 (f\_atime)

ファイルのデータを最後にリードした、 またはインデックス部を最後に更新した日時。 ファイルの生成時には -1 (サポートされていない場合)、 あるいはファイルの生成日時が設定される。

最新の更新日時 (f\_mtime)

ファイルのデータを最後に更新した日時。 ファイルの生成時には生成日時が設定される。

ファイルの作成日時 (f\_ctime)

ファイルを最初に生成した日時。

保存期限 (f\_ltime)

ファイルの保存期限。 ファイルの生成時には -1 が設定される。 このデータはアプリケーションで設定 / 使用するもので、 ファイル管理では使用しない。

ファイルの所在情報：

各ファイルが属するファイルシステムの情報であり、この内容はファイルシステムの管理情報の一部である。

```
typedef struct {
    STIME    fs_ctime;           /* ファイルシステムの生成日時 */
    TC      fs_name[L_FSNM];    /* ファイルシステム名 */
    TC      fs_locat[L_DLNM];   /* デバイス所在名 */
    TC      fs_dev[L_DEVNM];    /* 論理デバイス名 */
} F_LOCATE;
```

- 論理デバイス名は、その時点でファイルシステムが存在しているブロック型デバイスの名称である。

リンクファイル情報：

リンクファイルに対してはリンクファイル自体に保持されている以下の参照先のファイルの情報が得られる。この情報は参照先のファイルシステムが接続されていない場合でも取り出すことが可能である。

```
typedef struct {
    STIME    f_ctime;           /* リンクファイル自体の生成日時 */
    UH      f_atype;           /* 参照ファイルのアプリタイプ */
    TC      f_name[L_FNM];     /* 参照ファイル名 */
    UH      f_id;              /* 参照ファイルID */
    STIME    rf_ctime;         /* 参照ファイルの生成日時 */
    TC      fs_name[L_FSNM];    /* ファイルシステム名 */
    TC      fs_locat[L_DLNM];   /* ファイルシステムのデバイス所在名 */
} F_LINK;
```

リンクの構造

ファイルをアクセスするために使用される、リンクは以下のデータ構造となる。

```
typedef struct {
    TC      fs_name[L_FSNM];    /* ファイルシステム名 */
    UH      f_id;              /* ファイル ID */
    UH      atr1;              /* 属性データ1 */
    UH      atr2;              /* 属性データ2 */
    UH      atr3;              /* 属性データ3 */
    UH      atr4;              /* 属性データ4 */
    UH      atr5;              /* 属性データ5 */
} LINK;
```

- ファイルシステム名は接続されたファイルシステム名そのものであり、ファイルシステムを絶対的に識別するために使用される。固定リンクとした場合は、この情報はファイルに格納されない。
- ファイルIDはファイルシステム名により 特定されたファイルシステム内のファイルIDである。
- 属性データ 1~5 はリンク自体として保持される属性データであり、ファイル管理としてはその内容に基本的に関知せず、上位レベルでその用途が決められているものである。新規にリンクを生成した場合のデフォルト値はすべて0とされる。このデータ

は、固定リンクとした場合に、ファイルに格納され、固定リンクを読み出した場合に、ファイルに格納されている内容が取り出される。

ファイル管理ではファイルシステム名とファイルIDのみを使用して実際のファイルアクセスが行なわれる。

通常、リンクはファイル管理機能から得られたものを使用するが、アプリケーションがファイルシステム名とファイルIDを直接設定してリンクを作成することも可能である。

例えばファイルシステムのルートファイルへのリンクは、ファイルID=0であるため、ファイルシステム名が判っていればアプリケーションで直接リンクを作成することが可能となる。

## 1.6.7 データ / 定数の定義

### 各種名前の長さ(文字数)

```
#define L_FSNM      20      /* ファイルシステム名 */
#define L_DLNM      20      /* デバイス所在名 */
#define L_DEVM      8       /* 論理デバイス名 */
#define L_CONNM     8       /* 接続名 */
#define L_USRNM     (12+2)  /* ユーザ(グループ)名 + 隠し名 */
#define L_FNM       20      /* ファイル名 */
#define L_PATHNM    256     /* パス名 */
```

### 各種サイズ

```
#define N_GRP      4       /* 所属グループ数 */
```

### パス名の特殊記号

```
#define TC_FDLM    0xff21   /* '/'   パス名の区切り */
#define TC_FSEP    0xff22   /* ':'   出現順の区切り */
#define TC_FOWN    0xff23   /* ''    自分自身(作業ファイル) */
```

### get\_Ink()

```
#define F_NORM     0x0000   /* 通常指定 */
#define F_BASED    0x0001   /* ベース指定 */
#define F_DIRECT   0x0002   /* 直接取り出し指定 */
```

### opn\_fil()

```
#define F_READ     0x0004   /* 読み込み専用オープン */
#define F_WRITE    0x0002   /* 書き込み専用オープン */
#define F_UPDATE   0x0006   /* 更新用(読込 / 書込)オープン */

#define F_EXCL     0x0100   /* 排他モード */
```

```

#define F_WEXCL      0x0200      /* 排他書き込みモード */

    cre_fil() / cre_lnk()

#define F_FLOAT      0x0000      /* 固定リンクとしない */
#define F_FIX        0x0001      /* 固定リンクとする */
#define F_FILEID     0x0002      /* ファイル ID 指定 */

    fnd_rec() / fnd_lnk()

#define F_FWD        0x0000      /* 前向きサーチ */
#define F_NFWD       0x0001      /* 前向きサーチ(次から) */
#define F_BWD        0x0002      /* 後向きサーチ */
#define F_NBWD       0x0003      /* 後向きサーチ(手前から) */
#define F_TOPEND     0x0004      /* 先頭から終端 */
#define F_ENDTOP     0x0005      /* 終端から先頭 */

#define F_SFILE      0x4000      /* 同一ファイル */
#define F_SNAME      0x2000      /* 同一ファイル名 */
#define F_SATR1      0x1000      /* 同一属性データ1 */
#define F_SATR2      0x0800      /* 同一属性データ2 */
#define F_SATR3      0x0400      /* 同一属性データ3 */
#define F_SATR4      0x0200      /* 同一属性データ4 */
#define F_SATR5      0x0100      /* 同一属性データ5 */

    loc_rec()

#define F_UNLOCK     0x0000      /* アンロック */
#define F_LOCK       0x0001      /* ロック */
#define F_TSLOCK     0x0002      /* テスト & ロック */
#define F_CKLOCK     0x0003      /* ロック状態のチェック */

    map_rec()

#define F_READ       0x0004      /* R アクセス */
#define F_WRITE      0x0002      /* W アクセス */
#define F_EXCUTE     0x0001      /* E アクセス */
#define F_COMMON     0x0100      /* 共有メモリー空間にマップ */
#define F_SYSTEM     0x0300      /* システムメモリー空間にマップ */

    chk_fil()

#define F_READ       0x0004      /* R アクセスチェック */
#define F_WRITE      0x0002      /* W アクセスチェック */
#define F_EXCUTE     0x0001      /* E アクセスチェック */
#define F_EXIST      0x0000      /* ファイルの存在チェック */

```



```
#define F_PASWD      0x0008      /* パスワードの有無 */
```

chg\_fat()

```
#define F_SETONLY    0x0001      /* 書込不可属性のセット */
#define F_RSTRONLY   0x0002      /* 書込不可属性のリセット*/
#define F_SETPERM    0x0003      /* 削除不可属性のセット */
#define F_RSTPERM    0x0004      /* 削除不可属性のリセット*/
#define F_SETA1      0x0005      /* アプリ属性1のセット */
#define F_RSTA1      0x0006      /* アプリ属性1のリセット */
#define F_SETA2      0x0007      /* アプリ属性2のセット */
#define F_RSTA2      0x0008      /* アプリ属性2のリセット */
```

att\_fls()

```
#define FS_SYNC      0x0000      /* 同期指定 */
#define FS_ASYNC     0x0002      /* 非同期指定 */
#define FS_RDONLY    0x0001      /* 書き込み禁止 */
```

lst\_fls()

```
#define F_GETDEV     (-1)         /* 対応するデバイス名を得る */
#define F_GETNAM     (-2)         /* 対応する接続名を得る */
```

syn\_lnk()

```
#define F_SYNC       0           /* 一致している */
#define F_DNAME      1           /* ファイル名が異なっていた */
#define F_DDATE      2           /* 生成日時が異なっていた */
#define F_DBOTH      3           /* ファイル名と生成日時が異なる */
```

リンク

```
typedef struct {
    TC  fs_name[L_FSNM];          /* ファイルシステム名 */
    UH  f_id;                     /* ファイル ID */
    UH  atr1;                     /* 属性データ1 */
    UH  atr2;                     /* 属性データ2 */
    UH  atr3;                     /* 属性データ3 */
    UH  atr4;                     /* 属性データ4 */
    UH  atr5;                     /* 属性データ5 */
} LINK;
```

ファイルアクセスモード

```

typedef struct {
    UH  f_ownacc;      /* 所有者アクセスモード */
    UH  f_grpacc;     /* グループアクセスレベル */
    UH  f_pubacc;     /* 一般アクセスレベル */
    H   f_grpno;      /* グループ番号 */
} A_MODE;

```

## デフォルトファイルアクセスモード

```

typedef struct {
    UH  f_ownacc;      /* 所有者アクセスモード */
    UH  f_grpacc;     /* グループアクセスレベル */
    UH  f_pubacc;     /* 一般アクセスレベル */
    H   f_grpno;      /* グループ番号 */
    UH  f_gacc[N_GRP]; /* グループアクセスレベル */
} DA_MODE;

```

## ファイル日時

```

typedef struct {
    STIME  f_ltime;      /* ファイルの保存期限(日時) */
    STIME  f_atime;     /* 最新のアクセス日時 */
    STIME  f_mtime;     /* 最新の更新日時 */
} F_TIME;

```

## ファイル管理情報

```

typedef struct {
    UH      f_type;      /* ファイルタイプ/所有者アクセスモード */
    UH      f_atype;     /* アプリケーションタイプ */
    TC      f_owner[L_USRNM]; /* ファイル所有者名 (隠し名は常に0) */
    TC      f_group[L_USRNM]; /* 所有グループ名 (隠し名は常に0) */
    UH      f_grpacc;     /* グループアクセスレベル */
    UH      f_pubacc;     /* 一般アクセスレベル */
    H       f_nlink;     /* 含まれているリンク数 */
    H       f_index;     /* インデックスレベル */
    W       f_size;      /* ファイルの総バイト数 */
    W       f_nblk;      /* 総使用論理ブロック数 */
    W       f_nrec;      /* 総レコード数 */
    STIME   f_ltime;     /* ファイルの保存期限(日時) */
    STIME   f_atime;     /* 最新のアクセス日時 */
    STIME   f_mtime;     /* 最新の更新日時 */
    STIME   f_ctime;     /* ファイルの作成日時 */
} F_STATE;

```

## ファイルタイプ(f\_type)

```

#define F_FILE      0x1000      /* 通常ファイル */

#define F_APLATR1   0x0040      /* アプリケーション属性1 */
#define F_APLATR2   0x0080      /* アプリケーション属性2 */
#define F_RDONLY    0x0010      /* 書込不可属性 */
#define F_PERM      0x0020      /* 削除不可属性 */

#define F_OWNACC    0x0007      /* 所有者アクセス属性(RWE) */
#define F_OWNACR    0x0004      /* 所有者のRアクセス属性 */
#define F_OWNACW    0x0002      /* 所有者のWアクセス属性 */
#define F_OWNACE    0x0001      /* 所有者のEアクセス属性 */

#define F_NOCHG     0x8000      /* 変更しない */

```

## ファイル所在情報

```

typedef struct {
    STIME    fs_ctime;          /* ファイルシステムの生成日時 */
    TC       fs_name[L_FSNM];  /* ファイルシステム名 */
    TC       fs_locat[L_DLNM]; /* デバイス所在名 */
    TC       fs_dev[L_DEVM];   /* 論理デバイス名 */
} F_LOCATE;

```

## リンク情報

```

typedef struct {
    STIME    f_ctime;          /* リンクファイル自体の生成日時 */
    UH       f_atype;          /* 参照ファイルのアプリタイプ */
    TC       f_name[L_FNM];    /* 参照ファイル名 */
    UH       f_id;             /* 参照ファイルID */
    STIME    rf_ctime;         /* 参照ファイルの生成日時 */
    TC       fs_name[L_FSNM];  /* ファイルシステム名 */
    TC       fs_locat[L_DLNM]; /* ファイルシステムのデバイス所在名 */
} F_LINK;

```

## ファイルシステム管理情報

```

typedef struct {
    H        fs_bsize;         /* 論理ブロックのバイト数 */
    UH       fs_nfile;         /* 最大ファイル数 */
    H        fs_lang;          /* ファイルシステムでの使用言語 */
    H        fs_level;         /* ファイルシステムのアクセス管理レベル */
    W        fs_nblk;          /* 全体のブロック数 */
    W        fs_nfree;         /* 未使用ブロックの総数 */
    STIME    fs_mtime;         /* 最新のシステムブロックの更新日時 */
    STIME    fs_ctime;         /* ファイルシステムの生成日時 */
}

```

```
TC      fs_name[L_FSNM];    /* ファイルシステム名 */
TC      fs_locat[L_DLNM];   /* デバイス所在名 */
} FS_STATE;
```

## 使用言語

```
#define F_JPN      0x0021    /* 日本語 (日本語文字族) */
#define F_ENG      0x0080    /* 英語 (ラテン文字族1) */
```

## ファイルシステム接続情報

```
typedef struct {
    TC a_name[L_CONNM];      /* 接続名 */
    TC dev[L_DEVNM];        /* 論理デバイス名 */
} F_ATTACH;
```

## 1.6.8 システムコール

関数のパラメータの説明では、以下に示す記述方法を使用している。

( x y z ) -- x, y, z のいずれか1つを意味する。  
| -- OR で指定可能なことを意味する。  
[ ] -- 省略可能なことを意味する。

例: mode := (F\_NORM F\_BASED) | [F\_DIRECT] の場合、  
mode の指定は、以下の 4 種のいずれか 1 つとなる。

```
F_NORM
F_BASED
F_NORM | F_DIRECT
F_BASED | F_DIRECT
```

## get\_Ink

ファイルのリンク獲得

### 【形式】

```
WERR get_Ink(TC *path, LINK *Ink, W mode)
```

### 【パラメータ】

```
TC *path 対象パス名
NULL 作業ファイルを対象
```

LINK \*Ink 獲得したリンクの格納領域 (出力)  
作業ファイル指定 (入力：F\_BASED 指定時)

W mode リンク獲得モード  
( F\_NORM F\_BASED ) | [ F\_DIRECT ]  
F\_NORM 通常指定  
F\_BASED 作業ファイル指定  
F\_DIRECT 直接リンク獲得指定

### 【リターン値】

=0 正常(通常ファイルのリンク)  
=1 正常(リンクファイルのリンク：F\_DIRECT 指定なし)  
=2 正常(リンクファイルが参照する通常ファイルのリンク：F\_DIRECT 指定時)  
<0 エラー(エラーコード)

### 【解説】

パス名で指定したファイルのリンクを獲得する。パス名の指定が NULL の時は現在の作業ファイルのリンクを獲得する。

パス名が相対パス名るとき、F\_NORM 指定のときは現在の作業ファイルをベースとするが、F\_BASED 指定のときは Ink で指定したファイルを実作業ファイルとみなしてベースとする。

指定したファイルがリンクファイルるとき、F\_DIRECT 指定なしのときはリンクファイル自体へのリンクを獲得する。このとき、得られたリンクファイルが参照する通常ファイルの存在は保証されない。

F\_DIRECT 指定のときはリンクファイルが参照する通常ファイルへの直接のリンクを獲得する。

ファイルのリンクを取り出すためには、パス名に含まれる各ファイルに対しての実行/サーチ(E)アクセス権が必要となるが、対象ファイル自体の実行/サーチ(E)アクセス権は必要ない。

### 【エラーコード】

ER\_ACCES : パス名(path)内の経路ファイルのアクセス権(E)がない。  
ER\_ADR : アドレス(path, Ink)のアクセスは許されていない。  
ER\_FNAME : パス名(path)が空、不正、または長すぎる。  
ER\_IO : 入出力エラーが発生した。  
ER\_NOFS : パス名(path)内のファイル、リンクファイルの参照ファイル(F\_DIRECT指定時)の属するファイルシステムは接続されていない。  
ER\_NOEXS : パス名(path)内のファイル、リンクファイルの参照ファイル(F\_DIRECT指定時)は存在していない、または作業ファイルが未定義。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_PAR : パラメータが不正である(mode が不正)。

# chg\_wrk

作業ファイル変更

## 【形式】

ERR chg\_wrk(LINK \*Ink)

## 【パラメータ】

|      |      |               |
|------|------|---------------|
| LINK | *Ink | 変更する作業ファイル    |
|      | NULL | 作業ファイルを未定義とする |

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

指定したファイルを自プロセスの作業ファイルとする。

作業ファイルとするためには、そのファイルに対しての実行/サーチ(E)アクセス権が必要となる。

## 【エラーコード】

|          |   |                                 |
|----------|---|---------------------------------|
| ER_ACCES | : | ファイル(Ink)のアクセス権(E)がない。          |
| ER_ADR   | : | アドレス(Ink)のアクセスは許されていない。         |
| ER_IO    | : | 入出力エラーが発生した。                    |
| ER_NOEXS | : | ファイル(Ink)は存在していない。              |
| ER_NOFS  | : | ファイル(Ink)の属するファイルシステムは接続されていない。 |
| ER_NOSPC | : | システムのメモリ領域が不足した。                |

# cre\_fil

ファイル生成

## 【形式】

WERR cre\_fil(LINK \*Ink, TC \*name, A\_MODE \*mode, UH atype, W opt)

## 【パラメータ】

LINK \*Ink 生成したファイルのリンク格納領域(出力)  
 ファイルシステム指定 (入力: F\_FLOAT 指定時)  
 親ファイル指定 (入力: F\_FIX 指定時)  
 生成ファイル指定 (入力: F\_FILEID 指定時)

TC \*name ファイル名(0または最大ファイル名文字数まで有効)

A\_MODE \*mode アクセスモード  
 NULL デフォルトアクセスモード適用

UH atype ファイルアプリケーションタイプ

W opt 生成属性  
 ( FLOAT F\_FIX F\_FILEID )  
 F\_FLOAT 浮動リンク指定  
 F\_FIX 固定リンク指定  
 F\_FILEID ファイルID指定

### 【リターン値】

>0 正常(ファイルディスクプリタ)  
 <0 エラー(エラーコード)

### 【解説】

Ink で指定したファイル(リンクファイルのときはリンクファイルが参照する通常ファイル)が存在するファイルシステム上に新規に通常ファイルを生成し、更新用にオープンする。

生成したファイルのリンクの属性データはすべて0に設定され、Ink で指定した領域に格納される。

F\_FLOAT 指定のときは、単純にファイルを生成する。生成したファイルの参照カウントは0となる。この場合、Ink で指定した内容のファイルシステム名のみが有効でありファイルIDは無視されるため Ink で指定したファイルは存在していなくてもよい。

F\_FIX 指定のときは、生成したファイルのリンクを Ink で指定したファイルの最後のレコード位置にリンクレコード(サブタイプ=0)として追加する。生成したファイルの参照カウントは1となる。この場合、Ink で指定したファイルは存在し、かつ書き込みオープンできなくてはならない。

F\_FILEID 指定のときは、Ink で指定した内容のファイルIDと同じファイルIDのファイルを生成する。生成したファイルの参照カウントは0となる。この場合、Ink で指定したファイルは存在してはいけない。

A\_MODE は生成したファイルのアクセスモードを指定する。ファイルの所有者はファイルを生成したプロセスのユーザとなる。

生成したファイルの所有者アクセスモードが書き込み不可のときでも、ファイルは更新用にオープンされる。この状態ではレコードは1つも存在しないため現在レコードは終端レコードであり、そのレコード番号は0となる。

## 【エラーコード】

|           |   |
|-----------|---|
| ER_ACCES  | : ファイル(Ink)のアクセス権(W)がない(F_FIX指定時)。                              |
| ER_ADR    | : アドレス(Ink, name, mode)のアクセスは許されていない。                           |
| ER_BUSY   | : ファイル(Ink)は既に排他的にオープンされている為、同時にファイルをオープンすることができなかった(F_FIX指定時)。 |
| ER_EXS    | : ファイル(Ink)は既に存在している(F_FILEID指定時)。                              |
| ER_FNAME  | : ファイル名(name)は空または不正である。  |
| ER_IO     | : 入出力エラーが発生した。  |
| ER_LIMIT  | : 最大ファイル数を越えた、または同時オープン可能な最大ファイル数を越えた。                          |
| ER_NODSK  | : ディスクの領域が不足した。   |
| ER_NOEXS  | : ファイル(Ink)は存在していない(F_FIX指定時)。                                  |
| ER_NOFS   | : ファイル(Ink)の属するファイルシステムは接続されていない。                               |
| ER_NOSPC  | : システムのメモリ領域が不足した。  |
| ER_PAR    | : パラメータが不正である(f_grpno<0,>4 ,opt が不正)。                           |
| ER_RDONLY | : ファイル(Ink)は書込不可である、または属するファイルシステムは書込不可である。                     |
| ER_SZOVR  | : ファイル(Ink)のサイズがシステムの制限を越えた(F_FIX指定時)。                          |

## cre\_Ink

リンクファイルの生成

### 【形式】

ERR cre\_Ink(LINK \*Ink, F\_LINK \*ref, W opt)

### 【パラメータ】

LINK \*Ink 生成したファイルのリンク格納領域(出力)  
ファイルシステム指定 (入力: F\_FLOAT 指定時)  
親ファイル指定 (入力: F\_FIX 指定時)  
生成ファイル指定 (入力: F\_FILEID 指定時)

F\_LINK \*ref 生成するリンクファイルの内容

W opt 生成属性  
( F\_FLOAT F\_FIX F\_FILEID )

F\_FLOAT:

浮動リンク指定

生成したリンクファイルのリンクを単に Ink に戻す。この場合、生成したリンクファイルの参照カウントは0となる。この指定を行なった場合は、Ink のファイルシステム名のみが使用され、ファイルIDで示されるファイルは実際に存在しなくてもよい。

F\_FIX:



## 固定リンク指定

生成したリンクファイルのリンクを、Ink で指定したファイルの適当な位置のレコードとして追加する。追加されるレコードの位置は保証されない。この場合、生成したリンクファイルの参照カウントは1となる。追加されるリンクレコードのサブタイプは0であり、属性データはすべて0となる。この指定を行なった場合は、Ink で指定したファイルは存在し、かつ F\_WRITE オープンできなくてはならない。

F\_FILEID:

## ファイルID指定

Ink で指定したファイルIDを持つリンクファイルを生成し、生成したリンクファイルのリンクを単に Ink に戻す。この場合、生成したリンクファイルの参照カウントは0となる。この指定を行なった場合は、Ink で指定したファイルIDを持つファイルが既に存在していた場合(リンクファイルも含む)は、エラー (ER\_EXS) となる。

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

Ink で指定したファイル(リンクファイルのときはリンクファイルが参照する通常ファイル)が存在するファイルシステム上に新規に ref で指定した内容のリンクファイルを生成する。

生成したリンクファイルのリンクの属性データはすべて0に設定され、Ink で指定した領域に格納される。

F\_FLOAT, F\_FIX, F\_FILEID の意味は cre\_fil() と同一である。

生成したリンクファイルの内容は ref で指定した内容となるが、その生成日時は、ref->f\_ctime ではなく、本システムコールを実行した日時となる。

ref で指定したファイルの実際の存在はチェックされない。

ref->fs\_name が Ink で指定したファイルシステム名と同一のときはリンクファイルは生成できないためエラーとなる。

## 【エラーコード】

ER\_ACCES : ファイル(Ink)のアクセス権(W)がない(F\_FIX指定時)。  
ER\_ADR : アドレス(Ink, ref)のアクセスは許されていない。  
ER\_BUSY : ファイル(Ink)は既に排他的にオープンされている為、同時にファイルをオープンすることができなかった(F\_FIX指定時)。  
ER\_EXS : ファイル(Ink)は既に存在している(F\_FILEID指定時)。  
ER\_FNAME : ファイル名(ref->f\_name)ファイルシステム名(ref->fs\_name)は空または不正である。  
ER\_IO : 入出力エラーが発生した。  
ER\_LIMIT : 最大ファイル数を越えた。  
ER\_NODSK : ディスクの領域が不足した。

ER\_NOEXS : ファイル(lnk)は存在していない(F\_FIX指定時)。  
ER\_NOFS : ファイル(lnk)の属するファイルシステムは接続されていない。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_PAR : パラメータが不正である(opt が不正、同一ファイルシステム)。  
ER\_RDONLY : ファイル(lnk)は書込不可である、  
または属するファイルシステムは書込不可である(F\_FIX指定時)。  
ER\_SZOVR : ファイル(lnk)のサイズがシステムの制限を越えた(F\_FIX指定時)。

## gen\_fil

ファイルの直接生成

### 【形式】

WERR gen\_fil(LINK \*lnk, TC \*name, F\_STATE \*stat, F\_LINK \*ref, W opt)

### 【パラメータ】

LINK \*lnk 生成したファイルのリンク格納領域(出力)  
ファイルシステム指定 (入力: F\_FLOAT 指定時)  
親ファイル指定 (入力: F\_FIX 指定時)  
生成ファイル指定 (入力: F\_FILEID 指定時)

TC \*name ファイル名(0または最大ファイル文字数まで有効)  
(通常ファイル生成時のみ有効で、このとき name が NULL ならばエラーとなる)  
(リンクファイルの生成の場合は一切参照されない)

F\_STATE \*stat 生成するファイルの内容

F\_LINK \*ref 生成するリンクファイルの内容  
(リンクファイル生成時のみ有効)

W opt 生成属性  
( F\_FLOAT F\_FIX F\_FILEID )  
F\_FLOAT 浮動リンク指定  
F\_FIX 固定リンク指定  
F\_FILEID ファイルID指定

### 【リターン値】

>0 正常(ファイルディスクプリタ: 通常ファイル生成時)  
=0 正常(リンクファイル生成時)  
<0 エラー(エラーコード)

### 【解説】

Ink で指定したファイル (リンクファイルのときはリンクファイルが参照する通常ファイル) が存在するファイルシステム上に新規に通常ファイルまたはリンクファイルを生成し、通常ファイルのときは更新用にオープンする。

生成したリンクファイルのリンクの属性データはすべて 0 に設定され、Ink で指定した領域に格納される。

F\_FLOAT, F\_FIX, F\_FILEID の意味は `cre_fil()` と同一である。

生成したファイルの内容は `stat` で指定し、`stat->f_type` により通常ファイルかリンクファイルか区別される。

通常ファイル生成のときは、`name` で指定した名前の通常ファイルを生成し、生成したファイルの管理情報を `stat` で指定した内容に設定する。ただし、`f_nlink`, `f_index`, `f_size`, `f_nblk`, `f_nrec` の値は無視されファイルの生成時に初期化される。

リンクファイル生成のときは、`stat` の他の内容はすべて無視され、`ref` の内容のリンクファイルを生成する、`cre_Ink()` と同様の動作であるが、`ref->f_ctime` も有効となる。

本システムコールはファイルシステムの復元などの特殊な用途に使用されるため、ユーザレベル 0 のプロセスでのみ実行可能である。

通常ファイルを生成したときはファイルは更新用にオープンされる。この状態ではレコードは 1 つも存在しないため現在レコードは終端レコードであり、そのレコード番号は 0 となる。

## 【エラーコード】

- ER\_ACCES : レベル0のユーザでない。
- ER\_ADR : アドレス(`Ink`, `ref`, `name`, `stat`)のアクセスは許されていない。
- ER\_BUSY : ファイル(`Ink`)は既に排他的にオープンされている為、同時にファイルをオープンすることができなかった(`F_FIX`指定時)。
- ER\_EXS : ファイル(`Ink`)は既に存在している(`F_FILEID`指定時)。
- ER\_FNAME : ファイル名(`name`)、ファイル名(`ref->f_name`)、ファイルシステム名(`ref->fs_name`)は空または不正である。
- ER\_IO : 入出力エラーが発生した。
- ER\_LIMIT : 最大ファイル数を越えた、または同時オープン可能な最大ファイル数を越えた。
- R\_NODSK : ディスクの領域が不足した。
- ER\_NOEXS : ファイル(`Ink`)は存在していない。
- ER\_NOFS : ファイル(`Ink`)の属するファイルシステムは接続されていない。
- ER\_NOSPC : システムのメモリ領域が不足した。
- ER\_PAR : パラメータが不正である  
(`opt` が不正、同一ファイルシステム、`ref`, `stat` の内容が不正)。
- ER\_RDONLY : ファイル(`Ink`)は書込不可である、または属するファイルシステムは書込不可である。
- ER\_SZOVR : ファイル(`Ink`)のサイズがシステムの制限を越えた(`F_FIX`指定時)。

## opn\_fil

ファイルのオープン

## 【形式】

WERR opn\_fil(LINK \*lnk, W o\_mode, TC \*pwd)

## 【パラメータ】

LINK \*lnk 対象ファイル

W o\_mode オープンモード  
( F\_READ F\_WRITE F\_UPDATE ) | [ F\_EXCL F\_WEXCL ]  
F\_READ 読み込み用オープン  
F\_WRITE 書き込み用オープン  
F\_UPDATE 更新(読み込み/書き込み)用オープン  
F\_EXCL 排他モード  
F\_WEXCL 排他書き込みモード

TC \*pwd パスワード  
NULL パスワード指定なし

## 【リターン値】

> 0 正常(ファイルディスクリプタ)  
< 0 エラー(エラーコード)

## 【解説】

lnk で指定したファイルを指定したモードでオープンする。 ファイルをオープンするためにはオープンモードに対応するアクセス権が必要である。

pwd はファイルにパスワードが設定されている場合に有効であり、パスワードが一致しない場合はエラーとなる。

オープンしたファイルの先頭レコードが現在レコードとなる。 レコードが1つも存在しないときは終端レコードが現在レコードとなる。

## 【エラーコード】

ER\_ACCES : ファイル(lnk)のアクセス権(o\_modeに対応)がない。  
ER\_ADR : アドレス(lnk,pwd)のアクセスは許されていない。  
ER\_BUSY : ファイル(lnk)は既に排他的にオープンされている為、同時にファイルをオープンすることができなかった。  
ER\_IO : 入出力エラーが発生した。  
ER\_LIMIT : 同時オープン可能な最大ファイル数を越えた。  
ER\_NOEXS : ファイル(lnk)は存在していない。  
ER\_NOFS : ファイル(lnk)の属するファイルシステムは接続されていない。  
ER\_NOSPC : システムのメモリ領域が不足した。

- ER\_PAR : パラメータが不正である(o\_modeが不正)。  
ER\_PWD : ファイル(lnk)の合言葉が不一致である。  
ER\_RDONLY : ファイル(lnk)は書込不可である、  
または属するファイルシステムは書込不可である。

## cls\_fil

ファイルのクローズ

### 【形式】

ERR cls\_fil(W fd)

### 【パラメータ】

W fd ファイルディスクリプタ

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

オープンしたファイルをクローズする。  
ファイルをオープンしたプロセスが終了したときには、自動的にそのファイルはクローズされる。

### 【エラーコード】

- ER\_FD : ファイルディスクリプタは存在していない。  
ER\_IO : 入出力エラーが発生した。

## del\_fil

ファイルの削除

### 【形式】

WERR del\_fil(LINK \*org, LINK \*lnk, W force)

### 【パラメータ】

LINK \*org 削除対象ファイルの親ファイル  
NULL 親ファイル指定なし

LINK \*Ink 削除対象ファイル

W force 強制削除指定

= 0 :

削除対象ファイルがリンクレコードを含むときはエラーとして削除しない。

0 :

削除対象ファイルがリンクレコードを含むときも削除し、含まれるリンクレコードが示すファイルの参照カウントを -1 し、その結果、参照カウントが 0 となったリンクレコードの数をリターン値とする。

### 【リターン値】

0 正常(削除した結果参照カウント0となったリンクレコードの数)  
< 0 エラー(エラーコード)

### 【解説】

org で指定した親ファイル内の Ink で指定したファイルを示すリンクレコードを削除し、そのファイルの参照カウントを -1 する。その結果、参照カウントが 0 になった場合には、Ink で指定されたファイル自体を削除する。この場合、親ファイルの書き込み(W)アクセス権が必要となる。

親ファイルの指定なし (org = NULL) のときは、Ink で指定したファイルの参照カウントが 0 の場合にファイルを削除する。参照カウントが 0 でない場合はエラーとなる。

削除対象ファイルがリンクファイルのときは、リンクファイル自体が削除の対象となり、リンクファイルの参照先のファイルは削除されることはない。

削除対象ファイルが以下のいずれかのときは削除されずにエラーとなる。

- 削除不可属性が設定されているとき
- オープンしているプロセスが存在しているとき
- 作業ファイルとしているプロセスが存在しているとき

書込不可属性がセットされていた場合でも削除可能である。

### 【エラーコード】

ER\_ACCES : ファイル(org)のアクセス権(W)がない(org NULLの時)。  
ER\_ADR : アドレス(org, Ink)のアクセスは許されていない。  
ER\_BUSY : ファイル(org)は既に排他的にオープンされている為、同時にファイルをオープンすることができなかった(org NULLの時)。  
: ファイル(Ink)はオープン中である、または作業ファイルである。  
: ファイル(Ink)の参照カウントは0でない(org=NULLの時)。  
: ファイル(Ink)を示すリンクレコードは他のオープンで現在レコードとして使用されている(org=NULLの時)。

ER\_IO : 入出力エラーが発生した。  
ER\_LOCK : ファイル(lnk)を示すリンクレコードは、他からロックされている (org NULLの時)。  
ER\_NOEXS : ファイル(org, lnk)は存在していない(または、org内に指定されたファイル(lnk)を示すリンクレコードが存在しない)。  
ER\_NOFS : ファイル(org, lnk)の属するファイルシステムは接続されていない。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_PERM : ファイル(lnk)は削除不可である(削除不可属性がセットされている)。  
ER\_REC : ファイル(lnk)は リンクレコードを含んでいる(force=0の時)。  
ER\_RDONLY : ファイル(org)は書込不可である、  
または属するファイルシステムは書込不可である(org NULLの時)。  
: ファイル(lnk)の属するファイルシステムは書込不可である。

## see\_rec

現在レコード移動

### 【形式】

ERR see\_rec(W fd, W offset, W mode, W \*recnum)

### 【パラメータ】

W fd       ファイルディスクリプタ

W offset   移動オフセット

W mode     移動モード  
=0    現在レコード番号 + offset のレコード番号位置に移動。  
>0    offset のレコード番号位置に移動。  
      offset 0でなくてはならない。  
<0    終端レコード番号 + offset のレコード番号位置に移動。  
      offset 0でなくてはならない。

W \*recnum  移動後の現在レコード番号の格納領域  
          NULL   格納しない

### 【リターン値】

=0    正常  
<0    エラー(エラーコード)

### 【解説】

オープンしたファイルの現在レコード位置を指定した位置に移動する。

定した移動先が存在するレコードの範囲を越える場合はエラーとなり、現在レコードは変化しない。

## 【エラーコード】

ER\_ADR : アドレス(recnum)のアクセスは許されていない。  
ER\_FD : ファイルディスクリプタは存在していない。  
ER\_IO : 入出力エラーが発生した。  
ER\_REC : 存在するレコードの範囲を越えた。

# fnd\_rec

## レコード検索

## 【形式】

WERR fnd\_rec(W fd, W mode, UW typemask, UH subtype, W \*recnum)

## 【パラメータ】

W fd ファイルディスクリプタ

W mode 検索モード(検索開始位置 / 方向の指定)  
( F\_FWD F\_NFWD F\_BWD F\_NBWD F\_TOPEND F\_ENDTOP )  
F\_FWD 現在レコードから終端レコードまで  
F\_NFWD 現在レコードの次のレコードからから終端レコードまで  
F\_BWD 現在レコードから先頭レコードまで  
F\_NBWD 現在レコードの前のレコードから先頭レコードまで  
F\_TOPEND 先頭レコードから終端レコードまで  
F\_ENDTOP 終端レコードから先頭レコードまで

UW typemask 検索対象レコードタイプのビットマスク  
LSB タイプ 0 に対応  
MSB タイプ 31 に対応

UH subtype 検索対象レコードサブタイプ  
0 全サブタイプが対象(サブタイプのチェックなし)

W \*recnum 検索結果の現在レコード番号の格納領域  
NULL 格納しない

## 【リターン値】

0 正常(検索したレコードタイプ)  
<0 エラー(エラーコード)



## 【解説】

オープンしたファイル内の指定したレコードを検索し、見つけたレコードを現在レコードとする。

対象レコードが見つからなかったときはエラーとなり、現在レコードは変化しない。

## 【エラーコード】

ER\_ADR : アドレス(recnum)のアクセスは許されていない。  
ER\_FD : ファイルディスクリプタは存在していない。  
ER\_IO : 入出力エラーが発生した。  
ER\_PAR : パラメータが不正である(modeが不正)。  
ER\_REC : 指定した検索条件に合うレコードは存在しない  
(typemask=0の場合も含む)。

# fnd\_Ink

リンクレコード検索

## 【形式】

ERR fnd\_Ink(W fd, W mode, LINK \*Ink, UH subtype, W \*recnum)

## 【パラメータ】

W fd           ファイルディスクリプタ

W mode          検索モード(検索開始位置/方向/内容の指定)  
( F\_FWD    F\_NFWD    F\_BWD    F\_NBWD    F\_TOPEND    F\_ENDTOP )  
| [ F\_SFILE ] | [ F\_SNAME ]  
| [ F\_SATR1 ] | [ F\_SATR2 ] | [ F\_SATR3 ] |  
              [ F\_SATR4 ] | [ F\_SATR5 ]  
F\_FWD ~ F\_ENDTOP    fnd\_rec()と同じ  
F\_SFILE    Ink と同一のファイルを示すリンクレコード  
F\_SNAME    Ink と同一のファイル名のファイルを示すリンクレコード  
F\_SATR1    Ink と同一の属性データ1を持つリンクレコード  
F\_SATR2    Ink と同一の属性データ2を持つリンクレコード  
F\_SATR3    Ink と同一の属性データ3を持つリンクレコード  
F\_SATR4    Ink と同一の属性データ4を持つリンクレコード  
F\_SATR5    Ink と同一の属性データ5を持つリンクレコード

LINK \*Ink      検索対象リンク  
              F\_SFILE ~ F\_SATR5 を指定したときのみ有効

UH subtype     検索対象レコードサブタイプ  
              0 全サブタイプが対象(サブタイプのチェックなし)

W \*recnum 検索結果の現在レコード番号の格納領域  
NULL 格納しない

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

オープンしたファイル内の指定したリンクレコードを検索し、見つけたリンクレコードを現在レコードとする。

対象レコードが見つからなかったときはエラーとなり、現在レコードは変化しない。

### 【エラーコード】

ER\_ADR : アドレス(Ink, recnum)のアクセスは許されていない  
(Inkは検索条件を指定した場合のみアクセスされる)。  
ER\_FD : ファイルディスクリプタは存在していない。  
ER\_IO : 入出力エラーが発生した。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_NOEXS : ファイル(Ink)は存在していない。  
ER\_NOFS : ファイル(Ink)の属するファイルシステムは接続されていない。  
ER\_PAR : パラメータが不正である(mode が不正)。  
ER\_REC : 指定した検索条件に合うレコードは存在しない。

## rea\_rec

レコード読み込み

### 【形式】

WERR rea\_rec(W fd, W offset, B \*buf, W size, W \*r\_size, UH \*subtype)

### 【パラメータ】

W fd ファイルディスクリプタ  
W offset 読み込み開始バイト位置( 0)  
B \*buf 読み込みデータ格納領域  
NULL 格納しない  
W size 読み込みデータ格納領域のバイトサイズ( 0)

W \*r\_size 開始バイト位置からの残りバイトサイズ  
(レコードサイズ - offset)の格納領域  
NULL 格納しない

UH \*subtype レコードサブタイプの格納領域  
NULL 格納しない

### 【リターン値】

0 正常(現在レコードのレコードタイプ)  
<0 エラー(エラーコード)

### 【解説】

オープンしたファイルの現在レコードを読み込む。

レコードサイズ < offset + size のときは、buf には (レコードサイズ - offset) バイトのデータのみ読み込まれて格納される。

offset レコードサイズ、buf = NULL、または size = 0 のときは buf には何も格納されずに \*r\_size、\*subtype に対応する値が格納される。これは、レコードサイズやサブタイプのみを取り出す場合に使用される。

現在レコードがリンクレコードのときは、LINK 構造体全体の内容が buf に読み出され、\*r\_size には LINK 構造体のサイズが格納される。この場合、offset = 0、size LINK 構造体のサイズ (または size = 0) でなくてはならない。

現在レコードが終端レコードのとき、または他プロセスからロックされているときはエラーとなる。

### 【エラーコード】

ER\_ADR : アドレス(buf, r\_size, subtype)のアクセスは許されていない。  
ER\_ENDR : 現在レコードは終端レコードである。  
ER\_FD : ファイルディスクリプタは存在していない、  
または F\_WRITE オープンである。  
ER\_IO : 入出力エラーが発生した。  
ER\_LOCK : 現在レコードは他からロックされている。  
ER\_PAR : パラメータが不正である(size<0,offset<0,  
リンクレコードでoffset,sizeが不正)。

## wri\_rec

レコード書き込み

### 【形式】

ERR wri\_rec(W fd, W offset, B \*buf, W size, W \*r\_size, UH \*subtype, UW units)

## 【パラメータ】

|    |          |   |
|----|----------|---|
| W  | fd       | ファイルディスクリプタ   |
| W  | offset   | 書き込み開始バイト位置(-1 offset < レコードサイズ)<br>-1: レコードの最後への書き込み(追加)         |
| B  | *buf     | 書き込みデータへのポインタ<br>NULL 書き込まない                                      |
| W  | size     | 書き込みデータのバイトサイズ( 0)  |
| W  | *r_size  | 開始バイト位置からの残りバイトサイズ<br>(書き込み後のレコードサイズ - offset)の格納領域<br>NULL 格納しない |
| UH | *subtype | 変更するレコードサブタイプへのポインタ<br>NULL 変更しない                                 |
| UW | units    | ブロック獲得単位(Kバイト)<br>0 任意  |

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

オープンしたファイルの現在レコードに書き込む。

レコードサイズ < offset + size のときは、書き込み後にレコードサイズは増加する。

units はレコードサイズが増加したために必要となった追加ブロックの獲得の単位をKバイト単位で指定するもので、units サイズ以上 (size 以下) の連続ブロック領域を割り当てることを指定する。

units = 0 はブロックの割り当て方法は任意であることを意味する。

size = 0、または buf = NULL のときは、データの書き込みは行なわれないが subtype NULL であればレコードサブタイプは変更する。

buf = NULL でレコードサイズ < offset + size のときは、レコードサイズを増加するが増加した部分のデータは不定となる。これは units 指定と組み合わせてレコードの追加ブロック領域をあらかじめ確保するとき使用する。

offset = -1 のときは、常にその時点のレコードの最後にデータを書き込み、\*r\_size には size の値が格納される。同一レコードを複数のプロセスがオープンして同時に書き込む場合でもこの指定により別のプロセスが書き込んだデータを上書きしないことが保証される。

現在レコードがリンクレコードのときは buf の内容は LINK 構造体となるが、属性データ部分のみが書き込まれ、参照するファイル自体を変更することはできない。この場合、offset = 0、size LINK 構造体のサイズ (または size = 0) でなくてはならない。

現在レコードが終端レコードのとき、または他プロセスからロックされているときはエラーとなる。

### 【エラーコード】

- ER\_ADR : アドレス(buf, r\_size, subtype)のアクセスは許されていない。
- ER\_ENDR : 現在レコードは終端レコードである。
- ER\_FD : ファイルディスクリプタは存在していない、または F\_READ オープンである。
- ER\_IO : 入出力エラーが発生した。
- ER\_LOCK : 現在レコードは他からロックされている。
- ER\_NODSK : ディスクの領域が不足した、または指定された連続ブロック領域が獲得できなかった。
- ER\_PAR : パラメータが不正である(size<0, offsetが不正, リンクレコードで offset, sizeが不正)。
- ER\_SZOVR : ファイルのサイズがシステムの制限を越えた。

## ins\_rec

レコード挿入

### 【形式】

ERR ins\_rec(W fd, B \*buf, W size, W type, UH subtype, UW units)

### 【パラメータ】

- W fd : ファイルディスクリプタ
- B \*buf : 挿入レコードのデータへのポインタ  
NULL : データは書き込まない
- W size : 挿入レコードのバイトサイズ( 0)
- W type : 挿入レコードのレコードタイプ
- UH subtype : 挿入レコードのレコードサブタイプ
- UW units : ブロック獲得単位(Kバイト)  
0 : 任意

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

オープンしたファイルの現在レコードの直前に新規のレコードを挿入する。

units は挿入したレコードで必要とするブロックの獲得の単位を K バイト単位で指定するもので、units サイズ以上 (size 以下) の連続ブロック領域を割り当てることを指定する。units = 0 はブロックの割り当て方法は任意であることを意味する。

buf = NULL のときは、挿入したレコードのサイズは size となるが、そのデータは不定となる。これは units 指定と組み合わせて、レコードのブロック領域をあらかじめ確保するために使用する。

現在レコードがリンクレコードのときは buf の内容は LINK 構造体となるが、属性データ部分のみが書き込まれ、参照するファイル自体を変更することはできない。この場合、offset = 0、size LINK 構造体のサイズ (または size = 0) でなくてはならない。

type = 0 のときはリンクレコードの挿入であり、buf の内容は LINK 構造体となる。リンクレコードの挿入により、そのリンクの示すファイルの参照カウントは +1 される。この場合、buf NULL、size = LINK 構造体のサイズ、かつリンクの示すファイルは同一ファイルシステム上に存在していなくてはならない。

### 【エラーコード】

ER\_ADR : アドレス(buff)のアクセスは許されていない。  
ER\_FD : ファイルディスクリプタは存在していない、  
または F\_READ オープンである。  
ER\_IO : 入出力エラーが発生した。  
ER\_LIMIT : リnkの参照先のファイルの参照カウントがシステムの制限(255)を越えた。  
ER\_NODSK : ディスクの領域が不足した、または指定された連続ブロック領域が  
獲得できなかった。  
ER\_NOEXS : リnkの参照先のファイルは存在していない。  
ER\_PAR : パラメータが不正である(type が不正、size<0、units が不正、  
type=0の時の size, buff が不正)。  
ER\_REC : リnkは別ファイルシステムを参照している。  
ER\_SZOVR : ファイルのサイズがシステムの制限を越えた。

apd\_rec

レコード追加

### 【形式】

ERR apd\_rec(W fd, B \*buf, W size, W type, UH subtype, UW units)

### 【パラメータ】

|    |         |                                     |
|----|---------|-------------------------------------|
| W  | fd      | ファイルディスクリプタ                         |
| B  | *buf    | 追加レコードのデータへのポインタ<br>NULL データは書き込まない |
| W  | size    | 追加レコードのバイトサイズ( 0)                   |
| W  | type    | 追加レコードのレコードタイプ                      |
| U  | subtype | 追加レコードのレコードサブタイプ                    |
| UW | units   | ブロック獲得単位(Kバイト)<br>0 任意              |

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

オープンしたファイルの最後に新規のレコードを挿入する。

本システムコールは、現在レコードの位置の関係なく常に最後のレコード(終端レコードの直前)にレコードを挿入する点を除いて ins\_rec() と同一である。

### 【エラーコード】

|          |   |   |
|----------|---|---|
| ER_ADR   | : | アドレス(buff)のアクセスは許されていない。  |
| ER_FD    | : | ファイルディスクリプタは存在していない、または F_READ オープンである。                           |
| ER_IO    | : | 入出力エラーが発生した。  |
| ER_LIMIT | : | リンクの参照先のファイルの参照カウントがシステムの制限(255)を越えた。                             |
| ER_NODSK | : | ディスクの領域が不足した、または指定された連続ブロック領域が獲得できなかった。                           |
| ER_NOEXS | : | リンクの参照先のファイルは存在していない。   |
| ER_PAR   | : | パラメータが不正である(type が不正、size<0、units が不正、type =0の時の size, buff が不正)。 |
| ER_REC   | : | リンクは別ファイルシステムを参照している。   |
| ER_SZOVR | : | ファイルのサイズがシステムの制限を越えた。   |

## del\_rec

レコード削除

## 【形式】

WERR del\_rec(W fd)

## 【パラメータ】

W fd ファイルディスクリプタ

## 【リターン値】

= 1 正常(リンクレコードを削除した結果、参照カウンタ=0となった)

= 0 正常(上記以外)

< 0 エラー(エラーコード)

## 【解説】

オープンしたファイルの現在レコードを削除し、削除した次のレコードに現在レコードをを移動する。

削除したレコードがリンクレコードのときは、リンクレコードが示すファイルの参照カウンタは -1 され、その結果参照カウンタが0となった場合はリターン値は 1 となる。

現在レコードが終端レコードのとき、他プロセスからロックされているとき、または他のオープンで現在レコードとなっているときはエラーとなる。

## 【エラーコード】

ER\_BUSY : 他のオープンで現在レコードとして使用されている。

ER\_ENDR : 現在レコードは終端レコードである。

ER\_FD : ファイルディスクリプタは存在していない、  
または F\_READ オープンである。

ER\_IO : 入出力エラーが発生した。

ER\_LOCK : 現在レコードは他からロックされている。

ER\_NOSPC : システムのメモリ領域が不足した。

**trc\_rec**

レコードサイズ縮小

## 【形式】

ERR trc\_rec(W fd, W size)

## 【パラメータ】



W fd ファイルディスクリプタ

W size 縮小するレコードバイトサイズ( 0)

### 【リターン値】

= 0 正常

< 0 エラー(エラーコード)

### 【解説】

オープンしたファイルの現在レコードのレコードサイズを size バイトに縮小する。レコードサイズ size のときは何もしない。

現在レコードが終端レコードのとき、リンクレコードのとき、または他プロセスからロックされているときはエラーとなる。

### 【エラーコード】

ER\_ENDR : 現在レコードは終端レコードである。  
ER\_FD : ファイルディスクリプタは存在していない、  
または F\_READ オープンである。  
ER\_IO : 入出力エラーが発生した。  
ER\_LOCK : 現在レコードは他からロックされている。  
ER\_PAR : パラメータが不正である(size<0)。  
ER\_REC : 現在レコードはリンクレコードである。

## xch\_fil

ファイルの内容交換

### 【形式】

ERR xch\_fil(W fd\_1, W fd\_2)

### 【パラメータ】

W fd\_1 ファイルディスクリプタ 1

W fd\_2 ファイルディスクリプタ 2

### 【リターン値】

= 0 正常

<0 エラー(エラーコード)

【解説】

オープンした2つのファイルの内容を交換する。

交換するのはファイルのデータ部分であり、ファイルの管理情報はアクセス日付と更新日付を除いて元のままとなる。

交換する2つのファイルは、同一のファイルシステムに存在し、かつ排他モードで更新用オープンしてなくてはならない。

交換後の現在レコードは、それぞれ先頭レコードとなる。

【エラーコード】

- ER\_FD : ファイルディスクリプタは存在していない、または (F\_UPDATE|F\_EXCL) でない オープンである。
- ER\_IO : 入出力エラーが発生した。
- ER\_LOCK : 現在レコードは他からロックされている (レコードにロックがかかっている)。
- ER\_NOSPC : システムのメモリ領域が不足した。
- ER\_PAR : パラメータが不正である (fd\_1とfd\_2は同一ファイル)。
- ER\_XFS : ファイル(fd\_1, fd\_2)は異なるファイルシステムに属している。

## loc\_rec

レコードロック

【形式】

ERR loc\_rec(W fd, W mode)

【パラメータ】

- W fd      ファイルディスクリプタ
  
- W mode    ロックモード  
( F\_UNLOCK    F\_LOCK    F\_TSLOCK    F\_CKLOCK )  
F\_LOCK      ロック設定(待ち)  
F\_UNLOCK    ロック解除  
F\_TSLOCK    ロック設定(待ちなし)  
F\_CKLOCK    ロック状態チェック

【リターン値】

=0    正常

## <0 エラー(エラーコード)

### 【解説】

オープンしたファイルの現在レコードに対するロック操作を行なう。

#### F\_LOCK ロック設定(待ち)

他プロセスからロックされていたときはロックが解除されるまで待つ(待ちはプロセス優先度順で同一優先度の場合は待ちに入った順)。自プロセスの同一ファイルディスクリプタからロックしていた場合は何もせずに正常終了。自プロセスの他ファイルディスクリプタからロックしていた場合はエラー。

#### F\_UNLOCK ロック解除

レコードがロックされていない場合は何もせずに正常終了。自プロセスの同一ファイルディスクリプタからロックした場合のみ解除可能で、その他の場合はエラー。

#### F\_TSLOCK ロック設定(待ちなし)

他プロセスまたは他ファイルディスクリプタからロックされていたときはエラー。

#### F\_CKLOCK ロック状態チェック

他プロセスまたは他ファイルディスクリプタからロックされていたときはエラー、そうでないときは何もせずに正常終了。

ロックしたレコードは、ロックしたファイルディスクリプタ以外からの読み込み、書き込み、サイズ変更、および削除が禁止される。

ファイルをクローズ時にはオープンしたプロセスで設定したロックは解除される。

### 【エラーコード】

- ER\_ENDR : 現在レコードは終端レコードである。
- ER\_FD : ファイルディスクリプタは存在していない。
- ER\_IO : 入出力エラーが発生した。
- ER\_LIMIT : 同時にロック可能なレコードの最大数を越えた。
- ER\_LOCK : 現在レコードは他からロックされている現在レコードは既に他からロックされている (F\_TSLOCK / F\_CKLOCK 指定時)。  
: 自プロセスの他ファイルディスクリプタから既にロックされている (F\_LOCK 指定時)。  
: 他ファイルディスクリプタからのロックであり解除できない (F\_UNLOCK 指定時)。
- ER\_MINTR : メッセージハンドラが起動されたため待ち処理が中断された (F\_LOCK 指定時)。
- ER\_NOSPC : システムのメモリ領域が不足した。
- ER\_PAR : パラメータが不正である (mode が不正)。

## chk\_fil

ファイルアクセス権チェック

### 【形式】

WERR chk\_fil(LINK \*Ink, W mode, TC \*pwd)

## 【パラメータ】

LINK \*Ink 対象ファイル

W mode チェックモード  
( [ F\_READ ] | [ F\_WRITE ] | [ F\_EXCUTE ] ) [ F\_EXIST ]  
F\_READ 読み込み(R)アクセス権チェック  
F\_WRITE 書き込み(W)アクセス権チェック  
F\_EXCUTE 実行/サーチ(E)アクセス権チェック  
F\_EXIST ファイルの存在チェック

TC \*pwd パスワード(F\_READ または F\_WRITE 指定時のみ有効)  
NULL パスワード指定なし

## 【リターン値】

0 正常(ファイルのアクセス情報 : F\_EXIST 指定時)  
=0 正常(F\_EXIST 以外指定時)  
<0 エラー(エラーコード)

## 【解説】

指定したファイルの指定したアクセスが可能か否かのチェックを行なう。

F\_READ、F\_WRITE、F\_EXCUTE の組み合わせで指定したアクセスが不可のときはエラーとなる。パスワードはF\_READ、またはF\_WRITE を指定したときのみチェックされる。

F\_EXIST 指定のときは、ファイルが存在しない場合はエラーとなり、存在する場合は以下のアクセス情報をリターン値として戻す。

0.....0 BAPO SRWE

B: アプリケーション属性2 (1:ON、0:OFF)  
A: アプリケーション属性1 (1:ON、0:OFF)  
P: 削除不可属性 (1:ON、0:OFF)  
O: 書込不可属性 (1:ON、0:OFF)  
S: パスワードの有無 (1:有、0:無 )  
R: 読み込み(R)アクセス権 (1:有、0:無 )  
W: 書き込み(W)アクセス権 (1:有、0:無 )  
E: 実行/サーチ(E)アクセス権 (1:有、0:無 )

## 【エラーコード】

ER\_ACCES : ファイル(Ink)のアクセス権(F\_EXIST以外指定時)がない。  
ER\_ADR : アドレス(Ink)のアクセスは許されていない。  
ER\_IO : 入出力エラーが発生した。

ER\_NOEXS : ファイル(lnk)は存在していない。  
ER\_NOFS : ファイル(lnk)の属するファイルシステムは接続されていない。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_PAR : パラメータが不正である(modeが不正)。  
ER\_PWD : ファイル(lnk)の合言葉が不一致である(F\_READ/F\_WRITE指定時)。  
ER\_RDONLY : ファイル(F\_WRITE指定時)の書込不可属性が設定されている、  
または属するファイルシステムは書込不可である。

## chg\_fmd

ファイルアクセスモード変更

### 【形式】

ERR chg\_fmd(LINK \*lnk, A\_MODE \*mode)

### 【パラメータ】

LINK \*lnk 対象ファイル

A\_MODE \*mode 変更するアクセスモード

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

指定したファイルのアクセスモードを変更する。

アクセスモードの以下のそれぞれのデータに対して F\_NOCHG 指定を行うと、その項目は変更しないことを意味する。

- 所有者アクセスモード(f\_ownac)
- グループアクセスレベル(f\_grpacc)
- 一般アクセスレベル(f\_pubacc)
- 所有グループ番号(f\_grpno)

アクセスモードの変更は、ファイルシステムのアクセスレベルが 0 のときは誰でも変更可能であるが、アクセスレベルが 0 でないときはファイルの所有者のプロセスのみ変更可能となる。

すでにオープンされているファイルのアクセスモードを変更した場合、その変更はすでにオープンされているものには影響を与えない。

### 【エラーコード】

ER\_ACCES : ファイル(lnk)の所有者でない、またはレベル0のユーザでない。  
ER\_ADR : アドレス(lnk,mode)のアクセスは許されていない。  
ER\_IO : 入出力エラーが発生した。  
ER\_NOEXS : ファイル(lnk)は存在していない。  
ER\_NOFS : ファイル(lnk)の属するファイルシステムは接続されていない。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_PAR : パラメータが不正である(modeの内容が不正)。  
ER\_RDONLY : ファイル(lnk)は書込不可である、  
または属するファイルシステムは書込不可である。

## chg\_fat

ファイルのアクセス属性変更

### 【形式】

ERR chg\_fat(LINK \*lnk, W attr)

### 【パラメータ】

LINK \*lnk 対象ファイル

W attr 変更するアクセス属性  
( F\_SETONLY F\_RSTRONLY F\_SETPERM F\_RSTPERM  
F\_SETA1 F\_RSTA1 F\_SETA2 F\_RSTA2 )  
F\_SETONLY 書き込み不可属性のセット  
F\_RSTRONLY 書き込み不可属性のリセット  
F\_SETPERM 削除不可属性のセット  
F\_RSTPERM 削除不可属性のリセット  
F\_SETA1 アプリケーション属性1のセット  
F\_RSTA1 アプリケーション属性1のリセット  
F\_SETA2 アプリケーション属性2のセット  
F\_RSTA2 アプリケーション属性2のリセット

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

指定したファイルのアクセス属性を変更する。

アクセスモードの変更は、ファイルシステムのアクセスレベルが0のときは誰でも変更可能であるが、アクセスレベルが0でないときはファイルの所有者のプロセスのみ変更可能となる。

すでにオープンされているファイルのアクセスモードを変更した場合、その変更はすでにオー

プンされているものには影響を与えない。

### 【エラーコード】

ER\_ACCES : ファイル(lnk)の所有者でない、またはレベル0のユーザでない。  
ER\_ADR : アドレス(lnk)のアクセスは許されていない。  
ER\_IO : 入出力エラーが発生した。  
ER\_NOEXS : ファイル(lnk)は存在していない。  
ER\_NOFS : ファイル(lnk)の属するファイルシステムは接続されていない。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_PAR : パラメータが不正である(attrが不正)。  
ER\_RDONLY : ファイル(lnk)は書込不可である、または属するファイルシステムは書込不可である。

## chg\_fpw

ファイルパスワード変更

### 【形式】

ERR chg\_fpw(W fd, TC \*pwd)

### 【パラメータ】

W fd ファイルディスクリプタ

TC \*pwd パスワード(TNULL、または最大パスワード文字数まで有効)  
NULL パスワード解除

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

オープンしたファイルのパスワードを変更する。

パスワードの変更は、ファイルシステムのアクセスレベルが0のときは誰でも変更可能であるが、アクセスレベルが0でないときはファイルの所有者のプロセスのみ変更可能となる。ただし、ファイルの所有者が定義されていない場合は、ユーザーレベル0のプロセスからも変更可能となる。

### 【エラーコード】

ER\_ACCES : ファイルの所有者ではない、またはレベル0のユーザでない。  
ER\_FD : ファイルディスクリプタは存在していない。  
ER\_ADR : アドレス(pwd)のアクセスは許されていない。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_IO : 入出力エラーが発生した。  
ER\_RDONLY : ファイルは書き込み不可である。  
または属するファイルシステムは書き込み不可である。

## chg\_fnm

ファイル名変更

### 【形式】

ERR chg\_fnm(LINK \*Ink, TC \*name)

### 【パラメータ】

LINK \*Ink 対象ファイル

TC \*name 変更するファイル名(TNULL、または最大ファイル名文字数まで有効)

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

指定したファイルのファイル名を変更する。

ファイル名の変更は、ファイルシステムのアクセスレベルが0のときは誰でも変更可能であるが、アクセスレベルが0でないときはファイルの所有者のプロセスのみ変更可能となる。

書き込み不可属性、または削除不可属性のセットされているファイルのファイル名の変更はできない。

指定したファイルがリンクファイルのときは、参照先のファイル名およびリンクファイル内に保持されている参照ファイル名の両方が変更される。

### 【エラーコード】

ER\_ACCES : ファイル(Ink)の所有者でない、またはレベル0のユーザでない。  
ER\_ADR : アドレス(Ink,name)のアクセスは許されていない。  
ER\_IO : 入出力エラーが発生した。  
ER\_NOEXS : ファイル(Ink)は存在していない。  
ER\_NOFS : ファイル(Ink)の属するファイルシステムは接続されていない。



ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_FNAME : ファイル名(name)は空または不正である。  
ER\_PERM : ファイル(lnk)は削除不可である(削除不可属性がセットされている)。  
ER\_RDONLY : ファイル(lnk)は書込不可である、  
または属するファイルシステムは書込不可である。

## chg\_ftm

ファイル日時変更

### 【形式】

ERR chg\_ftm(LINK \*lnk, F\_TIME \*times)

### 【パラメータ】

|        |        |         |
|--------|--------|---------|
| LINK   | *lnk   | 対象ファイル  |
| F_TIME | *times | 変更する日時  |
|        | NULL   | 現在日時に設定 |

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

指定したファイルの保存期限、最新アクセス日時、および最新更新日時を変更する。

F\_TIME のそれぞれの値が 0 のときはその項目は変更しない。

ファイル日時の変更は、ファイルシステムのアクセスレベルが 0 のときは誰でも変更可能であるが、アクセスレベルが 0 でないときはファイルの所有者のプロセスのみ変更可能となる。

### 【エラーコード】

ER\_ACCES : ファイル(lnk)の所有者でない、またはレベル0のユーザでない。  
ER\_ADR : アドレス(lnk, times)のアクセスは許されていない。  
ER\_IO : 入出力エラーが発生した。  
ER\_NOEXS : ファイル(lnk)は存在していない。  
ER\_NOFS : ファイル(lnk)の属するファイルシステムは接続されていない。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_RDONLY : ファイル(lnk)は書込不可である、  
または属するファイルシステムは書込不可である。

# fil\_sts

ファイル情報の取得

## 【形式】

WERR    fil\_sts(LINK \*Ink, TC \*name, F\_STATE \*stat, F\_LOCATE \*locat)

## 【パラメータ】

LINK    \*Ink    対象ファイル

TC       \*name    ファイル名の格納領域(最大ファイル名+1文字分の領域)  
         NULL    格納しない

F\_STATE \*stat    ファイル管理情報の格納領域  
         NULL    格納しない

F\_LOCATE \*locat ファイル所在情報の格納領域  
         NULL    格納しない

## 【リターン値】

0        正常(ファイルの参照カウント)  
<0       エラー(エラーコード)

## 【解説】

指定したファイルの情報を取り出す。

## 【エラーコード】

ER\_ADR        : アドレス(Ink,name,stat,locat)のアクセスは許されていない。  
ER\_FD         : ファイルディスクリプタは存在していない(ofl\_stsの場合)。  
ER\_IO         : 入出力エラーが発生した。  
ER\_NOEXS      : ファイル(Ink)は存在していない(fil\_stsの場合)。  
ER\_NOFS       : ファイル(Ink)の属するファイルシステムは接続されていない  
               (fil\_stsの場合)。  
ER\_NOSPC      : システムのメモリ領域が不足した。

# ofl\_sts

ファイル情報の取得

## 【形式】

WERR ofl\_sts(W fd, TC \*name, F\_STATE \*stat, F\_LOCATE \*locat)

## 【パラメータ】

|          |        |   |
|----------|--------|---|
| W        | fd     | ファイルディスクリプタ                                 |
| TC       | *name  | ファイル名の格納領域(最大ファイル名 + 1文字分の領域)<br>NULL 格納しない |
| F_STATE  | *stat  | ファイル管理情報の格納領域<br>NULL 格納しない                 |
| F_LOCATE | *locat | ファイル所在情報の格納領域<br>NULL 格納しない                 |

## 【リターン値】

0 正常(ファイルの参照カウント)  
<0 エラー(エラーコード)

## 【解説】

オープンしたファイルの情報を取り出す。

## 【エラーコード】

ER\_ADR : アドレス(Ink,name,stat,locat)のアクセスは許されていない。  
ER\_FD : ファイルディスクリプタは存在していない(ofl\_stsの場合)。  
ER\_IO : 入出力エラーが発生した。  
ER\_NOEXS : ファイル(Ink)は存在していない(fil\_stsの場合)。  
ER\_NOFS : ファイル(Ink)の属するファイルシステムは接続されていない  
(fil\_stsの場合)。  
ER\_NOSPC : システムのメモリ領域が不足した。

# Ink\_sts

リンクファイル情報の取得

## 【形式】

WERR Ink\_sts(LINK \*Ink, F\_LINK \*stat)

## 【パラメータ】

LINK \*Ink 対象リンクファイル

F\_LINK \*stat リンクファイル情報の格納領域  
NULL 格納しない

### 【リターン値】

0 正常(リンクファイルの参照カウント)  
<0 エラー(エラーコード)

### 【解説】

指定したリンクファイルのリンクファイル情報を取り出す。

指定したファイルがリンクファイルでないときはエラーとなる。

### 【エラーコード】

ER\_ADR : アドレス(Ink,stat)のアクセスは許されていない。  
ER\_IO : 入出力エラーが発生した。  
ER\_NOEXS : ファイル(Ink)は存在していない。  
ER\_NOFS : ファイル(Ink)の属するファイルシステムは接続されていない。  
ER\_NOLNK : リンクファイルではない。  
ER\_NOSPC : システムのメモリ領域が不足した。

## syn\_Ink

リンクファイルの同期

### 【形式】

WERR syn\_Ink(LINK \*Ink, W opt)

### 【パラメータ】

LINK \*Ink 対象リンクファイル

W opt 同期属性  
=0 チェックのみ  
0 チェックおよび更新

### 【リターン値】

0 正常(同期状態)

## <0 エラー(エラーコード)

### 【解説】

指定したリンクファイルが保持しているファイル名、生成日時と、参照先のファイルの実際のファイル名、生成日時が一致しているかどうかをチェックする。

opt = 0 のときはチェックのみを行ない、opt 0 のときは異なっていた場合は、指定したリンクファイルが保持している情報を参照先のファイルの実際のファイル名、生成日時と一致するように更新する。

リターン値は以下の同期状態となる。

|         |                        |
|---------|------------------------|
| F_SYNC  | 一致している。                |
| F_DNAME | ファイル名が異なっていた           |
| F_DDATE | 生成日時が異なっていた            |
| F_DBOTH | ファイル名と生成日時の両方が共に異なっていた |

指定したファイルがリンクファイルでないときはエラーとなる。

### 【エラーコード】

|           |  |
|-----------|--|
| ER_ADR    | : アドレス((lnk))のアクセスは許されていない。                            |
| ER_IO     | : 入出力エラーが発生した。   |
| ER_NOEXS  | : ファイル(lnk)は存在していない。                                   |
| ER_NOFS   | : ファイル(lnk)またはファイル(lnk)の参照先のファイルの属するファイルシステムは接続されていない。 |
| ER_NOLNK  | : リンクファイルではない。   |
| ER_NOSPC  | : システムのメモリ領域が不足した。                                     |
| ER_RDONLY | : ファイル(lnk)は書込不可である、または属するファイルシステムは書込不可である。            |

## get\_dfm

デフォルトアクセスモードの取得

### 【形式】

ERR get\_dfm(DA\_MODE \*mode)

### 【パラメータ】

DA\_MODE \*mode デフォルトアクセスモードの格納領域

```
typedef struct {
    UH f_ownacc; /* 所有者アクセスモード */
    UH f_grpacc; /* グループアクセスレベル */
}
```

```
    UH  f_pubacc;      /* 一般アクセスレベル */
    H   f_grpno;      /* グループ番号(0~4) */
    UH  f_gacc[N_GRP]; /* グループアクセスレベル */
} DA_MODE;
```

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

自ユーザのデフォルトアクセスモードを取り出す。

デフォルトアクセスモードは、ファイルの生成時にアクセスモードを指定しなかったときに適用されるデフォルトのアクセスモードであり、ユーザごとに定義される。

f\_gacc[4] は、ユーザの所属するグループのそれぞれに対して設定されているグループアクセスレベルを示す単なる参照用のデータであり、f\_grpacc が実際のグループアクセスレベルとして適用される。

### 【エラーコード】

ER\_ADR : アドレス(mode)のアクセスは許されていない。

## set\_dfm

デフォルトアクセスモードの設定

### 【形式】

ERR set\_dfm(DA\_MODE \*mode)

### 【パラメータ】

DA\_MODE \*mode 設定するデフォルトアクセスモード

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

自ユーザのデフォルトアクセスモードを設定する。

変更したデフォルトアクセスモードは、同一ユーザの全プロセスに対して有効となる。

### 【エラーコード】

ER\_ADR : アドレス(mode)のアクセスは許されていない。  
ER\_PAR : パラメータが不正である(mode の内容が不正)。

## att\_fls

ファイルシステムの接続

### 【形式】

ERR att\_fls(TC \*dev, TC \*name, LINK \*lnk, UW mode)

### 【パラメータ】

TC \*dev デバイス名

TC \*name 接続名(TNULL、または最大接続名文字数まで有効)

LINK \*lnk 接続したファイルシステムのルートファイルのリンクの格納領域  
NULL 格納しない

UW mode 接続モード  
( FS\_SYNC FS\_ASYNC FS\_RDONLY )

FS\_SYNC 同期書き込み

ファイルへの書き込みは書き込みのシステムコールを実行した時点で必ず行われる。

FS\_ASYNC 非同期書き込み

ファイルへの書き込みは書き込みのシステムコールを実行した時点で行われるとは限らない。

FS\_RDONLY 書き込み禁止

ファイルへの書き込みはすべて禁止される。

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

指定したデバイス上に存在するファイルシステムを指定した接続名でシステムに接続する。

接続名は、接続したファイルシステムのルートファイルを絶対パス名により指定するために使用され、すでに接続済みの接続名と同じであってははいけない。

すでに接続済みのファイルシステムを再度接続しようとしたとき、また、接続しようとしたファイルシステムのファイルシステム名と同一のファイルシステム名を持つファイルシステムがすでに接続されていたときエラーとなる。

ファイルシステムを接続するには、デバイスに対する接続アクセス権が必要となる。

### 【エラーコード】

- ER\_ACCES : 論理デバイス(dev)のアクセス権(接続)がない。
- ER\_ADR : アドレス(dev, name, lnk)のアクセスは許されていない。
- ER\_BUSY : 論理デバイス(dev)は既にオープンされている、または接続されている。
- ER\_EXS : 接続名(name)は既に存在しているまたは同一のファイルシステム名のファイルシステムが既に接続されている。
- ER\_FNAME : ファイル名(name)は空または不正である。
- ER\_IO : 入出力エラーが発生した。
- ER\_LIMIT : 同時接続可能な最大ファイルシステム数を越えた。
- ER\_NODEV : デバイス(dev)へのアクセスができない。
- ER\_NOEXS : デバイス(dev)は登録されていない、またはブロック型デバイスでない。
- ER\_NOMDA : デバイス(dev)のメディアが存在しない。
- ER\_NOSPC : システムのメモリ領域が不足した。
- ER\_TRON : TRON形式のファイルシステムではない。

## det\_fls

ファイルシステムの切断

### 【形式】

ERR det\_fls(TC \*dev, W eject)

### 【パラメータ】

- TC \*dev     デバイス名
- W   eject    イジェクト指定
  - =0   イジェクトしない
  - 0   イジェクトする(イジェクト不可能なデバイスの時は無視)

### 【リターン値】

- =0    正常
- <0    エラー(エラーコード)



## 【解説】

指定したデバイス上の接続済みのファイルシステムをシステムから切り離す。この時、メモリ上に一時的に保持されている内容等があればファイルシステム上にすべて書き出す。

切断の対象となるファイルシステム内のファイルがオープンされているとき、または作業ファイルとして存在しているプロセスが存在しているときは、切断できない。

ファイルシステムを切断するには、デバイスに対する接続アクセス権が必要となる。

## 【エラーコード】

|           |  |
|-----------|--|
| ER_ACCES  | : 論理デバイス(dev)のアクセス権(接続)がない。            |
| ER_ADR    | : アドレス(dev)のアクセスは許されていない。              |
| ER_BUSY   | : ファイルシステムは使用中である。                     |
| ER_IO     | : 入出力エラーが発生した。                         |
| ER_NOEXS  | : デバイス(dev)は登録されていない、またはブロック型デバイスでない。  |
| ER_NOFS   | : 論理デバイス(dev)上の属するファイルシステムは接続されていない。   |
| ER_NOMDA  | : デバイスのメディアが存在しない。                     |
| ER_NOSPC  | : システムのメモリ領域が不足した。                     |
| ER_RDONLY | : ファイルは書込不可である、または属するファイルシステムは書込不可である。 |

## syn\_fs

ファイルシステムの同期

## 【形式】

ERR syn\_fs(VOID)

## 【パラメータ】

なし

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

メモリ上に一時的に保持されていた内容等をファイルシステム上にすべて書き出し、ファイルシステム全体を矛盾ないように更新する。  
接続されているすべてのファイルシステムに対して行う。

## 【エラーコード】

- ER\_IO : 入出力エラーが発生した。
- ER\_NOMDA : デバイスのメディアが存在しない。
- ER\_NOSPC : システムのメモリ領域が不足した。
- ER\_RDONLY : ファイルは書込不可である、または属するファイルシステムは書込不可である。

## fls\_sts

ファイルシステム管理情報の取得

### 【形式】

WERR fls\_sts(TC \*dev, FS\_STATE \*buff)

### 【パラメータ】

TC \*dev デバイス名

FS\_STATE \*buff ファイルシステム管理情報の格納領域

### 【リターン値】

- 0 正常(ファイルシステムの接続モード)  
( 0:ファイルシステムは書込可、1:ファイルシステムは書込不可)
- <0 エラー(エラーコード)

### 【解説】

指定したデバイス上の接続済みのファイルシステムの管理情報を取り出す。

ファイルシステムの管理情報を取り出すためには、デバイスに対する接続アクセス権が必要となる。

### 【エラーコード】

- ER\_ACCES : 論理デバイス(dev)のアクセス権(接続)がない。
- ER\_ADR : アドレス(dev, buf)のアクセスは許されていない。
- ER\_BUSY : 論理デバイスは既にオープンされている
- ER\_IO : 入出力エラーが発生した。
- ER\_NODEV : デバイス(dev)へのアクセスができない。
- ER\_NOEXS : デバイス(dev)は登録されていない、またはブロック型デバイスでない。
- ER\_NOMDA : デバイス(dev)のメディアが存在しない。
- ER\_NOSPC : システムのメモリ領域が不足した。
- ER\_TRON : TRON形式のファイルシステムではない。

# chg\_fls

## ファイルシステム情報の変更

### 【形式】

ERR chg\_fls(TC \*dev, TC \*fs\_name, TC \*fs\_locate)

### 【パラメータ】

TC \*dev            デバイス名  
                  NULL の場合エラーとなる

TC \*fs\_name        変更するファイルシステム名  
                  NULL 変更しない

TC \*fs\_locate     変更するデバイス所在名  
                  NULL 変更しない

### 【リターン値】

= 0     正常  
< 0    エラー(エラーコード)

### 【解説】

指定したデバイス上の接続済みのファイルシステムのファイルシステム名およびデバイス所在名を変更する。

ファイルシステム情報を変更するには、デバイスに対する接続アクセス権、および書込みアクセス権が必要となる。

### 【エラーコード】

ER\_ACCES        : 論理デバイス(dev)のアクセス権(接続、書込)がない。  
ER\_ADR          : アドレス(dev, fs\_name, fs\_locate)のアクセスは許されていない。  
ER\_BUSY         : 論理デバイスは既にオープンされている  
ER\_EXS          : 指定したファイルシステム名のファイルシステムは既に存在している  
                  (接続されている)。  
ER\_FNAME        : ファイルシステム名は空、または不正  
ER\_IO           : 入出力エラーが発生した。  
ER\_NODEV        : デバイス(dev)へのアクセスができない。  
ER\_NOEXS        : デバイス(dev)は登録されていない、またはブロック型デバイスでない。  
ER\_NOMDA        : デバイス(dev)のメディアが存在しない。  
ER\_NOSPC        : システムのメモリ領域が不足した。

ER\_RDONLY : ファイルは書込不可である、または属するファイルシステムは書込不可である。

ER\_TRON : TRON形式のファイルシステムではない。

## get\_nlk

リンクの順次取得

### 【形式】

WERR get\_nlk(LINK \*lnk)

### 【パラメータ】

LINK \*lnk 開始ファイルのリンク (入力)  
次のファイルのリンクの格納領域 (出力)

### 【リターン値】

0 正常(取り出したファイルの参照カウント)  
<0 エラー(エラーコード)

### 【解説】

指定した開始ファイルのファイルIDより大きなファイルIDを持つファイルのうち最小のファイルIDを持つファイルへのリンクを取り出す。

開始ファイルのファイルIDは実際には存在しないファイルでも良い。取り出したファイルのリンクの属性データはすべて0となる。

本システムコールにより、ファイルシステム上に存在するすべてのファイル(参照カウント=0のファイルも含む)へのリンクを取り出すことができる。

### 【エラーコード】

ER\_ADR : アドレス(lnk)のアクセスは許されていない。

ER\_IO : 入出力エラーが発生した。

ER\_NOFS : ファイル(lnk)の属するファイルシステムは接続されていない。

ER\_NOEXS : ファイル(lnk)より大きなファイルIDを持つファイルは存在していない。

ER\_NOSPC : システムのメモリ領域が不足した。

## lst\_fls

ファイルシステムの取得

## 【形式】

WERR    lst\_fls(F\_ATTACH \*buff, W cnt)

## 【パラメータ】

F\_ATTACH \*buff    ファイルシステム接続情報の格納領域(配列)  
    typedef struct {  
        TC a\_name[L\_CONNM];    /\* 接続名 \*/  
        TC dev[L\_DEVNM];      /\* 論理デバイス名 \*/  
    } F\_ATTACH;

W    cnt > 0      接続されている全てのファイルシステムの接続情報を  
                  buff に取り出す。cnt は buff の要素数を示す。  
= F\_GETDEV    buff->a\_name[] に設定した接続名に対応する  
                  デバイス名を buff->dev[] に取り出す。  
= F\_GETNAM    buff->dev[] に設定したデバイス名に対応する  
                  接続名を buff->a\_name[] に取り出す。

## 【リターン値】

= 1    正常(F\_GETDEV, F\_FETNAM 指定時)  
= 0    正常(接続済みのファイルシステム数)  
< 0    エラー(エラーコード)

## 【解説】

接続済みのファイルシステムの接続名とデバイス名と取り出す。

接続されているファイルシステムの数が指定した要素数(cnt)より多い場合は、最初の要素数(cnt)個のみを取り出す。

## 【エラーコード】

ER\_ADR        : アドレス(buff)のアクセスは許されていない。  
ER\_NOFS      : ファイルシステムは接続されていない(cnt= - 1, - 2の時)。  
ER\_PAR        : パラメータが不正である(cnt=0,< - 2)。

# map\_rec

レコードのマップ

## 【形式】

WERR    map\_rec(W fd, W offset, B \*\*addr, W size, W mode)

## 【パラメータ】

|   |  |                    |
|---|--|--------------------|
| W | fd   | ファイルディスクリプタ        |
| W | offset   | マップ開始バイトオフセット      |
| B | **addr   | マップされたメモリアドレスの格納領域 |
| W | size   | マップするバイトサイズ        |
| W | mode   | マップモード             |
|   | ( [ F_READ ]   [ F_WRITE ]   [ F_EXECUTE ] )   [ F_COMMON F_SYSTEM ] |                    |
|   | F_READ   | 読み込み用マップ           |
|   | F_WRITE  | 書き込み用マップ           |
|   | F_EXECUTE  | 実行用マップ             |
|   | F_COMMON   | 共有メモリー空間にマップ       |
|   | F_SYSTEM   | システムメモリー空間にマップ     |

## 【リターン値】

|     |             |
|-----|-------------|
| > 0 | 正常(マップID)   |
| < 0 | エラー(エラーコード) |

## 【解説】

オープンしたファイルの現在レコードの offset から size バイトをメモリー空間上にマップする。マップされたレコードの内容はメモリーとしてアクセスすることができる。

F\_COMMON を指定した場合には、共有メモリー空間上にマップされる。この場合、すべてのプロセスからアクセス可能となる。F\_SYSTEM を指定した場合には、システムメモリー空間上にマップされる。この場合、すべてのシステムプロセスからアクセス可能となる。マップしたプロセス自身であっても、一般のアプリケーションプロセスからはアクセスできない。したがって、アプリケーションプロセスからはこの指定を使用すべきではない。

F\_COMMON, F\_SYSTEM のいずれも指定しなければ、マップしたプロセスのローカルメモリー空間上にマップされる。この場合、マップしたプロセス以外からアクセスすることは出来ない。

マップされるアドレスはシステムで決定され、アプリケーション側から指定することはできない。

マップモードの指定はオープンしたモードに矛盾してはいけない。(ER\_FD)

- F\_READ オープンでは、F\_WRITE でマップできない。
- F\_WRITE オープンでは、F\_READ および F\_EXECUTE でマップできない。

リンクレコードはマップできない。(ER\_REC)

マップ中は、次の操作は禁止され ER\_BUSY となる。

- del\_rec マップしているレコードを対象とする場合。
- wri\_rec マップしているレコードを対象とする場合。

- trc\_rec マップしているレコードを対象とする場合。
- xch\_fil マップしているレコードを含むファイルを対象とする場合。

インプリメントに依存して以下の制限がつけられる場合がある。

- マップ可能なファイルシステム(デバイス)
- マップモードの組み合わせ
- offset、および size の値

## 【エラーコード】

|          |                            |
|----------|----------------------------|
| ER_FD    | : ファイルディスクリプタは存在していない。     |
|          | : マップモードとオープンモードが矛盾している。   |
| ER_REC   | : 現在レコードはリンクレコードである。       |
| ER_ADR   | : アドレス(addr)のアクセスは許されていない。 |
| ER_ENDR  | : 現在レコードは終端レコードである。        |
| ER_LOCK  | : 現在レコードは他からロックされている。      |
| ER_IO    | : 入出力エラーが発生した。             |
| ER_PAR   | : パラメータが不正である。             |
| ER_NOSPT | : システムの制限によりマップできない。       |

## ump\_rec

レコードのアンマップ

## 【形式】

ERR ump\_rec(W fd, W mapid)

## 【パラメータ】

W fd ファイルディスクリプタ

W mapid マップID

## 【リターン値】

= 0 正常  
 < 0 エラー(エラーコード)

## 【解説】

オープンしているファイルのマップIDで指定したマップを解除する。ファイルのクローズ時にはオープンしたプロセスで設定したマップは解除される。

## 【エラーコード】

ER\_FD : ファイルディスクリプタは存在していない。  
ER\_NOEXS : マップIDは存在していない。  
ER\_IO : 入出力エラーが発生した。

## chg\_fsm

ファイルシステム接続モードの変更

### 【形式】

WERR chg\_fsm( TC \*dev, UW mode )

### 【パラメータ】

TC \*dev 論理デバイス名

UW mode 接続モード (att\_fls の mode と同じ)  
( FS\_SYNC FS\_ASYNC FS\_RDONLY )  
FS\_SYNC 同期  
FS\_ASYNC 非同期  
FS\_RDONLY 書き込み禁止

### 【リターン値】

0 正常(変更前の接続モード)  
<0 エラー(エラーコード)

### 【解説】

dev のデバイスの接続状態を mode で指定した接続モードに変更する。 dev はすでに接続されているデバイスでなければならない。リターン値に変更前の接続モードを返す。

接続モードを変更するためには、dev に対する接続アクセス権が必要である。

接続モード変更前が書き込み可であった場合、書き込み禁止に変更しても変更前に行われた map\_rec() による書き込みモードのレコードマップはそのまま有効となり、書き込みも行われる。 F\_WRITE または F\_UPDATE でオープンしているファイルがあるとき、接続モードを書き込み禁止にすると、そのファイルに対する wri\_rec() 等の書き込みが ER\_RDONLY となる。

### 【エラーコード】

ER\_ACCES : 論理デバイス(dev)に接続アクセス権がない。  
ER\_ADR : アドレス(dev)のアクセスが許されていない。  
ER\_IO : 入出力エラーが発生した。  
ER\_PAR : パラメータが不正である。



ER\_NOFS : 論理デバイス(dev)は、ファイルシステムとして接続されていない。

---

[この章の目次にもどる](#)

[前頁:1.5 メモリ管理にもどる](#)

[次頁:1.7 イベント管理にすすむ](#)

[この章の目次にもどる](#)

[前頁:1.6 ファイル管理にもどる](#)

[次頁:1.8 デバイス管理にすすむ](#)

---

## 1.7 イベント管理

### 1.7.1 イベント管理機能の概要

イベント管理機能は、インタラクティブなヒューマンインタフェースを実現するために、ユーザとの対話に使用するキーボード(KB)とポインティングデバイス(PD)の操作を「イベント」という形で統一的に取り扱う機能であり、この機能により、柔軟なインタラクティブ操作を実現することが可能となる。

キーボードおよび、ポインティングデバイスの操作は「イベント」としてシステムで1つのイベントキューに順次格納され、アプリケーションはイベントキューからイベントを順次取り出し、それに対応するアクションを実行する「イベント駆動形式」の形態を取るようになる。

マルチプロセス環境の場合、ユーザとのインタラクティブ操作を必要とするプロセスはある時点では必ず1つであり、ユーザとの「入力受付状態」を複数のプロセス間で相互に受け渡していく形態をとる。

従って、ある時点で入力受付状態となっているプロセスのみがイベント管理機能を使用してイベントを取り出すというルールを前提としており、アプリケーションはこのルールに従った動作をしなければいけない。イベント管理機能自体では、このルールを保証する機構は特に提供しておらず、外殻レベルで提供することになる。

イベント管理機能には、イベントの管理以外にも、キーボード、およびポインティングデバイスの状態/属性の取出し、設定等の機能も含まれており、インタラクティブな入力装置に対する統一的なインタフェースを提供している。

### 1.7.2 イベント

#### イベントの種類

以下のタイプのイベントが定義される。

ボタндаウンイベント (EV\_BUTDWN) :

ポインティングデバイスのボタンが押された時に発生する。

ボタンアップイベント (EV\_BUTUP) :

ポインティングデバイスのボタンが離された時に発生する。

キーダウンイベント (EV\_KEYDWN) :

シフトキー等の特殊キー(メタキー)以外の通常キーが押された時に発生する。

キーアップイベント (EV\_KEYUP) :

シフトキー等の特殊キー(メタキー)以外の通常キーが離された時に発生する。

自動リピートキーイベント (EV\_AUTKEY) :

自動リピートの対象となるキーが押され続けていた時に周期的に発生する。自動リピートの対象となるキーを押してから、最初に自動リピートキーイベントが発生するまでの時間(オフセット)、およびその後の発生間隔(インターバル)は任意に設定可能である。

デバイスイベント (EV\_DEVICE) :

キーボード、ポインティングデバイス以外のデバイスのある種の操作に伴って発生する汎用的なイベントであり、その内容はデバイスに依存する。フロッピーディスク等の取り外し可能メディアを装着した場合は、このイベントが発生する。

ヌルイベント (EV\_NULL) :

対象とするイベントが発生していないことを示す擬似的なイベント。

アプリケーションイベント (EV\_APPL1 ~ EV\_APPL8) :

アプリケーションにより定義され使用されるイベントであり、アプリケーション間での通信機能として使用される。アプリケーションイベントのいくつかは外殻により、その意味が定義される。

物理的なキーボードが存在しない場合は、イベント管理の段階では EV\_KEYDOWN, EV\_KEYUP, EV\_AUTOKEY は発生しない。ソフトキーボードのようなフロントエンドプロセスで、ボタンイベントが適切なキーイベントに変換されて、アプリケーションに渡される。

イベントのタイプは、0 ~ 15 のタイプ番号で区別される。また各イベントタイプに対応したタイプマスクが定義されており、このマスクにより、対象とするイベントのタイプを指定することができる。タイプマスクはビット対応となっており、"1"のビットに対応するタイプのイベントが対象とされる。ただし、ヌルイベントに対しては、その性質上マスクは定義されない。

| イベント       | タイプ番号 | タイプマスク                 |
|------------|-------|------------------------|
| EV_NULL    | 0     | -----                  |
| EV_BUTDWN  | 1     | EM_BUTDWN (0x0001)     |
| EV_BUTUP   | 2     | EM_BUTUP (0x0002)      |
| EV_KEYDOWN | 3     | EM_KEYDOWN (0x0004)    |
| EV_KEYUP   | 4     | EM_KEYUP (0x0008)      |
| EV_AUTKEY  | 5     | EM_AUTKEY (0x0010)     |
| EV_DEVICE  | 6     | EM_DEVICE (0x0020)     |
| EV_RSV     | 7     | EM_RSV (0x0040) ( 予約 ) |
| EV_APPL1   | 8     | EM_APPL1 (0x0080)      |
| : :        | : :   |                        |
| EV_APPL8   | 15    | EM_APPL8 (0x4000)      |

なお、タイプマスクとしては以下の特殊なマスクも用意されている。

EM\_NULL 0x0000  
EM\_ALL 0x7fff

イベントの構造

イベントは以下に示す構造体で定義される。

```
typedef struct {
    W    type;          /* イベントタイプ */
    UW   time;         /* イベント発生時刻 */
    PNT  pos;          /* イベント発生時のPD位置 */
    EVDATA data;      /* イベントの固有データ */
    UW   stat;        /* メタキー、PDボタン状態 */
} EVENT;
```

type:

イベントのタイプを示す 0 ~ 15 の値である。

time:

イベント発生時刻を示すミリ秒単位の相対的な時刻であり、この値はイベントの発生順序、発生間隔を示しているもので絶対時刻としての意味は持たない。

内部的にイベントタイマーと呼ばれる、ミリ秒単位で加算される32ビットのタイマーが存在し、イベントが発生した時点のタイマーの値がイベント発生時刻としてセットされる。32ビットをオーバーフローした場合は単に上位ビットが捨てられる。イベントタイマーの分解能はインプリメントに依存する。

なお、アプリケーションイベントでの time の意味はイベントの定義に依存する。

pos:

イベント発生時点のポインティングデバイスの座標位置を、スクリーンの左上を (0, 0) とした絶対座標値で示すもので、以下の PNT タイプの値である。

```
typedef struct point {
    H    x; /* 水平座標値 */
    H    y; /* 垂直座標値 */
} PNT;
```

なお、アプリケーションイベントでの pos の意味はイベントの定義に依存する。

data:

イベントの固有データであり、イベントのタイプに依存した内容である。

```
typedef union {
    struct {          /* EV_KEYUP, EV_KEYDOWN, EV_AUTKEY */
        UH keytop; /* キートップコード */
        TC code;  /* 文字コード */
    } key;
    struct {          /* EV_DEVICE */
        H kind;   /* デバイスイベント種別 */
        H devno;  /* デバイス番号 */
    } dev;
    W info;         /* その他のイベント用データ */
} EVDATA;
```

EV\_KEYDOWN, EV\_KEYUP, EV\_AUTKEY の場合は、key が適用され、キーの物理的な位置を示すキートップコードと、エンコードされた文字コードからなる。

EV\_DEVICE の場合は、dev が適用され、デバイスイベントの種別(kind)とイベントが発生したデバイスを示すデバイス番号(devno)からなる。イベントの種別は以下に示す

ものである。

|                   |        |                 |
|-------------------|--------|-----------------|
| kind = DE_unknown | 0      | -- 未定義          |
| DE_MOUNT          | 0x01   | -- メディア挿入       |
| DE_EJECT          | 0x02   | -- メディア排出       |
| DE_ILLMOUNT       | 0x03   | -- メディア不正挿入     |
| DE_ILLEJECT       | 0x04   | -- メディア不正排出     |
| DE_REMOUNT        | 0x05   | -- メディア再挿入      |
| DE_CARDBATLOW     | 0x06   | -- カードバッテリー残量警告 |
| DE_CARDBATFAIL    | 0x07   | -- カードバッテリー異常   |
| DE_REQEJECT       | 0x08   | -- メディア排出要求     |
|                   | 0x09 ~ | -- 予約           |

これらのデバイスイベントは、デバイスドライバからの事象通知により発生する。EV\_NULL, EV\_BUTDWN, EV\_BUTUP の場合は、この data は使用されず、info は常に 0 となる。

アプリケーションイベント(EV\_APPL1 ~ 8) の場合は、イベントの定義に依存した内容となる。

stat:

イベント発生時点のメタキーとポインティングデバイスのボタンの状態をビット対応で示すものである。各ビットは、&quot;0&quot;が離されている(OFF)状態、"1" が押されている(ON)状態を意味する。

|          |        |                |
|----------|--------|----------------|
| ES_BUT   | 0x0001 | -- PDのメインボタン状態 |
| ES_BUT2  | 0x0002 | -- PDのサブボタン状態  |
| ES_ALPH  | 0x0004 | -- 英語ロックキー状態   |
| ES_KANA  | 0x0008 | -- カタカナロックキー状態 |
| ES_LSHFT | 0x0010 | -- 左シフトキー状態    |
| ES_RSHFT | 0x0020 | -- 右シフトキー状態    |
| ES_EXT   | 0x0040 | -- 拡張シフトキー状態   |
| ES_CMD   | 0x0080 | -- 命令シフトキー状態   |

ポインティングデバイスのサブボタン(メニューボタン)は、押されてもイベントは発生しない。また、上記のキーはメタキーであり、押されてもイベントは発生しない。また、下記の状態を示す情報も含まれる。

|           |            |               |
|-----------|------------|---------------|
| ES_LLSHFT | 0x00000100 | -- 左シフト簡易ロック  |
| ES_LRSHT  | 0x00000200 | -- 右シフト簡易ロック  |
| ES_LXT    | 0x00000400 | -- 拡張簡易ロック    |
| ES_LCMD   | 0x00000800 | -- 命令簡易ロック    |
| ES_TLSHFT | 0x00001000 | -- 左シフト一時シフト  |
| ES_TRSHFT | 0x00002000 | -- 右シフト一時シフト  |
| ES_TXT    | 0x00004000 | -- 拡張一時シフト    |
| ES_TCMD   | 0x00008000 | -- 命令一時シフト    |
| ES_HAN    | 0x00010000 | -- 半角キー       |
| ES_NODSP  | 0x00200000 | -- ポインタ非表示    |
| ES_PDSIM  | 0x00C00000 | -- PDシュミレーション |

これらの状態変化はイベントを発生しない。

## イベントキュー

イベントキューは、システムに唯1つ用意されているイベント格納用のキューであり、イベントが発生順に格納される。キューに空きが無い場合は、新規に発生したイベント、即ち、一番新しいイベントはキューに入れられず、捨てられることになる。

イベントキューに格納されるイベントは、システムイベントマスクにより制限される。即ち、システムイベントマスクの "1" のビットに対応するタイプのイベントのみがイベントキューに入り、"0" のビットに対応するタイプのイベントはシステム全体として無視されて捨てられる。

システムのスタートアップ時点では、イベントキューは空、システムイベントマスクは0となっており、イベント管理機能は事実上動作していない状態であるため、必ずシステムイベントマスクを適当な値に設定する必要がある。

イベントには、そのタイプに応じた以下に示す優先度が付けられており、高い優先度のイベントから取り出される。同一優先度の場合は、発生順に取り出される。

( 1: 最高優先度 ~ 6:最低優先度 )

1. EV\_APPL1 ~ 4
2. EV\_BUTDWN, EV\_BUTUP, EV\_KEYDWN, EV\_KEYUP
3. EV\_AUTKEY
4. EV\_DEVICE, EV\_RSV
5. EV\_APPL5 ~ 8
6. EV\_NULL

ヌルイベント(EV\_NULL)、および自動リピートキーイベント(EV\_AUTKEY)は、実際にはイベントキューには入れられず、イベントの取り出し要求時に自動的に生成されることになる。

ヌルイベント(EV\_NULL) :

要求されたタイプのイベントが発生していない場合に生成されて取り出される。

自動リピートキーイベント(EV\_AUTKEY) :

以下の条件が満足された場合、生成されて取り出される。

1. EV\_AUTKEY は要求されたタイプの1つである。
2. EV\_AUTKEY に対応するシステムイベントマスクのビットは "1" である。
3. EV\_AUTKEY より優先度の高いイベントが発生していない。
4. 最新の EV\_KEYDWN の発生キーが押され続けられている。
5. 最新の EV\_KEYDWN の発生以降、設定されているオフセット時間、またはインターバル時間が経過した。
6. 自動リピートの対象キーである。

なお、1つのキーを押し続けている状態で、さらに別のキーを押し続けた場合は、最後に押したキーの自動リピートキーイベントのみが発生することになる。

## 1.7.3 キーボード

### メタキー

TRONキーボードにおいて以下のキーはメタキーであり、押したり離したりした場合でもイベントは発生しないが、その状態はイベント内の stat フィールドにセットされ、また文字コードのエンコードに使用される。

- 日本語 / 英語キー (ロックキー)
- ひらがな / カタカナキー(英語の時は、CAPSLOCK キー) (ロックキー)
- 左シフトキー
- 右シフトキー
- 拡張シフトキー
- 命令シフトキー

メタキーは先に押された場合のみ文字のエンコードに対して有効となる。

### 自動リピートキー

自動リピートキーイベント(EV\_AUTKEY)は、自動リピートの対象キーが押し続けられた場合に発生する。

自動リピートの対象キーはイベントを発生しないメタキーを除いて任意に設定可能であり、システム立ち上げ時には、メタキーを除いたすべてのキーを自動リピートの対象とする。

押してから最初に発生するまでの時間(オフセット時間)と、その後の発生間隔(インターバル時間)の設定 / 取出し用のシステムコールが提供されている。この時間はミリ秒単位である。

但し、自動リピートキーイベントの発生間隔はインプリメントに依存しており、オフセット時間とインターバル時間は、イベント発生時刻の単位へ丸められる。

### 文字コード

キーイベントでは、キーの物理的位置を示すキートップコードと、エンコードされた文字コードを戻す。

キートップコードは、キーの物理的位置に応じた固定的な8ビットのコード(0 ~ 255)であり、メタキーの状態により異なった文字コードにエンコードされる。

文字コードへのエンコードは「文字コード変換表」を使用して行なわれる。通常、ユーザーの使用する言語、入力方式(ローマ字入力 / かな入力等)に対応した変換表が、ユーザー毎に設定されることになる。

文字コード変換表は以下の構造を持つ。

```
typedef struct {  
    W    keymax;          /* 実際の最大キー数 (1 ~ 256) */  
    W    kctmax;         /* 実際の変換表の数 (1 ~ 64) */  
    UH   kctsel[KCTSEL]; /* 変換表の番号 (0 ~ kctmax-1の値) */  
};
```

```
UH kct[KCTMAX]; /* 変換表本体 (keymax × kctmax個の要素) */
} KeyTab;
```

- keymax はキーボードにより規定される最大キー数であり、キートップコードは0 ~ (keymax-1)となる。
- kctmax は文字コード表の数で最大64。
- keymax × kctmaxの最大許容値はインプリメントに依存するが、少なくとも 2048 が保証される。
- kctsel[] は、6種のメタキー状態に対応する文字コード表の番号(0 ~ kctmax - 1)を定義する配列である。メタキー状態を以下の6ビットの数値とした場合、kctsel[メタキー状態]の値が対応する文字コード表の番号(0 ~ kctmax - 1)となる。複数のエントリから、共通の文字コード表を指すことができる。

メタキー状態：CERLKA

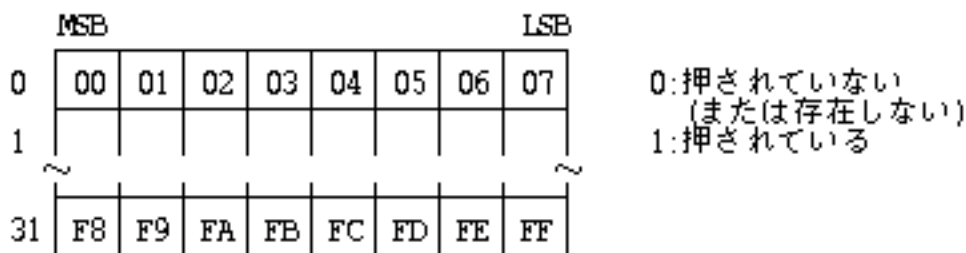
- A: 「日本語 / 英語」キー (0:日本語)
- K: 「ひらがな / カタカナ」キー (0:ひらがな)
- L: 「左シフト」キー
- R: 「右シフト」キー
- E: 「拡張シフト」キー
- C: 「命令シフト」キー

- kct[] は文字コード表本体であり、実際には kct[kctmax][keymax] の2次元配列となる。kct[kctsel[メタキー状態(0 ~ 63)]][キー位置コード(0 ~ keymax-1)]がエンコードされた文字コードを示す。

## キー状態

キーが押されているか否かの状態は、1つのキーを1ビットに対応させた以下に示す KeyMap 配列により定義され、この KeyMap 配列を取り出すためのシステムコールが提供されている。

```
typedef UB KeyMap[KEYMAX/8]; /* キー状態配列
```



番号は16進のキートップコード

図 13: キー状態 (KeyMap)

なお、この KeyMap 配列は、自動リピートの対象となるキーの設定 / 取出しにも使用される。この場合、"1"のビットに対応するキーが自動リピートの対象となる。

## キーボードID

現在接続されているキーボードの種類を知るために以下のように定義されるキーボードIDを



取り出すことが可能である。

```
typedef struct {
    UH kind;          /* キーボードタイプ */
    UH maker;        /* メーカーID */
    UB id[4];        /* メーカー依存キーボードID */
} KBD_ID;

kind : 0000 000T TTTT TTMP
```

|             |                               |
|-------------|-------------------------------|
| P= 0        | -- ポインティングデバイスがキーボードに内蔵されていない |
| 1           | -- ポインティングデバイスがキーボードに内蔵されている  |
| M= 0        | -- ロックキーは電子ロック式               |
| 1           | -- ロックキーはメカニカルロック方式           |
| T=0x00      | -- 未定義キーボード                   |
| 0x01        | -- TRON日本語キーボード               |
| 0x02 ~ 0x3F | -- 予約 (TRON 仕様のキーボード)         |
| 0x40        | -- IBM 101 系英語キーボード           |
| 0x41        | -- IBM 106 系日本語キーボード          |
| 0x42 ~ 0x7f | -- 予約 (TRON 仕様以外のキーボード)       |

maker:

メーカーを表わすIDであり、get\_ver() で得られるメーカーの値と同じものである。

id[4]:

メーカーに依存したキーボードの種別を表わすIDで4バイトから構成される。

## 1.7.4 ポインティングデバイス

### ポインティングデバイス

ポインティングデバイスは、スクリーン上に表示されているオブジェクトを選択するために使用され、その現在位置として、スクリーンの解像度に対応したレンジの絶対座標値を持つ。絶対座標値とは、スクリーンの左上の点を(0,0)とし、スクリーン上の1ピクセルを単位とした座標値である。

### ポインティングデバイスの動作タイプ

ポインティングデバイスは、その動作方式から以下の2種類に大きく分類される。

絶対動作タイプ:

電子ペン等のタブレットタイプのもので、ポインティングデバイスの物理的な位置により、絶対的に座標値が決定されるもの。

相対動作タイプ(差動タイプ):

マウス等で、ポインティングデバイスの物理的な位置の移動により、相対的に座標値が決定されるもの。

相対動作タイプの場合は、ポインティングデバイスの(基準)位置を任意に設定することが可能であるが、絶対動作タイプの場合は、位置の設定は不可となる。

なお、ポインティングデバイスによっては、絶対 / 相対動作の両方の動作を切り換えることが可能なものもある。

## ポインティングデバイスの属性

ポインティングデバイスの属性として以下のものが定義され、取り出し/設定が可能である。

### 感度:

相対動作タイプの場合のみ定義され、ポインティングデバイスの移動量と、座標値の移動量と比率を表わすもので、感度が高い程、ポインティングデバイスの移動量に対する、座標値の移動量が大きくなる。

最低感度(0) ~ 最高感度(15) の16段階で定義される。

### スキャン速度

ポインティングデバイスの位置を取り出すことが可能な時間間隔を示すものであり、スキャン速度が高速な程、時間間隔が狭まることになる。

最低速(0) ~ 最高速(15) の16段階で定義される。

## 1.7.5 データ/定数の定義

### イベント固有データ

```
typedef union {
    struct {
        UH keytop; /* EV_KEYUP, EV_KEYDOWN, EV_AUTKEY */
        TC code; /* キートップコード */
        /* 文字コード */
    } key;
    struct {
        /* EV_DEVICE */
        H kind; /* デバイスイベント種別 */
        H devno; /* デバイス番号 */
    } dev;
    W info; /* その他のイベント用データ */
} EVDATA;
```

### イベント

```
typedef struct {
    W type; /* イベントタイプ */
    UW time; /* イベント発生時間(m sec) */
    PNT pos; /* イベント発生時のPD位置 */
    EVDATA data; /* イベントの固有データ */
    UW stat; /* メタキー、PDボタン状態 */
} EVENT;
```

### イベントタイプ

```
#define EV_NULL 0 /* ヌルイベント */
```

```

#define EV_BTUDWN 1 /* ボタンダウン */
#define EV_BTUP 2 /* ボタンアップ */
#define EV_KEYDWN 3 /* キーダウン */
#define EV_KEYUP 4 /* キーアップ */
#define EV_AUTKEY 5 /* 自動キーリピート */
#define EV_DEVICE 6 /* デバイスイベント */
#define EV_RSV 7 /* 予約 */
#define EV_APPL1 8 /* アプリケーションイベント#1 */
#define EV_APPL2 9 /* アプリケーションイベント#2 */
#define EV_APPL3 10 /* アプリケーションイベント#3 */
#define EV_APPL4 11 /* アプリケーションイベント#4 */
#define EV_APPL5 12 /* アプリケーションイベント#5 */
#define EV_APPL6 13 /* アプリケーションイベント#6 */
#define EV_APPL7 14 /* アプリケーションイベント#7 */
#define EV_APPL8 15 /* アプリケーションイベント#8 */

```

## イベントマスク

```

#define EM_NULL 0x0000
#define EM_ALL 0x7fff
#define EM_BTUDWN 0x0001
#define EM_BTUP 0x0002
#define EM_KEYDWN 0x0004
#define EM_KEYUP 0x0008
#define EM_AUTKEY 0x0010
#define EM_DEVICE 0x0020
#define EM_RSV 0x0040
#define EM_APPL1 0x0080
#define EM_APPL2 0x0100
#define EM_APPL3 0x0200
#define EM_APPL4 0x0400
#define EM_APPL5 0x0800
#define EM_APPL6 0x1000
#define EM_APPL7 0x2000
#define EM_APPL8 0x4000

```

## メタキー、PDボタン状態(0: OFF、1:ON)

```

#define ES_BT 0x00000001 /* PDメインボタン */
#define ES_BT2 0x00000002 /* PDサブボタン */
#define ES_ALPH 0x00000004 /* 英語ロックキー */
#define ES_KANA 0x00000008 /* カタカナロックキー */
#define ES_LSHFT 0x00000010 /* 左シフトキー */
#define ES_RSHFT 0x00000020 /* 右シフトキー */
#define ES_EXT 0x00000040 /* 拡張キー */
#define ES_CMD 0x00000080 /* 命令キー */
#define ES_LLSHFT 0x00000100 /* 左シフト簡易ロック */

```

```

#define ES_LRSHT 0x00000200 /* 右シフト簡易ロック */
#define ES_LXT 0x00000400 /* 拡張簡易ロック */
#define ES_LCMD 0x00000800 /* 命令簡易ロック */
#define ES_TLSHFT 0x00001000 /* 左シフト一時シフト */
#define ES_TRSHFT 0x00002000 /* 右シフト一時シフト */
#define ES_TEXT 0x00004000 /* 拡張一時シフト */
#define ES_TCMD 0x00008000 /* 命令一時シフト */
#define ES_HAN 0x00010000 /* 半角キー */
#define ES_NODSP 0x00400000 /* ポインタ非表示 */
#define ES_PDSIM 0x00800000 /* PDシュミレーション */

```

## 入力モード

```

#define IM_HIRA 0x0000 /* 日本語ひらがな */
#define IM_ALPH (ES_ALPH) /* 英語(小文字) */
#define IM_KATA (ES_KANA) /* 日本語カタカナ */
#define IM_CAPS (ES_ALPH | ES_KANA) /* 英語(大文字) */
#define IM_MASK (ES_ALPH | ES_KANA)

#define KIN_KANA 0x0000 /* かな入力モード */
#define KIN_ROMAN 0x0001 /* ローマ字入力モード */

```

## デバイスイベント(EV\_DEVICE)の種別(EVDATA.dev.kind)

```

typedef enum {
    DE_unknown = 0, /* 未定義 */
    DE_MOUNT = 0x01, /* メディア挿入 */
    DE_EJECT = 0x02, /* メディア排出 */
    DE_ILLMOUNT = 0x03, /* メディア不正挿入 */
    DE_ILLEJECT = 0x04, /* メディア不正排出 */
    DE_REMOUNT = 0x05, /* メディア再挿入 */
    DE_CARDBATLOW = 0x06, /* カードバッテリー残量警告 */
    DE_CARDBATFAIL = 0x07, /* カードバッテリー異常 */
    DE_REQEJECT = 0x08 /* メディア排出要求 */
} DevEvtKind;

```

## put\_evt()

```

#define EP_NONE 0x0000 /* time,pos,stat はそのまま */
#define EP_POS 0x0001 /* pos に現在PD位置を設定 */
#define EP_STAT 0x0002 /* stat に現在メタキー状態を設定 */
#define EP_TIME 0x0004 /* time に現在時間を設定 */
#define EP_ALL 0x0007 /* pos,stat,time に現在値を設定 */

```

## chg\_pda()

```

#define PD_REV      0x1000    /* 左右反転 */
#define PD_ABS      0x0100    /* 絶対座標タイプ */
#define PD_REL      0x0000    /* 相対座標タイプ */
#define PD_SCMSK    0x00f0    /* スキャン速度 (マスク) */
#define PD_SNMSK    0x000f    /* 感度 (マスク) */

```

## キーボード

```

typedef struct {
    UH kind;          /* キーボードタイプ */
    UH maker;        /* メーカーID */
    UB id[4];        /* メーカー依存キーボードID */
} KBD_ID;

```

## キーマップ

```

#define KEYMAX      256

```

```

typedef UB KeyMap[KEYMAX/8];

```

## キーテーブル

```

#define KCTSEL      64
#define KCTMAX      4000

```

```

typedef struct {
    W keymax;        /* 実際の最大キー数 */
    W kctmax;        /* 実際の変換表の数 */
    UH kctsel[KCTSEL]; /* 変換表の番号 */
    UH kct[KCTMAX];  /* 変換表本体 */
} KeyTab;

```

## ctl\_buz()

```

/* <取り出し> */
#define GET_BUZ_BEEP    0x0000    /* 音色：標準の警告音 */
#define GET_BUZ_PDON    0x0001    /* 音色：ボタンONのクリック音 */
#define GET_BUZ_PDOFF   0x0002    /* 音色：ボタンOFFのクリック音 */
#define GET_BUZ_KEYON   0x0003    /* 音色：キーONのクリック音 */
#define GET_BUZ_KEYOFF  0x0004    /* 音色：キーOFFのクリック音 */
#define GET_BUZ_VOLUME  0x0010    /* 音量 */
/* <設定> */
#define SET_BUZ_BEEP    0x0100    /* 音色：標準の警告音 */
#define SET_BUZ_PDON    0x0101    /* 音色：ボタンONのクリック音 */

```

```
#define SET_BUZ_PD OFF    0x0102    /* 音色：ボタンOFFのクリック音 */
#define SET_BUZ_KEY ON    0x0103    /* 音色：キーONのクリック音 */
#define SET_BUZ_KEY OFF   0x0104    /* 音色：キーOFFのクリック音 */
#define SET_BUZ_VOLUME   0x0110    /* 音量 */
```

## 1.7.6 システムコール

### get\_evt

イベントの取得

#### 【形式】

```
WERR    get_evt(W t_mask, EVENT* evt, W opt)
```

#### 【パラメータ】

W t\_mask 対象イベントタイプマスク

EVENT\* evt 取得したイベントの格納領域

W opt 取得属性  
( CLR NOCLR )  
CLR イベントキューから取り除く。  
NOCLR イベントキューから取り除かない。

#### 【リターン値】

0 正常(得られたイベントのタイプ)  
<0 エラー(エラーコード)

#### 【解説】

指定したタイプのイベントをイベントキューから取り出す。  
指定したタイプのイベントが発生していないときは、EV\_NULL が取り出される。

#### 【エラーコード】

ER\_ADR : アドレス(evt)のアクセスは許されていない。  
ER\_IO : 入出力エラーが発生した(何らかのデバイスエラーが発生した)。  
ER\_PAR : パラメータが不正である(t\_mask 0、optが不正)。  
ER\_NOSPC : システムのメモリ領域が不足した。

# put\_evt

イベントの発生

## 【形式】

ERR put\_evt(EVENT\* evt, W opt)

## 【パラメータ】

EVENT\* evt 発生するイベント

W opt 発生属性  
( EP\_NONE EP\_ALL ([ EP\_POS ] | [ EP\_STAT ] | [EP\_TIME]) )  
EP\_NONE time, pos, stat は evt の内容のままとする。  
EP\_ALL time, pos, stat のすべてを設定する。  
EP\_POS pos に現在のポインティングデバイスの位置を設定する。  
EP\_STAT stat に現在のメタキー/ PD ボタン状態を設定する。  
EP\_TIME time に現在のイベントタイマーの値を設定する。

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

指定したイベントを発生させてイベントキューに入れる。  
イベントキューに空きがない場合、または、EV\_NULL および EV\_AUTKEY の指定したときはエラーとなる。  
システムイベントマスクで対象外のイベントは実際には発生せず無視される。  
発生したイベントは time の値にかかわらず、本システムコールを実行したときに発生したものとみなされ、常にイベントキューの最後に入れられる。

## 【エラーコード】

ER\_ADR : アドレス(evt)のアクセスは許されていない。  
ER\_IO : 入出力エラーが発生した(何らかのデバイスエラーが発生した)。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_PAR : パラメータが不正である(イベントのタイプが不正、optが不正)。

# clr\_evt

イベントのクリア

## 【形式】

ERR clr\_evt(W t\_mask, W last\_mask)

## 【パラメータ】

W t\_mask      クリア対象イベントタイプマスク  
              = EM\_ALL      全イベントタイプ

W last\_mask    クリア終了イベントタイプマスク  
              = EM\_ALL      イベントを1つだけクリア  
              = EM\_ALL      対象はイベントキューの最後まで

## 【リターン値】

= 0      正常  
< 0      エラー(エラーコード)

## 【解説】

発生済みのイベントをクリアする。  
イベントキューに入っているイベントのうち、t\_mask で指定したタイプのイベントを last\_mask で指定したタイプのイベントの直前までクリアする。last\_mask で指定したタイプのイベントはクリアされないが、last\_mask = EM\_ALL のときは、特別にイベントを1つだけクリアすることを意味する。  
t\_mask と last\_mask の指定例を以下に示す。

| t_mask | last_mask | 動作                       |
|--------|-----------|--------------------------|
| EM_ALL | EM_NULL   | 全てのイベントをクリア              |
| --     | EM_ALL    | t_mask で指定したイベントを1つだけクリア |
| EM_ALL | EM_ALL    | 先頭のイベントを1つだけクリア          |

## 【エラーコード】

ER\_PAR      : パラメータが不正である(t\_mask 0、last\_mask<0)。

# get\_pdp

PD位置の取得



## 【形式】

WERR get\_pdp(PNT\* pos)

## 【パラメータ】

PNT\* pos PD位置の格納領域

## 【リターン値】

0 正常(発生しているイベントのタイプ)  
<0 エラー(エラーコード)

## 【解説】

現在のポインティングデバイスの位置を絶対座標値で取り出す。  
同時に発生しているイベントのタイプを戻す。何のイベントも発生していないときはEV\_NULLを戻す。  
イベントキューの内容は一切変化しない。

## 【エラーコード】

ER\_ADR : アドレス(pos)のアクセスは許されていない。  
ER\_BUSY : PDはビジー状態である。  
ER\_DEV : デバイスに対しての操作は許されていない。  
ER\_ERDEV : 装置異常が発生した。  
ER\_IO : 入出力エラーが発生した。  
ER\_NODEV : デバイスへのアクセスができない。  
ER\_NOSPC : システムのメモリ領域が不足した。

# set\_pdp

PD位置の設定

## 【形式】

WERR set\_pdp(PNT pos)

## 【パラメータ】

PNT pos 設定するPD位置(絶対座標)

## 【リターン値】

= 0 正常 (PD位置設定可)  
= 1 正常 (PD位置設定不可)  
< 0 エラー (エラーコード)

## 【解説】

指定した絶対座標位置を現在のポインティングデバイスの位置として設定する。  
指定する位置はポインティングデバイスの座標レンジの範囲内であってはならない。  
位置の設定ができたときは関数値 "0" を戻し、ポインティングデバイスの動作が絶対動作タイプのため位置の設定ができなかったときは関数値 "1" を戻す。

## 【エラーコード】

ER\_BUSY : PDはビジー状態である。  
ER\_DEV : デバイスに対する操作は許されていない。  
ER\_ERDEV : 装置異常が発生した。  
ER\_IO : 入出力エラーが発生した。  
ER\_NODEV : デバイスへのアクセスができない。  
ER\_PAR : パラメータが不正である (pos が座標レンジ外)。  
ER\_NOSPC : システムのメモリ領域が不足した。

## get\_etm

イベントタイマー値の取得

## 【形式】

ERR get\_etm(UW\* time)

## 【パラメータ】

UW \*time イベントタイマー値の格納領域

## 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

## 【解説】

イベントタイマーの現在の値を取り出す。  
イベントタイマーはミリ秒単位の相対的な時間であるが、実際の分解能はインプリメントに

依存する。

本システムコールは一般的にミリ秒単位の相対時間を取り出すために使用される。

### 【エラーコード】

ER\_ADR : アドレス(time)のアクセスは許されていない。

## get\_kmp

キー状態の取得

### 【形式】

ERR get\_kmp(KeyMap keymap)

### 【パラメータ】

KeyMap keymap キー状態の格納領域

### 【リターン値】

= 0 正常

< 0 エラー(エラーコード)

### 【解説】

現在のキー状態を取り出す。

### 【エラーコード】

ER\_ADR : アドレス(keymap)のアクセスは許されていない。

ER\_BUSY : キーボードはビジー状態である。

ER\_DEV : デバイスに対しての操作は許されていない。

ER\_ERDEV : 装置異常が発生した。

ER\_IO : 入出力エラーが発生した。

ER\_NODEV : デバイスへのアクセスができない。

ER\_NOSPC : システムのメモリ領域が不足した。

## chg\_emk

システムイベントマスクの変更

### 【形式】

WERR chg\_emk(W mask)

### 【パラメータ】

W mask 設定するシステムイベントマスク  
<0 変更しない(現在のシステムイベントマスクの取得)

### 【リターン値】

0 正常(変更前のシステムイベントマスク)

### 【解説】

システムイベントマスクを指定した値に変更し、変更前のシステムイベントマスクの値を関数値として戻す。

mask <0 のときは変更せずに現在のシステムイベントマスクの値を関数値として戻す。

### 【エラーコード】

発生しない。

## set\_krp

自動リピート間隔の設定

### 【形式】

ERR set\_krp(W offset, W interval)

### 【パラメータ】

W offset 自動リピート最初の発生までの時間(ミリ秒)

W interval 自動リピート発生間隔(ミリ秒)

### 【リターン値】

= 0 正常

< 0 エラー(エラーコード)

### 【解説】

自動リピートキーイベント(EV\_AUTKEY)の発生までの時間、および間隔を設定する。設定する時間はミリ秒単位であるが、実際の分解能はインプリメントに依存する。

### 【エラーコード】

ER\_PAR : パラメータが不正である(offset 0, interval 0)。

## get\_krp

自動リピート間隔の取得

### 【形式】

ERR get\_krp(W\* offset, W\* interval)

### 【パラメータ】

W \*offset 自動リピート最初の発生までの時間(ミリ秒)の格納領域

W \*interval 自動リピート発生間隔(ミリ秒)の格納領域

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

自動リピートキーイベント(EV\_AUTKEY)の発生までの時間、および間隔を取り出す。

### 【エラーコード】

ER\_ADR : アドレス(offset, interval)のアクセスは許されていない。

## set\_krm

自動リピート対象キーの設定

### 【形式】

ERR set\_krm(KeyMap keymap)

## 【パラメータ】

KeyMap keymap 自動リピート対象キーマップ

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

自動リピートの対象となるキーを設定する。  
指定したキーマップの "1" のビットに対応するキーを自動リピートの対象とし、"0" のビットに対応するキーを自動リピートの対象としない。  
イベントを発生しないメタキーは自動リピートの対象としても無視される。

## 【エラーコード】

ER\_ADR : アドレス(keymap)のアクセスは許されていない。

get\_krm

自動リピート対象キーの取得

## 【形式】

ERR get\_krm(KeyMap keymap)

## 【パラメータ】

KeyMap keymap 自動リピート対象キーマップの格納領域

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

自動リピートの対象となるキーを取り出す。  
取り出したキーマップの "1" のビットに対応するキーが自動リピートの対象で、"0" のビットに対応するキーは自動リピートの対象でない。

## 【エラーコード】

ER\_ADR : アドレス(keymap)のアクセスは許されていない。

# get\_kid

キーボードIDの取得

## 【形式】

ERR get\_kid(KBD\_ID\* id)

## 【パラメータ】

KBD\_ID\* id キーボードIDの格納領域

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

キーボードIDを取り出す。

## 【エラーコード】

ER\_ADR : アドレス(id)のアクセスは許されていない。  
ER\_BUSY : キーボードはビジー状態である。  
ER\_DEV : デバイスに対しての操作は許されていない。  
ER\_ERDEV : 装置異常が発生した。  
ER\_IO : 入出力エラーが発生した。  
ER\_NODEV : デバイスへのアクセスができない。  
ER\_NOSPC : システムのメモリ領域が不足した。

# get\_ktb

文字コード変換表の取得

## 【形式】

WERR get\_ktb(KeyTab\* keytab)

### 【パラメータ】

KeyTab\* keytab 文字コード変換表の格納領域  
NULL 格納しない

### 【リターン値】

0 正常(文字コード変換表の全体のバイトサイズ)  
<0 エラー(エラーコード)

### 【解説】

現在設定されている文字コード変換表を取り出し、文字コード変換表全体のバイトサイズを関数値として戻す。

keytab = NULL は、文字コード変換表のバイトサイズを取り出すために使用する。

### 【エラーコード】

ER\_ADR : アドレス(keytab)のアクセスは許されていない。  
ER\_BUSY : キーボードはビジー状態である。  
ER\_DEV : デバイスに対しての操作は許されていない。  
ER\_ERDEV : 装置異常が発生した。  
ER\_IO : 入出力エラーが発生した。  
ER\_NODEV : デバイスへのアクセスができない。  
ER\_NOSPC : システムのメモリ領域が不足した。

## set\_ktb

文字コード変換表の設定

### 【形式】

ERR set\_ktb(KeyTab\* keytab)

### 【パラメータ】

KeyTab\* keytab 設定する文字コード変換表

### 【リターン値】

= 0 正常



<0 エラー(エラーコード)

### 【解説】

指定した文字コード変換表を設定する。

### 【エラーコード】

ER\_ADR : アドレス(keytab)のアクセスは許されていない。  
ER\_BUSY : キーボードはビジー状態である。  
ER\_DEV : デバイスに対しての操作は許されていない。  
ER\_ERDEV : 装置異常が発生した。  
ER\_IO : 入出力エラーが発生した。  
ER\_NODEV : デバイスへのアクセスができない。  
ER\_PAR : パラメータが不正である(keytab の内容が不正)。  
ER\_NOSPC : システムのメモリ領域が不足した。

## chg\_pda

ポインティングデバイス属性の変更

### 【形式】

WERR chg\_pda(W atr)

### 【パラメータ】

W atr ポインティングデバイス属性  
<0 変更しない(現在のポインティングデバイス属性の取得)

### 【リターン値】

0 正常(変更前のPD属性)  
<0 エラー(エラーコード)

### 【解説】

ポインティングデバイスの属性を変更し、変更前の属性を関数値として戻す。  
atr <0 のときは変更せずに現在のポインティングデバイス属性を関数値として戻す。  
ポインティングデバイス属性は以下の通りである。

xx..xx xxxV xxxA RRRR SSSS

A : 動作タイプ ( 0: 相対座標タイプ、 1: 絶対座標タイプ )  
S : 感度 ( 0: 最低感度 ~ 15: 最高感度 )

R: スキャン速度 ( 0: 最低速 ~ 15: 最高速 )  
V: 左右反転 ( 0: 右手モード、 1: 左手モード )  
x: 予約 ( = 0 )

動作タイプの変更が不可能の場合はエラー(ER\_DEV)となる。

## 【エラーコード】

ER\_BUSY : PDはビジー状態である。  
ER\_DEV : デバイスに対しての操作は許されていない。  
ER\_ERDEV : 装置異常が発生した。  
ER\_IO : 入出力エラーが発生した。  
ER\_NODEV : デバイスへのアクセスができない。  
ER\_PAR : パラメータが不正である(atr)。  
ER\_NOSPC : システムのメモリ領域が不足した。

## sig\_buz

ブザー音の発生

## 【形式】

ERR sig\_buz(W type)

## 【パラメータ】

W type ブザー音の種別  
= 0 標準の警告ブザー  
= -1 ブザーの停止  
0xTTTTHHHH

TTTT 0の場合

HHHH 0の場合 (ブザー)

T: 鳴らす時間 (1 ~ 32767 msec)

H: 鳴らす周波数 (0 ~ 32767 Hz)

HHHH<0の場合 (メロディー)

T: 繰り返し回数 (1 ~ 32767)

H: メロディー番号 (-1 ~ -100)

TTTT= 0の場合 (ブザー)

type = 0x0000thhh

t: 鳴らす時間 (100msec 単位)

h: 鳴らす周波数 (10Hz 単位)

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

指定した種類のブザー音またはメロディーを発生する。  
メロディーは beep ドライバーに登録しておく必要がある。メロディーの登録は、beep ドライバーへ直接行う。  
beep ドライバーがメロディーに対応していない場合は、メロディーの機能は使用できない。

### 【エラーコード】

ER\_DEV : デバイスに対しての操作は許されていない。  
ER\_ERDEV : 装置異常が発生した。  
ER\_IO : 入出力エラーが発生した。  
ER\_NODEV : デバイスへのアクセスができない。  
ER\_PAR : パラメータが不正である。

## ctl\_buz

ブザー音の制御

### 【形式】

ERR ctl\_buz(W kind, UW \*val)

### 【パラメータ】

W kind 制御操作の種別

UW \*val 制御操作の値

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

ブザー音に対して各種の制御操作を行う。

kind: GET\_BUZ\_BEEP 音色：標準の警告音の取り出し  
GET\_BUZ\_PDON 音色：ボタンONのクリック音の取り出し

|                 |                      |
|-----------------|----------------------|
| GET_BUZ_PD OFF  | 音色：ボタンOFFのクリック音の取り出し |
| GET_BUZ_KEY ON  | 音色：キーONのクリック音の取り出し   |
| GET_BUZ_KEY OFF | 音色：キーOFFのクリック音の取り出し  |
| GET_BUZ_VOLUME  | 音量：取り出し              |
| SET_BUZ_BEEP    | 音色：標準の警告音の設定         |
| SET_BUZ_PD ON   | 音色：ボタンONのクリック音の設定    |
| SET_BUZ_PD OFF  | 音色：ボタンOFFのクリック音の設定   |
| SET_BUZ_KEY ON  | 音色：キーONのクリック音の設定     |
| SET_BUZ_KEY OFF | 音色：キーOFFのクリック音の設定    |
| SET_BUZ_VOLUME  | 音量：設定                |

val: 音色: DTTT TTTT TTTT TTTT HHHH HHHH HHHH HHHH  
D: 0: 有効、1: 無効(音なし)  
H 0 の場合 (ブザー)  
T: 時間(msec)  
H: 周波数(Hz)  
H<0 の場合 (メロディー)  
T: 繰り返し回数  
H: メロディー番号

音量: LLLL LLLL LLLL LLLL RRRR RRRR RRRR RRRR  
L: 左チャンネル音量 (無音=0 ~ 最大=0xffff)  
R: 右チャンネル音量 (無音=0 ~ 最大=0xffff)  
音量は、すべての警告音、クリック音に共通である。

## 【エラーコード】

ER\_PAR : パラメータが不正である。  
ER\_ADR : アドレス(val)のアクセスは許されていない。

# req\_evt

イベントメッセージ要求

## 【形式】

ERR req\_evt(W t\_mask)

## 【パラメータ】

W t\_mask 対象イベントタイプマスク

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

t\_mask で指定したタイプのイベントが発生した場合に、そのイベントをメッセージとして自プロセス(req\_evt をコールしたプロセス)に送信することを要求する。  
ただし、EV\_NULL、EV\_AUTKEY はメッセージとして送信することはできない。  
メッセージとして送信されても、そのイベントはイベントキューから取り除かれない。

## メッセージ形式

```
struct {  
    W      msg_type;    /* メッセージタイプ = MS_SYS5 */  
    W      msg_size;    /* メッセージサイズ */  
    EVENT  evt;         /* イベント */  
    VB     info[];      /* 追加情報 */  
}
```

msg\_type は、MS\_SYS5 固定となる。

メッセージには、標準では発生したイベントが evt に格納される。また、イベントの種類によって、追加情報が info に格納される場合がある。したがって、msg\_size は最低で sizeof(EVENT) となり、追加情報があればそのサイズ分大きくなる。追加情報はその内容によりサイズが異なる。

デバイスイベント(EV\_DEVICE)の場合に、追加情報が付加される。追加情報は、デバイスドライバからの事象通知の内容そのものとなる。

rcv\_msg() によりメッセージを受信したときの送信プロセス ID は次のようになる。

- put\_evt() により発生されたイベントの場合には、put\_evt() を呼び出したプロセスの ID となる。
- その他の場合には、ID = 1 (初期プロセス)となる。

t\_mask に EM\_NULL を指定することにより、イベントメッセージの要求が解除される。また、プロセスが終了したときにも自動的に解除される。

プロセスのメッセージキューが一杯で送信できなかった場合には、そのイベントメッセージは単に捨てられる。

イベントメッセージ要求を同時に行えるプロセスの最大数は、システムで制限される。

## 【エラーコード】

ER\_PAR : パラメータが不正である。  
ER\_LIMIT : イベントメッセージ要求の登録数がシステムの制限を超えた。

## las\_evt

最終イベント発生からの経過時間の取得

## 【形式】

WERR las\_evt(W t\_mask)

### 【パラメータ】

W t\_mask 対象イベントタイプマスク

### 【リターン値】

0 正常(最終イベント発生からの経過時間)  
<0 エラー(エラーコード)

### 【解説】

t\_mask で指定したタイプのイベントの内、最後に発生したイベントから現在までの経過時間をミリ秒単位で返す。

t\_mask に EM\_BUTDWN または EM\_BUTUP が指定された場合には、ポインタの移動およびメニューボタンの操作もイベント発生と扱う。また、EM\_KEYDWN または EM\_KEYUP が指定された場合には、メタキーの状態変化もイベント発生と扱う。

t\_mask に EM\_NULL を指定することで、すべてのタイプのイベントの最終発生時刻に現在時刻がセットされる。

### 【エラーコード】

ER\_PAR : パラメータが不正である。

---

[この章の目次にもどる](#)

[前頁:1.6 ファイル管理にもどる](#)

[次頁:1.8 デバイス管理にすすむ](#)

## 1.8 デバイス管理

### 1.8.1 デバイス管理機能の概要

デバイス管理機能では、アプリケーションが各種のデバイスを統一的に取り扱うためのインタフェース機能、および各種のデバイスに対応したデバイスドライバの登録/管理機能、デバイスドライバとのインタフェース機能を提供している。ここでは、アプリケーションインタフェースのみについて記述する。

また、OS 核のファイル管理機能やイベント管理機能では、内部的にデバイス管理機能を経由して実際のデバイス操作を行なうことになる。

以下にデバイス管理機能の位置付けを示す。

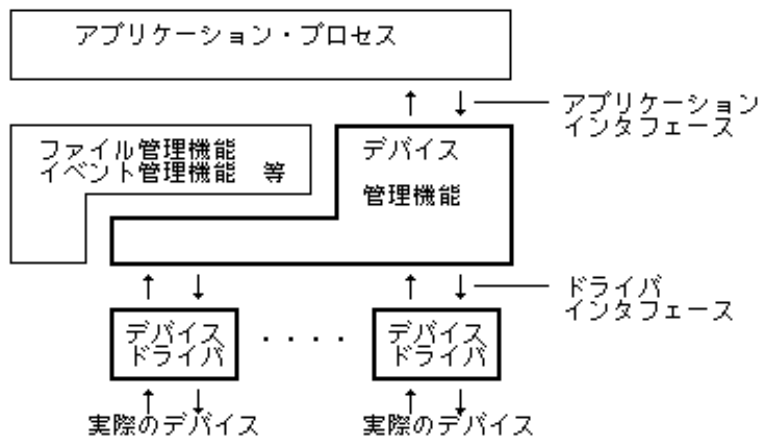


図 14: デバイス管理機能の位置付け

使用されるデバイスとしては以下のようなものがある。

1. ビットマップディスプレイ (CRT, LCD, 等)
2. キーボード (KD)
3. ポインティングデバイス (PD)
4. 各種ディスク装置 (FD, HD, OD, CD-ROM, 等)
5. 各種のプリンタ
6. 通信回線 (RS-232C, モデム、各種ネットワーク, 等)
7. スピーカ
8. イメージ処理装置
9. その他

これらのデバイスのうち、ビットマップディスプレイに対するアプリケーションインタフェースは、デバイス管理機能とは独立したディスプレイ関数群として提供される。

これらのディスプレイ関数群は通常ドライバと一体となっているためドライバインタフェースも適用されないことになり、デバイス管理機能とは基本的に独立した存在となる。

また、キーボードとポインティングデバイスに対するアプリケーションインタフェースはイベント管理機能として提供されるが、ドライバとのインタフェースはデバイス管理でサポートしている。

各種のディスク装置の場合は、通常はファイルとして使用されるが、デバイスとして直接取り扱うためのアプリケーションインタフェースも用意される。

### 1.8.2 デバイス

#### デバイスのタイプ

デバイスは文字型デバイスとブロック型デバイスの2種類に分類される。文字型デバイスは基本的にバイト単位でデータの入出力をシーケンシャルに行なうもので、ブロックデバイスはデバイス毎に定められた大きさのデータブ

ロック単位での入出力を行なうものでブロック単位でのランダムなアクセスが可能である。

各種のディスク装置はブロック型デバイスであり、その他のほとんどのデバイスは文字型デバイスとなる。

また、デバイスは実デバイスと仮想デバイスに分類される。実デバイスは物理的に実際に存在する通常の意味でのデバイスであり、仮想デバイスはある種の資源を仮想的に1つのデバイスとして取り扱うものである。仮想デバイスの代表的なものとしては画面上の1つのウィンドウを仮想的に1つのコンソールとして取り扱う仮想コンソールがあげられる。

### 論理デバイス名

デバイスは、論理デバイス名と呼ばれるユニークな文字列によりシステムに登録され、識別される。論理デバイス名は、基本的に以下に示す3つの要素から構成される。

種別:

デバイスの種類を表わす名称であり、"hd"(ハードディスク)、"fd"(フロッピーディスク)等がある。

ユニット:

同一種別のデバイスが複数個存在する場合に、個々のデバイスを識別するために使用され、通常、"a" ~ の英文字1文字が使用される。

サブユニット:

1つのユニットを論理的に複数に分割した場合に、個々の分割単位を識別するために使用され、"0" ~ "254"の最大3桁の連続した数字で表わされる。

種別 + ユニット を、特にユニット名と呼び、OS で識別される単位となる。OS 側では種別とユニットの区別は認識せず、両方を合わせた名称であるユニット名により、デバイスを識別する。ユニット名の最後の文字は数字であってはいけない。

論理デバイス名は、「ユニット名 + サブユニット番号」で表わされ、全体として最大8文字(16バイト)の文字列となる。サブユニット番号は前に0の付かない3桁までの数字である。サブユニットが存在しない場合、ユニット名がそのまま論理デバイス名と成り得る。論理デバイス名の最後の文字が数字である場合は、その論理デバイス名はサブユニットを表わしているものと見なされる。サブユニット全体を表わす場合、サブユニット番号は付けず、ユニット名そのものが論理デバイス名となる。

サブユニット(パーティション)を表わすために最大3桁の数字(サブユニット番号)が使用されるが、サブユニットの分割を無視した1つのユニットとして取り扱う場合は、サブユニット番号を含めない論理デバイス名(ユニット名)を使用する。これは、主にサブユニットの分割(即ち、ディスクのパーティショニング)を行なうために使用される。なお、サブユニット番号を含まない論理デバイス名(ユニット名)を特に物理デバイス名と呼ぶ。



図 15: 論理デバイス名

デバイスの登録は、実際には対応するデバイスドライバの登録を意味し、デバイスドライバはユニット単位でユニット名により定義される。即ち、1つのユニットに含まれるすべてのサブユニットは1つのデバイスドライバで処理される。通常、デバイスドライバはデバイスの種別に対応して用意され、1つのデバイスドライバで複数のユニットを処理するケースが多いが、複数のユニットを複数のデバイスドライバで処理してもよい。

なお、今後単にデバイス名と言った場合は、論理デバイス名を意味するものとする。

### デバイスのアクセス制御

ディスク等のデバイスでは、システムの安全性を保つために、そのデバイスを使用できるユーザを制限することが望ましい。このため、プロセスのユーザレベルによるデバイスのアクセス制御が行なわれる。

各デバイスには以下に示す1ワードのアクセスモードが保持されており、これによりデバイスのアクセス権がチェックされる(これはファイルに於ける一般アクセスレベルに相当するものである)。





R : 読込可能な最低のユーザレベル (0 ~ 15)  
W : 書込可能な最低のユーザレベル (0 ~ 15)  
F : ファイルシステムとしての接続、切断、管理情報読込みが可能な最低のユーザレベル (0 ~ 15)  
x : 未使用 (0)

図 16 : デバイスのアクセスモード

即ち、読込可能なユーザレベルを持つプロセスからのみデバイスの読込用 (D\_READ) のオープンが可能で、書込可能なユーザレベルを持つプロセスからのみデバイスの書込用 (D\_WRITE) のオープンが可能となる。また、アクセス可能なユーザレベルを持つプロセスからのみ、そのデバイスをファイルシステムとして取り扱うことが可能であり、ファイルシステムの接続 / 切断 / 管理情報読込みのファイル管理のシステムコールを実行できる。

デバイスのアクセスモードは、デバイスの登録時に R = W = F = 15 に設定される。登録後はユーザレベル 0 のプロセスからのみ変更可能となる。

### 1.8.3 データ / 定数の定義

#### オープンモード

```
#define D_READ      0x0001    /* 読み込み専用オープン */
#define D_WRITE     0x0002    /* 書き込み専用オープン */
#define D_UPDATE   0x0003    /* 更新用オープン */
#define D_EXCL     0x0100    /* 排他モード指定 */
#define D_WEXCL    0x0200    /* 排他書き込みモード指定 */
#define D_NOWAIT   0x8000    /* 待ちなしモード指定 */
```

#### デバイス管理情報

```
typedef struct dev_state {
    UW attr;      /* デバイスの属性 */
    UW mode;     /* アクセスモード */
    W  blksize;  /* 物理ブロックサイズ( -1 :不明) */
    W  wrpt;     /* 書き込み可否( 0:許可 1:禁止 -1:不明) */
} DEV_STATE;
```

#### デバイス情報

```
#define L_DEVM    8    /* デバイス名の長さ */
```

```
typedef struct {
    UW attr;      /* デバイスの属性 */
    W  nsub;     /* サブユニット数 */
    TC name[L_DEVM]; /* ユニット名 */
} DEV_INFO;
```

デバイスの属性 (attr : IIII IIII IIII IIII CRxx xxxx KKKK KKKK)

```
#define DA_DEVINFO 0xFFFF0000 /* デバイス/メディア依存情報 */
#define DA_CHARDEV 0x8000    /* 文字型デバイス */
#define DA_REMOVABLE 0x4000  /* 取り外し可能 */

#define DA_DEVKIND 0xFF      /* デバイス/メディア種別 */
#define DA_DEVTYPE 0xF0     /* デバイス/メディアタイプマスク */

#define DK_UNDEF   0x00     /* 未定義/不明 */
#define DK_DISK    0x10     /* ディスクタイプ */
```

```

#define DK_DISK_UNDEF    0x10    /* その他のディスク */
#define DK_DISK_RAM      0x11    /* RAM ディスク */
#define DK_DISK_ROM      0x12    /* ROM ディスク */
#define DK_DISK_FLA     0x13    /* FLASH ROM / SS ディスク */
#define DK_DISK_FD       0x14    /* フロッピーディスク */
#define DK_DISK_HD       0x15    /* ハードディスク */
#define DK_DISK_CDROM    0x16    /* CD-ROM */

```

## サスペンド要求モード

```

#define D_SUSPEND    0x0001 /* サスペンドする */
#define D_DISSUS    0x0002 /* サスペンドを禁止する */
#define D_ENASUS    0x0003 /* サスペンドを許可する */
#define D_CHECK     0x0004 /* サスペンド禁止カウントを調べる */
#define D_FORCE     0x8000 /* 強制サスペンド指定 */

```

## 1.8.4 システムコール

### opn\_dev

デバイスのオープン

#### 【形式】

WERR opn\_dev(TC\* dev, W o\_mode, W\* error)

#### 【パラメータ】

|          |  |
|----------|--|
| TC* dev  | 対象デバイス名  |
| W o_mode | オープンモード<br>( D_READ D_WRITE D_UPDATE )   [ D_EXCL D_WEXCL ]<br>  [ D_NOWAIT ]  |
| D_READ   | 読み込み用オープン(Rアクセス権が必要)   |
| D_WRITE  | 書き込み用オープン(Wアクセス権が必要)   |
| D_UPDATE | 更新(読み込み / 書き込み)用オープン(RWアクセス権が必要)   |
| D_EXCL   | 排他モード  |
| D_WEXCL  | 排他書き込みモード  |
| D_NOWAIT | 待ち無し指定   |
|          | 文字型デバイスの場合、オープン時にデバイスがレディ状態になるまで待たずにリターンする。この指定でオープンした場合は、rea_dev(), wri_dev() で入力データが得られなかったり、出力がビジー状態の場合には、待たずにリターンする。 |
| W* error | 詳細エラー情報格納領域<br>*error には、デバイスドライバから戻されたエラー情報が設定される。エラー情報の内容は、各デバイスドライバによって異なる。error が NULL の場合、エラー情報は格納しない。              |

#### 【リターン値】

>0 正常(デバイスディスクリプタ)  
<0 エラーコード

#### 【解説】

指定したデバイスを指定したモードでオープンする。デバイスをオープンするためにはオープンモードに対応するアクセス権が必要である。排他モード、および排他書き込みモードはファイルに対するものと全く同様の機能であり、1つのデバイスの同時オープン

を制限することになる。なお、物理デバイス名によるオープンは、対象とするユニットのすべての論理デバイスに対するオープンと同時に進んだものとみなされた排他制御が行なわれる。例えば、"hda" の D\_EXCL オープンは、"hda#" が1つもオープンされていない場合のみ可能であり、逆に "hda" を D\_EXCL オープンした後は、どの "hda#" のオープンも不可となる。

#### 【エラーコード】

ER\_ACCES : デバイス(dev)のアクセス権(R,W)がない。  
ER\_ADR : アドレス(dev,error)のアクセスは許されていない。  
ER\_BUSY : デバイス(dev)は既に排他的にオープンされている為、同時にデバイスをオープンすることができない、またはファイルシステムとして接続されている。  
ER\_DEV : デバイス(dev)に対しての読み込みまたは書き込み操作は許されていない。  
ER\_ERDEV : 装置異常が発生した。  
ER\_IO : 入出力エラーが発生した。  
ER\_LIMIT : 同時オープン可能な最大デバイス数を越えた。  
ER\_MINTR : メッセージハンドラが起動されたため、処理が中断された。  
ER\_NODEV : デバイス(dev)へのアクセスができない。  
ER\_NOEXS : デバイス(dev)は存在していない(登録されていない)。  
ER\_NOMDA : デバイス(dev)のメディアが存在しない。  
ER\_NOSPC : システムのメモリ領域が不足した。  
ER\_PAR : パラメータが不正である(o\_modeが不正)。  
ER\_RDONLY : デバイス(dev)は書込不可である。

## cls\_dev

デバイスのクローズ

#### 【形式】

ERR cls\_dev(W dd, W eject, W\* error)

#### 【パラメータ】

W dd デバイスディスクリプタ  
W eject イジェクト指定  
= 0 イジェクトしない  
0 イジェクトする(イジェクト不可能なデバイスの時は無視)  
W\* error 詳細エラー情報格納領域  
\*error には、デバイスドライバから戻されたエラー情報が設定される。  
エラー情報の内容は、各デバイスドライバによって異なる。  
error が NULL の場合、エラー情報は格納しない。

#### 【リターン値】

= 0 正常  
< 0 エラーコード

#### 【解説】

オープンしたデバイスをクローズする。  
デバイスをオープンしたプロセスが終了したときには、自動的にそのデバイスはクローズされる。

#### 【エラーコード】

ER\_ADR : アドレス(error)のアクセスは許されていない。  
ER\_DD : デバイスディスクリプタは存在していない。

ER\_IO : 入出力エラーが発生した。  
ER\_MINTR : メッセージハンドラが起動されたため、処理が中断された。

## rea\_dev

デバイスのデータ読み込み

### 【形式】

ERR rea\_dev(W dd, W start, B\* buf, W size, W\* a\_size, W\* error)

### 【パラメータ】

W dd デバイスディスクリプタ

W start 読み込み開始データ番号  
0 固有データ  
< 0 属性データ

B\* buf 読み込みデータの格納領域

W size 読み込みデータサイズ  
固有データ 単位はデバイス依存  
属性データ 単位はバイト

W\* a\_size 読み込んだデータサイズ  
固有データ 単位はデバイス依存  
属性データ 単位はバイト

W\* error 詳細エラー情報格納領域  
\*error には、デバイスドライバから戻されたエラー情報が設定される。  
エラー情報の内容は、各デバイスドライバによって異なる。  
error が NULL の場合、エラー情報は格納しない。

### 【リターン値】

= 0 正常

< 0 エラーコード

### 【解説】

オープンしたデバイスのデータを指定した開始データ番号から指定したサイズだけ読み込む。  
size = 0 の時は実際の読み込みは行わずに、a\_size に読み込み可能なデータサイズを戻す。  
文字型デバイスの場合は、オープン時の指定により以下に示す動作となる(詳細は個々のデバイスに依存する)。

D\_NOWAIT 指定の時:

その時点で得られたバイト数(size で指定したバイト数以下)のデータだけを読み込む。何もデータが無い場合は、\*a\_size に 0 がセットされ、ER\_BUSY のエラーとなる。

D\_NOWAIT 指定でない時:

size で指定したバイト数のデータが得られるか、データが終了となるまで待つ。

### 【エラーコード】

ER\_ADR : アドレス(buf, a\_size, error)のアクセスは許されていない。  
ER\_BUSY : データは得られなかった(D\_NOWAITオープン時)  
ER\_DD : デバイスディスクリプタは存在していない、  
または D\_WRITE オープンである。  
ER\_DEV : デバイス(dd)に対しての読み込み操作は許されていない。

ER\_ERDEV : 装置異常が発生した。  
ER\_IO : 入出力エラーが発生した。  
ER\_MINTR : メッセージハンドラが起動されたため、処理が中断された。  
ER\_NODEV : デバイス(dd)へのアクセスができない。  
ER\_NOMDA : デバイス(dd)のメディアが存在しない。  
ER\_PAR : パラメータが不正である(size< 0,start不正)。

## wri\_dev

デバイスへのデータ書き込み

### 【形式】

ERR wri\_dev(W dd, W start, B\* buf, W size, W\* a\_size, W\* error)

### 【パラメータ】

W dd デバイスディスクリプタ

W start 書き込み開始データ番号  
0 固有データ  
< 0 属性データ

B\* buf 書き込みデータ

W size 書き込みデータサイズ  
固有データ 単位はデバイス依存  
属性データ 単位はバイト

W\* a\_size 書き込んだデータサイズ  
固有データ 単位はデバイス依存  
属性データ 単位はバイト

W\* error 詳細エラー情報格納領域  
\*error には、デバイスドライバから戻されたエラー情報が設定される。  
エラー情報の内容は、各デバイスドライバによって異なる。  
error が NULL の場合、エラー情報は格納しない。

### 【リターン値】

=0 正常  
<0 エラーコード

### 【解説】

オープンしたデバイスへ指定した開始データ番号から指定したサイズだけデータを書き込む。  
size = 0 の時は実際の書き込みは行わずに、a\_size に書き込み可能なデータサイズを戻す。  
文字型デバイスの場合は、オープン時の指定により以下に示す動作となる(詳細は個々のデバイスに依存する)。

D\_NOWAIT 指定の時:

その時点で得られたバイト数(size で指定したバイト数以下)のデータだけを読み込む。何もデータが無い場合は、\*a\_size に 0 がセットされ、ER\_BUSY のエラーとなる。

D\_NOWAIT 指定でない時:

size で指定したバイト数のデータが得られるか、データが終了となるまで待つ。

### 【エラーコード】

ER\_ADR : アドレス(buf,a\_size,error)のアクセスは許されていない。  
ER\_BUSY : データは書き込めなかった(D\_NOWAITオープン時)。

ER\_DD : デバイスディスクリプタは存在していない、  
または D\_READ オープンである。  
ER\_DEV : デバイス(dev)に対しての書込み操作は許されていない。  
ER\_ERDEV : 装置異常が発生した。  
ER\_IO : 入出力エラーが発生した。  
ER\_MINTR : メッセージハンドラが起動されたため、処理が中断された。  
ER\_NODEV : デバイス(dd)へのアクセスができない。  
ER\_NOMDA : デバイス(dd)のメディアが存在しない。  
ER\_PAR : パラメータが不正である(size< 0,start不正)。  
ER\_RDONLY : デバイス(dev)は書込不可である。

## chg\_dmd

デバイスのアクセスモード変更

### 【形式】

ERR chg\_dmd(TC\* dev, W mode)

### 【パラメータ】

TC\* dev 対象デバイス名

W mode アクセスモード

### 【リターン値】

=0 正常  
<0 エラーコード

### 【解説】

指定したデバイスのアクセスモードを変更する。本システムコールはユーザレベル0のプロセスからのみ可能である。

### 【エラーコード】

ER\_ACCES : レベル0のユーザでない。  
ER\_ADR : アドレス(dev)のアクセスは許されていない。  
ER\_NOEXS : デバイス(dev)は存在していない(登録されていない)。  
ER\_PAR : パラメータが不正である(modeが不正)。

## dev\_sts

デバイスの管理情報の取得

### 【形式】

ERR dev\_sts(TC\* dev, DEV\_STATE\* buf)

### 【パラメータ】

TC\* dev 対象デバイス名

DEV\_STATE\* buf デバイス管理情報の格納領域

```

typedef struct dev_state {
    UW attr;      /* デバイスの属性 */
    UW mode;     /* アクセスモード */
    W  blksize;  /* 物理ブロックサイズ(- 1:不明) */
    W  wrpt;     /* 書き込み可否(0:許可 1:禁止 - 1:不明) */
} DEV_STATE;

```

**attr** デバイスの属性を示す。  
**mode** デバイスのアクセスモードを示す。  
**blksize** デバイスの物理ブロックサイズをバイト数で示したもので、  
 rea\_dev(), wri\_dev() での単位となる値である。  
 文字型デバイスの場合は常に "1" となる。  
**wrpt** デバイスへの書き込みが禁止されているか否かを示す状態であり、書き込み禁止  
 の時は "1" となり、書き込み許可の時は "0" となる。

### 【リターン値】

= 0 正常  
 < 0 エラーコード

### 【解説】

指定したデバイスの管理情報を取り出す。

### 【エラーコード】

ER\_ADR : アドレス(dev, buf)のアクセスは許されていない。  
 ER\_NOEXS : デバイス(dev)は存在していない。

## get\_dev

デバイス名の取得

### 【形式】

WERR get\_dev(TC\* dev, W devno)

### 【パラメータ】

TC\* dev デバイス名の格納領域

W devno デバイス番号

### 【リターン値】

0 正常(デバイスのサブユニット数)  
<0 エラーコード

### 【解説】

指定したデバイス番号のデバイスのユニット名を取り出し、デバイスのサブユニット数を関数値として戻す。サブユニットが存在しないときは関数値は "0" となる。

指定するデバイス番号はデバイスイベントとして得られたデバイス番号である。

### 【エラーコード】

ER\_ADR : アドレス(dev)のアクセスは許されていない。  
ER\_NOEXS : デバイス番号(num)は存在していない。

## lst\_dev

登録済みデバイスの取得

### 【形式】

WERR lst\_dev(DEV\_INFO\* dev, W ndev)

### 【パラメータ】

DEV\_INFO\* dev デバイス情報の格納領域(配列)  
NULL 格納しない

```
typedef struct {  
    UW attr;          /* デバイスの属性 */  
    W nsub;           /* サブユニット数 */  
    TC name[L_DEVNM]; /* ユニット名 */  
} DEV_INFO;
```

W ndev デバイス情報の格納領域(配列)の要素数  
0 格納しない



## 【リターン値】

0 正常(登録されているデバイス数)  
<0 エラーコード

## 【解説】

登録済みのデバイス(ユニット)の情報を取り出し、登録済みのデバイス(ユニット)数を関数値として戻す。

登録済みのデバイスの数が ndev より小さいときは登録されている数だけ dev に格納され、登録済みのデバイスの数が ndev より多い場合は ndev 個のみ dev に格納される。

dev = NULL または ndev = 0 は、登録済みのデバイスの数を取り出すために使用する。

## 【エラーコード】

ER\_ADR : アドレス(dev)のアクセスは許されていない。  
ER\_PAR : パラメータが不正である(ndev< 0)。

**sus\_dev**

サスペンド要求

## 【形式】

WERR sus\_dev(UW mode)

## 【パラメータ】

mode ::= ( D\_SUSPEND D\_DISSUS D\_ENASUS D\_CHECK ) | [D\_FORCE]

|           |        |                 |
|-----------|--------|-----------------|
| D_SUSPEND | 0x0001 | サスペンドする         |
| D_DISSUS  | 0x0002 | サスペンドを禁止する      |
| D_ENASUS  | 0x0003 | サスペンドを許可する      |
| D_CHECK   | 0x0004 | サスペンド禁止カウントを調べる |
| D_FORCE   | 0x8000 | 強制サスペンド指定       |

## 【リターン値】

0 正常 (D\_CHECK のときはサスペンド禁止要求カウント数)

<0 エラーコード

### 【解説】

システムのサスペンドの制御を行う。

D\_SUSPEND

サスペンドおよび強制サスペンド許可状態なら、サスペンドを行う。サスペンド状態から復帰(リジューム)すると、システムコールからリターンする。

D\_DISSUS

サスペンドを禁止する。

D\_ENASUS

サスペンドを許可する。

D\_DISSUS の要求回数はカウントされ、同じ回数だけ D\_ENASUS を要求して初めてサスペンドが許可される。

初期状態は、サスペンド許可である。

D\_CHECK

現在のサスペンド禁止要求カウント数を調べ、リターン値に返す。0ならサスペンド許可状態である。

D\_FORCE が指定されると、強制サスペンドに関する操作となる。

D\_FORCE|D\_SUSPEND

強制サスペンド許可状態なら、サスペンドを行う。D\_DISSUS による通常のサスペンド禁止は無視される。

D\_FORCE|D\_DISSUS

強制サスペンドを禁止する。

D\_FORCE|D\_ENASUS

強制サスペンドを許可する。

D\_FORCE|D\_DISSUS の要求回数はカウントされ、同じ回数だけ D\_FORCE|D\_ENASUS を要求して初めて強制サスペンドが許可される。

初期状態は、強制サスペンド許可である。

D\_FORCE|D\_CHECK

現在の強制サスペンド禁止要求カウント数を調べ、リターン値に返す。0なら強制サスペンド許可状態である。

サスペンドおよび強制サスペンドを禁止したプロセスが終了すると、そのプロセスが行った禁止要求は解除される。

### 【エラーコード】

ER\_BUSY : サスペンド禁止状態のためサスペンドできなかった。  
ER\_PAR : パラメータが不正である。  
ER\_LIMIT : サスペンド禁止要求カウントがシステムの制限を超えた。  
ER\_MINTR : メッセージハンドラが起動されたため、処理が中断された。

[前頁:1.7 イベント管理にもどる](#)

[次頁:1.9 時計管理にすすむ](#)

## 1.9 時計管理

### 1.9.1 時計管理機能の概要

時計管理機能では、システム内部で保持している基準時間であるシステム時間の取出し / 設定、およびカレンダー日付時刻との変換機能を提供している。

システム時間はグリニッチ標準時 GMT (Greenwich Mean Time) の 1985年1月1日 00:00 からの秒数で表わされた32ビットの値であり、ファイルの生成 / 更新 / アクセス日時等のシステム内部の時間表現として使用される。

システム時間に対して、マシンが実際に存在する地域の時間をローカル時間と呼ぶ。時計管理機能ではシステム時間とローカル時間との時差を保持し、ローカル時間での日付時刻の取出し / 設定機能も提供している。

システム時間とローカル時間の関係は、以下に示す時間補正データとして定義される。

```
typedef struct {
    W      adjust;          /* システム時間との時差(秒) */
    W      dst_flg;         /* DST の適用タイプ */
    W      dst_adj;        /* DST 調整時間(分) */
} TIMEZONE;
```

adjust はローカル時間とシステム時間 (GMT) との時差であり、グリニッチから西方向への秒単位で表現される、 $-(12 \times 60 \times 60) \sim +(12 \times 60 \times 60)$  の範囲の値である。

dst\_flg はサマータイム (Daylight Saving Time, DST) の適用タイプを示し 0 は適用しないことを示す。0 以外の値は適用することを示すが、その意味は時計管理機能では関知せず、単に 0 か 0 でないかの判断のみを行なっている。

dst\_adj は DST による時間の調整時間 (分) を示す  $-(12 \times 60) \sim +(12 \times 60)$  の値である。時計管理機能では DST が適用されるか否かの判断は行なわず、システムプログラムが DST が実際に適用される期間の開始時に、dst\_adj を適当な値に設定し、DST が適用される期間の終了時に、dst\_adj を 0 に設定することを想定している。

時間補正データにより、ローカル時間は以下の式で定義されることになる。

$$\text{ローカル時間(秒)} = \text{システム時間(秒)} - \text{adjust} + (\text{dst\_flg} ? (\text{dst\_adj} \times 60) : 0)$$

なお、時間の設定はユーザレベル0のプロセス以外から行なうことは禁止されている。

また、時計管理機能では、以下に示す構造体で定義されるカレンダー日付時刻をサポートしており、システム時間との変換機能が提供されている。

```
typedef struct {
```

```

W      d_year;    /* 1990年からのオフセット(85~) */
W      d_month;  /* 月 ( 1 ~ 12, 0) */
W      d_day;    /* 日 ( 1 ~ 31 ) */
W      d_hour;   /* 時 ( 0 ~ 23 ) */
W      d_min;    /* 分 ( 0 ~ 59 ) */
W      d_sec;    /* 秒 ( 0 ~ 59 ) */
W      d_week;   /* 週 ( 1 ~ 54 ) */
W      d_wday;   /* 曜日 ( 0 ~ 6, 0が日曜) */
W      d_days;   /* 日 ( 1 ~ 366 ) */
} DATE_TIM;

```

d\_week は、その年の1月1日の週を1とした場合の週の数という意味し、d\_days は、その年の1月1日を1とした場合の日数を意味する。また、d\_month = 0 は、特殊な意味として使用される。

## 1.9.2 データ / 定数の定義

時計管理システムコールはすべて WORD 型の関数値をとり、何らかのエラーがあった場合は「負」のエラーコードが戻る。正常終了時には「0」が戻る。

### 時間補正データ定義

```

typedef struct {
W      adjust;   /* システム時間との時差(秒) */
W      dst_flg;  /* DSTの適用タイプ */
W      dst_adj;  /* DST調整時間(分) */
} TIMEZONE;

```

### カレンダー日付時刻定義

```

typedef struct {
W      d_year;    /* 1900年からのオフセット(85~) */
W      d_month;  /* 月 ( 1 ~ 12, 0) */
W      d_day;    /* 日 ( 1 ~ 31 ) */
W      d_hour;   /* 時 ( 0 ~ 23 ) */
W      d_min;    /* 分 ( 0 ~ 59 ) */
W      d_sec;    /* 秒 ( 0 ~ 59 ) */
W      d_week;   /* 週 ( 1 ~ 54 ) */
W      d_wday;   /* 曜日 ( 0 ~ 6, 0が日曜) */
W      d_days;   /* 日 ( 1 ~ 366 ) */
} DATE_TIM;

```

## 1.9.3 システムコール

# get\_tim

システム時間の取得

## 【形式】

ERR get\_tim(STIME\* time, TIMEZONE\* tz)

## 【パラメータ】

STIME\* time      システム時間の格納領域  
                  NULL 格納しない

TIMEZONE\* tz     時間補正データの格納領域  
                  NULL 格納しない

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

現在のシステム時間、および時間補正データを取り出す。

## 【エラーコード】

ER\_ADR           : アドレス(time,tz)のアクセスは許されていない。

# set\_tim

システム時間の設定

## 【形式】

ERR set\_tim(STIME time, TIMEZONE\* tz)

## 【パラメータ】

STIME time        設定するシステム時間  
                  0    設定しない

TIMEZONE\* tz 設定する時間補正データ  
NULL 設定しない

### 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

現在のシステム時間、および時間補正データ設定する。

本システムコールの実行はユーザレベル 0 のプロセスのみ許可される。

### 【エラーコード】

ER\_ACCES : レベル 0 のユーザでない。  
ER\_ADR : アドレス(tz)のアクセスは許されていない。  
ER\_PAR : パラメータが不正である(tz の内容が範囲外)。

## get\_tod

日時の取得 / 変換

### 【形式】

ERR get\_tod(DATE\_TIM\* dt, STIME time, Bool local)

### 【パラメータ】

DATE\_TIM\* dt 変換した日時の格納領域

STIME time 変換するシステム時間  
0 現在のシステム時間を変換

Bool local ローカル時間指定  
= 0 GMT 時間  
dt には時間補正を行わない結果を戻す。  
0 ローカル時間  
dt には時間補正を行った結果を戻す。

### 【リターン値】

=0 正常  
<0 エラー(エラーコード)

### 【解説】

指定したシステム時間または現在のシステム時間をカレンダー日付時刻に変換する。

### 【エラーコード】

ER\_ADR : アドレス(data\_tim)のアクセスは許されていない。

## set\_tod

日時の設定 / 変換

### 【形式】

ERR set\_tod(DATE\_TIM\* dt, STIME\* time, Bool local)

### 【パラメータ】

|           |       |   |
|-----------|-------|---|
| DATE_TIM* | dt    | 設定 / 変換する日時   |
| STIME*    | time  | 変換したシステム時間の格納領域<br>NULL 現在のシステム時間として設定する<br>(ユーザレベル 0 のプロセスのみ許可)                  |
| Bool      | local | ローカル時間指定<br>= 0 GMT 時間<br>dt は時間補正を行わない日時とみなす。<br>0 ローカル時間<br>dt は時間補正を行った日時とみなす。 |

### 【リターン値】

=0 正常  
<0 エラー(エラーコード)

### 【解説】

指定したカレンダー日付時刻をシステム時間に変換する。

指定した日時の内容のうち、d\_week ( 週 ), d\_wday ( 曜日 ) は無視される。 d\_days ( 日数 ) の値は d\_month = 0 のときのみ有効であり、 d\_month 0 のときは無視される。



## 【エラーコード】

ER\_ACCES : レベル 0 のユーザでない(time=NULLの時)。  
ER\_ADR : アドレス(tz)のアクセスは許されていない。  
ER\_PAR : パラメータが不正である(dat\_tim の内容が範囲外)。

---

[この章の目次にもどる](#)

[前頁:1.8 デバイス管理にもどる](#)

[次頁:1.10 システム管理機能にすすむ](#)

[この章の目次にもどる](#)

[前頁:1.9 時計管理にもどる](#)

[次頁:第2章 ディスプレイプリミティブにすすむ](#)

---

## 1.10 システム管理機能

システム管理機能では、下記のような機能を提供している。

- 例外処理
- OS構成情報の取得

### 1.10.1 データ / 定数の定義

OSバージョン

システム管理機能として OS バージョンの取得機能が用意されている。

```
typedef struct {
    UH maker;      /* メーカー */
    UH id;         /* 形式番号 */
    UH spver;     /* 仕様書バージョン */
    UH prver;     /* 製品バージョン */
    UH prno[4];   /* 製品管理情報 */
    UH cpu;       /* CPU 情報 */
    UH var;       /* バリエーション記述子 */
} T_VER;
```

### 1.10.2 システムコール

#### def\_exc

例外処理ハンドラ定義

【形式】

ERR def\_exc(W exckind, FP exchr)

【パラメータ】

W exckind 例外要因(インプリメント依存)  
FP exchr 例外処理ハンドラ開始アドレス  
NULL 例外処理ハンドラの定義解除

【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

### 【解説】

指定した例外要因に対応する自タスクの例外処理ハンドラを定義する。

同じ要因に対する例外処理ハンドラがすでに定義されている場合には、後から定義した例外処理ハンドラが有効となる。

例外要因の内容はインプリメントに依存する。

### 【エラーコード】

ER\_ADR : アドレス(exc\_hdr)のアクセスは許されていない。  
ER\_CTX : コンテキストエラー(使用不可のシステムコールを発行した)。  
ER\_PAR : パラメータが不正である(exc\_vecで定義した例外要因が不正)。

## ret\_exc

例外処理ハンドラ終了

### 【形式】

VOID ret\_exc(W ret)

### 【パラメータ】

W ret リターン指定

= 0

例外が発生した位置から実行を再開し、本システムコールからは戻らない。再開が不可能な種類の例外の場合には、プロセスは異常終了する。

0

例外が発生した位置からは再開せずに、本システムコールから戻って、そのまま実行を続行する。この場合、通常はハンドラの最後で longjmp () により制御を他へ移行することになる。

### 【リターン値】

リターンしない (ret = 0 の時)

= 0 正常 (ret = 0 の時)

### 【解説】

例外処理ハンドラを終了する。

例外処理ハンドラの最後には必ず本システムコールを実行しなくてはならない。また、本システムコールは例外処理ハンドラ内でのみ発行することができ、それ以外の部分で発行された時はシステムエラーとなる。

## 【エラーコード】

発生しない。

get\_ver

バージョンの取得

## 【形式】

ERR get\_ver(T\_VER\* version)

## 【パラメータ】

T\_VER\* version バージョンの格納領域

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

OSのバージョンを取り出す。

T\_VERはTRON体系で共通の以下の構造体である。

```
typedef struct {
    UH maker;      -- メーカー
    UH id;         -- 形式番号
    UH spver;     -- 仕様書バージョン
    UH prver;     -- 製品バージョン
    UH prno[4];   -- 製品管理情報
    UH cpu;       -- CPU 情報
    UH var;       -- バリエーション記述子
} T_VER;
```

## 【エラーコード】

ER\_ADR : アドレス(version)のアクセスは許されていない。

## get\_cnf

コンフィグレーション情報の取得

### 【形式】

WERR get\_cnf(TC\* name, VP value, W len)

### 【パラメータ】

TC\* name コンフィグレーション項目名

VP value コンフィグレーション情報の格納領域

W len コンフィグレーション情報の格納領域のバイトサイズ

### 【リターン値】

= 0 正常(W の数値情報)  
> 0 正常(TC 文字列情報 : 文字数)  
< 0 エラー(エラーコード)

### 【解説】

指定した項目名のシステムのコンフィグレーション情報を取り出す。

コンフィグレーション情報は W の数値または TC 文字列であり、関数値により区別される。

関数値 = 0 :

W の数値情報

関数値 > 0 :

TC 文字列情報

関数値は文字数で、文字列の最後には 0 が格納される。

格納領域が不足するときは格納領域のサイズ分のみ格納される。

コンフィグレーション情報の項目名はインプリメントに依存する。

### 【エラーコード】

ER\_ADR : アドレス(name)のアクセスは許されていない。

ER\_NOEXS : 項目名(name)は存在していない。

ER\_PAR : パラメータが不正である。

[この章の目次にもどる](#)

[前頁:1.9 時計管理にもどる](#)

[次頁:第2章 ディスプレイプリミティブにすすむ](#)

[この章の目次にもどる](#)

[前頁:第2章 ディスプレイプリミティブにもどる](#)

[次頁:2.2 基本概念にすすむ](#)

---

## 2.1 はじめに

### 2.1.1 ディスプレイプリミティブの概要

ディスプレイプリミティブは VRAM や主メモリ上のメモリビットマップ上に各種の図形や文字を表示するための基本的な関数群であり、HMI 機能を提供する各種外殻の他、アプリケーションでも使用される。

ディスプレイプリミティブは、マルチウィンドウ機能やビジネスアプリケーションで要求される機能をビットマップ上で効率良くサポートするための基本的な関数群であり、CAD、イメージ処理等を目的としたディスプレイ関数群とは基本的に目的が異なっている。また、ディスプレイプリミティブの一部として、ポインティングデバイスのポインタ表示に関する関数群も含んでいる。

---

[この章の目次にもどる](#)

[前頁:第2章 ディスプレイプリミティブにもどる](#)

[次頁:2.2 基本概念にすすむ](#)

## 2.2 基本概念

### 2.2.1 座標平面 / 点 / 長方形

ディスプレイリミティブで取り扱う座標系は、16ビット整数値(-32768 ~ 32767) で表わされる整数座標系であり、水平座標値は右方向に増加し、垂直座標値は下方向に増加する。

点 (Point) は、座標系上での位置を表わすための基本的な概念であり、以下の構造体により定義される(基本データタイプに関しては「[第1編 第1章 基本データタイプ](#)」を参照の事)。

```
/* 点 */
typedef struct Point {
    H    x;      /* 水平座標値 */
    H    y;      /* 垂直座標値 */
} PNT;
```

また、長方形 (Rectangle) は、座標系上での領域を表わすための重要な概念であり、以下の構造体により定義される。

```
/* 長方形 */
typedef union Rectangle {
    struct {
        H    left;      /* 左の座標値 */
        H    top;       /* 上の座標値 */
        H    right;     /* 右の座標値 */
        H    bottom;   /* 下の座標値 */
    } c;
    struct {
        PNT lefttop;   /* 左上の点 */
        PNT rightbot;  /* 右下の点 */
    } p;
} RECT;
```

ここで取り扱う長方形は、実際には右と下の辺を含まない領域であり、以下に示す性質を持つことに注意が必要である。(このように右と下の辺を含まない性質を長方形の half-open property と呼ぶ。)

- ある点(x, y) が長方形に含まれるのは、次の条件が成り立つ場合であり、点(lefttop)は長方形に含まれるが、点(rightbot)は含まれない。

$$(left \leq x) \ \&\& \ (x < right) \ \&\& \ (top \leq y) \ \&\& \ (y < bottom)$$

- top bottom または left right の場合は、その長方形の領域は空である。

また、長方形の集合を表現するための長方形のリスト(RectangleList) は以下のように定義される。リストの最後の要素の next は NULL(0) となる。



```

/* 長方形リスト */
typedef struct RectangleList {
    RectangleList *next;    /* 次の要素へのポインタ */
    RECT          comp;     /* 要素としての長方形 */
} RLIST;

```

ディスプレイプリミティブで取り扱う座標系は、次項で説明するビットマップにより実際の表示上のイメージに対応付けられる。

## 2.2.2 ビットマップ

### ビットマップ

ビットマップは、図形や文字等の表示イメージを大きさ、および色を持った点(8ピクセル)の集合として表わし、その点をメモリ上の1ビット、あるいは複数ビットに一対一対応させて、実際の表示イメージの保持/操作を行なうためのデータ構造である。

ビットマップ上では以下に示す図のように、1つのピクセルが整数座標系上の1つの点に対応づけられる。

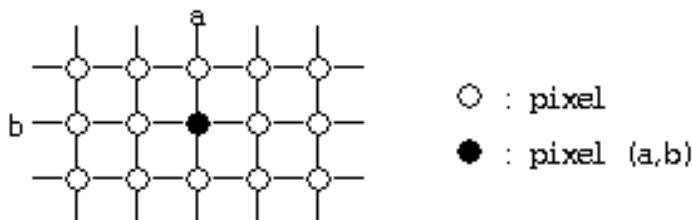


図 17: 座標系とピクセル

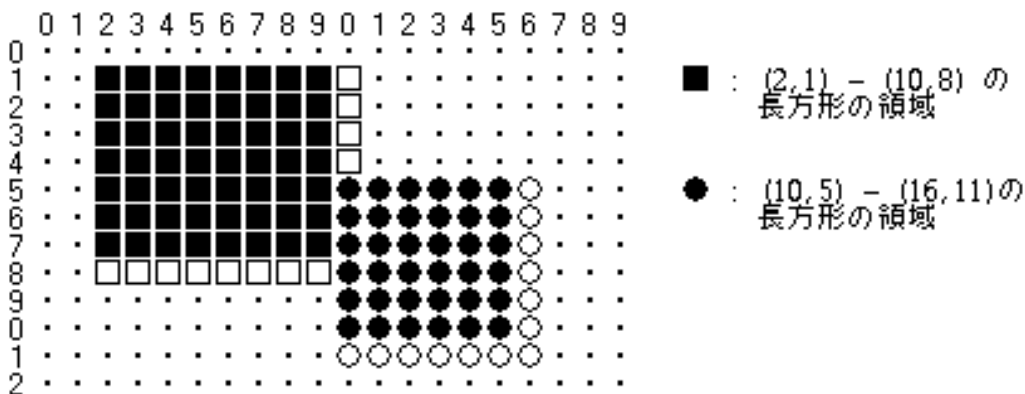


図 18: ビットマップ上の長方形

ビットマップはイメージを表現するためのデータ構造であるが、実際のイメージの色、大きさ等の属性は、ビットマップ構造としては定義されず、ビットマップの対象とするデバイスの属性として定義されることになる。

### ビットマップの構造

ビットマップは、基本的に連続した1次元メモリ配列を、1つの長方形領域内の点(ピクセル)の2次元配列として解釈することにより定義される。このメモリ上のビットと表示上のピクセルとの対応付けは、以下のように定義される。

- ビットマップは、連続したアドレス空間を持つ1つ以上のプレーンにより構成され、各プレーン内では1つ以上のビットが1つのピクセルに対応する。各プレーン内でのピクセル当

たりのビット数をピクセルビット数と呼び、ピクセルビット数×プレーン数、即ち、ピクセル当たりの総ビット数を深さ(depth)と呼ぶ。

- プレーン数とピクセルビット数の組み合わせとしては、以下の4種類が考えられる。なお、1.以外の構成の場合、即ち深さ>1の場合は、ビットマップではなく、ピクセルマップと呼ぶこともあるが、ディスプレイプリミティブでは、ピクセルマップの意味を含めてビットマップと呼ぶことにする。
  1. ピクセルビット数 = 1、プレーン数 = 1 -- 白黒
  2. ピクセルビット数 = 1、プレーン数 = n -- カラー / 白黒多階調
  3. ピクセルビット数 = n、プレーン数 = 1 -- カラー / 白黒多階調
  4. ピクセルビット数 = n、プレーン数 = m -- カラー / 白黒多階調
- ビットマップの最大の深さは28とする。
- 横方向のピクセル数は16の倍数であり、偶数バイト(16ビット)境界に一致する。
- メモリ上の低位アドレスのバイトのMSB(最上位ビット)が左上のピクセルに対応する。アドレス順に右側に対応し、右側の境界に達した場合は1つ下の左端に対応する。最後のバイトのLSB(最下位ビット)が右下のピクセルに対応する。

ビットマップは以下の構造体により定義される。

```
/* ビットマップ */
#define PLANES      (1)      /* ダミー */
typedef struct Bitmap {
    UW    planes;           /* プレーンの数 */
    UH    pixbits;         /* (境界 / 有効)ピクセルビット数 */
    UH    rowbytes;        /* プレーンの横幅のバイト数(偶数) */
    RECT  bounds;          /* 境界長方形(座標定義) */
    UB    *baseaddr[PLANES]; /* 開始アドレス(planes 個の配列) */
} BMP;
```

planes はプレーン数を表わし、1~28の範囲の値でなくてはならない。また、プレーン数 × 有効ピクセルビット数(下記)は、28以下でなくてはならない。

pixbits は、上位バイトがピクセルビットの境界ビット数を表わし、下位バイトが有効ビット数を表わす。境界ビット数は、ビットマップメモリ上での1ピクセル当たりのビット数を示す1~32の範囲の値であり、通常は2のべき乗の値である。有効ビット数は、境界ビット数のうち実際に有効な下位からのビット数を示す1~28の範囲の値である。

なお、単にピクセルビット数と言った場合は、有効ビット数を示すものとし、後に述べるピクセル値は有効ビット数をもとにしている。

rowbytes は、1つのプレーンの横幅のバイト数を示し、必ず偶数バイトとなる。

bounds は、ビットマップの大きさと座標系を示す長方形で、左上のピクセルの座標値と、右下のピクセルの座標値を指定する。これにより、ビットマップの対象とする全体の大きさを指定するとともに、座標系の原点(0,0)を任意の位置に設定することができる。この bounds は通常、rowbytes で示されるビットマップのメモリ領域の大きさに一致する大きさに設定される。

baseaddr は、各プレーン毎のビットマップの開始メモリアドレスを示す、planes 個の要素からなる配列であり、baseaddr の値が、NULL の場合は、対応するプレーンが存在しないことを表わす。

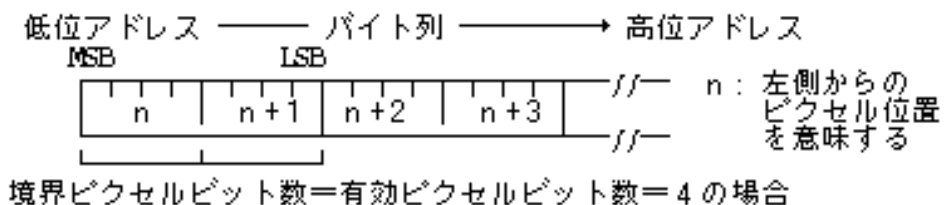
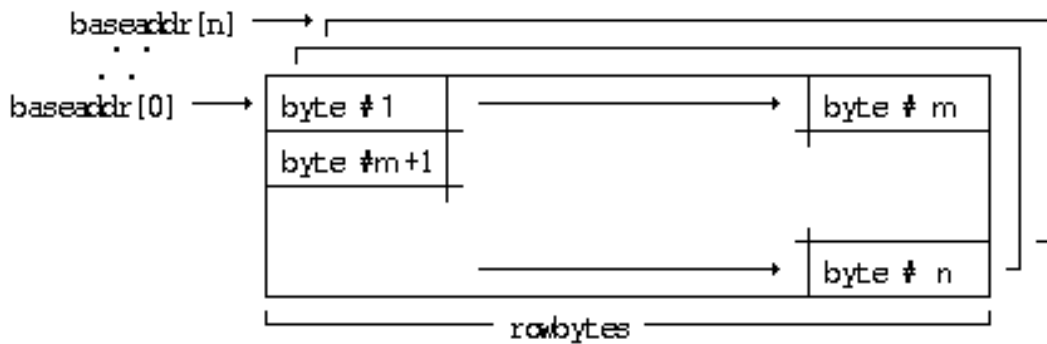


図 19: ビットマップの構造

### ビットマップの座標系

ビットマップ上の左上のピクセルを常に(0, 0)と考えた座標値を「絶対座標」と呼び、boundsにより規定される座標値、即ち、bounds.p.lefttop を左上のピクセルの座標値としたものを、「相対座標」と呼ぶ。通常は「相対座標」が使用される。

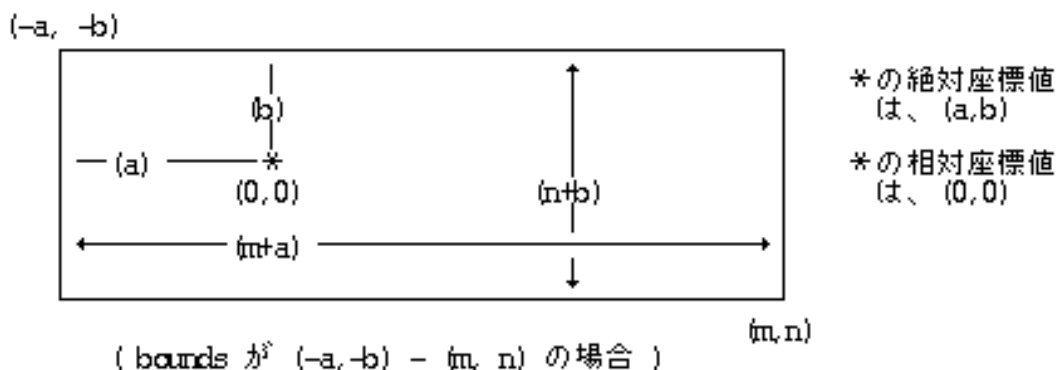


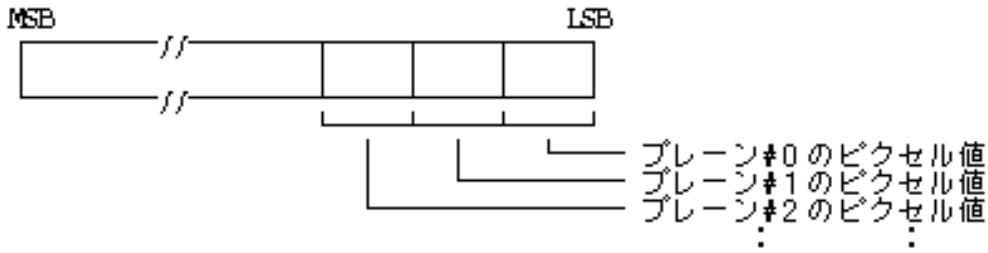
図 20: ビットマップの座標定義

### ピクセル値

ビットマップ上で、1つのピクセルに対応する複数のビットを並べて1つの数値として見た値をピクセル値と呼ぶ。ピクセル値は以下のように定義され、 $0 \sim (2^{**} \text{depth} - 1)$ の範囲の値をとる。

ピクセル値は、そのピクセルの色や白黒の階調を表わすが、実際の色、階調との対応は、対象と

するデバイスの属性として定義される。



各プレーン毎のピクセル値は、左側のピクセルに隣接するビットをMSBとしたピクセルビット数で示されたビット数の数値であり、例えばピクセルビット数 = 4 の場合は、以下ようになる。

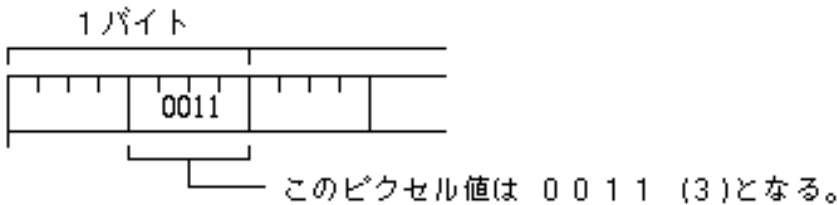


図 21 : ビットマップのピクセル値

なお、ディスプレイプリミティブでは depth は最大28までであり、ピクセル値として32ビットのデータを使用し、下位の depth(最大28)ビットが有効となる。

```
/* ピクセル値 */
typedef W PIXVAL; /* ピクセル値 */
```

### 圧縮ビットマップ

圧縮ビットマップは、ビットマップデータを圧縮し、サイズを小さくしたもので、主にビットマップデータの保存のために使用される。

圧縮ビットマップは、以下のように定義される。

```
/* 圧縮ビットマップ */
typedef struct CompactedBitmap {
    W compac; /* イメージ圧縮方法 */
    BMP bmp; /* ビットマップ */
} CBMP;
```

compac はイメージの圧縮方法を示すもので以下のものとなる。

```
/* イメージ圧縮方法 */
enum ImageCompactionMethod {
    NOCOMPAC = 0, /* 非圧縮 */
    MHCOMPAC = 1, /* MH符号 */
    MR2COMPAC = 2, /* MR符号 (K = 2) */
    MR4COMPAC = 3, /* MR符号 (K = 4) */
    /* = 4 ~ : 予約 */
    /* < 0 : インプリメントに依存した特殊圧縮方法 */
};
```

NOCOMPAC (非圧縮) の場合は、ビットマップの内容は通常のビットマップと全く同一となる。圧縮の場合は、ビットマップの baseaddr[] で示されるデータ部分が圧縮された形式で格納され

る。

- MH 符号 (MHCOMPAC) :

先頭のヘッダ部には以下の内容が入り、ヘッダ部の後にライン毎のMH符号列データが並び、

- 全体のバイト数 (UW: 4バイト)
- 全体のライン数 (UH: 2バイト)
- 各ライン毎のデータへのヘッダ部の先頭からのバイトオフセット (UW: 4バイト)

各ラインのデータは、MH符号列、FILL(任意個の0)、EOL(000000000001)の並びであり、最後のEOLが偶数バイト境界にくるようにFILLによりパッドされる。MH符号列、FILL、EOLはすべて、低位アドレスのMSBから高位アドレスのLSBに向かう順番となる。

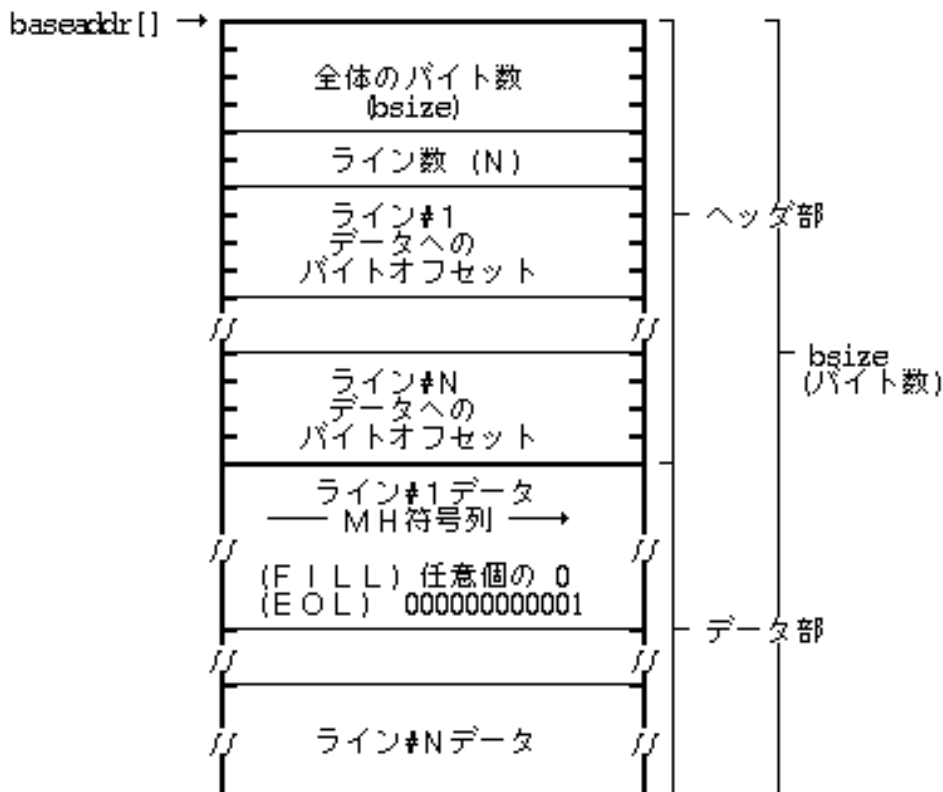


図 22 : ビットマップの座標定義

- MR符号(MR2COMPAC / MR4COMPAC) :

MH符号の場合と同様であり、各ラインのデータがMR符号列となる。

### 2.2.3 カラー表現

一般的にカラーの表示方式は、以下の2つの方法に分類される。

固定カラー方式 :

ピクセル値が、直接実際のカラーを示す方式。

カラーマップ方式 :

ピクセル値は、カラーマップのインデックスであり、対応するカラーマップのエントリの内容が実際のカラーを示す方式。限られたビットマップの深さで多くのカラーを表示する場合に使用される。

カラー表示の方式はデバイスに固有なものであり、デバイスごとに以下のカラー表示方式の情報を持っている。(詳細は後述)

- 固定カラー / カラーマップ方式の別
- カラータイプ:
  - 白黒
  - RGBカラー
  - CMYカラー(プリンタ)
  - その他
- 各色要素の階調数 / ビット位置

ディスプレイプリミティブでは、カラー指定の方法として、ピクセル値による直接指定と、RGBの各要素の輝度による絶対指定の2種類の方法が取られる。

各種描画のカラー指定:

ピクセル値による直接指定、および RGB による絶対指定

カラーマップの設定:

RGB による絶対指定

カラー指定は以下に定義される構造体により行なわれる。

```

/* カラー表現 */
struct ColorDescription {
    UW  trans:1;      /* 透明 */
    UW  crep:3;      /* カラー表現 */
    UW  cval:28;     /* カラー値 */
};

typedef UW  COLOR;  /* カラー指定 */
  
```

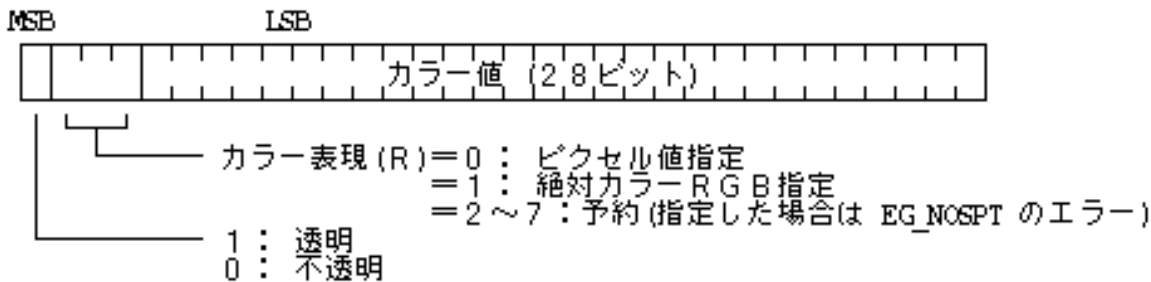


図 23: カラー指定

- 透明 / 不透明は、描画カラー指定にのみ使用され、透明を指定した場合は実際には描画を行なわないことを意味し、下位31ビットの値は無視される。
- カラー値は、カラー表現により異なる意味を持つ。

crep = 0

直接ピクセル値を示す。

crep = 1

赤 / 緑 / 青の各色の輝度を0 ~ 255の正規化されたレンジで示す。すなわち、0が0%、255が100%を示す。

XXXX RRRRRRRR GGGGGGGG BBBBBBBB  
 赤(8) 緑(8) 青(8)

crep = 2 ~ 7 予約

実際の描画はピクセル値により行なわれるため、描画カラーとして指定したCOLOR から実際のピクセル値への変換を行なう必要があり、この変換は以下のように行なわれる。

crep = 0 : ピクセル値指定

COLOR の下位Nビットをそのままピクセル値とし、上位ビットは無視する。ここでNは、対象ビットマップの深さを意味する。

crep > 0 : 絶対カラー指定

COLOR で指定したカラーに最も近いカラーのピクセル値とする。カラーマップ方式の場合はカラーマップの検索を行ない、指定した COLOR に最も近い色のエントリのインデックスをピクセル値とする。

固定カラー方式の場合は、COLOR での標準RGB表記に最も近い色のデバイス固有のRGB表記に変換した値をピクセル値とする。

カラーマップ方式の場合、ピクセル値 0 は白、最大値 ( $2^{**} \text{depth} - 1$ ) は黒を定義することが望ましい。

## 2.2.4 描画環境

### 描画環境

ディスプレイプリミティブの実行環境を描画環境と呼ぶ。描画環境は動的に生成され、生成時に得られる描画環境IDにより識別される。実際の描画では、この描画環境IDを指定して対象とする描画環境を指定することになる。

```
/* 描画環境ID */  
typedef W    GID;          /* 描画環境ID */
```

1つの描画環境に対して以下のデータが対応づけられる。

- 対象デバイス
- 対象ビットマップの本体
- カラー情報
- ポインタ

さらに、個々の描画環境ごとに以下のデータが対応づけられる。

- 対象ビットマップの座標系 (境界長方形)
- マスクピクセル値
- クリッピング領域
- 文字属性 (描画位置 / 方向 / 指定 / 描画カラー / フォント)
- 文字セット
- 色変換用カラー情報

異なる描画環境に対するディスプレイプリミティブの関数が同時に実行されても問題が起きないようにディスプレイプリミティブ内部で排他制御が自動的に行なわれる。ただし、対象ビットマップが重なっている場合には、重なっている部分の描画結果は保証されない。

逆に、同一の描画環境に対するディスプレイプリミティブの関数が同時に実行された場合の結果は保証されない。

描画環境の生成時には、描画対象とするデバイスを示す文字列で指定する。あらかじめディスプレイプリミティブに登録されているデバイスのみが指定可能である。

ディスプレイプリミティブでかならずサポートしているデバイスとして以下のものがあげらる。

- "SCREEN"
- "MSGPNL"

描画対象デバイスには、以下のデバイス固有の情報(仕様)が定義されており、これはビットマップ上に描画したイメージの具体的な表現の定義を意味する。

```
typedef struct {
    H  attr;          /* デバイスの属性 */
    H  planes;       /* プレーン数 */
    H  pixbits;      /* ピクセルビット数(境界 / 有効) */
    H  hpixels;      /* 横のピクセル数 */
    H  vpixels;      /* 縦のピクセル数 */
    H  hres;         /* 横の解像度 */
    H  vres;         /* 縦の解像度 */
    H  color[4];     /* カラー情報 */
    H  resv[6];      /* 予約 */
} DEV_SPEC;
```

- attr は、デバイスの属性を示す以下の内容である。

OLMX XXXX XXXX PRRR

L:

- 0 = このデバイスに対して生成したデバイス描画環境レコードはロックの対象とされない。
- 1 = このデバイスに対して生成したデバイス描画環境レコードはロックの対象とされる。

M:

- 0 = 固有イメージ領域(ビットマップ)は持っていない(デバイス描画環境は生成不可)。
- 1 = 固有イメージ領域(ビットマップ)を持っている(デバイス描画環境を生成可能)。

P:

- 0 = カラーマップ無(直接指定)
- 1 = カラーマップ有(マップエントリ数はプレーン数 × ピクセルビット数で決まる)

R:

- 0 = 白黒
- 1 = RGBカラー
- 2 = CMY カラー
- 3 ~ その他

X:

予約

- planes は、デバイスに適用されるビットマップのプレーン数を示し、pixbits は、境界 / 有効ピクセルビット数を示すもので、BITMAP の定義と同じ意味を持つ。なお、planes < 0 の場合は特殊ビットマップであることを意味し、pixbits の意味はデバイスに依存する。
- hpixels と vpixels は、デバイスに適用されるビットマップの横と縦のピクセル数を示す。大きさが規定されていない場合は0となる。
- hres と vres は、デバイスに実際に出力した場合のビットマップの横と縦の解像度を示し、1センチメートルまたは1インチ当たりのピクセル数で示される。インチの場合は、負数で表わされる。解像度が規定されていない場合は0となる。



- color[] は、カラー情報であり、attr で指定されるカラー方式によりその内容が異なる。

R = (白黒)の場合

color[0] 白黒の階調ビット数を示す。  
color[1] ~ 未使用(0)

R = 1(RGB)の場合

color[0] 赤の階調ビット数とその位置を示す。  
color[1] 緑の階調ビット数とその位置を示す。  
color[2] 青の階調ビット数とその位置を示す。  
color[3] 未使用(0)

R = 2(CMY)の場合

color[0] シアンの階調ビット数とその位置を示す。  
color[1] マゼンタの階調ビット数とその位置を示す。  
color[2] 黄の階調ビット数とその位置を示す。  
color[3] 黒の階調ビット数とその位置を示す。

下位バイトが階調ビット数を表わし、上位バイトがビット位置を表わす。マップ方式の場合は、ビット位置は意味を持たず0となり、マップ方式でない場合は、ピクセル値上でのLSBを0としたビット位置を示す。

例としてピクセル値が以下のように定義されている場合は、

```
....BBBBGGGGGRRRRR
```

```
color[0] = 0x0006  
color[1] = 0x0606  
color[2] = 0x0c04
```

となる。

カラー情報を持たない汎用メモリ描画環境の場合には、ピクセル値とカラーとの対応は定義されないことになり、COLOR で指定されるカラーについては、透明 / 不透明は意味あるものとして処理されるが、カラー表現についてはピクセル値 / 絶対値指定にかかわらず、単にピクセルの深さでマスクを取った値をピクセル値として処理されることになる。

## 対象ビットマップ

描画環境には、生成時に指定したメモリビットマップが対応付けられ、描画環境に対する描画は、そのビットマップ上に行なわれる。

VRAM 等のビットマップでは特殊な構成を持つ場合があり、その場合にはプレーン数として負の特殊な値とする。この特殊な値はインプリメントに依存し、ディスプレイプリミティブの関数はあらかじめその値の意味を知っていることを前提とする。

プレーン数が正の値の場合は、そのビットマップのデータの意味は、標準のビットマップの定義そのものであるが、一般に主メモリ上のビットマップ以外は、ビットマップ領域のメモリ内容を直接アクセス可能であることは保証されない。

## カラー情報

汎用メモリ描画環境には、対象とするビットマップのピクセル値の実際の意味を示すためのカラー情報を描画環境の生成時に設定することができる。

カラー情報が設定されている場合は、絶対RGBによるカラー指定はカラー情報に従ってピクセル値に変換されて実際の描画が行われる。カラー情報が設定されていない場合は、絶対RGBカラー

による指定は無視され、常にピクセル値とみなされて描画される。

ビットマップ操作関数や描画パターンで使用されるビットマップの色変換を行うため、描画環境に色変換用のカラー情報を設定することができる。

```
/* カラー指定 */
struct ColorSpec {
    W      attr;          /* カラー属性 */
    H      info[4];      /* カラー情報 */
    COLOR  *colmap;      /* カラーマップへのポインタ */
};
```

```
typedef ColorSpec CSPEC; /* カラー指定 */
```

- attr は DEV\_SPEC の attr の下位4ビットと同じ意味を持つ。上位28ビットは予約済み。
- info は attr & 0x8 が0以外(カラーマップ形式)の場合は以下の意味を持つ。

info[0]

カラーマップのエントリ数

info[1] ~ [3]

未使用

info は attr & 0x8 が0の場合は DEV\_SPEC の color と同じ意味を持つ。

- colmap は、attr & 0x8 が0以外場合に、実際のカラーマップへのポインタを示す。0の場合は、この値は意味を持たない。

実際のカラーマップは、(最大ピクセル値 + 1)個の要素からなる、COLOR 値の配列であり、その配列のインデックス(0 ~ 最大ピクセル値)がピクセル値となる。配列の要素は、COLOR の絶対RGB表現の値となる。

## マスクピクセル値

マスクピクセル値は、描画対象とするピクセルを限定するために使用されるピクセル値であり、後述する描画モードの指定により、以下のいずれかの描画動作となる。

1. マスクピクセル値に無関係に全ピクセルを描画対象とする。
2. マスクピクセル値と同一の値のピクセルのみ描画対象とする。
3. マスクピクセル値と異なる値のピクセルのみ描画対象とする。

## クリッピング領域

描画環境において、実際に対象ビットマップ上に描画が行なわれる領域は、対象ビットマップの境界長方形 (bounds)、フレーム長方形(FrameRect)、表示長方形(VisibleRect)、設定されたクリッピング領域に含まれ、かつ、前置長方形リストに含まれない領域となる。

クリッピング領域が未定義の場合は、対象ビットマップの境界長方形 (bounds)に含まれる領域がすべて描画対象となる。

クリッピング領域は、以下の任意領域構造体により定義され、その座標系は相対座標で指定される。

```
/* 任意領域 */
#define NX      1 /* ダミー */
#define NS      1 /* ダミー */
```

```

struct HorizontalRegion {
    UH  nx;          /* 水平座標の数(偶数) */
    UH  x[NX];      /* 水平座標の値(nx 個の要素で値の小さい順) */
};
typedef HorizontalRegion HRGN; /* 水平領域 */

struct GenericRegion {
    RECT  r;        /* 任意領域全体を囲む最小の長方形(相対座標系) */
    UW    ns;       /* 垂直区間数 */
    struct {
        UH    y;    /* 区間の開始垂直座標値 */
        HRGN  *hp;  /* 区間の水平領域へのポインタ */
    } s[NS];
};

typedef GenericRegion  GRGN; /* 任意領域 */

```

水平領域は1つの垂直座標値(ラスタ)に対応する水平座標の領域を指定するものであり、nx は、水平座標値の個数を示し、x[0] ~ x[nx-1] は、水平座標値を小さい順に並べたものである。水平座標値は、任意領域全体を囲む最小の長方形の r.c.left の値を0とする非負の値で示される。

水平座標値は2つ1組であり、第一の水平座標値(小さい値)から、第二の水平座標値(大きい値) - 1 の水平座標値までが、対象領域となる。(第一の値は含まれるが、第二の値は含まれない : half-open)。

同一の水平座標値が2つ以上含まれていた場合、偶数回のときにはその座標値は存在しないものと見なされ、奇数回のときには1つのみ存在しているものと見なす。nx が偶数でない場合は、最後の座標値は無視される。

同一の水平領域を持つ連続した垂直座標の領域を(垂直)区間と呼び、任意領域は、いくつかの(垂直)区間により定義される。ns は区間の数を示し、ns = 0の場合は、任意領域は長方形 r に等しいことを意味する。

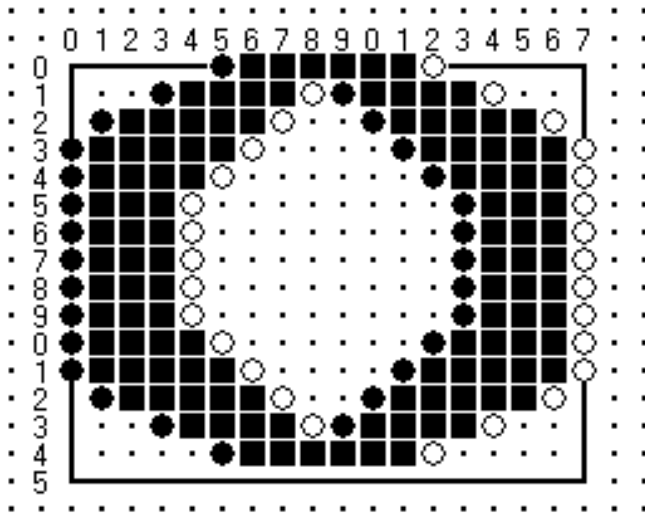
s[N].y は区間 N の開始垂直座標値を示し、その区間の終了垂直座標位置は、s[N+1].y - 1、または r.c.bottom - 1(最後の区間の時)となる。

s[N].hp は区間 N に対応する水平領域へのポインタを示す。s[N].hp = NULLの時は、その区間は空であることを意味する。

```

r = (0, 0, 17, 15)
ns = 11
s[0].y = 0    s[0].hp = &(amp;2, 5, 12)
s[1].y = 1    s[1].hp = &(amp;4, 3, 8, 9, 14)
s[2].y = 2    s[2].hp = &(amp;4, 1, 7, 10, 16)
s[3].y = 3    s[3].hp = &(amp;4, 0, 6, 11, 17)
s[4].y = 4    s[4].hp = &(amp;4, 0, 5, 12, 17)
s[5].y = 5    s[5].hp = &(amp;4, 0, 4, 13, 17)
s[6].y = 10   s[6].hp = &(amp;4, 0, 5, 12, 17)
s[7].y = 11   s[7].hp = &(amp;4, 0, 6, 11, 17)
s[8].y = 12   s[8].hp = &(amp;4, 1, 7, 10, 16)
s[9].y = 13   s[9].hp = &(amp;4, 3, 8, 9, 14)
s[10].y = 14  s[10].hp = &(amp;2, 5, 12)

```



$\bullet, \circ$  :  $r$   
 水平領域の指定座標値

任意領域は、 $\bullet, \blacksquare$  で示された領域となる。

図 24 : 任意領域の例 - 1

```

r = (x0, y0, x3, y3)
ns = 3
s[0].y = y0 s[0].hp = &(2, x0, x2)
s[1].y = y1 s[1].hp = NULL
s[2].y = y2 s[2].hp = &(2, x1, x3)
  
```

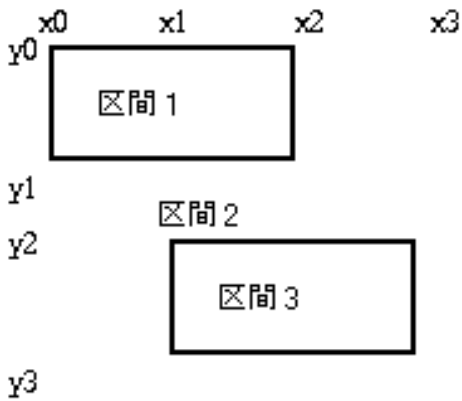


図 25 : 任意領域の例 - 2

```

r = (x0, y0, x4, y4)
ns = 4
s[0].y = y0 s[0].hp = &(2, x0, x3)
s[1].y = y1 s[1].hp = &(2, x0, x4)
s[2].y = y2 s[2].hp = &(2, x1, x4)
s[3].y = y3 s[3].hp = &(2, x1, x2)
  
```

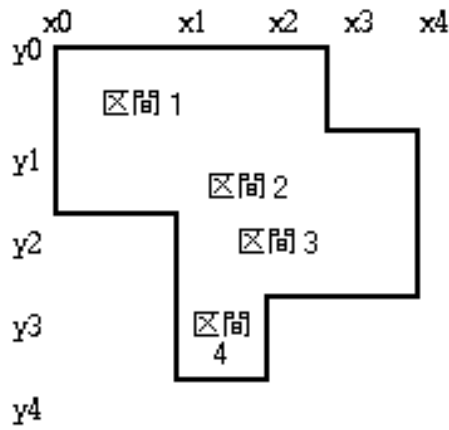


図 26 : 任意領域の例 - 3

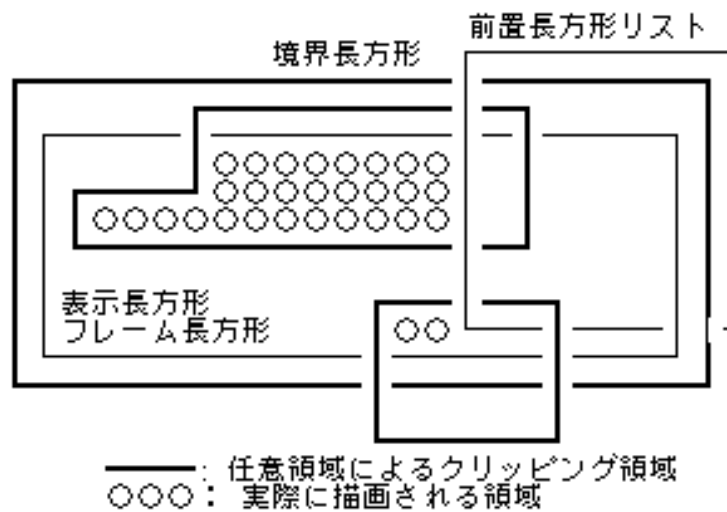


図 27 : クリッピング領域

## 描画モード

実際の描画は、図形描画 / 文字描画時に指定した描画モードに従って行なわれる。

描画モードは、以下の形式の32ビットデータで指定される。

```
0000 0000 0000 0000 0000  PPCF MMMM MMMM
```

- M: 描画演算モード指定
- F: ビットマップ形式変換指定
- C: カラー変換指定
- P: マスクピクセル指定

描画演算モードは、描画の対象とするピクセルに、書き込むべきピクセル値(src)と既に現在のピクセル値(dest)と、描画後のピクセル値(result)との関係を規定するものであり、以下に示すものが用意されている。

```
/* 描画演算モード */
enum DrawingCopyMode {
    /* ピクセル値のビット単位の論理演算 */
    G_STORE = 0, /* (src) --> (result) */
    G_XOR = 1, /* (src) XOR (dest) --> (result) */
};
```

```

G_OR    = 2,    /* (src) OR (dest) --> (result) */
G_AND   = 3,    /* (src) AND (dest) --> (result) */
G_CPYN  = 4,    /* NOT (src) --> (result) */
G_XORN  = 5,    /* NOT (src) XOR (dest) --> (result) */
G_ORN   = 6,    /* NOT (src) OR (dest) --> (result) */
G_ANDN  = 7,    /* NOT (src) AND (dest) --> (result) */
G_NOP   = 8     /* (dest) --> (result) */

```

```
};
```

ビットマップ形式変換指定は、ソースとディスティネーションのビットマップの形式(プレーン数、ピクセルビット数)が異なる場合は以下に示す変換を行なうことを指定する。

- 変換後の双方の下位Nビットのピクセル値が等しくなるように変換する。ここで、Nは、ソースとディスティネーションの depth の最小値を意味する。
- ビットマップの操作を行う関数に適用されるほか、描画パターン(後述)として指定するビットマップ(MPAT)の形式変換としても適用される。

```

/* ビットマップ形式変換 */
enum BitMapFormConversion {
    G_CVFORM    = 0x0100    /* ビットマップ形式変換 */
};

```

カラー変換指定は、ソースとディスティネーションのカラー表現(カラー情報、カラーマップ)が異なる場合は以下に示す変換を行なうことを指定する。

- どちらか一方のカラー情報が設定されていない場合は変換されない。
- ソースのピクセル値を絶対カラーに変換し、その絶対カラーに最も近い絶対カラーに対応するディスティネーションのピクセル値に変換する。
- ビットマップの操作を行う関数に適用されるほか、描画パターン(後述)として指定するビットマップ(MPAT)のカラー変換としても適用される。

```

/* カラー変換 */
enum ColorConversion {
    G_CVCOLOR   = 0x0200    /* カラー変換 */
};

```

マスクピクセル指定は、描画環境に定義されたマスクピクセル値の効果を指定する。

```

/* マスクピクセル指定 */
enum MackPixelMode {
    G_MASK      = 0x0400 /* マスクピクセル値と同一の値の dest ピクセルのみ描画 */
    G_MASKN    = 0x0800 /* マスクピクセル値と異なる値の dest ピクセルのみ描画 */
};

```

```

/* 描画モード */
typedef W      DCM;    /* 描画モード */

```

## 描画パターン

描画パターンは、描画の対象となるピクセルに書き込むピクセル値を指定するために使用されるデータであり、任意の大きさの長方形のデータとして定義される。

描画対象のビットマップの境界長方形(bounds)の原点(0,0)に描画パターンデータの左上の点を揃えて、規則正しくタイルを詰めるように境界長方形の全域に置いた場合、描画の対象となるピク

セルの位置のピクセル値が書き込むべきピクセル値となる。

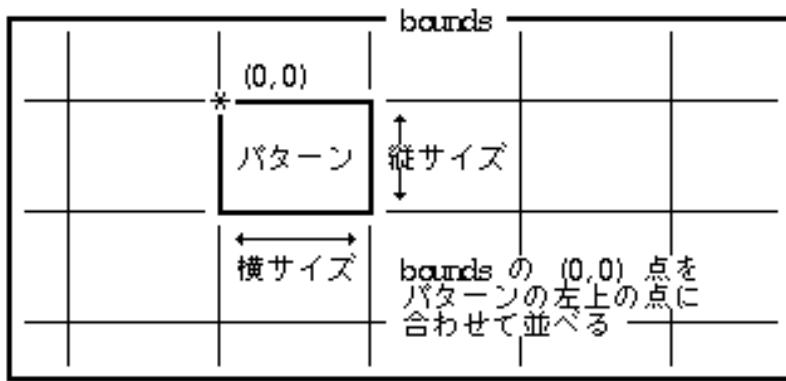


図 28 : 描画パターンの配置

描画パターンは、以下の3種類の方法により指定することができる。

1. マスクデータ、前景色、背景色により描画パターンを指定する。前景色、および背景色はそれぞれ透明とすることが可能であり、透明を含めた2色までの混合パターンが指定できる。
2. ビットマップデータにより描画パターンを指定する。指定したビットマップ上の左上隅の正方形をパターンとして指定するため、任意の色の混合パターンが可能となる。また、別にマスクデータが指定可能であり、これによりパターンの一部を透明とすることができる。
3. 上記1または2の方法で定義したパターンをより高速に描画するために内部形式に変換したものを指定する。この場合は、あらかじめ対象とする描画環境に対して、使用する描画パターン(上記1,2)を、指定したメモリ領域(通常はプロセスのローカルメモリ領域)上に内部パターンイメージとして生成しておく必要がある。

内部パターンイメージの形式とサイズは、描画環境、パターンタイプ、パターンのサイズ、およびインプリメントに依存する。内部パターンイメージへの変換関数、および内部パターンイメージの大きさを得るための関数が用意されている。

描画パターンは以下に示すデータにより描画時に指定することになる。

```
/* 描画パターン */
typedef W      PatKind;    /* パターンタイプ */

union pattern {
    struct {
        PatKind kind;      /* パターンタイプ(=0) */
        UH      hsize;     /* パターンの横サイズ(ピクセル数) */
        UH      vsize;     /* パターンの縦サイズ(ピクセル数) */
        COLOR   fgcol;     /* 前景色 */
        COLOR   bgcol;     /* 背景色 */
        UB      *mask;     /* パターンマスクデータへのポインタ */
    } spat;
    struct {
        PatKind kind;      /* パターンタイプ(=1) */
        UH      hsize;     /* パターンの横サイズ(ピクセル数) */
        UH      vsize;     /* パターンの縦サイズ(ピクセル数) */
        UB      *mask;     /* パターンマスクデータへのポインタ */
        BMP     *bmap;     /* ビットマップへのポインタ */
    };
};
```

```

} mpat;
struct {
    PatKind kind;          /* パターンタイプ(=2) */
    UH      hsize;        /* パターンの横サイズ(ピクセル数) */
    UH      vsize;        /* パターンの縦サイズ(ピクセル数) */
    UB      *pat;         /* 内部パターンイメージへのポインタ */
} ipat;
}

```

```

typedef Pattern PAT /* 描画パターン */

```

kind :

パターンのタイプを示す以下の値である。

- = 0 : spat タイプ
- = 1 : mpat タイプ
- = 2 : ipat タイプ
- = 3 ~ : 予約 (指定した場合は、EG\_NOSPT のエラーとなる)

hsize / vsize :

hsize は、パターンの横サイズ、vsize はパターンの縦サイズを、ピクセル数で表わしたものである。

fgcol / bgcol :

fgcol はパターンマスクデータの"1"の部分の色を指定し、bgcol は"0"の部分の色を指定する。fgcol および bgcol < 0 の場合は、それぞれ透明を示す。

mask :

以下の形式でパターンマスクイメージを示す。即ち、パターンサイズで示される長方形が入る最小のビットマップデータ(ピクセルビット数 = 1)である。

- mpat 形式の場合、"0"の部分は描画を行わず、透明とすることを意味する。
- また、mask = NULL の場合は、透明部分は存在しないことを意味する。
- spat 形式の場合は、NULL であってははいけない。



図 29 : パターンマスクの構成

なお、以下のパターンマスクデータが組込みで用意されているため mask として使用することができる。(これらは、パターンのサイズに無関係に使用できる。)

```

/* 標準パターンマスクポインタ */
enum StdPatternMask {
    _FILL0 = (1), /* 0 % 塗り潰し */
    _FILL12 = (2), /* 12.5% 塗り潰し */
}

```



```

_FILL25 = (3), /* 25 % 塗り潰し */
_FILL50 = (4), /* 50 % 塗り潰し */
_FILL75 = (5), /* 75 % 塗り潰し */
_FILL87 = (6), /* 87.5% 塗り潰し */
_FILL100 = (7) /* 100 % 塗り潰し */
};

```

```

#define FILL0      ((B*)_FILL0)
#define FILL12     ((B*)_FILL12)
#define FILL25     ((B*)_FILL25)
#define FILL50     ((B*)_FILL50)
#define FILL75     ((B*)_FILL75)
#define FILL87     ((B*)_FILL87)
#define FILL100    ((B*)_FILL100)

```

bmap :

指定したビットマップの左上隅のパターンサイズで定義された長方形部分をパターンとみなす。bounds はパターンサイズより小さくてはいけない。また、rowbytes はパターンサイズ以上でもよい。

ビットマップの形式(プレーン数、ピクセルビット数)が対象とする描画環境のビットマップ形式と同一でない場合、描画時に形式変換指定、(色変換指定)を行う必要がある。

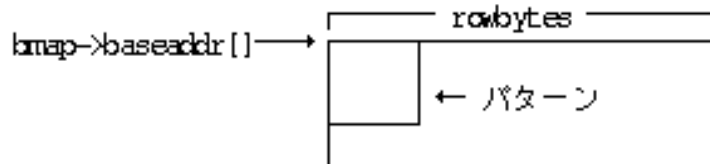


図 30 : ビットマップでのパターン指定

pat :

変換した内部パターンイメージへのポインタである。

標準パターンとして以下のパターンが用意されている。

```

/* 標準パターンポインタ */
enum StdPattern { /* すべて、前景色 : 黒、背景色 : 白 */
_WHITE0    = (1), /* 0 % 塗り潰し */
_BLACK12   = (2), /* 12.5% 黒塗り潰し */
_BLACK25   = (3), /* 25 % 黒塗り潰し */
_BLACK50   = (4), /* 50 % 黒塗り潰し */
_BLACK75   = (5), /* 75 % 黒塗り潰し */
_BLACK87   = (6), /* 87.5% 黒塗り潰し */
_BLACK100  = (7) /* 100 % 黒塗り潰し */
};

```

```

#define WHITE0      ((PAT*)_WHITE0)
#define BLACK12     ((PAT*)_BLACK12)
#define BLACK25     ((PAT*)_BLACK25)
#define BLACK50     ((PAT*)_BLACK50)
#define BLACK75     ((PAT*)_BLACK75)
#define BLACK87     ((PAT*)_BLACK87)
#define BLACK100    ((PAT*)_BLACK100)

```

## 線属性

直線、弧や閉じた図形の枠の描画の場合は、描画する線の属性を以下の形で指定することができる。

```
0000 0000 0000 0000 TTTT TTTT WWWW WWWW
```

W: 線幅 (下位8ビット符号無し) (0~255)  
線の幅(太さ)をピクセル数で指定したもの

T: 線種 (上位8ビット符号無し) (0~5,255)  
以下に示す線種を表わす番号

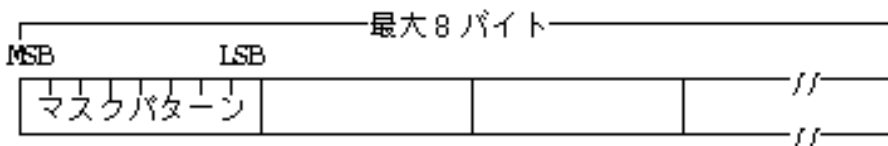
```
/* 線種 */  
enum LineKind {  
    LN_SOLID      = 0,      /* 実線 */  
    LN_DASH       = 1,      /* 破線 */  
    LN_DOT        = 2,      /* 点線 */  
    LN_DDASH      = 3,      /* 一点鎖線 */  
    LN_DDDASH     = 4,      /* 二点鎖線 */  
    LN_LDASH      = 5,      /* 長破線 */  
    /* 予約      6~254 (指定した場合は、EG_NOSPT のエラー) */  
    LN_MASK       = 255     /* 指定マスク */  
};
```

```
/* 線属性 */  
typedef W          LATTR; /* 線属性 */
```

描画環境には、線マスクパターンを登録しておくことが可能であり、T=255の場合は、登録しておいた線マスクパターンを使用した線種が適用される。この線マスクパターンはデフォルトでは実線となっている。

線マスクパターンは、線幅1の時に描画対象となるピクセルを"1"のビットで示した周期的なマスクパターンとして定義され、周期の単位として8~64ピクセルまで、8ピクセル単位で指定することができる。即ち、1~8バイトまでバイト配列により定義される。

なお、太い線幅で描画した場合は、定義したパターンは自然な形で拡大されて描画されることになる。



※ 例えば、周期1バイトで、その内容が11100010の場合は、以下に示すパターンの線種となる。

```
1 1 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 1 0 0 0 . . .
```

図 31 : 線マスクパターン

## 2.2.5 文字描画

### 文字描画カラー

文字描画は指定した文字に対応する文字イメージデータを描画することにより行なわれる。文字イメージデータは、文字(前景)部分と背景部分からなる単一プレーンのビットマップであり、文字部分と背景部分をそれぞれ何色で描画するかの以下の2種類の文字描画カラーの指定が必要になる。

- 文字前景色 (chfgc) : 文字イメージデータの文字(前景)部分("1"の部分)に対しての色指定。
- 文字背景色 (chbgc) : 文字イメージデータの背景部分("0"の部分)に対しての色指定。

実際の文字描画は文字描画カラーに従って以下に示す方法で行なわれる。

文字部分 :

chfgc 0 : 文字前景色をピクセル値として各プレーンに対して、文字描画演算指定に従って描画する。

chfgc < 0 : 描画を行なわない(透明)

背景部分 :

chbgc 0 : 文字背景色をピクセル値として各プレーンに対して、文字描画演算指定に従って描画する。

chbgc < 0 : 描画を行なわない(透明)

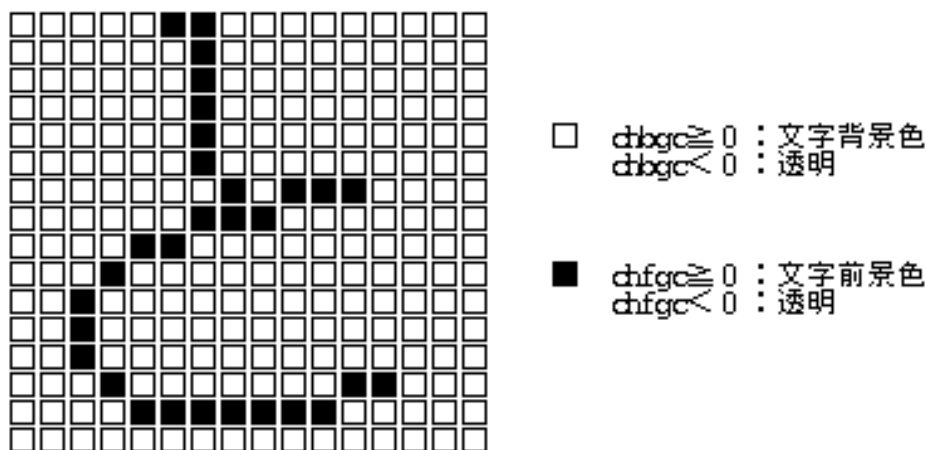


図 32 : 文字描画カラー

描画環境を生成した時点のデフォルトの文字描画カラーは、文字背景色は白(カラー情報を持たない場合は0)、文字前景色は黒(カラー情報を持たない場合は最大ピクセル値)となる。

### 文字描画位置 / 方向

描画環境には、文字描画の際に使用される、文字描画位置、文字描画方向、文字間隔のデータが保持されている。

文字描画位置は、次に描画する文字の座標位置であり、文字(列)を描画した後は、自動的に次の描画位置に更新される。

文字描画方向は、文字の描画方向を指定するものであり、以下の値で表される。文字描画方向により、文字イメージ矩形(文字枠)のどの位置を文字描画位置とするかが一意的に決定される。

```

/* 文字描画方向 */
enum CharacterDirection {
    TORIGHT = 0; /* 右向き(文字枠のベースラインの左端が描画位置) */
    TOLEFT  = 1; /* 左向き(文字枠のベースラインの右端が描画位置) */
    TOUP    = 2; /* 上向き(文字枠の左上の点が描画位置) */
    TODOWN  = 3; /* 下向き(文字枠の左下の点が描画位置) */
    TOARB   = 4; /* 任意の向き(文字枠の左上の点が描画位置) */
};

```

文字間隔は、文字と文字の間隔を指定するものであり、以下の構造体データにより指定される。

```

/* 文字間隔 */
struct SignedGap {
    H   dir;      /* 方向 */
    UH  gap;      /* 文字間隔 */
}
typedef SignedGap  SGAP; /* 文字間隔 */

struct CharacterGap {
    SGAP  hgap;    /* 水平文字間隔 */
    SGAP  vgap;    /* 垂直文字間隔 */
    SGAP  hspgap; /* 空白水平間隔 */
    SGAP  vspgap; /* 空白垂直間隔 */
};
typedef CharacterGap  CGAP; /* 文字間隔 */

```

水平文字間隔 (hgap) :

水平方向の文字間隔  
文字描画方法が、TOUP / TODOWN の場合は無視される。

垂直文字間隔 (vgap) :

垂直方向の文字間隔  
文字描画方法が、TORIGHT / TOLEFT の場合は無視される。

空白水平間隔 (hspgap) :

空白文字の直後に追加される水平方向の文字間隔文字描画方法が、TOUP / TODOWN / TOARB の場合は無視される。

空白垂直間隔 (vspgap) :

空白文字の場合に追加される垂直方向の文字間隔文字描画方法が、TORIGHT / TOLEFT / TOARB の場合は無視される。

水平 / 垂直文字間隔は、文字描画方向が TOARB 以外の場合は、文字枠の端と次の文字枠の端との間隔を示すが、TOARB の場合は、文字描画位置と次の文字描画位置の間隔を示す。

空白水平 / 垂直間隔は、空白文字(可変幅空白、コード = 0x20) の直後に追加される文字間隔を示し、空白文字(可変幅空白)の幅、または高さを変更することに相当する。

文字間隔指定、SGAP の要素は以下の意味を持つ。

dir

- = 0 : 文字間隔を文字描画方向にとる。
- 0 : 文字間隔を文字描画方向の反対方向にとる。(重なって描画される)

gap

文字間のピクセル値を絶対値、または文字幅の比例値で指定する。

絶対ピクセル数指定： 1NNN NNNN NNNN NNNN  
文字間隔はNピクセル数

文字サイズ比例指定： OAAA AAAA BBBB BBBB

文字間隔は、文字幅(水平方向の時)または文字高さ(垂直方向の時)× A / Bのピクセル数。  
(B = 0の時は比率1とみなす)文字幅、文字高さが固定長さでない場合は、最大幅、最大高さを基準とする。

文字間隔の指定により、文字枠と次の文字枠の間に隙間ができる場合、その隙間には何も描画されないため、注意が必要である。

文字描画位置は文字を描画した後、以下に示すように更新される。

水平位置：

TORIGHT：文字幅+水平文字間隔+[空白水平間隔]だけ増加

TOLEFT：文字幅+水平文字間隔+[空白水平間隔]だけ減少

TODOWN：変化しない

TOUP：変化しない

TOARB：水平文字間隔だけ増加

垂直位置：

TOLEFT：変化しない

TORIGHT：変化しない

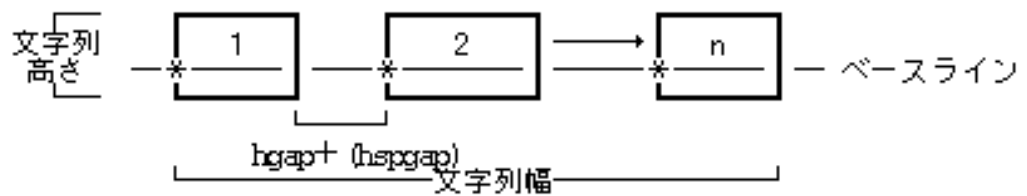
TODOWN：文字高さ+垂直文字間隔+[空白垂直間隔]だけ増加

TOUP：文字高さ+垂直文字間隔+[空白垂直間隔]だけ減少

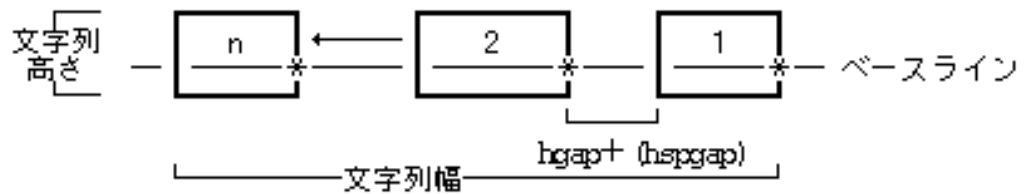
TOARB：垂直文字間隔だけ増加

- [空白水平 / 垂直間隔]は可変幅空白文字を描画した場合のみ適用される。

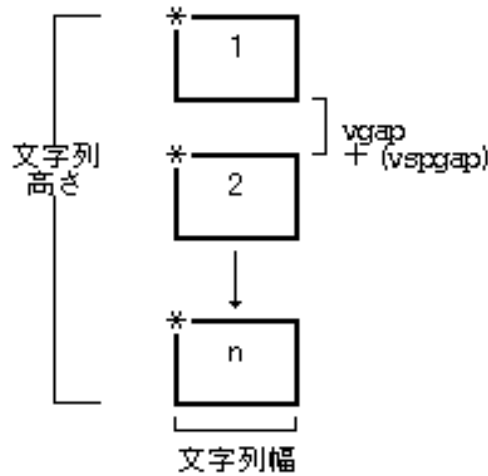
TORIGHT: 右向き (\*が文字描画位置)



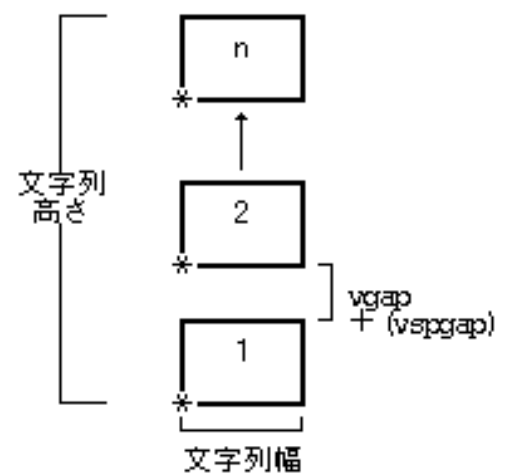
TOLEFT: 左向き



TODOWN: 下向き



TOUP: 上向き



TOARB: 任意方向

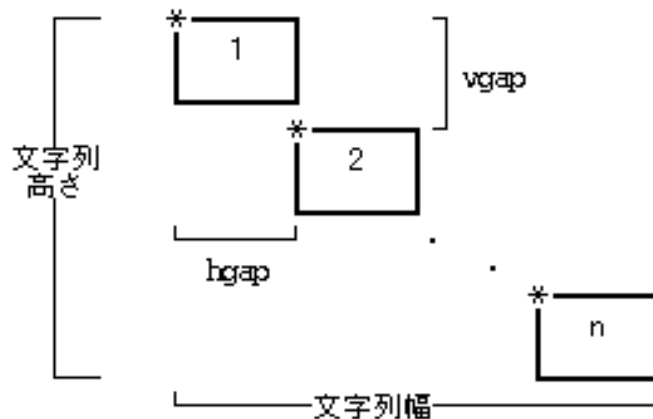


図 33: 文字描画方向 / 文字間隔

描画環境を生成した時点では、以下のデフォルト値が設定される。

文字描画位置:

(0,0)

文字描画方向:

TORIGHT (0)

文字間隔:

すべて0

文字フォント

文字描画で使用する文字フォントは、以下のフォント指定データにより指定される。このフォント指定により、フォントの種類、字の太さ等の属性、および文字サイズ/変形の指定を同時に行なう。指定できる文字サイズの最大はインプリメントに依存する。

```
/* フォント指定 */
typedef struct ExtendFontSpecifier {
    TC    name[L_FONTFAMNM]; /* フォントファミリ名 */
    UW    fclass;           /* フォントクラス */
    UW    attr;             /* フォント属性 */
    SIZE  size;            /* 文字サイズ */
} FSSPEC;
```

また、指定したフォントに関する以下の情報を得ることができる。

```
/* フォント情報 */
struct FontInfo {
    UH    height;          /* 最大文字高 (文字サイズ) */
    UH    width;           /* 最大文字幅 */
    UH    base;           /* ベース位置 */
    UH    leading;        /* レディング */
};
```

```
typedef FontInfo    FNTINFO; /* フォント情報 */
```

文字の太さ、幅、斜体(イタリック)、袋文字等に関しては、フォント属性として指定することにより描画することが可能であるが、下線、上線、網掛け、反転、上付き、下付き、ルビ等の処理はディスプレイプリミティブとしてはサポートしてしないため、アプリケーション側で行なう必要がある。

なお、フォントに関する詳細は「[第3章 3.9 フォントマネージャ](#)」を参照のこと。

```
/* 文字コード */
typedef UH    TC    /* 文字コード(1文字) */
```

---

[この章の目次にもどる](#)

[前頁:2.1 はじめににもどる](#)

[次頁:2.3 基本関数にすすむ](#)

## 2.3 基本関数

ディスプレイプリミティブの関数はすべて W 型の関数値をとり、何らかのエラーがあった場合は、「負」のエラーコードが戻る。正常終了時には、「0」または「正」の値が戻る。

ディスプレイプリミティブの関数から戻されるエラーコードは以下のものである。

EG\_ADR ((-64) << 16)

パラメータで指定したアドレスが不正である。

EG\_PAR ((-65) << 16)

パラメータが不正、または範囲外である。

EG\_NOSPT ((-66) << 16)

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。

EG\_NOSPC ((-67) << 16)

システムのメモリが不足、または制限を越えた。

EG\_GID ((-68) << 16)

指定した描画環境IDが不正(存在しない)。

EG\_LOCK ((-69) << 16)

画環境はロックされている。

EG\_LIMIT ((-70) << 16)

システムの制限を越えた

EG\_NOEXS ((-71) << 16)

指定したデバイスは存在しない。(登録されていない。)

EG\_DEV ((-72) << 16)

指定したデバイスのタイプが不正である。

EG\_ERDEV ((-73) << 16)

デバイスでエラーが発生した。

EG\_FORM ((-80) << 16)

ビットマップの形式が異なる。

EG\_ENV ((-81) << 16)

カラーマップが存在しない。

なお、以下のエラーコードは、各関数毎の説明では特に記述していないが各種の描画関数で発生する可能性がある。

EG\_NOSPT

インプリメント上サポートされていない場合、もしくは予約されているパラメータ値が指定された場合。

基本関数としては、描画環境の生成 / 削除、描画環境の各データのアクセス、および座標変換を行なう関数群が用意されている。各種の関数内での「相対」「絶対」座標変換でオーバーフローが発生した場合、即ち、-32768 ~ 32767 の範囲を越えた場合の描画は保証されないため注



意が必要である。また、以下に示すように描画すべき図形の一部が -32768 ~ 32767 の座標系をはみ出した場合も描画は保証されない。

- 描画指定点は範囲内であるが、線幅が太いため枠の一部が範囲外となる場合。
- 回転した図形の描画で、回転後の図形の一部が範囲外となる場合。

### 2.3.1 描画環境の生成 / 削除

## gini\_dsp

Initialize Display Primitive

gini\_dsp

#### 【形式】

ERR gini\_dsp (W mode)

#### 【解説】

ディスプレイプリミティブの初期化処理を行う。

通常、アプリケーションは使用しない。

アプリケーションがこの関数を呼び出したときの動作は保証しない。

#### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_NOSP:

システムのメモリが不足した。

## gfin\_dsp

Finish Display Primitive

gfin\_dsp

#### 【形式】

ERR gfin\_dsp ( )

#### 【解説】

ディスプレイプリミティブの終了処理を行う。

通常、アプリケーションは使用しない。

アプリケーションがこの関数を呼び出したときの動作は保証しない。

#### 【リターン値】

=0 : 正常終了

<0 : エラー (発生しない)

#### 【エラーコード】

発生しない。

## gcls\_env

Delete Drawing Environment

gcls\_env

#### 【形式】

ERR gcls\_env ( GID gid )

### 【解説】

gid で指定した描画環境を削除する。

描画環境を生成したプロセスのみがその描画環境を削除可能であり、 そうでない場合には EG\_GID のエラーとなる。

### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_GID:

指定した描画環境IDが不正(存在しない)。

**gopn\_dev**

Open Devide Drawing Environment

gopn\_dev

### 【形式】

GID gopn\_dev ( TC \*dev, B \*par )

### 【解説】

dev で指定したデバイスに対する「デバイス描画環境」を新規に生成し、その 描画環境IDを関数値として戻す。 固有ビットマップを持たないデバイスの場合は、EG\_DEVのエラーとなる。

生成された描画環境の各データは、以下のように初期化される。

|           |   |
|-----------|---|
| 対象デバイス    | : dev                                   |
| 対象ビットマップ  | : dev の固有ビットマップ                         |
| マスクピクセル値  | : 白(0)                                  |
| フレーム長方形   | : ビットマップの bounds 長方形                    |
| 表示長方形     | : ビットマップの bounds 長方形                    |
| 前置長方形リスト  | : 未定義                                   |
| クリッピング領域  | : 未定義                                   |
| 線マスクパターン  | : 実線                                    |
| 文字カラー     | : 前景 : 黒(0x10000000)、背景 : 白(0x10FFFFFF) |
| 文字描画方向    | : TORIGHT (0)                           |
| 文字描画位置    | : (0,0)                                 |
| 文字間隔      | : すべて0                                  |
| 文字フォント    | : システムフォント                              |
| 文字セット     | : システムフォントの文字セット                        |
| 色変換用カラー情報 | : 未定義                                   |

### 【リターン値】

0 : 正常終了(関数値は描画環境ID)

<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(dev)のメモリのアクセスは許されていない。

EG\_DEV

指定したデバイスのタイプが不正である。

EG\_LIMIT

システムの制限を越えた。(オープン可能な描画環境の最大数を越えた)

EG\_NOEXS

指定されたデバイス(dev)は登録されていない。

EG\_NOSPC

システムのメモリが不足した。

## gopn\_mem

Open Memory Drawing Environment

gopn\_mem

### 【形式】

GID gopn\_mem ( TC \*dev, BMP \*bmap, B \*par )

### 【解説】

dev で指定したデバイスに対する「メモリ描画環境」を新規に生成し、その描画環境IDを関数値として戻す。描画用のビットマップは bmap で指定したものが使用される。指定したデバイスがカラーマップを持つ場合は、カラーマップを共有することになる。

bmap の内容は、描画環境生成後にディスプレイプリミティブにより直接アクセスされるため、変更してはいけない。

bmap で指定したビットマップのプレーン数とピクセルビット数が、dev で指定したデバイスで定義されるものと一致していない場合は、EG\_DEV のエラーとなる。

dev = NULL の場合は、「汎用メモリ描画環境」の生成となる。B \*par == NULL の場合、カラー情報を持たない汎用メモリ描画環境を生成する。

B \*par != NULL の場合、CSPEC \*par とみなされ、カラー情報付きの汎用メモリ描画環境を生成する。

CSPEC.colmap にカラーマップへのアドレスを設定することにより、カラーマップ方式の汎用メモリ描画環境の生成も可能である。

生成された描画環境の対象デバイス、ビットマップ、カラー情報は生成する描画環境の種類に応じてパラメータとして指定されたものになり、他の各データはデバイス描画環境の生成時と同様に初期化される。

### 【リターン値】

0 : 正常終了(関数値は描画環境ID)

< 0 : エラー(関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(dev)のメモリのアクセスは許されていない。

EG\_DEV

指定したデバイスのタイプが不正である。

EG\_LIMIT

システムの制限を越えた。(オープン可能な描画環境の最大数を越えた)

EG\_NOEXS

指定されたデバイス(dev)は登録されていない。

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。bmap の内容が不正、par の内容が不正(カラー情報付き汎用メモリ描画環境の場合)

## gnew\_env

Open New Drawing Environment

gnew\_env

### 【形式】

GID gnew\_env ( GID gid )

### 【解説】

gid で指定した描画環境と全く同じ内容を持つ描画環境を新たに生成(複製)し、その描画環境IDを関数値として戻す。デバイス情報、カラー情報、カラーマップ、ビットマップ (BMP.baseaddr のデータ領域)は生成元と共有される。

### 【リターン値】

0 : 正常終了(関数値は描画環境ID)  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。(オープン可能な描画環境の最大数を越えた)

EG\_NOSPC

システムのメモリが不足した。

## gini\_env

Initialize Drawing Environment

gini\_env

### 【形式】

ERR gini\_env ( GID gid )

### 【解説】

gid で指定した描画環境の各データを再初期化し、gopn\_dev() または gopn\_mem() を行なった直後の状態と同じ状態にする。ただし、ビットマップ上に描画されているデータは変更されない。

### 【リターン値】

= 0 (E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_GID : 指定した描画環境IDが不正(存在しない)。

## gget\_spc

Get Device Specification

gget\_spc

### 【形式】

ERR gget\_spc ( TC \*dev, DEVSPEC \*buf )

### 【解説】

dev で指定したデバイスの情報(仕様)を取り出し、buf で指定した領域に格納する。

### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(dev, spec)のメモリのアクセスは許されていない。

EG\_NOEXS

指定されたデバイス(dev)は登録されていない。

## gget\_dev

Get Device Name

gget\_dev

### 【形式】

ERR gget\_dev ( W gid, TC dev[11] )

### 【解説】

gid で指定した描画環境の対象デバイス名を dev に指定した領域に格納する。対象デバイスが存在しない場合(汎用メモリ描画環境)は空文字列が格納される。

### 【リターン値】

0 : 正常終了  
0=デバイス描画環境  
1=メモリ描画環境  
2=汎用メモリ描画環境  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(dev)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gget\_dsp

Get Device Specification

gget\_dsp

### 【形式】

ERR gget\_dsp ( W gid, DEVSPEC \*buf )

## 【解説】

gid で指定した描画環境の対象デバイス情報を buf に指定した領域に格納する。対象デバイスが存在しない場合(汎用メモリ描画環境)は EG\_DEV となる。

## 【リターン値】

0 : 正常終了(関数値は0=デバイス描画環境 / 1=メモリ描画環境)  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(buf)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

# gget\_csp

Get Color Specification

gget\_csp

## 【形式】

ERR gget\_csp ( W gid, CSPEC \*buf )

## 【解説】

gid で指定した描画環境のカラー情報を buf に指定した領域に格納する。ただし、colmap はカラーマップ方式であっても格納されず常に NULL が設定される。カラーマップは別に gget\_col() で取り出す必要がある。カラー情報が存在しない場合は EG\_DEV となる。

## 【リターン値】

0 : 正常終了(関数値は0=デバイス描画環境 / 1=メモリ描画環境 / 2=汎用メモリ描画環境)  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(buf)のメモリのアクセスは許されていない。

EG\_DEV

デバイスはカラー情報を持っていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

# gset\_bcs

Set Bitmap Convert Color Specification

gset\_bcs

## 【形式】

ERR gset\_bcs ( GID gid, CSPEC \*cspec )

## 【解説】

gid で指定した描画環境に対して、色変換用のCSPECを設定する。CSPEC がカラーマップ形式の場合、CSPEC.colmap にもエントリ数分の カラーマップが設定されていなければならない。

cspec = NULL とした場合、変換用CSPECを未定義にすることを意味する。色変換用カラー

情報は、カラー情報を持たない描画環境には設定できない。

【リターン値】

0: 正常終了  
<0: エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR  
指定されたアドレス(cspect)のメモリのアクセスは許されていない。  
EG\_ENV  
デバイスはカラー情報を持っていない。  
EG\_GID  
指定した描画環境IDが不正(存在しない)。

## gget\_bcs

Get Bitmap Convert Color Specification

gget\_bcs

【形式】

ERR gget\_bcs ( GID gid, CSPEC \*cspec )

【解説】

gid で指定した描画環境に対して、色変換用のCSPECを取り出す。  
CSPEC.colmap = NULL でない場合、カラーマップが取り出される可能性があるため、カラーマップを取り出すのに十分な領域が確保されていなければならない。  
CSPEC.colmap = NULL の場合、描画環境に設定されている色変換用カラー情報が カラーマップ形式であっても、カラーマップは取り出されない。

【リターン値】

0: 正常終了  
<0: エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR  
指定されたアドレス(cspect)のメモリのアクセスは許されていない。  
EG\_ENV  
デバイスはカラー情報を持っていない。または色変換用カラー情報は未定義である。  
EG\_GID  
指定した描画環境IDが不正(存在しない)。

### 2.3.2 カラーマップの操作

## gset\_col

Set Color Map Entries

gset\_col

【形式】

W gset\_col ( GID gid, UW p, COLOR \*cv, UW cnt )

【解説】

gid で指定した描画環境のカラーマップの p で指定したエントリから連続した cnt 個のエントリを cv で指定した値に設定し、実際に設定したエントリ の数を関数値として戻す。 cv は、COLOR の cnt 個の COLOR 配列へのポインタである。

カラーマップを持たないか、設定可能でないカラーマップを持つ描画環境の場合は EG\_ENV のエラーとなる。

【リターン値】

0 : 正常終了(関数値は実際に設定したエントリの数 ( 0 ~ ))  
<0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR

指定されたアドレス(cv)のメモリのアクセスは許されていない。

EG\_ENV

カラーマップが存在しない。(カラーマップは存在しない)

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(p は範囲外)

## gget\_col

Get Color Map Entries

gget\_col

【形式】

W gget\_col ( GID gid, UW p, COLOR OUT \*cv, UW cnt )

【解説】

gid で指定した描画環境の対象デバイスのカラーマップの p で指定したエントリから連続した cnt 個のエントリに対応する COLORVAL 値を cv で指定した領域に取り出し、実際に取り出したエントリの数を関数値として戻す。 cv は cnt 個の COLOR 配列へのポインタであり、その配列は予め確保しておく必要がある。

カラーマップ形式でないカラー情報を持つ描画環境の場合、 p をピクセル値とみなし、ピクセル値に対応する絶対 R G B カラーを cv で指定した cnt 個の領域に取り出すことができる。

カラー情報を持たない描画環境の場合は EG\_ENV のエラーとなる。

【リターン値】

0 : 正常終了(関数値は実際に取り出したエントリの数 ( 0 ~ ))  
<0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR

指定されたアドレス(cv)のメモリのアクセスは許されていない。

EG\_ENV

カラーマップが存在しない。(カラーマップは存在しない)

EG\_GID

指定した描画環境IDが不正(存在しない)。



EG\_PAR

パラメータが不正、または範囲外である。(p は範囲外)

**gcnv\_col**

Convert Color to Pixel Value

gcnv\_col

**【形式】**

W gcnv\_col ( GID gid, COLOR cv, PIXVAL OUT \*pixv )

**【解説】**

gid で指定した描画環境で cv で指定した絶対カラーにもっとも近いカラーのピクセル値を取り出し、pixv で指定した領域に格納する。cv の上位 4 ビットの値は無視され、下位 28 ビットが RGB 絶対カラー指定値と見なされる。

カラー情報を持たない描画環境の場合は EG\_ENV のエラーとなる。

関数値として、カラーの近似度の目安を示す 0 ~ 100 の範囲の値を戻す。0 の場合は完全に一致したカラーであり、100 は補色となる。この値はあくまで目安であり、厳密な意味はもたない。

**【リターン値】**

0 : 正常終了(関数値はカラーの近似度(0 ~ 100))  
<0 : エラー(関数値はエラーコード)

**【エラーコード】**

EG\_ADR

指定されたアドレス(pixv)のメモリのアクセスは許されていない。

EG\_ENV

カラー情報が存在しない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

### 2.3.3 描画環境レコードのアクセス

**gset\_msk**

Set Mask Pixel Value

gset\_msk

**【形式】**

ERR gset\_msk ( GID gid, PIXVAL pixv )

**【解説】**

gid で指定した描画環境の「マスクピクセル値」を pixv で指定した値に変更する。

**【リターン値】**

=0(E\_OK) : 正常終了  
<0 : エラー(関数値はエラーコード)

**【エラーコード】**

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。

## gget\_msk

Get Mask Pixel Value

gget\_msk

### 【形式】

ERR gget\_msk ( GID gid, PIXVAL OUT \*pixv )

### 【解説】

gid で指定した描画環境の「マスクピクセル値」を取り出し、pixv で指定した領域に格納する。

### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。

## gget\_clp

Get Clipping Region

gget\_clp

### 【形式】

ERR gget\_clp ( GID gid, GRGN \*clip, W csize, W \*a\_size )

### 【解説】

gid で指定した描画環境の「クリッピング領域」を csize バイトの clip の領域に設定する。

a\_size が NULL でない場合、「クリッピング領域」を設定するために必要なメモリサイズが設定される。

「クリッピング領域」に必要なメモリサイズを獲得するためには、clip = NULL, csize = 0 でこの関数を呼び出せばよい。

clip が NULL または「クリッピング領域」を設定するために必要なサイズよりも少ない場合、clip の領域にはなにも設定されない。

関数値として、実際に設定したクリッピング領域のサイズを戻す。0 の場合、なにも設定されなかったことを示す。a\_size = 0, 関数値 = 0 の場合、クリッピング領域が未定義であることを示す。

### 【リターン値】

0 : 正常終了(関数値は実際に設定されたクリッピング領域データサイズ)

<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(clip,a\_size)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

# gset\_clp

Set Clipping Region

gset\_clp

## 【形式】

ERR gset\_clp ( GID gid, GRGN \*clip )

## 【解説】

gid で指定した描画環境の「クリッピング領域」を clip で指定した任意領域 に変更する。

clip は任意領域へのポインタであり、NULL の場合は、「クリッピング領域」を未定義(なし)にする。任意領域の内容はコピーされてディスプレイプリミティブ 内で使用される。

## 【リターン値】

= 0 (E\_OK) : 正常終了

< 0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(clip) のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。(clp の内容が不正)

# gget\_fra

Get Frame Rectangle

gget\_fra

## 【形式】

ERR gget\_fra ( GID gid, RECT \*rp )

## 【解説】

gid で指定した描画環境の現在の「フレーム長方形」を rp で指定した領域に 取り出す。rp は長方形領域へのポインタであり、 その領域はあらかじめ確保 しておく必要がある。

## 【リターン値】

=0(E\_OK) : 正常終了

< 0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(rp)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gset\_fra

Set Frame Rectangle

gset\_fra

### 【形式】

ERR gset\_fra ( GID gid, RECT r )

### 【解説】

gid で指定した描画環境の「フレーム長方形」を r で指定した長方形に変更する。

### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。

## gget\_vis

Get Visible Rectangle

gget\_vis

### 【形式】

ERR gget\_vis ( GID gid, RECT \*rp )

### 【解説】

gid で指定した描画環境の現在の「表示長方形」を rp で指定した領域に取り出す。rpは長方形領域へのポインタであり、その領域はあらかじめ確保しておく必要がある。

### 【リターン値】

=0 (E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(rp)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gset\_vis

Set Visible Rectangle

gset\_vis

### 【形式】

ERR gset\_vis ( GID gid, RECT r )

【解説】

gid で指定した描画環境の「表示長方形」を r で指定した長方形に変更する。

【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。

## gget\_for

Get Foreground Rectangle List

gget\_for

【形式】

ERR gget\_for ( GID gid, RLIST \*rlp )

【解説】

gid で指定した描画環境の現在の「前置長方形リスト」を rlp で指定した領域に取り出す。rlp は前置長方形リストへのポインタであり、リストの各要素の r\_next は正しく設定されているものとする。

【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR

指定されたアドレス(rp)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gset\_for

Set Foreground Rectangle List

gset\_for

【形式】

ERR gset\_for ( GID gid, RLIST \*rlp )

【解説】

gid で指定した描画環境の「前置長方形リスト」を rlp で指定した長方形リストに変更する。rlp が NULL の場合、「前置長方形リスト」を空(未定義)にする。

【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。

**gsiz\_pat**

Get Pattern Image Size

gsiz\_pat

**【形式】**

ERR gsiz\_pat ( GID gid, UW kind, UW hsize, UW vsize, W \*size )

**【解説】**

gid で指定した描画環境上での描画で使用する内部パターンイメージを格納するために必要なメモリ領域全体のバイトサイズを取り出し、size で指定した領域に格納する。

kind は使用するパターンのタイプであり、内部パターンであってはいけない。hsize, vsize は、使用するパターンの横サイズ、縦サイズのピクセル数の指定である。

このサイズは、その描画環境用に生成される同一のパターンタイプ、同一のパターンのサイズを持つ全ての内部パターンイメージに対して同一の値である。即ち、得られたサイズ×N のメモリ領域内にN個の任意のパターンを生成できることが保証される。

**【リターン値】**

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

**【エラーコード】**

EG\_ADR

指定されたアドレス(rp)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。

EG\_PAR

パラメータが不正、または範囲外である。(hsize,vsizeが大きすぎる)

**gcre\_pat**

Create Pattern Image

gcre\_pat

**【形式】**

ERR gcre\_pat ( GID gid, PAT \*def, UB \*loc, UB \*\*pat, W \*size, W cvt )

**【解説】**

gid で指定した描画環境上で描画するための内部パターンイメージを生成する。def は生成する描画パターンの定義データへのポインタであり、パターンタイプは内部パターンであってはいけない。

loc は、生成した内部パターンイメージを格納する領域へのポインタであり、少なくとも gsiz\_pat() で得られた大きさがなくてはならない。

loc = NULL の場合、パターンイメージ用の領域(ローカルメモリ)が get\_mbk() により自動的に確保される。自動的に確保されたローカルメモリは rel\_mbk() によって解放できる。

\*pat には、生成された内部パターンイメージへのポインタが戻され、そのバイト数が \*size に戻される。得られたポインタを PAT 構造体の ipat.ptr に設定することにより、パターンとして使用できることになる。loc != NULL の場合には、\*pat には loc の値が設定されることになる。

cvt には G\_CVFORM および G\_CVCOLOR の指定が可能で、この指定は変換前のパターンが MPAT である場合のみ意味を持つ。色変換用の CSPEC が描画環境に設定されていない場合には G\_CVCOLOR 指定は無視される。MPAT のビットマップの有効ピクセルビット数が色変換用 CSPEC の色指定と矛盾する (カラーマップのエントリ数が足りないなど) 場合、EG\_ENV を戻す。

生成したパターンは、同一ビットマップ形式を持つ他の描画環境に対しても使用することが可能であるが、描画環境の CSPEC が異なる場合、描画色に関しては保証されない。

#### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(size) のメモリのアクセスは許されていない。

EG\_FORM

パターンで指定されたビットマップ形式が描画環境のビットマップ形式と異なる。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPC

システムのメモリが不足した。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。

EG\_PAR

パラメータが不正、または範囲外である。(def の内容が不正)

## gset\_lmk

Set Line Mask

gset\_lmk

#### 【形式】

ERR gset\_lmk ( GID gid, UW nbytes, UB \*mask )

#### 【解説】

gid で指定した描画環境の「線マスクパターン」を mask で指定した内容に設定する。

nbytes はマスクパターンのバイト数を示す 1 ~ 8 の値であり、パターンの周期を意味する。mask は nbytes で指定したバイト数のバイト配列へのポインタである。

### 【リターン値】

= 0(E\_OK) : 正常終了  
< 0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(mask) のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(nbytes < 1, > 8)

## gget\_lmk

Get Line Mask

gget\_lmk

### 【形式】

W gget\_lmk ( GID gid, UB \*mask )

### 【解説】

gid で指定した描画環境の「線マスクパターン」を取り出し、mask で指定した領域に格納し、関数値としてパターンのバイト数(1 ~ 8)を戻す。mask は 8 バイトの領域が確保されている必要がある。

### 【リターン値】

0 : 正常終了(関数値は線マスクパターンバイト数(1 ~ 8))  
< 0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(mask) のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gget\_bmp

Get Bitmap

gget\_bmp

### 【形式】

ERR gget\_bmp ( GID gid, BMP \*bmap )

### 【解説】

gid で指定した描画環境の「対象ビットマップ」を bmap で指定した領域に取り出す。bmap はビットマップ領域へのポインタであり、その領域は予め確保しておく必要がある。

### 【リターン値】

=0(E\_OK) : 正常終了 < 0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(rp)のメモリのアクセスは許されていない。



EG\_GID

指定した描画環境IDが不正(存在しない)。

## gget\_bnd

Get Bitmap Bounds

gget\_bnd

### 【形式】

ERR gget\_bnd ( GID gid, RECT \*rp )

### 【解説】

gid で指定した描画環境の対象ビットマップの境界長方形(bounds)を rp で指定した領域に取り出す。rp は長方形領域へのポインタであり、その領域は予め確保しておく必要がある。

### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(rp)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gmov\_bnd

Move Bitmap Bounds

gmov\_bnd

### 【形式】

ERR gmov\_bnd ( GID gid, W dh, W dv )

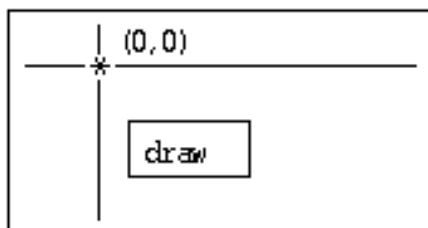
### 【解説】

gid で指定した描画環境のビットマップの境界長方形(bounds)を (dh, dv) だけ並行移動する。dh は水平方向の移動量、dv は垂直方向の移動量を示す。

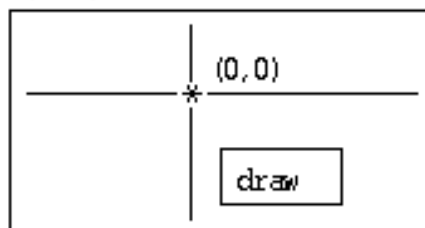
なお、クリッピング領域、文字描画位置等の相対座標は変更されない。

例： gmov\_bnd (gid, -20, -10) を行なった場合：

(-15, -10)



(-35, -20)



クリッピング領域(draw)のlefttopの相対座標は、(5,15)のまま変化しない。

図 34 : ビットマップ境界の移動

### 【リターン値】

= 0(E\_OK) : 正常終了

<0: エラー (関数値はエラーコード)

【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(座標値がオーバーフローした)

## gmov\_cor

Move Coordination

gmov\_cor

【形式】

ERR gmov\_cor ( GID gid, W dh, W dv )

【解説】

gid で指定した描画環境の座標系全体を(dh, dv)だけ並行移動する。即ち、ビットマップ境界(bounds)、フレーム長方形、表示長方形、文字描画位置の全てを(dh, dv)だけ平行移動する。dh は水平方向の移動量、dv は垂直方向の移動量を示す。

例: gmov\_cor (gid, -20, -10) を行なった場合:

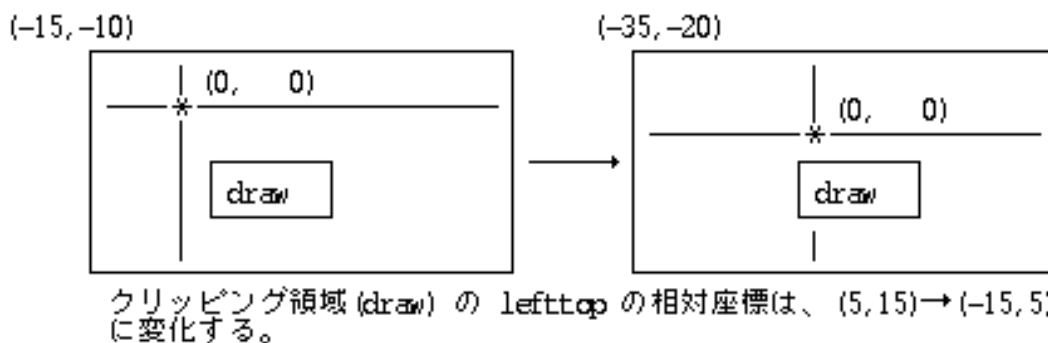


図 35 : 座標系の移動

【リターン値】

= 0(E\_OK) : 正常終了

<0: エラー (関数値はエラーコード)

【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(座標値がオーバーフローした)

### 2.3.4 座標変換

## gcnv\_abs

Convert to Absolute Coordination

gcnv\_abs

【形式】

ERR gcnv\_abs ( GID gid, PNT \*pp )

【解説】

gid で指定した描画環境の相対座標で示された点 pp の座標値を、絶対座標に変換する。結果は pp に直接戻される。

【リターン値】

= 0(E\_OK) : 正常終了  
< 0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR  
指定されたアドレス(pp)のメモリのアクセスは許されていない。  
EG\_GID  
指定した描画環境IDが不正(存在しない)。  
EG\_PAR  
パラメータが不正、または範囲外である。(座標値がオーバーフローした)

## gcnv\_rel

Convert to Relative Coordination

gcnv\_rel

【形式】

ERR gcnv\_rel ( GID gid, PNT \*pp )

【解説】

gid で指定した描画環境の絶対座標で示された点 pp の座標値を、相対座標に変換する。結果は pp に直接戻される。

【リターン値】

= 0(E\_OK) : 正常終了  
< 0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR  
指定されたアドレス(pp)のメモリのアクセスは許されていない。  
EG\_GID  
指定した描画環境IDが不正(存在しない)。  
EG\_PAR  
パラメータが不正、または範囲外である。(座標値がオーバーフローした)

## gloc\_env

Lock Drawing Environment

gloc\_env

【形式】

ERR gloc\_env ( GID gid W lock )

【解説】

gid で指定した「デバイス描画環境」と、同一のデバイスに対する他の「デバイス描画環境」からの書込みアクセスをロック(lock = 0の時)、またはロック解除(lock = 0の時)する。

デバイスがロック対象外のデバイスの場合、またはメモリ描画環境の場合は、ロックは適用されず無視される。

ロックされた描画環境での描画(書込み)を行なった場合は、ロックが解除されるまで待たされる。待たされている描画が複数存在した場合、解除後の描画順序は待たされた順番となる。待つことによりデッドロックとなるような場合は、待たずに EG\_LOCK のエラーが各種の描画関数で戻されることになる。ただし、このデッドロックの検出はインプリメントに依存する。

#### 【リターン値】

= 0(E\_OK) : 正常終了

< 0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LOCK

既にロックされている、またはロックが多すぎる。

EG\_PAR

パラメータが不正、または範囲外である。(ロックしたプロセスでない)

---

[この章の目次にもどる](#)

[前頁:2.2 基本概念にもどる](#)

[次頁:2.4 図形描画関数にすすむ](#)

## 2.4 図形描画関数

個別には示さないが、図形描画関数で PAT \*pat と DCM mode を引数に持つ関数は、描画モードとして G\_CVFORM と G\_CVCOLOR を指定することが可能である。

G\_CVFORM:ビットマップ形式変換

- パターン(kind=1, mpat)のビットマップの形式(プレーン数、ピクセルビット数)が描画環境のビットマップ形式と異なる場合は変換を行なう。
- この指定が無い場合、ソースとディスティネーションのビットマップの形式(プレーン数、ピクセルビット数)が異なる場合はEG\_PARのエラーとなる。
- 変換は、変換後の双方の下位Nビットのピクセル値が等しくなるように行なわれる(ここで、Nはソースとディスティネーションのdepthの最小値を意味する)。

G\_CVCOLOR: カラー変換

- 描画環境に設定されている変換用カラー情報と描画環境自体のカラー情報が異なり、パターンが kind = 1, mpat である場合に 変換を行なう。パターンとして指定されたビットマップのカラー形式として、描画環境に設定されている変換用カラー形式が適用される。どちらか一方のカラー表現方式が未定義の場合は変換は行なわれない。
- カラー変換はソースのピクセル値を絶対カラーに変換し、その絶対カラーに最も近い絶対カラーに対応するディスティネーションのピクセル値を求めることにより行なわれる。
- ディスティネーションのカラー表現形式がカラーマップ形式の場合は、その時点のカラーマップの内容を使用し、カラーマップの内容の自動変更は行なわない。

### 2.4.1 直線 / 点の描画

直線の描画では、指定した2点を結ぶ直線を、線幅(太さ)、線種、パターン、描画モードを指定して描画する。

直線の描画方法としては、指定した線幅の大きさの正方形の左上の点が線を構成する点の軌跡上を移動していった場合の、正方形全体の軌跡を直線として描画する。即ち、移動した正方形に重なったピクセルが描画されることになる。従って、線幅が1以上の直線は全体に右下にずれた形となる。(下図参照)。

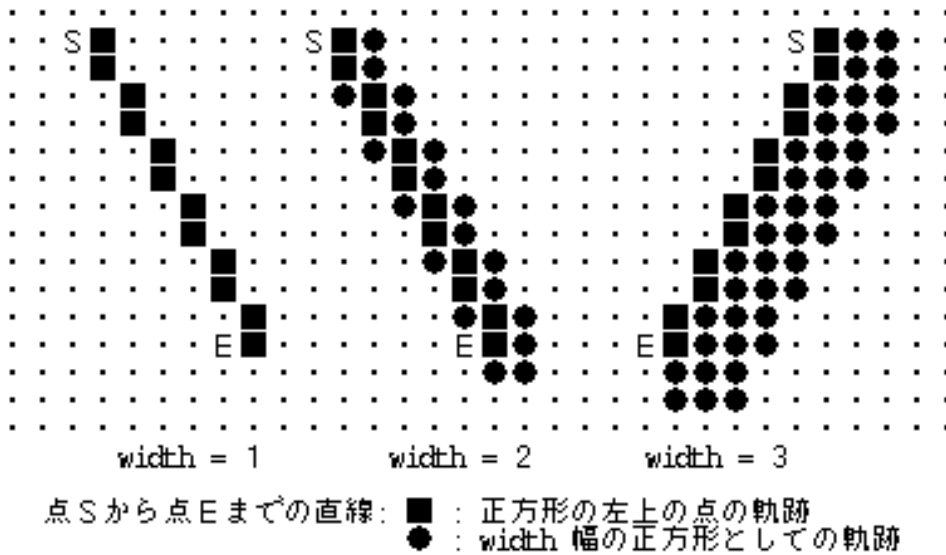


図 36 : 直線の描画

点の描画では、指定した点のピクセル値の取り出し、指定したピクセル値による描画、指定したパターンによる描画の関数が用意されている。

## gdra\_lin

Draw Line

直線の描画

- 【形式】  
ERR gdra\_lin ( GID gid, PNT p1, PNT p2, LATTR attr, PAT \*pat, DCM mode )
- 【解説】  
gidで指定した描画環境で、点 p1 と p2 結ぶ直線を attr で指定した属性、pat で指定した描画パターン、mode で指定した描画モードで描画する。
- 【リターン値】  
=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)
- 【エラーコード】  
EG\_ADR  
指定されたアドレス(pat)のメモリのアクセスは許されていない。  
EG\_GID  
指定した描画環境IDが不正(存在しない)。  
EG\_PAR  
パラメータが不正、または範囲外である。(patの内容, attr, modeが不正)

## gtst\_pnt

Test Point

点の状態の取り出し

- 【形式】  
Bool gtst\_pnt ( GID gid, PNT p, PIXVAL \*val )

## 【解説】

gid で指定した描画環境で、点 p がクリッピング領域の内部にあり描画される場合は関数値 "0" を戻し、そのピクセル値を val で指定した領域に取り出す。点 p がクリッピング領域の外部にあり描画されない場合は関数値 "1" を戻し、val には何も設定されない。

## 【リターン値】

0 : 正常終了(関数値は点のクリッピング状態(0/1))  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(val)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

# gdra\_pnt

Draw Point

点の描画

## 【形式】

ERR gdra\_pnt ( GID gid, PNT p, PIXVAL val, DCM mode )

## 【解説】

gid で指定した描画環境で、点 p を、val で指定したピクセル値、mode で指定した描画モードで描画する。

## 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(modeが不正)

# gfil\_pnt

Fill Point

点のパターン描画

## 【形式】

ERR gfil\_pnt ( GID gid, PNT p, PAT \*pat, DCM mode )

## 【解説】

gid で指定した描画環境で、点 p を、pat で指定したパターン、mode で指定した描画モードで描画する。

## 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容,modeが不正)

## 2.4.2 長方形の描画

長方形の描画関数としては、枠の描画、長方形領域のパターンによる塗り潰し、が用意されている。

長方形の左上の点を中心として、反時計方向に任意の角度回転させた傾いた長方形の描画も可能である。回転角度は0以上の度単位の非負の数値で指定し、360の剰余が有効となる。なお、回転した場合、線のパターン、および塗り潰しのパターン自体は回転しない。

```
/* 回転角度 */
typedef UW DEG /* 回転角度 */
```

長方形は、top, left, bottom, right の値により指定されるが、top bottom または left right の場合は、その長方形は「空」であり、描画関数では EG\_PAR のエラーとなる。長方形の領域は、half-open property のため、右と下の部分が1ピクセルだけ左および上にずれた形となる点に注意が必要である。(下図参照) 枠の描画で、太い枠の場合は必ず内側に太くなり決して外側にはみでることはない。また、枠以外の描画では、その対象となる領域は枠を含んだ長方形全体の領域となる。

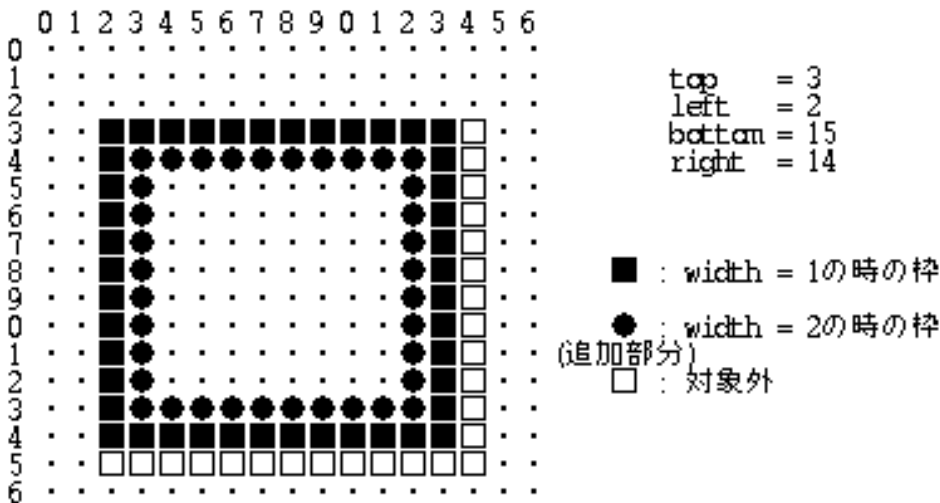


図 37 : 長方形の描画

### gfra\_rec

Frame Rectangle

長方形の枠の描画

#### 【形式】

ERR gfra\_rec ( GID gid, RECT r, LATTR attr, PAT \*pat, DEG angle, DCM mode )

#### 【解説】

gid で指定した描画環境で、r で指定した長方形の枠を attr で指定した属性、pat で指定した描画パターン、angle で指定した回転角度、mode で指定した描画モードで描画する。



線幅 > 1 の場合には枠は内側に線幅だけ太くなり、指定した枠の外側にはみでることはない。線幅が大き過ぎて枠をはみでるような場合は、枠の中を指定したパターンで塗り潰す。(エラーとはならない)

#### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, attr, modeが不正、rが空)

## gfil\_rec

Fill Rectangle

長方形の塗り潰し

#### 【形式】

ERR gfil\_rec ( GID gid, RECT r, PAT \*pat, DEG angle, DCM mode )

#### 【解説】

gid で指定した描画環境で、r で指定した長方形を、pat で指定したパターン、angle で指定した回転角度、mode で指定した描画モードで塗り潰す。

#### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, modeが不正、rが空)

## 2.4.3 角丸長方形の描画

角丸長方形は長方形の角を楕円弧により丸めた図形であり、長方形の場合と同様に、枠の描画、指定パターンによる塗り潰しの描画関数が用意されている。

長方形の左上の点を中心として、反時計方向に任意の角度回転させた傾いた角丸長方形の描画も可能である。回転角度は0以上の度単位の非負の数値で指定し、360の剰余が有効となる。なお、回転した場合、線のパターン、および塗り潰しのパターン自体は回転しない。

長方形は、top, left, bottom, right の値により指定されるが、top bottom または left right の場合は、その長方形は「空」であり、描画関数では EG\_PAR のエラーとなる。

角は、指定した垂直直径(rv)と水平直径(rh)により定義される楕円の弧により丸められる。どちら

かの直径が0の場合は、角は丸められずに通常の長方形が描画されることになる。rvが長方形の高さより大きい場合、またはrhが長方形の幅より大きい場合は、rv、rhはそれぞれ長方形の高さ、幅に等しいものとして描画され、エラーとはならない。

枠の描画で、太い枠の場合は必ず内側に太くなり決して外側にはみでることはない。また、枠以外の描画では、その対象となる領域は枠を含んだ長方形全体の領域となる。

## gfra\_rrc

Frame Round Rectangle

角丸長方形の枠の描画

### 【形式】

ERR gfra\_rrc ( GID gid, RECT r, UW rh, UW rv, LATTR attr, PAT \*pat, DEG angle, DCM mode )

### 【解説】

gidで指定した描画環境で、rおよびrh、rvで示された角丸長方形の枠をattrで指定した属性、patで指定したパターン、angleで指定した回転角度、modeで指定した描画モードで描画する。

線幅>1の場合には枠は内側に線幅だけ太くなり、指定した枠の外側にはみでることはない。線幅が大き過ぎて枠をはみでるような場合は、枠の中を指定したパターンで塗り潰す(エラーとはならない)。なお、rvまたはrhが0の場合は、gfra\_rec()と同じとなる。

### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR  
指定されたアドレス(pat)のメモリのアクセスは許されていない。  
EG\_GID  
指定した描画環境IDが不正(存在しない)。  
EG\_PAR  
パラメータが不正、または範囲外である。(patの内容、attr、modeが不正、rが空)

## gfil\_rrc

Fill Round Rectangle

角丸長方形の塗り潰し

### 【形式】

ERR gfil\_rrc ( GID gid, RECT r, UW rh, UW rv, PAT \*pat, DEG angle, DCM mode )

### 【解説】

gidで指定した描画環境で、rおよびrh、rvで示された角丸長方形をpatで指定したパターン、angleで指定した回転角度、modeで指定した描画モードで塗り潰す。rvまたはrhが0の場合は、gfil\_rec()と同じとなる。

### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容,modeが不正、rが空)

## 2.4.4 楕円(円)の描画

楕円(円)の描画関数としては、枠の描画、楕円(円)領域の指定パターンによる塗り潰し、が用意されている。

楕円(円)の描画は、外接する長方形を指定して行なう。即ち、指定した長方形の枠に内接する楕円(円)を描画することになる。この場合、長方形の枠と楕円(円)の枠は接点では重なることになる。また、指定した長方形が正方形の場合は、楕円ではなく円となる。

長方形の左上の点を中心として、反時計方向に任意の角度回転させた傾いた長方形で定義される楕円の描画も可能である。回転角度は0以上の度単位の非負の数値で指定し、360の剰余が有効となる。なお、回転した場合、線のパターン、および塗り潰しのパターン自体は回転しない。

長方形は、top, left, bottom, right の値により指定されるが、top bottom または left right の場合は、その長方形は「空」であり、描画関数では EG\_PAR のエラーとなる。

枠の描画で、太い枠の場合は必ず内側に太くなり決して外側にはみでることはない。枠の太さは長方形の枠と重なる部分(水平 / 垂直軸上の部分)の太さとして定義される。即ち、上下左右の辺を内側に枠の幅だけ小さくした長方形に対する楕円(円)の部分を取り貫いた形となる。

また、枠以外の描画では、その対象となる領域は枠を含んだ楕円(円)全体の領域となる。

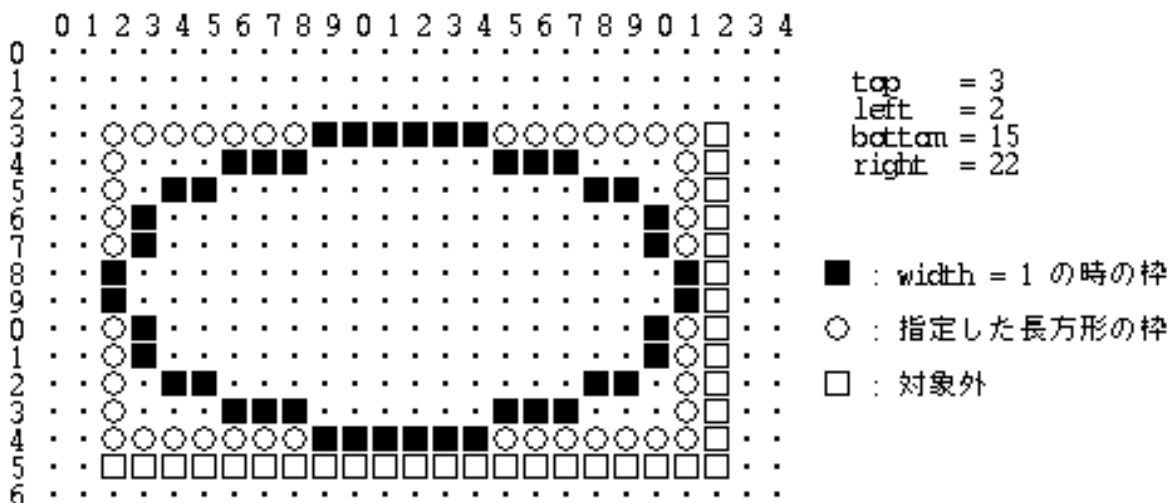


図 38 : 楕円(円)の描画

gfra\_ovl

Frame Oval

楕円(円)の枠の描画

### 【形式】

ERR gfra\_ovl ( GID gid, RECT r, LATTR attr, PAT \*pat, DEG angle, DCM mode )

### 【解説】

gid で指定した描画環境で、r で示された長方形(正方形)に内接する楕円(円)の枠を attr で指定した属性、pat で指定したパターン、angle で指定した回転角度、mode で指定した描画モードで描画する。

線幅 > 1 の場合には枠は内側に線幅だけ太くなり、垂直 / 水平の格子と交わる部分の幅が線幅になる。指定した枠の外側にはみでることはない。線幅が大き過ぎて枠をはみでるような場合は、枠の中を指定したパターンで塗り潰す(エラーとはならない)。

#### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, attr, modeが不正, rが空)

## gfil\_ovl

Fill Oval

楕円(円)の塗り潰し

#### 【形式】

ERR gfil\_ovl ( GID gid, RECT r, PAT \*pat, DEG angle, DCM mode )

#### 【解説】

gid で指定した描画環境で、r で示された長方形(正方形)に内接する楕円(円)を、patで指定したパターン、angle で指定した回転角度、mode で指定した描画モードで塗り潰す。

#### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, modeが不正, rが空)

## 2.4.5 弧 / 扇形 / 弓形の描画

弧 / 扇形 / 弓形の描画関数としては、弧の描画、扇形 / 弓形の枠の描画、扇形 / 弓形領域のパターンによる塗り潰し、が用意されている。

弧の描画は、外接する長方形(r)と開始点(sp)および終了点(ep)を指定して行なう。即ち、指定した長方形の枠に内接する楕円(円)の開始点から終了点までの弧を描画することになる。開始点、終了点は弧の線上にある必要はなく、実際には中心と開始点を結ぶ線から、時計回りで、中心と終了点を結ぶ線までの範囲の弧の描画を行なう。開始点と終了点が等しい場合は一周しているもの

と見なすため 楕円(円)となる。

弧の実際の描画方法はインプリメントに依存し特に規定しないが、正確に楕円(円) 枠を切り取った形として描画する方法と、直線と同様に指定した幅を持つ正方形が移動した軌跡を描く方法が考えられる。どちらの方法によっても、幅が1の場合は同一図形となり、指定した長方形に対する楕円(円)の枠と一致するが、幅が2以上の場合には、後者の方法では、全体として右下にずれた形となり楕円(円)の枠と一致しないことになる。

扇形の描画では、弧の描画に加えて弧の開始点と中心を結ぶ線、および弧の終了点と中心を結ぶ線を描画したものが枠となり、枠を含んだ内部の領域が扇形の領域となる。開始点と終了点等しい場合は中心からの線は描画されず、楕円(円)となる。

弓形の描画では、弧の描画に加えて弧の両端の点を結ぶ線(弦)を描画したものが枠となり、枠を含んだ内部の領域が弓形の領域となる。開始点と終了点等しい場合は弦は描画されず、楕円(円)となる。

扇形 / 弓形の枠(特に太い場合)の描画方法は、弧と同様にインプリメントに依存し特に規定しないが、幅1の枠の内側に太くなり外側にはみ出さない方法と、それぞれ指定された幅の弧と直線の組み合わせとして描く方法が考えられる。どちらの方法によっても、幅が1の場合は同一図形となるが、幅が2以上の場合には、後者の方法では、全体として右下にずれた形となる。

長方形の左上の点を中心として、反時計方向に任意の角度回転させた傾いた長方形で定義される楕円弧、扇形、弓形の描画も可能である。回転角度は0以上の度単位の非負の数値で指定し、360の剰余が有効となる。なお、回転した場合、線のパターン、および塗り潰しのパターン自体は回転しない。

長方形は、top, left, bottom, right の値により指定されるが、top bottom または left right の場合は、その長方形は「空」であり、描画関数では EG\_PAR のエラーとなる。

gdra\_arc

Draw Arc

弧の描画

### 【形式】

ERR gdra\_arc ( GID gid, RECT r, PNT sp, PNT ep, LATTR attr, PAT \*pat, DEG angle, DCM mode )

### 【解説】

gid で指定した描画環境で、r, sp, ep で指定した弧を attr で指定した属性、pat で指定したパターン、angle で指定した回転角度、mode で指定した描画モードで描画する。

### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, attr, modeが不正、rが空)

## gfra\_sec

Frame Sector

扇形の枠の描画

### 【形式】

ERR gfra\_sec ( GID gid, RECT r, PNT sp, PNT ep, LATTR attr, PAT \*pat, DEG angle, DCM mode )

### 【解説】

gid で指定した描画環境で、r, sp, ep で指定した扇形の枠を attr で指定した属性、pat で指定したパターン、angle で指定した回転角度、mode で指定した描画モードで描画する。

### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR  
指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID  
指定した描画環境IDが不正(存在しない)。

EG\_PAR  
パラメータが不正、または範囲外である。(patの内容, attr, modeが不正、rが空)

## gfil\_sec

Fill Sector

扇形の塗り潰し

### 【形式】

ERR gfil\_sec ( GID gid, RECT r, PNT sp, PNT ep, PAT \*pat, DEG angle, DCM mode )

### 【解説】

gid で指定した描画環境で、r, sp, ep で指定した扇形を pat で指定したパターン、angle で指定した回転角度、mode で指定した描画モードで塗り潰す。

### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR  
指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID  
指定した描画環境IDが不正(存在しない)。

EG\_PAR  
パラメータが不正、または範囲外である。(patの内容, modeが不正、rが空)

## gfra\_cho

Frame Chord

弓形の枠の描画

## 【形式】

ERR gfra\_cho ( GID gid, RECT r, PNT sp, PNT ep, LATTR attr, PAT \*pat, DEG angle, DCM mode )

## 【解説】

gid で指定した描画環境で、r, sp, ep で指定した弓形の枠を attr で指定した属性、pat で指定したパターン、angle で指定した回転角度、mode で指定した描画モードで描画する。

## 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, attr, modeが不正、rが空)

# gfil\_cho

Fill Chord

弓形の塗り潰し

## 【形式】

ERR gfil\_cho ( GID gid, RECT r, PNT sp, PNT ep, PAT \*pat, DEG angle, DCM mode )

## 【解説】

gid で指定した描画環境で、r, sp, ep で指定した弓形を pat で指定したパターン、angle で指定した回転角度、mode で指定した描画モードで塗り潰す。

## 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, modeが不正、rが空)

## 2.4.6 多角形(直線列)の描画

多角形の描画関数としては、枠の描画、多角形領域のパターンによる塗り潰し、が用意されている。また、開いた多角形としての直線列(polyline)の描画も用意されている。

多角形は、いくつかの点を与えて、それらの点を線で結んで得られる図形であり、点の配列として以下の構造体により定義される。

/\* 多角形 \*/

```
#define NP 1      /* ダミー */
struct Polygon {
    UH    round;      /* 角を丸める場合の円の直径 */
    UH    size;       /* 構成する頂点の数 */
    PNT    pt[NP];    /* 構成する頂点の配列(size 個の要素) */
};
```

```
typedef Polygon POLY; /* 多角形 */
```

即ち、pt[0] pt[1] pt[2] ... pt[size-2] pt[size-1] の順番に頂点を結んでいくことにより直線列が描画され、さらに pt[size-1] pt[0] を結ぶことにより多角形が描画される。従って、直線列では size>1、多角形では size>2; でない場合は EG\_PAR のエラーとなる。

多角形の頂点の最大数はインプリメントに依存するが、最低でも1000個はサポートするものとする。頂点の数が制限をオーバーした場合、または全体の図形が、描画環境で定義されている領域より(非常に)大きくて、描画できない場合は EG\_LIMIT のエラーとなり、その場合には、描画動作はいっさい行われぬ。

round は角を丸める指定であり、0の場合は角を丸めない多角形を意味し、0でない場合は、角を丸める場合の円の直径を示す。round が大きすぎて丸めた角の円弧部分が隣合う頂点で重なってしまうような場合は、エラーとならずに、重ならないような適当な大きさの円弧により角が丸められて描画される。

閉じた多角形の枠の実際の描画方法はインプリメントに依存し特に規定しないが、線幅1の枠を指定された太さだけ内側に太らす方法と、単に定義された頂点を順番に指定された太さの直線で結ぶ方法がある。幅が1の場合は同一図形となるが、幅が2以上の場合には後者の方法では全体として右下にずれた形となる。また、頂点や、線の交差する点では、線が2重書きされることがあるため、G\_XOR モードでの描画では注意する必要がある。

枠以外の描画では、その描画対象領域は、幅1で描かれた多角形の枠により生ずる内部閉領域であり、枠自体も含まれる。この場合、ある点Pが多角形の内部にあるかどうかは以下のように定義される。

- 点Pが多角形の辺(枠)上にあれば、点Pは多角形の内部にあると定義する。
- 多角形から十分に離れた任意の外部の点と、点Pを結ぶ線分が多角形の辺と交差する点の数が奇数であれば、点Pは多角形の内部にあり、偶数であれば、点Pは多角形の外部にあると定義する。

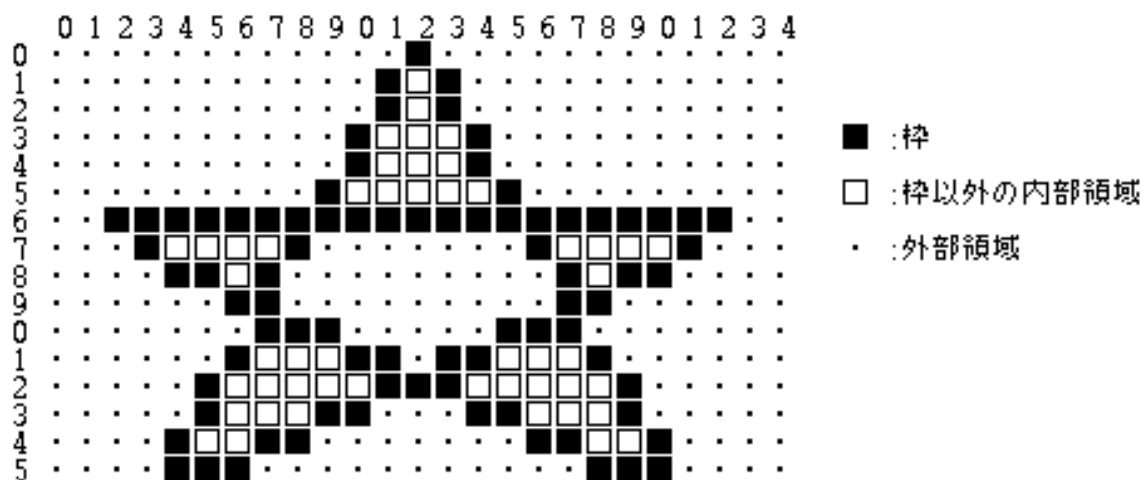


図 39 : 多角形の描画



# gdra\_pln

Draw Polyline

直線列の描画

## 【形式】

ERR gdra\_pln ( GID gid, POLY \*p, LATTR attr, PAT \*pat, DCM mode )

## 【解説】

gid で指定した描画環境で、p で指定した開いた多角形の枠、即ち直線列を attr で指定した属性、pat で指定したパターン、mode で指定した描画モードで描画する。指定した多角形の最後の点と最初の点は結ばれずに開いた形となる。

線幅>1の場合には、線幅の太さの直線を連続して書いたものになる。線幅が大きすぎて、全体の図形より大きくなってしまう場合にも、単に太い直線で枠を描画する。

## 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(p, pat) のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。(頂点の数が制限以上)

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, attr, modeが不正、poly.size 1)

# gfra\_pol

Frame Polygon

多角形の枠の描画

## 【形式】

ERR gfra\_pol ( GID gid, POLY \*p, LATTR attr, PAT \*pat, DCM mode )

## 【解説】

gid で指定した描画環境で、p で指定した多角形の枠を attr で指定した属性、pat で指定したパターン、mode で指定した描画モードで描画する。

線幅>1の場合の描画方法はインプリメントに依存する。

## 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(p, pat) のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。(頂点の数が制限以上)

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, attr, modeが不正、poly.size 2)

**gfil\_pol**

Fill Polygon

多角形の塗り潰し

**【形式】**

ERR gfil\_pol ( GID gid, POLY \*p, PAT \*pat, DCM mode )

**【解説】**

gid で指定した描画環境で、p で指定した多角形を、pat で指定したパターン、mode で指定した描画モードで塗り潰す。

**【リターン値】**

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

**【エラーコード】**

EG\_ADR

指定されたアドレス(p, pat) のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。(頂点の数が制限以上)

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, modeが不正、poly.size 2)

## 2.4.7 任意図形の描画

**gfil\_gen**

Fill Generic Region

任意領域の塗り潰し

**【形式】**

ERR gfil\_gen ( GID gid, GRGN \*rgn, PAT \*pat, DCM mode )

**【解説】**

gid で指定した描画環境で、rgn で指定した任意領域を、pat で指定したパターン、mode で指定した描画モードで塗り潰す。

## 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(rgn, pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。(rgn の定義が制限を越えた)

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容,rgnの内容,mode が不正)

# gfil\_rgn

Fill Region

任意閉領域の塗り潰し

## 【形式】

Bool gfil\_rgn ( GID gid, PIXVAL val, PNT p, PAT \*pat, DCM mode )

## 【解説】

gid で指定した描画環境で、p で指定した点を含む閉領域を、pat で指定したパターン、mode で指定した描画モードで塗り潰す。

塗り潰す閉領域は、ビットマップ境界(bounds)内で、val の値により以下のように決められる。

val 0 の時 :

val で指定した値と異なるピクセル値を持つピクセルの、点 p を含む閉領域を塗り潰す。即ち、val で指定したピクセル値を境界点のピクセル値とする。

val < 0 の時 :

p で指定した点のピクセル値と同じピクセル値を持つピクセルの、点 p を含む閉領域を塗り潰す。即ち、点 p のピクセル値以外のすべてを境界点のピクセル値とする。

指定した点 p がクリッピングにより、描画されない位置にある場合は、塗り潰しは一切行なわれず関数値として "1" が戻り、塗り潰しが行なわれた場合は関数値 "0" が戻る。

## 【リターン値】

0 : 正常終了(関数値は描画状態(0:行なわれた、1:行なわれなかった))  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容,modeが不正)

## 2.4.8 曲線の描画

### gdra\_spl

Draw Spline

スプライン曲線の描画

#### 【形式】

ERR gdra\_spl ( GID gid, UW np, PNT \*pt, LATTR attr, PAT \*pat, DCM mode )

#### 【解説】

gid で指定した描画環境で、np 個の pt で指定した点による 3 次 B - スプライン曲線を描画する。attr で指定した属性、pat で指定したパターン、mode で指定した描画モードが使用される。

pt は np 個の要素から構成される点の配列へのポインタである。pt[0] != pt [np-1] の場合は開曲線の描画であり、pt[0]==pt[np-1] の場合は、pt[0] ~ pt [np-2] の点を結ぶ閉曲線の描画となる。

np 3(閉曲線の場合は4)の場合に曲線が描画され、それ以下の場合には、点または直線が描画される。

B-スプライン曲線であるため、描画される曲線は一般に指定した点を通らない。但し、開曲線の場合は、曲線の両端は指定した端点(pt[0] と pt[np-1]) に一致するように描画される。

#### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(pt,pat) のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。(np が制限を越えた)

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容, attr, modeが不正, np=0)

### gfil\_spl

Fill Spline

閉スプライン曲線領域の塗り潰し

#### 【形式】

ERR gfil\_spl ( GID gid, UW np, PNT \*pt, PAT \*pat, DCM mode )

## 【解説】

gid で指定した描画環境で、np 個の pt で指定した点による3次の閉じたB-スプライン曲線で囲まれた領域を、pat で指定したパターン、mode で指定した描画モードで塗り潰す。

pt は np個の要素から構成される点の配列へのポインタであり、pt[0] ~ pt[np-1] の点を結ぶ閉曲線の描画となる。

np 3の場合に曲線が描画され、それ以下の場合は、点または直線が描画される。

B-スプライン曲線であるため、描画される曲線は一般に指定した点を通らない。但し曲線の両端は指定した端点(pt[0]) に一致するように描画される。

## 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(pt,pat) のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。(np が制限を越えた)

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容,modeが不正,np=0)

## 2.4.9 ビットマップの操作

ビットマップの操作関数としては、以下の図に示すものが用意されている。

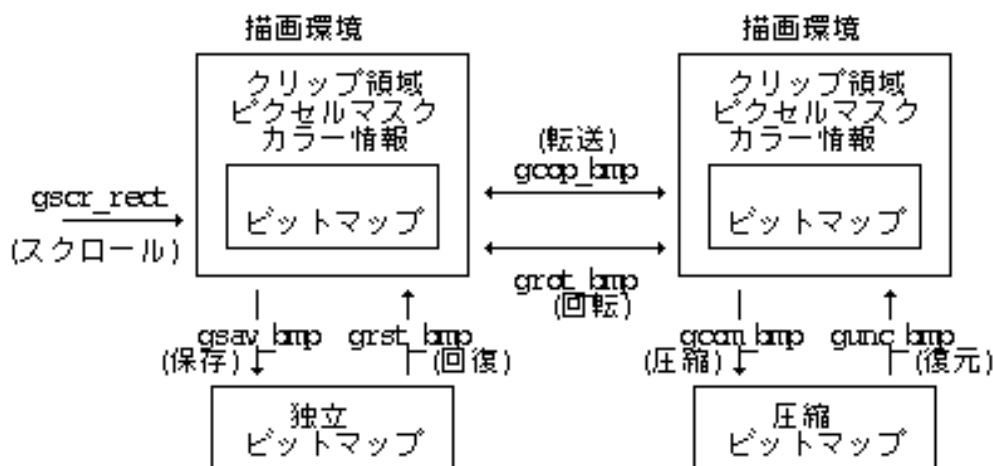


図 40 : ビットマップの操作関数

grop\_bmp(), gsav\_bmp(), grst\_bmp(), grst\_bmp() では、指定した描画モードに従って、データ転送を行なうが、以下の変換指定を演算モードの追加パラメータとして指定することができる。

G\_CVFORM : ビットマップ形式変換

- ソースとディスティネーションのビットマップの形式(プレーン数、ピクセルビット数)が異なる場合は変換を行なう。

- この指定が無い場合、ソースとディスティネーションのビットマップの形式(プレーン数、ピクセルビット数)が異なる場合は EG\_FORM のエラーとなる。
- 変換は、変換後の双方の下位Nビットのピクセル値が等しくなるように行なわれる(ここで、Nはソースとディスティネーションの depth の最小値を意味する)。

G\_CVCOLOR : カラー変換

- ソースとディスティネーションのカラー表現形式が異なる場合は、変換を行なう。どちらか一方のカラー表現方式が未定義の場合は変換は行なわれない。
- カラー変換はソースのピクセル値を絶対カラーに変換し、その絶対カラーに最も近い絶対カラーに対応するディスティネーションのピクセル値を求めることにより行なわれる。
- ディスティネーションのカラー表現形式がカラーマップ形式の場合は、その時点のカラーマップの内容を使用し、カラーマップの内容の自動変更は行なわない。

## gcop\_bmp

Copy Bitmap

ビットマップ間のデータ転送

### 【形式】

ERR gcop\_bmp ( GID srcid, RECT \*sr, GID dstid, RECT \*dr, GRGN \*mask, DCM mode )

### 【解説】

srcid で指定した描画環境のビットマップ中の sr で指定した領域を、dstid で指定した描画環境のビットマップ中の dr で指定した領域に mode で指定した演算により転送を行なう。dstid = srcid としても構わない。

dr と sr の大きさが異なる場合は、拡大 / 縮小が行なわれ、dr = NULL の場合は、dr = sr と見なされる。

dstid の描画環境で定義されているクリップ領域が適用され、さらに mask が NULL でない場合は、mask で定義された領域に含まれる部分のみが転送される。mask は、dstid の座標系で定義される。

srcid の描画環境のクリップ領域は無視され、sr とビットマップの bounds の共通部分が転送される。

mode は、描画モード指定に以下の変換指定を追加したものとなる。

G\_CVFORM

ビットマップ形式の変換

G\_CVCOLOR

カラー変換

### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(sr, dr,mask)のメモリのアクセスは許されていない。

EG\_FORM

ビットマップの形式が異なる。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。

EG\_NOSPC

システムのメモリが不足した。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。(変換をサポートしていないカラー形式/ビットマップ形式である。)

EG\_PAR

パラメータが不正、または範囲外である。(maskの内容,mode が不正, sr,drが空)

## gsav\_bmp

Save Bitmap

ビットマップの保存

### 【形式】

ERR gsav\_bmp ( GID gid, RECT \*gr, BMP \*bmap, RECT \*br, GRGN \*mask, DCM mode )

### 【解説】

gid で指定した描画環境のビットマップ中の gr で指定した領域を、bmap で指定した独立したビットマップ中の br で指定した領域に mode で指定した演算により転送を行なう。

gr と br の大きさが異なる場合は、拡大/縮小が行なわれ、br = NULL の場合は、br = gr と見なされる。

mask が NULL でない場合は、mask で定義された領域に含まれる部分のみが転送される。mask は、bmap の座標系で定義される。

gid の描画環境のクリップ領域は無視され、gr とビットマップの bounds の共通部分が転送される。

bmap の baseaddr が NULL の場合は、そのプレーンは転送されない。

mode は、描画モード指定に以下の変換指定を追加したものとなる。

G\_CVFORM

ビットマップ形式の変換

G\_CVCOLOR

カラー変換

カラー変換に関しては、描画環境に色変換用カラー情報が設定されている場合のみ有効で、色変換用カラー情報を転送先ビットマップのカラー情報として扱う。

色変換用カラー情報と転送先ビットマップ形式に矛盾がある場合、EG\_ENV のエラーを戻す。

この関数は、gcop\_bmp() のディスティネーションを独立したビットマップとしたものである。

### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(gr,bmap,br,mask)のメモリのアクセスは許されていない。

EG\_ENV

色変換用カラー情報とビットマップの形式に矛盾がある。(G\_CVCOLOR 指定時のみ)

EG\_FORM

ビットマップの形式が異なる。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。

EG\_NOSPC

システムのメモリが不足した。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。(変換をサポートしていないビットマップ形式である。)

EG\_PAR

パラメータが不正、または範囲外である。(bmapの内容, maskの内容, modeが不正, gr, brが空)

## grst\_bmp

Restore Bitmap

### ビットマップの回復

#### 【形式】

ERR grst\_bmp ( GID gid, RECT \*gr, BMP \*bmap, RECT \*br, GRGN \*mask, DCM mode )

#### 【解説】

bmap で指定した独立したビットマップ中の br で指定した領域を、gid で指定した描画環境のビットマップ中の gr で指定した領域に mode で指定した演算により転送を行なう。

br と gr の大きさが異なる場合は、拡大 / 縮小が行なわれ、gr = NULL の場合は、gr = br と見なされる。

gid の描画環境で定義されているクリップ領域が適用され、さらに mask が NULL でない場合は、mask で定義された領域に含まれる部分のみが転送される。mask は、gid の座標系で定義される。

br と bmap のビットマップの bounds の共通部分が転送され、bmap->baseaddr が NULL の場合は、そのプレーンは転送されない。

mode は、描画モード指定に以下の変換指定を追加したものとなる。

G\_CVFORM

ビットマップ形式の変換

G\_CVCOLOR

カラー変換

カラー変換に関しては、描画環境に色変換用カラー情報が設定されている場合のみ有効で、色変換用カラー情報を転送元ビットマップのカラー情報として扱う。

色変換用カラー情報と転送先ビットマップ形式に矛盾がある場合、EG\_ENV のエラーを戻



す。

この関数は、`gcop_bmp()` のソースを独立したビットマップとしたものである。

#### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(`gr,bmap,br,mask`)のメモリのアクセスは許されていない。

EG\_ENV

色変換用カラー情報とビットマップの形式に矛盾がある。(G\_CVCOLOR 指定時のみ)

EG\_FORM

ビットマップの形式が異なる。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。

EG\_NOSPC

システムのメモリが不足した。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。(変換をサポートしていないビットマップ形式である。)

EG\_PAR

パラメータが不正、または範囲外である。(bmapの内容, maskの内容, modeが不正, gr,brが空)

## grst\_mbm

Restore Masked Bitmap

ビットマップの回復

#### 【形式】

ERR `grst_mbm` ( GID `gid`, RECT \*`gr`, BMP \*`bmap`, RECT \*`br`, UB \*`mask`, DCM `mode` )

#### 【解説】

`bmap` で指定した独立したビットマップ中の `br` で指定した領域を、`gid` で指定した描画環境のビットマップ中の `gr` で指定した領域に `mode` で指定した演算により転送を行なう。

`br` と `gr` の大きさが異なる場合は、拡大 / 縮小が行なわれ、`gr = NULL` の場合は、`gr = br` と見なされる。

`gid` の描画環境で定義されているクリップ領域が適用され、さらに `mask` が `NULL` でない場合は、`mask` で定義された領域に含まれる部分のみが転送される。

`mask` は、`br` で示された領域に対応する `pixbits=0x0101` のビットマップであり、マスクイメージの1の部分の部分が実際に転送されることになる。以下のサイズの領域のビットマップイメージが設定されていなければならない。

$$(((br.c.right - vr.c.left)+15)/16) * 2 * (br.c.bottom - br.c.top)$$

br と bmap のビットマップの bounds の共通部分が転送され、 bmap->baseaddr が NULL の場合は、そのプレーンは転送されない。

mode は、描画モード指定に以下の変換指定を追加したものとなる。

G\_CVFORM

ビットマップ形式の変換

G\_CVCOLOR

カラー変換

カラー変換に関しては、描画環境に色変換用カラー情報が設定されている場合のみ有効で、色変換用カラー情報を転送元ビットマップのカラー情報として扱う。色変換用カラー情報と転送先ビットマップ形式に矛盾がある場合、EG\_ENV のエラーを戻す。

この関数は、grst\_bmp() のマスクをビットマップ表現にしたものである。

#### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(gr,bmap,br,mask)のメモリのアクセスは許されていない。

EG\_ENV

色変換用カラー情報とビットマップの形式に矛盾がある。(G\_CVCOLOR 指定時のみ)

EG\_FORM

ビットマップの形式が異なる。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_LIMIT

システムの制限を越えた。

EG\_NOSPC

システムのメモリが不足した。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。(変換をサポートしていないビットマップ形式である。)

EG\_PAR

パラメータが不正、または範囲外である。(bmapの内容, maskの内容, modeが不正, gr,br が空)

## grot\_bmp

Rotate Bitmap

ビットマップの回転

#### 【形式】

ERR grot\_bmp ( GID srcid, GID dstid, RECT \*rp, PNT \*dp, DEG angle, GRGN \*mask, DCM mode )

#### 【解説】

srcid で指定した描画環境の rp で指定した長方形の内容を回転し、dstid で指定した描画環境の dp で指定した点を左上の点とする長方形領域に mode で指定した演算により転送す

る。dp = NULL の場合は、rp の左上の点と同じとみなされる。

srcid と dstid のビットマップの構造が同一でない場合は EG\_FORM のエラーとなる。なお、srcid での rp の領域と dstid での転送先領域のビットマップは重なっても良い。

dstid の描画環境で定義されているクリップ領域が適用され、さらに mask が NULL でない場合は、mask で定義された領域に含まれる部分のみが転送される。mask は、dstid の座標系で定義される。srcid の描画環境のクリップ領域は適用されない。

angle により反時計回りの回転角度(0 ~)を指定する。実際は360の剰余の値が有効となり、0、90、180、270 以外の回転角度のサポートはインプリメントに依存する。

#### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(rp,dp,mask) のメモリのアクセスは許されていない。

EG\_FORM

ビットマップの形式が異なる。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPC

システムのメモリが不足した。

EG\_NOSPT

現在のインプリメントではサポートしていない。(angle)

EG\_PAR

パラメータが不正、または範囲外である。(maskの内容,modeが不正, angle < 0, rpが空)

## gscr\_rec

Scroll Rectangle

長方形領域のスクロール

#### 【形式】

W gscr\_rec ( GID gid, RECT r, W dh, W dv RLIST \*rlp, PAT \*pat )

#### 【解説】

gid で指定した描画環境で、r で指定した領域を dh, dv だけスクロール移動し、発生した隙間の部分を指定したパターンで塗りつぶす。前置長方形リストを含めたすべてのクリッピングが適用される。前置長方形リストによりオーバーラップされていた部分がでてきた場合は、その部分も指定したパターンで塗りつぶされる。

関数値として、パターンで塗りつぶされた部分の矩形領域の数が戻り、rlp が NULL でない場合は、その矩形領域のリストが設定される。rlpの要素が不足する場合は、rlpの要素分のみ設定される。

rlpの各要素は、水平線分解された矩形であり、垂直座標値の小さい順、同一垂直座標値の場合は水平座標値の小さい順に並べられる。

pat=NULLの場合は塗りつぶしは行わないが、矩形リストはrlpに設定される。

#### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(rlp,pat)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。(patの内容が不正、rが空)

## gsiz\_cbm

Get Compression Size

圧縮ビットマップのサイズ計算

#### 【形式】

ERR gsiz\_cbm ( GID gid, RECT \*rp, W compac, W \*size )

#### 【解説】

gid で指定した描画環境内の rp で指定した長方形領域の内容を、compac で指定した圧縮方法で圧縮した場合の各プレーン毎のデータのバイトサイズを size で指定した領域に格納する。size は、プレーン数個の W 配列へのポインタであり、size[0] にプレーン# 0の圧縮後データのバイトサイズ、size[1] にプレーン#1の圧縮後データのバイトサイズが戻されることになる。

#### 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(rp, size)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。

EG\_PAR

パラメータが不正、または範囲外である。(compacが不正, rpが空)

## gcom\_bmp

Compress Bitmap

ビットマップの圧縮

#### 【形式】

Bool gcom\_bmp ( GID gid, RECT \*rp, CBMP \*cbmp, W \*stat )

#### 【解説】

gid で指定した描画環境の rp で指定した長方形領域のイメージを圧縮して、cbmp で指定した圧縮ビットマップとする。拡大 / 縮小は行なわれない。

cbmp->compac には圧縮方法を、cbmp->bmp.baseaddr[] には圧縮イメージの格納領域の先頭アドレスをあらかじめ設定しておかなくてはならない。格納領域の先頭には、その領域の全体のバイトサイズを W size として設定しておく必要がある。cbmp の他の要素は、自動的に以下に示す値に設定されるので、特に設定しておく必要はない。

cbmp->compac

実行前に圧縮方法を設定しておく。

bmap.planes

実行後に gid と同じ値が設定される。

bmap.pixbits

実行後に gid と同じ値が設定される。

bmap.rowbytes

実行後に r を含む最小の値が設定される。

bmap.bounds

実行後に r と同じ値が設定される。

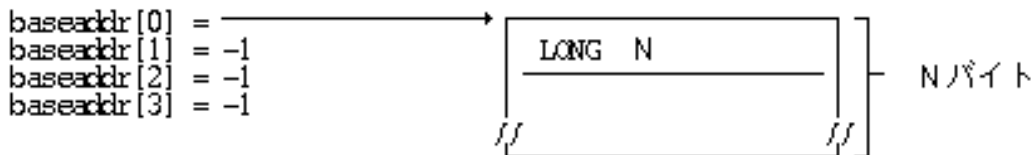
bmap.baseaddr[]

実行前に格納アドレスを設定しておく。

描画環境の描画ピクセルマスクは適用されず、描画環境のビットマップの全プレーンがデータ圧縮の対象となるが、cbmp->bmp.baseaddr[] が NULL の場合は、そのプレーンは対象とならない。

プレーンの圧縮イメージが、設定バイトサイズ以内の場合は、実際に圧縮したイメージのバイトサイズが、イメージ領域の先頭に入り、stat の対応するプレーンのビット(LSB がプレーン#0に対応する)は0にセットされる。プレーンの圧縮イメージが、設定バイトサイズを越える場合は、stat の対応するプレーンのビットは1にセットされる。この場合イメージ領域の内容は保証されないが設定したサイズを越えて書かれることはない。

cbmp->bmp.baseaddr[] = -1の場合は、そのビットマップ領域は、直前のプレーンの領域の直後を取ることを意味し、実行後に自動的にそのアドレスが設定され、対応する stat のビットは0にセットされる。領域が不足した場合は、実行後も - 1のままとなり、対応する stat のビットは1にセットされる。



実行後 (プレーン3の領域が不足したケース)

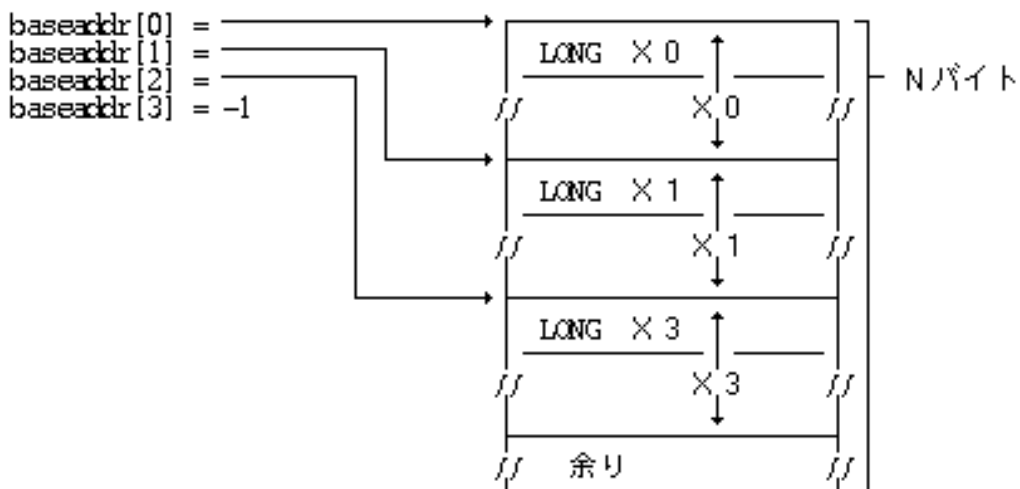


図 41 : ビットマップの圧縮

存在しないプレーン、または `cbmp->bmp.baseaddr[] = NULL` のプレーンに対応する `stat` のビットは常に0にセットされる。

すべてのプレーンが正しく圧縮された場合、即ち `stat=0` の場合は関数値として"0"が戻り、1つ以上のプレーンの領域が不足した場合、即ち、`stat` 0の場合は関数値として"1"が戻る。

なお、`gid` で指定した描画環境のビットマップの領域と、`cbmp->baseaddr[]` で指定した領域が重なる場合の動作は保証されない。

#### 【リターン値】

0 : 正常終了(関数値は(0:正常、1:領域不足))

<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(`rp,cbmp,stat`) のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。

EG\_PAR

パラメータが不正、または範囲外である。( `rp` が空、`cbmp->compac` が不正)

## gunc\_bmp

Uncompress Bitmap

圧縮ビットマップの復元

#### 【形式】

ERR gunc\_bmp ( GID gid, RECT \*dr, CBMP \*cbmp, RECT \*sr, DCM mode )

## 【解説】

cbmp で指定した圧縮ビットマップの sr で指定した長方形領域の内容を、gid で指定した描画環境の dr で指定した長方形領域に復元しながら mode で指定した演算により転送を行なう。

sr と dr の大きさが異なる場合は、拡大 / 縮小が行なわれ、dr = NULL の場合は、dr = sr と見なされる。

gid の描画環境で定義されているクリップ領域、描画ピクセルマスクは有効となる。

描画環境のビットマップと、圧縮ビットマップの形式は、同一でなくてはならない。また、cbmp->baseaddr[] が NULL の場合は、そのプレーンに対してのデータ復元は行なわれない。

なお、gid で指定した描画環境のビットマップの領域と cbmp->baseaddr[] で指定した領域が重なる場合の動作は保証されない。

## 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(dr, cbmp,sr)のメモリのアクセスは許されていない。

EG\_FORM

ビットマップの形式が異なる。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPT

現在のインプリメントではサポートしていない、もしくは予約されているパラメータ値である。

EG\_PAR

パラメータが不正、または範囲外である。(cbmpの内容が不正, sr,drが空)

## 2.4.10 ピッキング処理

ピッキング処理関数は、指定した点が、指定した図形の描画領域に含まれるか否かをチェックするための関数であり、個々の図形に対応した関数が用意される。

ピッキング処理関数では、基本的には、同一のパラメータで実際に図形描画を行なった場合に、指定した点が含まれているか否かを厳密にチェックするものであるが、描画環境の対象デバイスや、インプリメントによっては、実際に描画した場合と僅かに結果が異なる場合もある。

この関数群は描画環境とは無関係の独立した関数である。

図形のパラメータが不正で描画できないような図形の場合は、結果として含まれていないことを示す値"0"を戻し、エラーは発生しない。

gpick\_lin

Pick Line

直線のピッキング

## 【形式】

Bool gpic\_lin ( PNT pt, PNT p0, PNT p1, UW width )

## 【解説】

pt で指定した点が、p0 から p1 へ太さ width の直線に含まれているか否かをチェックし、含まれている場合は関数値 "1" を戻し、含まれていない場合は関数値 "0" を戻す。

## 【リターン値】

=1 : 正常終了(含まれている)  
=0 : 正常終了(含まれていない)  
<0 : エラー (発生しない)

## 【エラーコード】

発生しない。

# gpic\_rec

Pick Rectangle

長方形のピックアップ

## 【形式】

Bool gpic\_rec ( PNT pt, RECT r, DEG angle, UW width )

## 【解説】

pt で指定した点が、r で指定した、線幅 width、回転角度 angle の長方形に含まれているか否かをチェックし、含まれている場合は関数値 "1" を戻し、含まれていない場合は関数値 "0" を戻す。

width が大きすぎて長方形の枠をはみ出るような場合、または width=0 の場合は、長方形の内部に含まれているか否かをチェックする。

## 【リターン値】

=1 : 正常終了(含まれている)  
=0 : 正常終了(含まれていない)  
<0 : エラー (発生しない)

## 【エラーコード】

発生しない。

# gpic\_rrc

Pick Round Rectangle

角丸長方形のピックアップ

## 【形式】

Bool gpic\_rrc ( PNT pt, RECT r, UW rh, UW rv, DEG angle, UW width )

## 【解説】

pt で指定した点が、r および rv, rh で指定した、線幅 width、回転角度 angle の角丸長方形に含まれているか否かをチェックし、含まれている場合は関数値 "1" を戻し、含まれていない場合は関数値 "0" を戻す。

width が大きすぎて角丸長方形の枠をはみ出るような場合、または width=0 の場合は、角丸長方形の内部に含まれているか否かをチェックする。

## 【リターン値】

=1 : 正常終了(含まれている)  
=0 : 正常終了(含まれていない)



<0:エラー (発生しない)

#### 【エラーコード】

発生しない。

## gpic\_ovl

Pick Oval

楕円(円)のピックアップ

#### 【形式】

Bool gpic\_ovl ( PNT pt, RECT r, DEG angle, UW width )

#### 【解説】

pt で指定した点が、r で指定した、線幅 width、回転角度 angle の楕円(円)に含まれているか否かをチェックし、含まれている場合は関数値 "1" を戻し、含まれていない場合は関数値 "0" を戻す。

width が大きすぎて楕円(円)の枠をはみ出るような場合、または width =0 の場合は、楕円(円)の内部に含まれているか否かをチェックする。

#### 【リターン値】

=1:正常終了(含まれている)

=0:正常終了(含まれていない)

<0:エラー (発生しない)

#### 【エラーコード】

発生しない。

## gpic\_arc

Pick Arc

弧のピックアップ

#### 【形式】

Bool gpic\_arc ( PNT pt, RECT r, PNT sp, PNT ep, DEG angle, UW width )

#### 【解説】

pt で指定した点が、r および sp, ep で指定した、線幅 width、回転角度 angle の弧に含まれているか否かをチェックし、含まれている場合は関数値 "1" を戻し、含まれていない場合は関数値 "0" を戻す。

#### 【リターン値】

=1:正常終了(含まれている)

=0:正常終了(含まれていない)

<0:エラー (発生しない)

#### 【エラーコード】

発生しない。

## gpic\_sec

Pick Sector

扇形のピックアップ

#### 【形式】

Bool gpic\_sec ( PNT pt, RECT r, PNT sp, PNT ep, DEG angle, UW width )

## 【解説】

pt で指定した点が、r および sp, ep で指定した、線幅 width、回転角度 angle の扇形に含まれているか否かをチェックし、含まれている場合は関数値 "1" を戻し、含まれていない場合は関数値 "0" を戻す。

width=0の場合は、扇形の内部に含まれているか否かをチェックする。

## 【リターン値】

=1 : 正常終了(含まれている)  
=0 : 正常終了(含まれていない)  
<0 : エラー (発生しない)

## 【エラーコード】

発生しない。

# gpic\_cho

Pick Chord

弓形のピックアップ

## 【形式】

Bool gpic\_cho ( PNT pt, RECT r, PNT sp, PNT ep, DEG angle, UW width )

## 【解説】

pt で指定した点が、r および sp, ep で指定した、線幅 width、回転角度 angle の弓形に含まれているか否かをチェックし、含まれている場合は関数値 " 1 " を戻し、含まれていない場合は関数値 "0" を戻す。

width=0の場合は、弓形の内部に含まれているか否かをチェックする。

## 【リターン値】

=1 : 正常終了(含まれている)  
=0 : 正常終了(含まれていない)  
<0 : エラー (発生しない)

## 【エラーコード】

発生しない。

# gpic\_pln

Pick Polyline

直線列のピックアップ

## 【形式】

Bool gpic\_pln ( PNT pt, POLY \*poly, UW width )

## 【解説】

pt で指定した点が、\*poly で示された開いた多角形、即ち直線列に含まれているか否かをチェックし、含まれている場合は関数値 "1" を戻し、含まれていない場合は関数値 "0" を戻す。

## 【リターン値】

=1 : 正常終了(含まれている)  
=0 : 正常終了(含まれていない)  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(poly)のメモリのアクセスは許されていない。

## gpic\_pol

Pick Polygon

多角形のピックアップ

### 【形式】

Bool gpic\_pol ( PNT pt, POLY \*poly, UW width )

### 【解説】

ptで指定した点が、\*polyで示された多角形に含まれているか否かをチェックし、含まれている場合は関数値 "1" を戻し、含まれていない場合は関数値 "0" を戻す。

width =0の場合は、多角形の内部に含まれているか否かをチェックする。

### 【リターン値】

=1 :正常終了(含まれている)

=0 :正常終了(含まれていない)

<0 :エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(poly)のメモリのアクセスは許されていない。

## gpic\_gen

Pick Generic Region

任意図形のピックアップ

### 【形式】

Bool gpic\_gen ( PNT pt, GRGN \*rgn )

### 【解説】

ptで指定した点が、\*rgnで示された任意図形に含まれているか否かをチェックし、含まれている場合は関数値 "1" を戻し、含まれていない場合は関数値 "0" を戻す。

### 【リターン値】

=1 :正常終了(含まれている)

=0 :正常終了(含まれていない)

<0 :エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(rgn)のメモリのアクセスは許されていない。

## gpic\_spl

Pick Spline

スプライン曲線のピックアップ

### 【形式】

Bool gpic\_spl ( PNT p, UW np, PNT \*pt, UW width )

### 【解説】

pt で指定した点が、np, pt で示された太さ width の gdra\_spl() で描画した曲線に含まれているか否かをチェックし、含まれている場合は関数値 "1" を戻し、含まれていない場合は関数値 "0" を戻す。

width=0 で、閉じた曲線の場合は、曲線の内部に含まれているか否かをチェックする。

#### 【リターン値】

=1 : 正常終了(含まれている)

=0 : 正常終了(含まれていない)

<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(pt)のメモリのアクセスは許されていない。

---

[この章の目次にもどる](#)

[前頁:2.3 基本関数にもどる](#)

[次頁:2.5 文字\(列\)描画関数にすすむ](#)

[この章の目次にもどる](#)

[前頁:2.4 図形描画関数にもどる](#)

[次頁:2.6 ポインタ制御関数にすすむ](#)

## 2.5 文字(列)描画関数

### 2.5.1 描画環境レコードアクセス

#### gset\_chp

Set Character Position

文字描画位置の設定

##### 【形式】

ERR gset\_chp ( GID gid, W x, W y, Bool abs )

##### 【解説】

gid で指定した描画環境の文字描画位置の値を abs, x, y で指定した値に従って以下のように変更する。

abs =0 (相対指定)

(現在の水平位置) + x を新しい水平位置、(現在の垂直位置) + y を新しい垂直位置とする。

abs !=0 (絶対指定)

x を新しい水平位置、y を、新しい垂直位置とする。

##### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

##### 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(座標値がオーバーフローした(abs = 0の時))

#### gget\_chp

Get Character Position

文字描画位置の取り出し

##### 【形式】

ERR gget\_chp ( GID gid, W \*x, W \*y )

##### 【解説】

gid で指定した描画環境の文字描画位置を取り出し、水平位置を x、垂直位置 を y で指定した領域に格納する。

##### 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(x,y)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gset\_chc

Set Character Color

文字描画カラーの設定

## 【形式】

ERR gset\_chc ( GID gid, COLOR fgc, COLOR bgc )

## 【解説】

gid で指定した描画環境の文字描画の前景色を fgc で指定した値に、背景色を bgc で指定した値に変更する。

## 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gget\_chc

Get Character Color

文字描画カラーの取り出し

## 【形式】

ERR gget\_chc ( GID gid, COLOR \*fgc, COLOR \*bgc )

## 【解説】

gid で指定した描画環境の文字描画の色を取り出し、前景色を fgc で指定した領域に、背景色を bgc で指定した領域に格納する。

## 【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(fgc,bgc)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gset\_chd

Set Character Direction

文字描画方向 / 間隔の設定

## 【形式】

ERR gset\_chd ( GID gid, W dir, CGAP \*gap )

【解説】

gid で指定した描画環境の文字描画方向 を dir で指定した値に、文字間隔を \*gap で指定した値にそれぞれ変更する。

dir<0 の場合は、文字描画方向は変更しないことを意味し、gap = NULL の場合は、文字間隔は変更しないことを意味する。

【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR

指定されたアドレス(gap)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(dir,gapの内容が不正)

## gget\_chd

Get Character Direction

文字描画方向 / 間隔の取り出し

【形式】

W gget\_chd ( GID gid, CGAP OUT \*gap )

【解説】

gid で指定した描画環境の文字描画方向を関数値として戻し、文字間隔を gap で指定した領域に領域に格納する。gap = NULL の場合は、文字描画方向のみを関数値として取り出す。

【リターン値】

0 : 正常終了 (関数値は文字描画方向)

<0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR

指定されたアドレス(gap)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gset\_fon

Set Font

文字フォントの指定

【形式】

ERR gset\_fon ( GID gid, FSSPEC \*fnt )

【解説】

gid で指定した描画環境のフォント指定を fnt で指定したフォントに設定する。

fnt は FSSPEC 構造体へのポインタである。

## 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(fnt)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_NOSPC

システムのメモリが不足した。

EG\_PAR

パラメータが不正、または範囲外である。(文字サイズが不正)

# gget\_fon

Get Font

文字フォント情報の取り出し

## 【形式】

ERR gget\_fon ( GID gid, FSSPEC \*fnt, FNTINFO \*inf )

## 【解説】

gid で指定した描画環境のフォント指定、およびフォント情報を取り出し、それぞれ、 fnt および inf で指定した領域に格納する。 fnt = NULL または inf = NULL の場合は、対応する情報は格納されない。

\*fnt には、 gset\_fon() で設定したデータそのものが格納され、 \*inf にはフォント情報が格納される。この情報は全てピクセル数で表わされ、指定した文字サイズにスケールされた値である。即ち、 inf->height は、 fnt->size.v に常に等しくなる。

## 【リターン値】

=0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_ADR

指定されたアドレス(fnt,inf)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

# gset\_scr

Set SCRIPT

文字セット(SCRIP)の指定

## 【形式】

ERR gset\_scr ( GID gid, SCRIPT script )

## 【解説】

gid で指定した描画環境の文字セット(SCRIP)を script に設定する。

script は言語指定コードを含まない文字コードのデフォルトの文字セットとして使用され



る。

描画環境の文字セットに関しては、本システムコールの他に、文字(列)描画によっても、自動的に変更される可能性がある。

【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_PAR

script の内容が不正。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gget\_scr

Get SCRIPT

文字セット(SCRIP)の取り出し

【形式】

ERR gget\_scr ( GID gid, SCRIPT \*script)

【解説】

gid で指定した描画環境の文字セット(SCRIP)を取り出し、\*script に設定する。

【リターン値】

=0(E\_OK) : 正常終了

<0 : エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR

指定されたアドレス(script)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

## 2.5.2 文字(列)チェック関数

## gget\_chw

Get Character Width

文字幅の取り出し

【形式】

WERR gget\_chw ( GID gid, TC ch )

【解説】

gid で指定した描画環境で ch で指定した文字を描画した場合の、文字幅(ピクセル数)を関数値として戻す。

ch が言語指定コードであった場合、常に0を戻す。

【リターン値】

0 : 正常終了 (関数値は文字幅)

<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gget\_chh

Get Character Height

文字高さの取り出し

## 【形式】

WERR gget\_chh ( GID gid, TC ch )

## 【解説】

gid で指定した描画環境で ch で指定した文字を描画した場合の、文字高さ(ピクセル数)を関数値として戻す。

ch が言語コードであった場合、常に0を戻す。

## 【リターン値】

0 : 正常終了(関数値は文字高さ)  
<0 : エラー (関数値はエラーコード)

## 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

## gget\_stw

Get String Width

文字列幅の取り出し

## 【形式】

WERR gget\_stw ( GID gid, TC \*str, W len, CGAP \*gap, W \*pos )

## 【解説】

gid で指定した描画環境で str で指定した文字列の先頭から len 文字数を gap で指定した文字間隔で描画した場合の、文字列幅(ピクセル数)を関数値として戻す。gap = NULL の場合は、描画環境に設定されている文字間隔が適用される。

len には言語指定コードも含まれ、言語指定コード2バイトを1文字とみなす。

文字列幅とは、指定した文字列を描画した場合の全ての文字枠を囲む最小の長方形の幅であり、設定されている水平文字間隔を含んだ値となる。

str で指定した文字列が len で指定した値より短い場合は、そこまでの文字列幅が戻される。

pos は文字列中の各文字の描画位置の水平座標値を格納するための、len の大きさを持つ W 配列へのポインタである。先頭文字に対応する座標値は、現在設定されている文字描画位置の水平座標値となる。pos = NULL の場合は、各文字の描画位置は格納されない。

文字列中に言語指定コードが含まれる場合、言語指定コードに対応した pos[N] にはその時点での文字描画位置が設定される。言語指定コードによる文字描画位置の更新は行われず、pos[N]、pos[N+1] の値は等しくなる。

縦書きの場合は、この関数は単に文字幅を戻すことになり、pos にはすべて同一の値が設定されることになる。

この関数により文字描画位置、言語指定は変化しない。

#### 【リターン値】

- 0: 正常終了 (関数値は文字列幅)
- <0: エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(str,gap,pos)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(gapの内容が不正)

## gget\_sth

Get String Height

文字列高さの取り出し

#### 【形式】

WERR gget\_sth ( GID gid, TC \*str, W len, CGAP \*gap, W \*pos )

#### 【解説】

gid で指定した描画環境で str で指定した文字列の先頭から len 文字数を gap で指定した文字間隔で描画した場合の、文字列高さ(ピクセル数)を関数値として戻す。gap = NULL の場合は、描画環境に設定されている文字間隔が適用される。

len には言語指定コードも含まれ、言語指定コード2バイトを1文字とみなす。

文字列高さとは、指定した文字列を描画した場合の全ての文字枠を囲む最小の長方形の高さであり、設定されている垂直文字間隔を含んだ値となる。

str で指定した文字列が len で指定した値より短い場合は、そこまでの文字列高さが戻される。

pos は文字列中の各文字の描画位置の垂直座標値を格納するための、len の大きさを持つ W 配列へのポインタである。先頭文字に対応する座標値は、現在設定されている文字描画位置の垂直座標値となる。pos = NULL の場合は、各文字の描画位置は格納されない。

文字列中に言語指定コードが含まれる場合、言語指定コードに対応した pos[N] にはその時点での文字描画位置が設定される。言語指定コードによる文字描画位置の更新は行われず、pos[N]、pos[N+1] の値は等しくなる。

横書きの場合は、この関数は単に文字高さを戻すことになり、pos にはすべて同一の値が設定されることになる。

この関数により文字描画位置、言語指定は変化しない。

#### 【リターン値】

- 0: 正常終了(関数値は文字列高さ)
- <0: エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_ADR

指定されたアドレス(str,gap,pos)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(gapの内容が不正)

## gget\_str

Get String Rectangle

文字列描画領域の取り出し

### 【形式】

WERR gget\_str ( GID gid, TC \*str, W len, CGAP \*gap, PNT \*chp, RECT \*cr, RECT \*dr, W \*alen)

### 【解説】

gid で指定した描画環境で str で指定した文字列の先頭から len 文字数を gap で指定した文字間隔で描画した場合の文字描画位置、文字描画領域の情報を引数で指定した領域に設定する。gap = NULL の場合は、描画環境に設定されている文字間隔が適用される。

len には言語指定コードも含まれ、言語指定コード2バイトを1文字とみなす。

\*dr が NULL でない場合、対象描画環境の相対座標系で表現される、文字列描画領域が設定される。

文字描画領域とは、指定した文字列を描画した場合の全ての文字枠、文字枠を超えたカーニング、斜体描画を含む最小の矩形であり、設定されている文字間隔を含んだ値となる。

str で指定した文字列が len で指定した値より短い場合は、そこまでの文字列幅が戻される。

str で指定した文字列中に言語指定コードがあった場合、フォントを切り替えも考慮した描画領域情報の取得が可能である。

alen が NULL でない場合、処理した要素(文字)数(TCの個数)が設定される。通常はlenと同じ値が設定されるが、実際の文字列要素がlenよりも少なかった場合や、言語指定コードの途中でlenの要素数に達した場合、正常に処理できた部分の要素数が設定される。alen が len を超える値になることはない。(不完全な言語指定コード列があっても、エラーにはしない。)

chp は文字列中の各文字の描画位置を格納するための、len の要素数を持つ PNT 配列へのポインタである。chp = NULL の場合は、各文字の描画位置は格納されない。言語指定コードに対応したPNT配列の要素に関しては、その時点での文字描画位置が設定される。

cr は文字列中の各文字の描画領域を格納するための、len の要素数を持つ RECT 配列へのポインタである。cr = NULL の場合は、各文字の描画領域は格納されない。言語指定コードに対応したRECTに関しては、その時点での文字描画位置を(Px,Py)として、{Px,Py,Px,Py}が設定される。(RECT としては空ということになる。)

この関数により描画環境の文字描画位置、文字セット(SCRIP)は変化しない。関数値として、エラーコードまたは、描画終了時点でのSCRIPを戻す。ただし、SCRIPに変化がない場合には0を戻す。

### 【リターン値】

=0 : 正常終了(描画による言語切り替えは発生しない)

>0 : 正常終了(関数値は描画した場合に更新されるSCRIP)

<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(str, gap, cr, dr, alen)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(gap,lenの内容が不正)

## 2.5.3 文字(列)の描画関数

### gdra\_chr

Draw Character

文字の描画

#### 【形式】

WERR gdra\_chr ( GID gid, TC ch, DCM mode )

#### 【解説】

gid で指定した描画環境の文字描画位置に ch で指定した文字を mode で指定した描画モードで描画する。描画後、文字描画位置は文字描画方向 / 文字間隔に従って更新される。ch が言語指定コードであった場合、言語指定が更新される。

言語切り替えが発生した場合、切り替わった言語指定に対応したSCRIPTを戻す。

#### 【リターン値】

=0 : 正常終了。言語切り替えは発生しなかった。

>0 : 正常終了。言語切り替えが発生した。

<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(modeが不正)

### gdra\_chp

Draw Character at Position

文字の描画(位置指定)

#### 【形式】

WERR gdra\_chp ( GID gid, W x, W y, TC ch, DCM mode )

#### 【解説】

gid で指定した描画環境の文字描画位置を(x,y)に変更した後、ch で指定した文字を mode で指定した描画モードで描画する。描画後、文字描画位置は文字描画方向 / 文字間隔に従って更新される。ch が言語指定コードであった場合、言語指定が更新される。

言語切り替えが発生した場合、切り替わった言語指定に対応したSCRIPTを戻す。

#### 【リターン値】

=0 : 正常終了。言語切り替えは発生しなかった。

>0 : 正常終了。言語切り替えが発生した。

<0 : エラー (関数値はエラーコード)

#### 【エラーコード】

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(modeが不正)

## gdra\_str

Draw String

文字列の描画

### 【形式】

WERR gdra\_str ( GID gid, TC \*str, W len, DCM mode )

### 【解説】

gid で指定した描画環境の文字描画位置に、str で指定した文字列の先頭からlen 文字を mode で指定した描画モードで描画する。描画後、文字描画位置は文字描画方向 / 文字間隔に従って更新される。ch が言語指定コードであった場合、言語指定が更新される。

言語切り替えが発生した場合、切り替わった言語指定に対応したSCRIPTを戻す。

str で指定した文字列が len で指定した値より短い場合は、そこまでの文字列が描画される。

### 【リターン値】

=0 : 正常終了。言語切り替えは発生しなかった。

>0 : 正常終了。言語切り替えが発生した。

<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(str)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(modeが不正)

## gdra\_stp

Draw String at Position

文字列の描画(位置指定)

### 【形式】

WERR gdra\_stp ( GID gid, W x, W y, TC \*str, W len, DCM mode )

### 【解説】

gid で指定した描画環境の文字描画位置を(x,y)に変更した後、str で指定した文字列の先頭からlen 文字を mode で指定した描画モードで描画する。描画後、文字描画位置は文字描画方向 / 文字間隔に従って更新される。ch が言語指定コードであった場合、言語指定が更新される。

言語切り替えが発生した場合、切り替わった言語指定に対応したSCRIPTを戻す。

str で指定した文字列が len で指定した値より短い場合は、そこまでの文字列が描画される。

### 【リターン値】

=0 : 正常終了。言語切り替えは発生しなかった。

>0 : 正常終了。言語切り替えが発生した。

<0: エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR

指定されたアドレス(str)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(modeが不正)

## gcop\_chr

Copy Character Image

文字イメージの描画

【形式】

ERR gcop\_chr ( GID gid, SIZE asize, SIZE isize, UB \*img, DCM mode )

【解説】

gid で指定した描画環境で、img で指定した文字イメージを現在の文字描画位置に mode で指定した描画モード、設定されている文字描画カラーで描画する。描画後、文字描画位置は文字描画方向 / 文字間隔に従って更新される。

文字描画位置が文字イメージ矩形のどの位置に対応するかは、通常の文字描画と同様に文字描画方向に依存する。

描画すべき文字イメージは以下に示すビットマップデータとして取り扱われる。この文字イメージは isize で示される大きさを持ち、asize で示される大きさに拡大 / 縮小が行なわれて描画される。

```
planes    1
rowbytes  ((isize.h +15) ÷ 16) ×2 (偶数)
bounds    (0, 0, isize.h, isize.v)
baseaddr  img
```

ベース位置に関しては、描画時点でのフォント情報の文字高さとベース位置の比率をもとに、asize.v から決定される。

Base = asize.v \* FNTINFO.base / FNTINFO.height

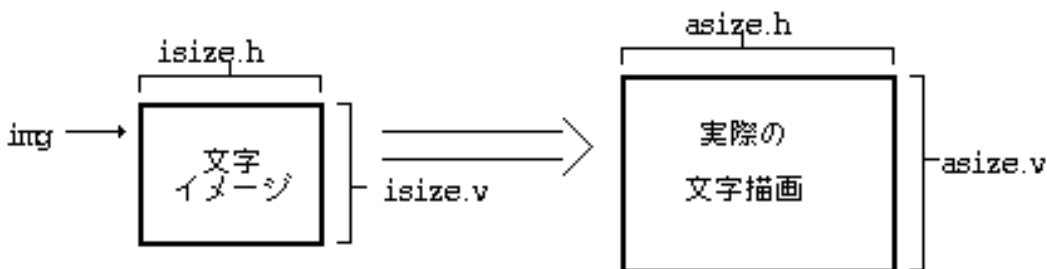


図 42 : 文字イメージの描画

【リターン値】

=0(E\_OK) : 正常終了

<0: エラー (関数値はエラーコード)

【エラーコード】

EG\_ADR

指定されたアドレス(img)のメモリのアクセスは許されていない。

EG\_GID

指定した描画環境IDが不正(存在しない)。

EG\_PAR

パラメータが不正、または範囲外である。(isize,asizeの内容,mode が不正)

---

[この章の目次にもどる](#)

[前頁:2.4 図形描画関数にもどる](#)

[次頁:2.6 ポインタ制御関数にすすむ](#)



## 2.6 ポインタ制御関数

### 2.6.1 ポインタ

ポインタは、スクリーン上に表示されているオブジェクトを指し示すために使用される特殊な図形であり、スクリーン上の表示にオーバーラップする形で表示される。

### 2.6.2 ポインタの状態 / 形状

ポインタの状態は以下の構造体により定義される。

```
/* ポインタ形状 */
typedef W          PTRSTL; /* ポインタの形状 */

/* ポインタ状態 */
struct PointerStatus {
    PNT      pos;          /* 現在のポインタの座標位置 */
    PTRSTL   style;       /* ポインタの形状 */
    COLOR    fgcol;       /* 前景色 */
    COLOR    bgcol;       /* 背景色 */
    W        motion;     /* 追従状態(0:非追従、1:追従) */
    W        hidden;     /* 表示状態(0:表示状態、正:消去状態) */
    W        size;       /* ポインタサイズ(16,24,32,etc) */
};
```

```
typedef PointerStatus PTRSTS; /* ポインタの状態 */
```

pos は、現在のポインタの座標位置であり、スクリーンの左上を(0,0)とした絶対座標で示される。

style は、ポインタの形状を示し、0 の場合は、システムで定義されている標準形状を示し、<0 の場合は、アプリケーションが独自に定義した形状を示す。標準形状としては、以下のものが定義されている。

```
/* 標準ポインタ形状 */
enum StdPointerShape {
/*          (意味) */
    PS_SELECT   = 0, /* 選択指 : 選択操作 */
    PS_MODIFY   = 1, /* 修正選択指 : 修正選択操作 */
    PS_MOVE     = 2, /* 移動手 : 移動可能 */
    PS_VMOVE    = 3, /* 移動手(縦方向) : 移動可能(縦方向のみ) */
    PS_HMOVE    = 4, /* 移動手(横方向) : 移動可能(横方向のみ) */
    PS_GRIP     = 5, /* 握り : 移動ドラッグ中 */
    PS_VGRIP    = 6, /* 握り(縦方向) : 移動ドラッグ中(縦方向のみ) */
    PS_HGRIP    = 7, /* 握り(横方向) : 移動ドラッグ中(横方向のみ) */
    PS_RSIZ     = 8, /* 変形手 : 変形可能 */
};
```

```

PS_VRSIZ    = 9,    /* 変形手(縦方向) : 変形可能(縦方向のみ) */
PS_HRSIZ    = 10,   /* 変形手(横方向) : 変形可能(横方向のみ) */
PS_PICK     = 11,   /* つまみ : 変形ドラッグ中 */
PS_VPICK    = 12,   /* つまみ(縦方向) : 変形ドラッグ中(縦方向のみ) */
PS_HPICK    = 13,   /* つまみ(横方向) : 変形ドラッグ中(横方向のみ) */
PS_BUSY     = 14,   /* 湯のみ(待ち状態) : 待ち状態 */
PS_MENU     = 15    /* プリメニュー : メニュー表示 */
/*          = 16 ~ : 予約 */
};

```

fgcol と bgcol はそれぞれポインタの前景色、背景色を示すピクセル値である。

motion は、ポインタの位置がポインティングデバイスに追従しているか否かの状態を示す。

hidden は、ポインタの消去要求数を示し、この値が0の時、ポインタが実際に表示されていることになる。

size は、ポインタサイズを示し、ポインタ形状を定義する正方形の大きさ(高さ)をピクセル数で示したもので、標準的には24であるが、インプリメントによっては、16、32等の場合もある。

ポインタの形状を示すイメージデータは、以下の構造体で表わされる。

```

/* ポインタイメージデータ */
#define MAX_PTR_SIZE    64 /* ポインタイメージデータサイズ */
#define P_SIZE    (MAX_PTR_SIZE * MAX_PTR_SIZE) / 8

```

```

struct PointerImage {
    PNT    hotpt;        /* ホットポイントの座標値 */
    SIZE    size;        /* ポインタサイズ */
    UB    data[P_SIZE]; /* データイメージ */
    UB    mask[P_SIZE]; /* マスクイメージ */
};

```

```

typedef PointerImage    PTRIMG;    /* ポインタイメージ */

```

hotpt は、イメージデータ中のどの点をポインタ位置とするかを示すもので、イメージ長方形の左上を(0,0)とした座標値で表わされる。即ち、hotpt がポインタ位置となるようにポインタイメージが表示される。

data と mask はそれぞれポインタのデータイメージとマスクイメージを表現する以下のビットマップである。n はポインタのサイズであり、標準では24であるが、インプリメントによっては、16、32 などとなる。

```

planes    : 1
rowbytes  : ((n + 15) ÷ 16) × 2 (偶数)
bounds    : (0,0,n,n)
baseaddr  : data / mask

```

P\_SIZE は、ポインタサイズに依存した値であり、 $n \times \text{rowbytes}$  である。(n = 16 の時は P\_SIZE = 32、n = 24 の時は P\_SIZE = 96、n = 32 の時は P\_SIZE = 128)

ポインタイメージの表示は、データイメージと、マスクイメージに従って、以下のように描画環境上に行なわれる。

| データ | マスク | 表示              |
|-----|-----|-----------------|
| 0   | 1   | 背景色             |
| 1   | 1   | 前景色             |
| 0   | 0   | 元のイメージのまま (透明)  |
| 1   | 0   | 元のイメージの反転 (XOR) |

前景色と背景色は、ポインタの色を指定したピクセル値である。この値は基本的には、スクリーンへの描画で使用されるピクセル値と同一であるが、インプリメントによっては、異なっている場合もある。

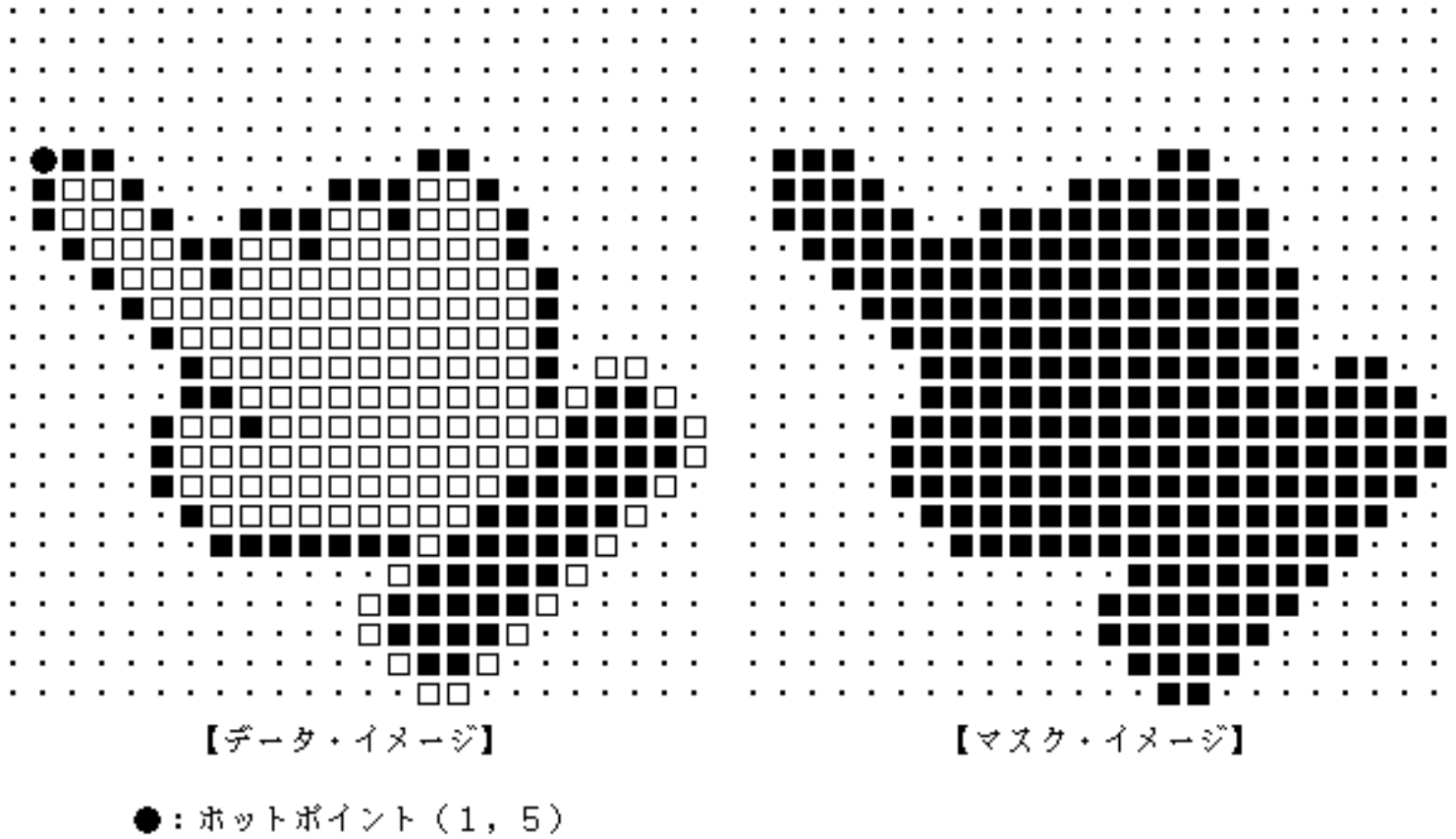


図 43 : ポインタイメージの例

### 2.6.3 ポインタの制御関数

**gini\_ptr**

Initialize Pointer

ポインタの初期化

**【形式】**

ERR gini\_ptr ( )

**【解説】**

ポインタの状態を以下に示す初期状態に設定して、現在のポインティングデバイスの位置に表示する。

表示 / 非表示  
表示状態

形状

style = 0の形状

前景色

デフォルト色（通常は黒）

背景色

デフォルト色（通常は白または肌色）

追従 / 非追従

追従状態

【リターン値】

= 0 : 正常終了

< 0 : エラー (発生しない)

【エラーコード】

発生しない。

## gdsp\_ptr

Display Pointer

ポインタの表示 / 消去

【形式】

ERR gdsp\_ptr ( W req )

【解説】

ポインタの消去 / 表示 / 強制表示を要求する。

req = 0

消去要求

req > 0

表示要求

req < 0

強制表示要求

消去要求では、ポインタの消去要求数を + 1 してポインタの表示を消去する。表示要求では、ポインタの表示要求数が0でない場合、- 1 して0になった場合にポインタの表示を行なう。強制表示要求では、ポインタの表示要求数を0としてポインタの表示を行なう。

結果としての消去要求数を関数値として戻す。即ち、関数値が0の場合は、ポインタは表示されており、> 0の場合はポインタは表示されていないことになる。

【リターン値】

0 : 正常終了 (関数値は消去要求数)

< 0 : エラー (発生しない)

【エラーコード】

発生しない。

## gset\_ptr

Set Pointer Shape and Color

ポインタ形状 / カラーの設定

【形式】

ERR gset\_ptr ( PTRSTL style, PTRIMG \*img, COLOR fgcol, COLOR bgcol )

### 【解説】

ポインタの形状 / カラーを、style(形状)、fgcol(前景色)、および bgcol(背景色) で指定した値に変更する。style<0の場合は、img で指定したポインタイメージが有効になり style 0の場合はimgは無視され、指定した標準イメージが適用される。

fgcol<0, bgcol<0の場合は、それぞれデフォルト色の指定となる。

### 【リターン値】

= 0(E\_OK) : 正常終了  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_ADR

指定されたアドレス(img)のメモリのアクセスは許されていない。

EG\_PAR

パラメータが不正、または範囲外である。(fgcol,bgcolが範囲外, imgの内容が不正)

## gmov\_ptr

Move Pointer Position

ポインタ表示位置の移動

### 【形式】

ERR gmov\_ptr ( PNT pos )

### 【解説】

ポインタが非追従状態の場合、ポインタの位置を pos で指定した位置に移動する。pos は画面の左上を (0,0)とした絶対座標であり、画面をはみ出る場合はEG\_PAR エラーとなる。ポインタが追従状態の場合は、何もしない。

関数値として、ポインタが非追従状態の場合は0、追従状態の場合は1が戻る。従って、関数値が0の場合のみポインタ表示位置が変更されたことになる。

### 【リターン値】

0 : 正常終了 (関数値は追従状態)  
<0 : エラー (関数値はエラーコード)

### 【エラーコード】

EG\_PAR

パラメータが不正、または範囲外である。( pos が範囲外 )

## gmot\_ptr

Set Pointer Motion

ポインタ動作の設定

### 【形式】

ERR gmot\_ptr ( W sts )

### 【解説】

ポインタ表示がポインティングデバイスの動きに追従するか否かを sts の値により指定する。

sts = 0

非追従動作指定

## 追従動作指定

非追従指定を行なった場合はその時点でポインタの位置は固定され、`gmov_ptr()` 関数によってのみポインタ表示を移動することができる。追従指定を行なった場合は、その時点からポインタの表示はポインティングデバイスに追従を始める。従って、ポインタの表示位置が、ポインティングデバイスの位置と異なっていた場合は、追従指定によりポインタ表示が移動することになる。

既に追従している時に追従指定を行なった場合や、非追従の時に非追従指定を行なった場合は何もしない。

### 【リターン値】

= 0 : 正常終了  
< 0 : エラー (発生しない)

### 【エラーコード】

発生しない。

## gget\_ptr

Get Pointer Status and Shape

ポインタ状態 / 形状の取り出し

### 【形式】

W gget\_ptr ( PTRSTS \*sts, PTRIMG \*img )

### 【解説】

ポインタの現在の状態を `sts` で指定した領域に、ポインタのイメージデータを `img` で指定した領域にそれぞれ格納する。

`sts` は `PTRSTS` 構造体データへのポインタであり、`img` は `PTRIMG` 構造体データへのポインタであり、ともに予めその領域は確保されている必要がある。`sts` または `img` が `NULL` の場合は、それぞれ対応するデータは格納されない。

関数値としてポインタのサイズが戻る。

### 【リターン値】

0 : 正常終了(関数値はポインタのサイズ)  
< 0 : エラー (関数値はエラーコード)

### 【エラーコード】

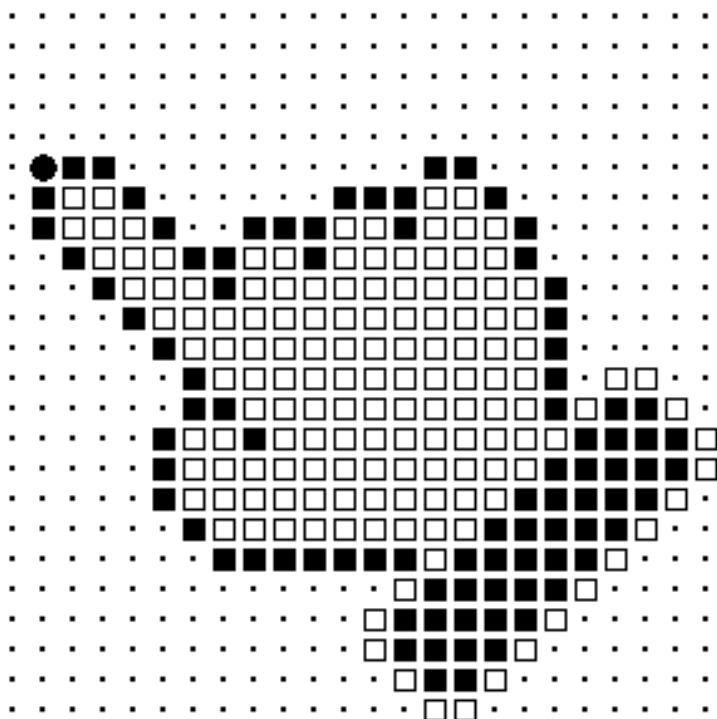
EG\_ADR

指定されたアドレス(`sts, img`)のメモリのアクセスは許されていない。

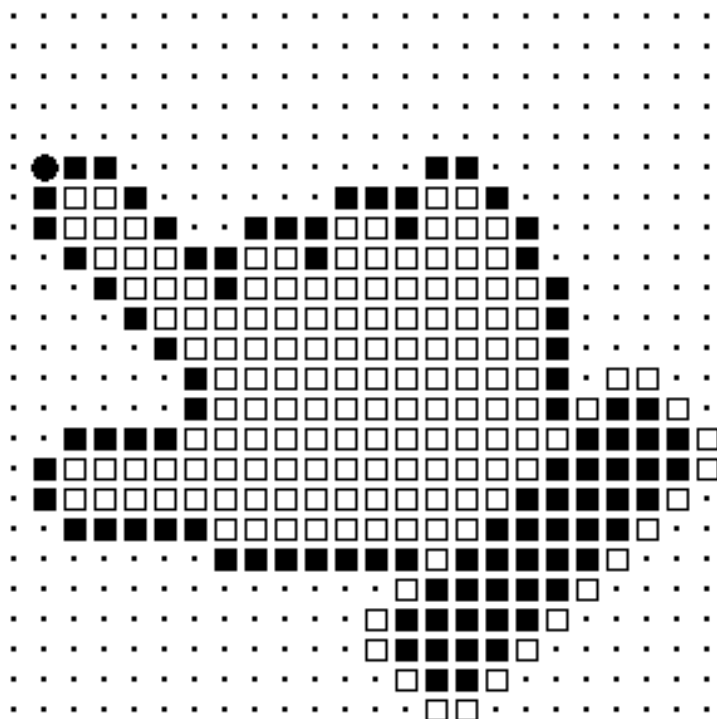
## 標準ポインタ形状(参考)

以下に、24 x 24 ピクセルの標準ポインタ形状を示す。

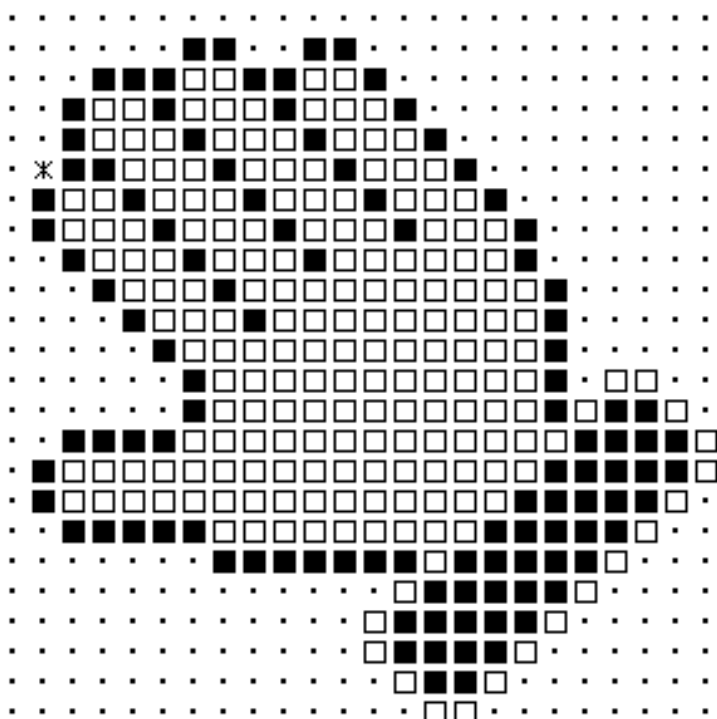
|     |                      |        |
|-----|----------------------|--------|
| •   | : mask = 0, data = 0 | -- 透明  |
| ▼   | : mask = 0, data = 1 | -- 反転  |
| □ ○ | : mask = 1, data = 0 | -- 背景色 |
| ■ ● | : mask = 1, data = 1 | -- 前景色 |
| ●   | : hotspot (前景色)      |        |
| ○   | : hotspot (背景色)      |        |
| *   | : hotspot (透明)       |        |



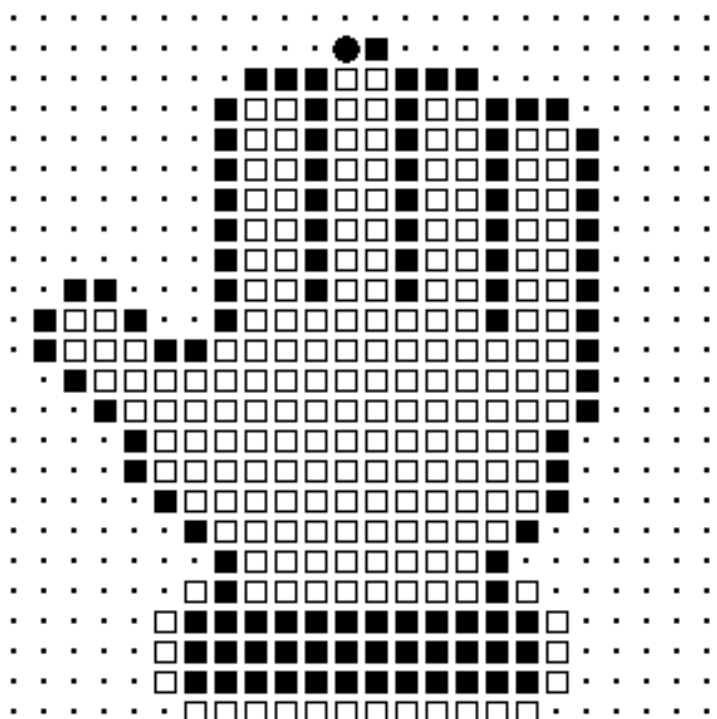
【 PS\_SELECT : 選択指 】



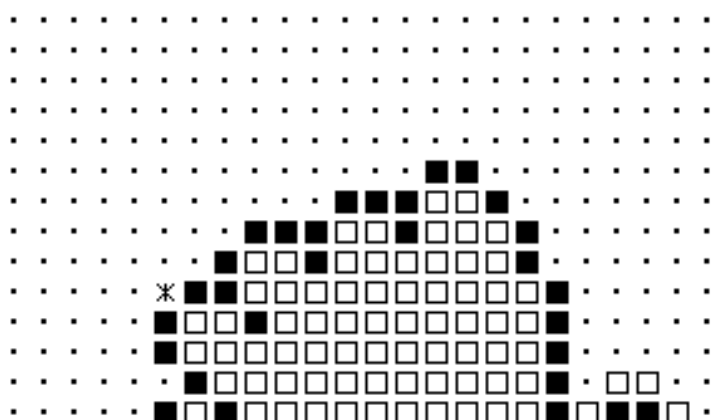
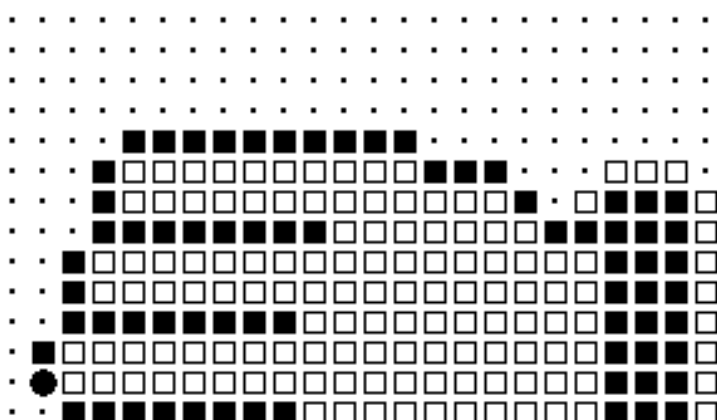
【 PS\_MODIFY : 修正選択指 】

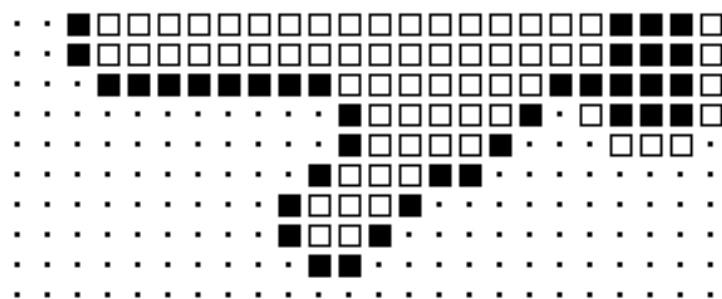


【 PS\_MOVE : 移動手 】

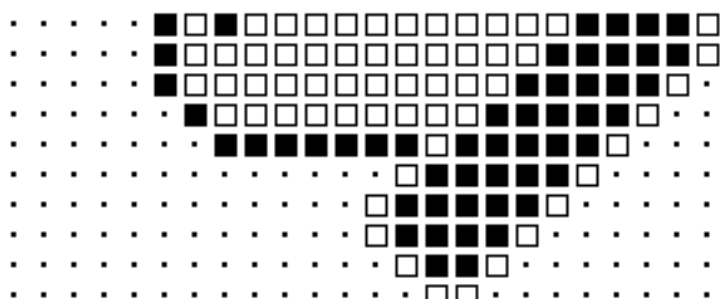


【 PS\_VMOVE : 移動手(縦方向) 】

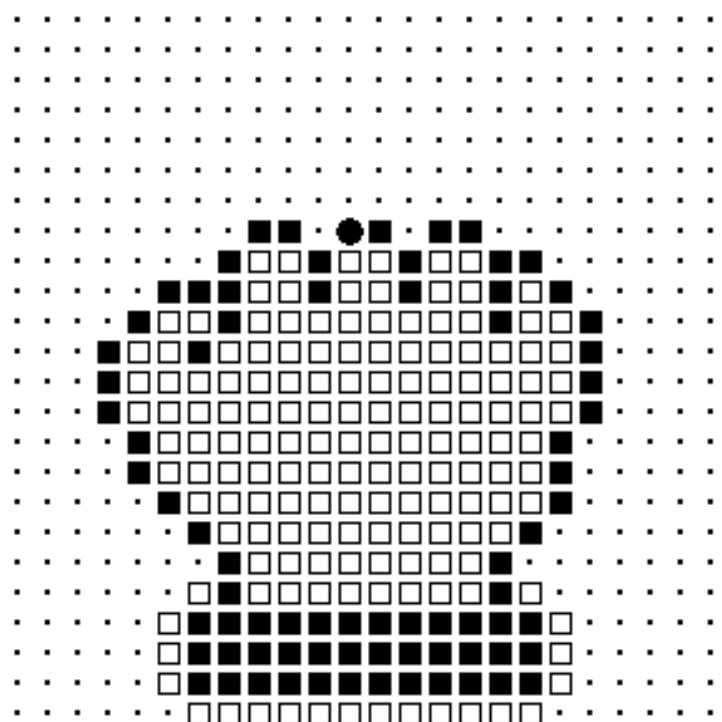




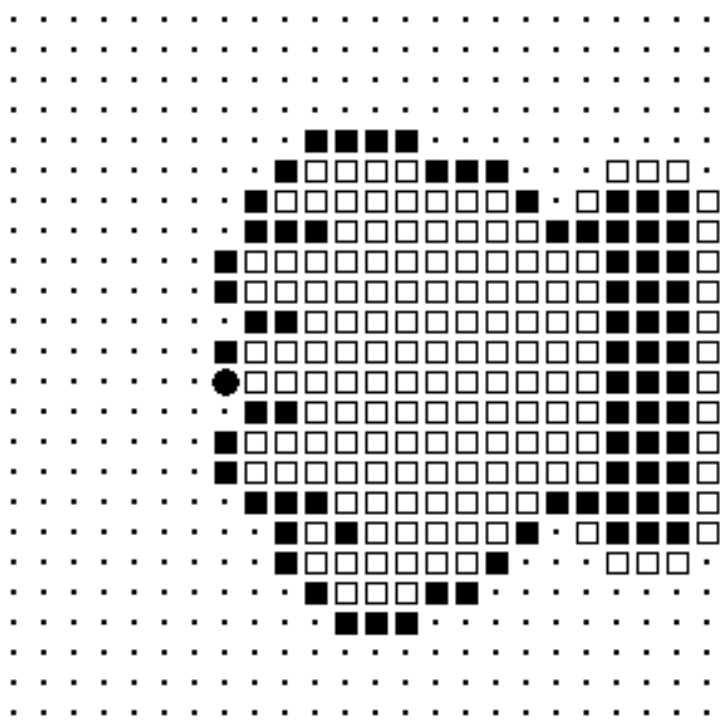
【 PS\_HMOVE : 移動手(横方向) 】



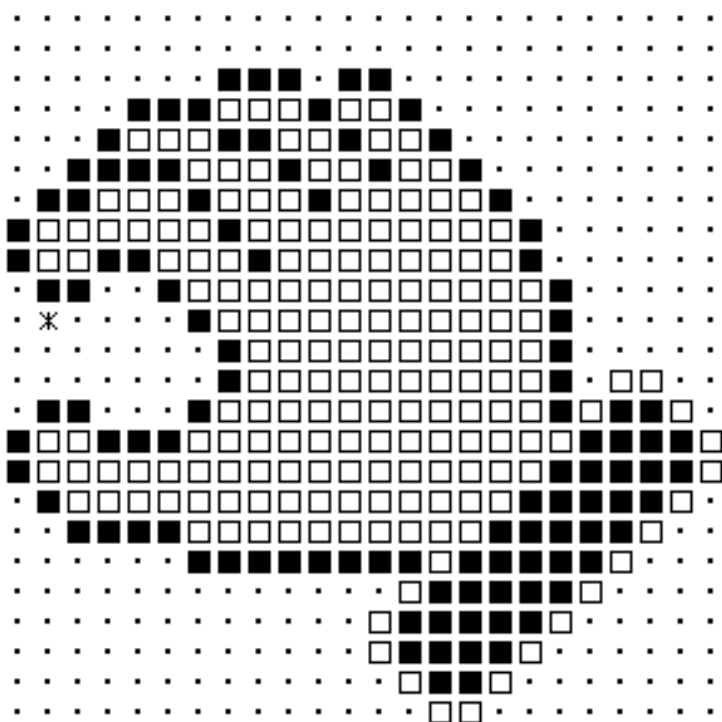
【 PS\_GRIP : 握り 】



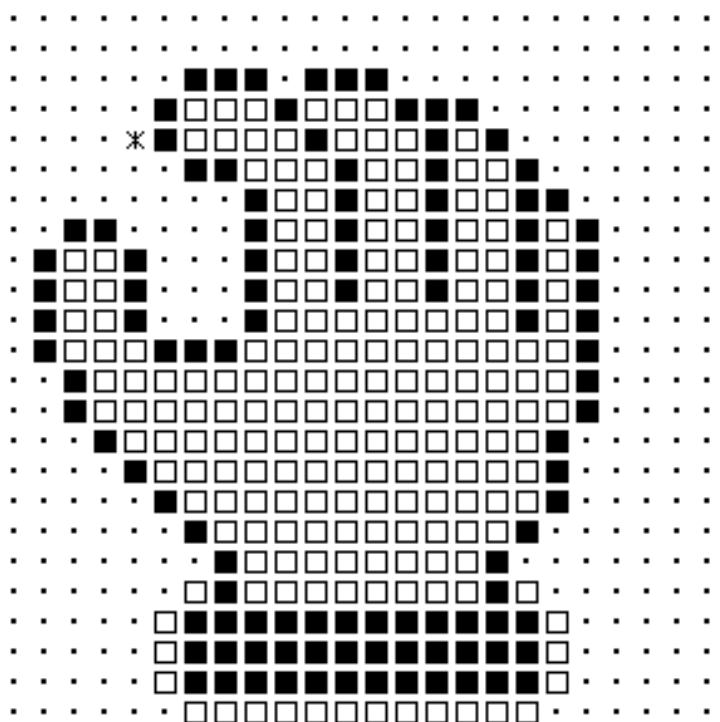
【 PS\_VGRIP : 握り(縦方向) 】



【 PS\_HGRIP : 握り(横方向) 】

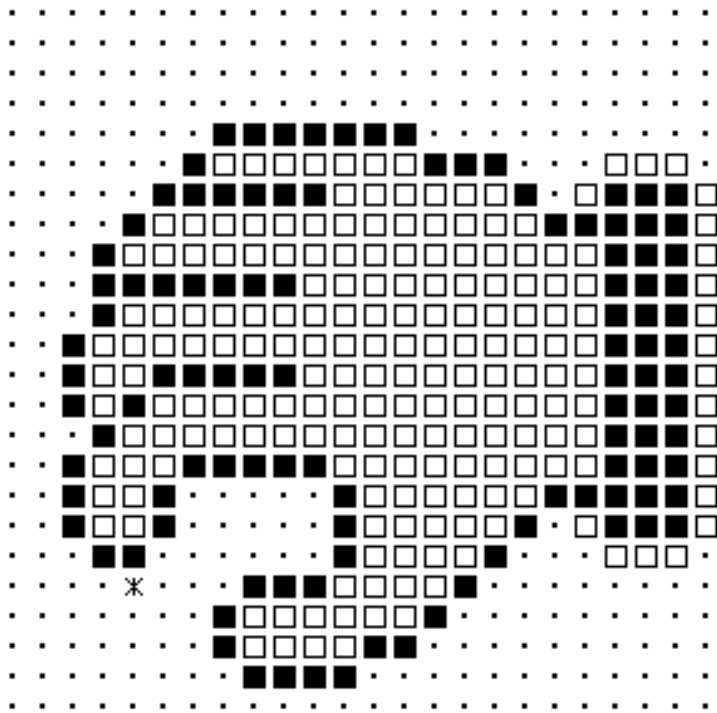


【 PS\_RSIZ : 変形手 】

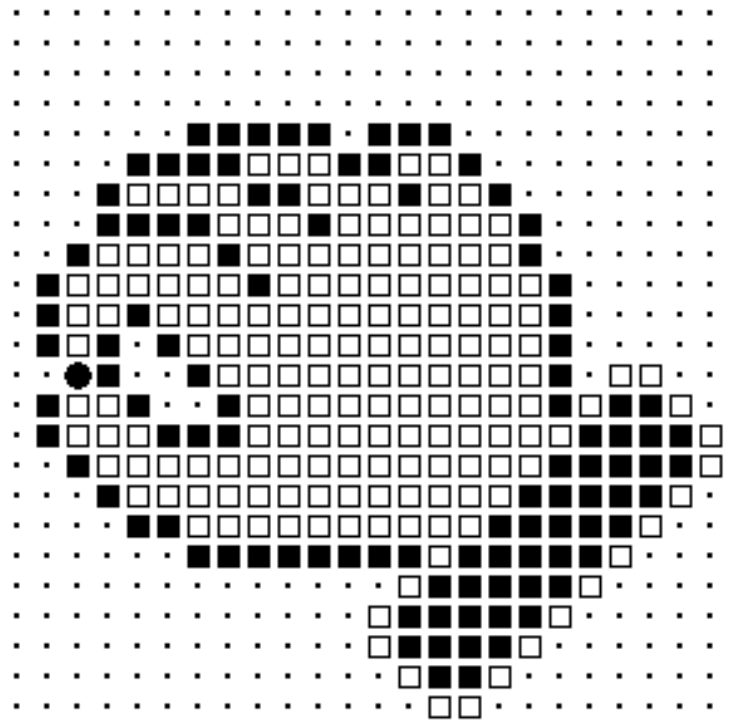


【 PS\_VRSIZ : 変形手(縦方向) 】

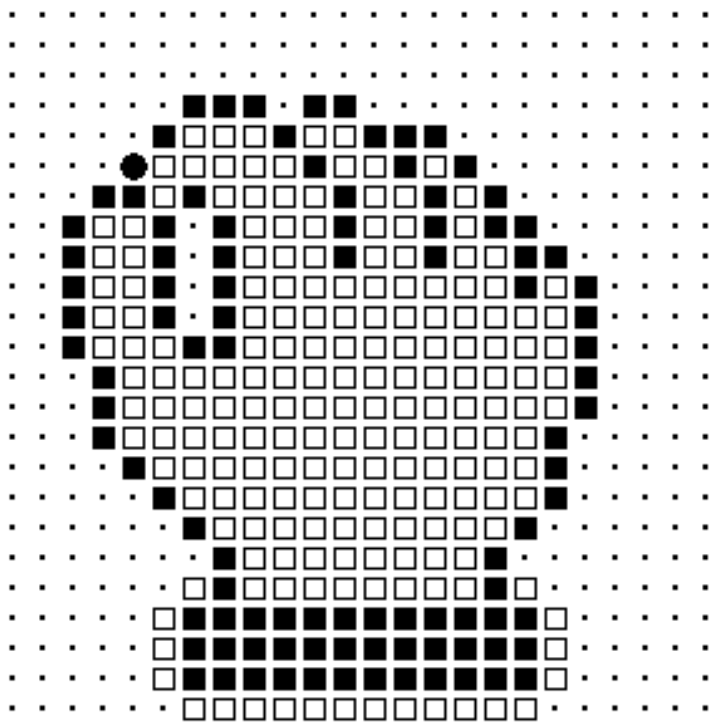




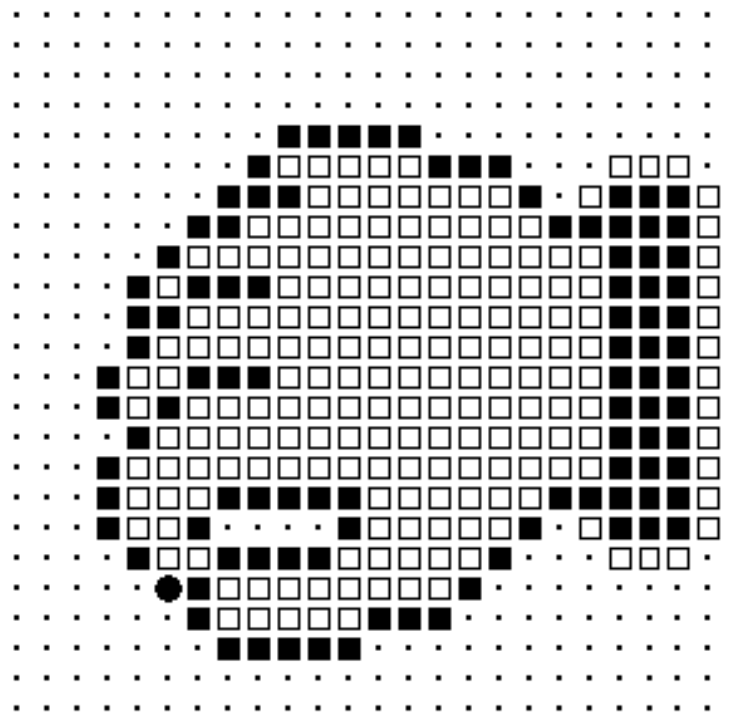
【 PS\_HRSIZ : 変形手(横方向) 】



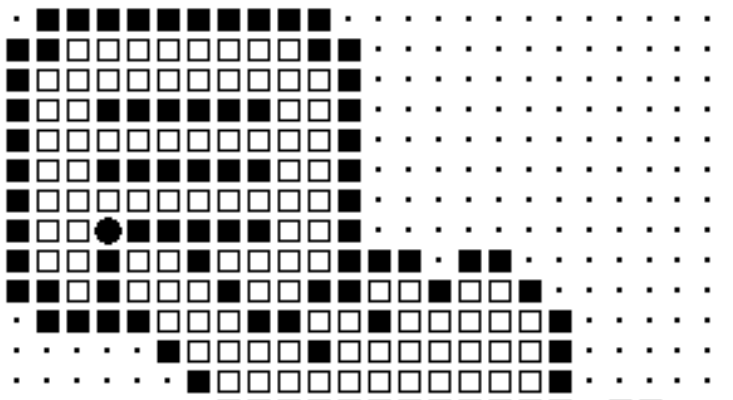
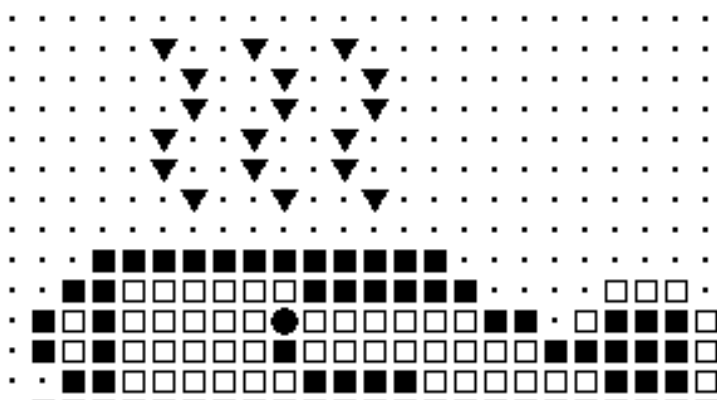
【 PS\_PICK : つまみ】

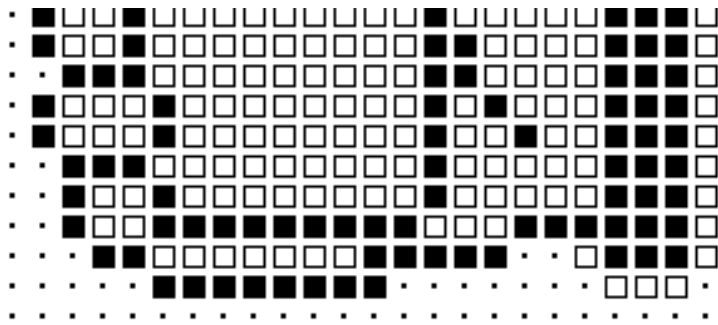


【 PS\_VPICK : つまみ(縦方向) 】

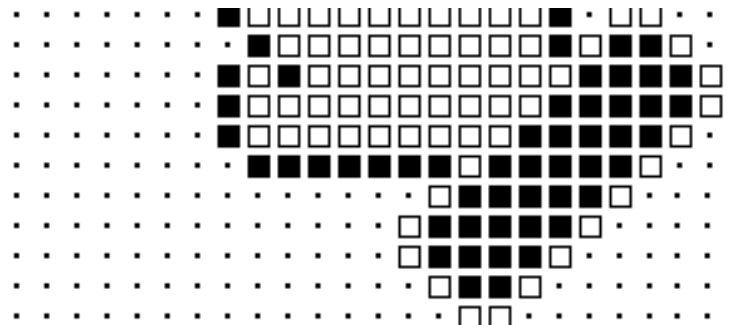


【 PS\_HPICK : つまみ(縦方向) 】





【 PS\_BUSY : 湯のみ 】



【 PS\_MENU : プリメニュー 】

図 44 : 標準ポインタ形状

[この章の目次にもどる](#)

[前頁:2.5 文字\(列\)描画関数にもどる](#)

[次頁:第3章 外殻にすすむ](#)

[この章の目次にもどる](#)

[前頁:2.6 ポインタ制御関数にもどる](#)

[次頁:3.1 ウィンドウマネージャにすすむ](#)

---

# 第3章 外殻

---

## [第3章 外殻](#)

### [3.1 ウィンドウマネージャ](#)

#### [3.1.1 ウィンドウマネージャの機能](#)

##### [3.1.1.1 概要](#)

##### [3.1.1.2 ウィンドウ](#)

##### [3.1.1.3 ウィンドウの表示](#)

##### [3.1.1.4 ウィンドウの操作](#)

#### [3.1.2 ウィンドウの管理](#)

##### [3.1.2.1 ウィンドウとプロセス](#)

##### [3.1.2.2 ウィンドウイベント](#)

##### [3.1.2.3 ウィンドウの再表示処理](#)

##### [3.1.2.4 ウィンドウの表示管理](#)

##### [3.1.2.5 ウィンドウの描画環境](#)

##### [3.1.2.6 ウィンドウ情報レコード](#)

#### [3.1.3 データ/定数の定義](#)

#### [3.1.4 ウィンドウマネージャの関数](#)

### [3.2 メニューマネージャ](#)

#### [3.2.1 メニューマネージャの機能](#)

##### [3.2.1.1 概要](#)

##### [3.2.1.2 標準メニュー](#)

##### [3.2.1.3 標準メニューの操作](#)

##### [3.2.1.4 標準メニューのデータ構造](#)

##### [3.2.1.5 汎用メニュー](#)

##### [3.2.1.6 汎用メニューの操作](#)

##### [3.2.1.7 汎用メニューのデータ構造](#)

#### [3.2.2 データ / 定数の定義](#)

#### [3.2.3 メニューマネージャの関数](#)

### [3.3 パーツマネージャ](#)

#### [3.3.1 パーツマネージャの機能](#)

##### [3.3.1.1 概要](#)

##### [3.3.1.2 パーツの種類](#)

##### [3.3.1.3 パーツのデータ構造](#)

##### [3.3.1.4 パーツの状態](#)

#### [3.3.2 パーツの種類](#)

- [3.3.2.1 テキストボックス](#)
- [3.3.2.2 シークレットテキストボックス](#)
- [3.3.2.3 数値ボックス](#)
- [3.3.2.4 シリアルボックス](#)
- [3.3.2.5 テキストオルタネートスイッチ](#)
- [3.3.2.7 ピクトグラムオルタネートスイッチ](#)
- [3.3.2.8 ピクトグラムモーメンタリスイッチ](#)
- [3.3.2.9 スイッチセレクタ](#)
- [3.3.2.10 スクロールセレクタ](#)
- [3.3.2.11 ボリューム](#)
- [3.3.3 データ / 定数の定義](#)
- [3.3.4 パーツマネージャの関数](#)

### [3.4 パネルマネージャ](#)

- [3.4.1 パネルマネージャの機能](#)
  - [3.4.1.1 概要](#)
  - [3.4.1.2 パネル](#)
  - [3.4.1.3 システムメッセージパネル](#)
  - [3.4.1.4 パネルの表示形状](#)
  - [3.4.1.5 パネルのデータ構造](#)
  - [3.4.1.6 パネルの描画環境](#)
  - [3.4.1.7 パネルの動作](#)
- [3.4.2 データ / 定数の定義](#)
- [3.4.3 パネルマネージャの関数](#)

### [3.5 トレーマネージャ](#)

- [3.5.1 トレーマネージャの機能](#)
  - [3.5.1.1 概要](#)
  - [3.5.1.2 トレー](#)
  - [3.5.1.3 トレーの格納データ](#)
  - [3.5.1.4 トレーの操作](#)
- [3.5.2 データ / 定数の定義](#)
- [3.5.3 トレーマネージャの関数](#)

### [3.6 データマネージャ](#)

- [3.6.1 データマネージャの機能](#)
  - [3.6.1.1 概要](#)
  - [3.6.1.2 データタイプID / データ番号](#)
  - [3.6.1.3 データマネージャの動作](#)
  - [3.6.1.4 データボックスの構造](#)
- [3.6.2 データ / 定数の定義](#)
  - [3.6.2.1 データ / 定数の定義](#)
  - [3.6.2.2 標準データ形式](#)
- [3.6.3 データマネージャの関数](#)

### [3.7 テキスト入力プリミティブ](#)

### [3.7.1 テキスト入力プリミティブの機能](#)

#### [3.7.1.1 概要](#)

#### [3.7.1.2 テキスト入力プリミティブ](#)

#### [3.7.1.3 テキスト入力ポート](#)

#### [3.7.1.4 カレット表示](#)

### [3.7.2 データタイプ/定数の定義](#)

### [3.7.3 テキスト入力プリミティブの関数](#)

## [3.8 実身 / 仮身マネージャ](#)

### [3.8.1 実身 / 仮身マネージャの機能](#)

#### [3.8.1.1 概要](#)

#### [3.8.1.2 仮身 / 実身 / 虚身](#)

#### [3.8.1.3 仮身の表示](#)

#### [3.8.1.4 仮身の操作](#)

### [3.8.2 仮身の移動 / 複写](#)

#### [3.8.2.1 付箋](#)

### [3.8.3 実身 / 仮身マネージャの詳細](#)

#### [3.8.3.1 仮身](#)

#### [3.8.3.2 付箋](#)

#### [3.8.3.3 アプリケーションプログラム管理](#)

#### [3.8.3.4 アプリケーションプログラムの起動](#)

#### [3.8.3.5 メニュー管理](#)

### [3.8.4 データ / 定数の定義](#)

### [3.8.5 実身 / 仮身マネージャの関数](#)

### [3.8.6 アプリケーション支援関数](#)

## [3.9 フォントマネージャ](#)

### [3.9.1 フォントマネージャの機能](#)

#### [3.9.1.1 概要](#)

#### [3.9.1.2 フォント](#)

#### [3.9.1.3 フォントデータ](#)

#### [3.9.1.4 フォントのアクセス](#)

### [3.9.2 データ / 定数の定義](#)

### [3.9.3 フォントマネージャの関数](#)

### [3.9.4 標準フォントデータ形式](#)

## [3.10 TCP/IPマネージャ](#)

### [3.10.1 構成](#)

### [3.10.2 TCP/IP](#)

### [3.10.3 エラーコード](#)

### [3.10.4 システムコール](#)

## [3.11 印刷マネージャ](#)

### [3.11.1 印刷マネージャの機能](#)

### [3.11.2 印刷管理](#)

[3.11.2.1 レイアウト用紙と印刷用紙](#)

[3.11.2.2 印刷管理の機能](#)

[3.11.2.3 印刷パラメータ](#)

[3.11.3 スプール管理](#)

[3.11.3.1 スプール管理の機能](#)

[3.11.3.2 印刷登録状態](#)

[3.11.3.3 印刷終了メッセージ](#)

[3.11.4 プリンタ管理](#)

[3.11.4.1 プリンタ管理の機能](#)

[3.11.4.2 プリンタ構成情報](#)

[3.11.5 データ / 定数の定義](#)

[3.11.6 印刷マネージャの関数](#)

---

[この章の目次にもどる](#)

[前頁:2.6 ポインタ制御関数にもどる](#)

[次頁:3.1 ウィンドウマネージャにすすむ](#)

## 3.1 ウィンドウマネージャ

### 3.1.1 ウィンドウマネージャの機能

#### 3.1.1.1 概要

ウィンドウマネージャは外殻の HMI 機能の中心的な機能であり、画面表示上の操作環境の基本となる空間であるパネルプレーン上に存在する複数のオーバーラップしうるパネルの生成 / 削除 / 表示 / 操作等に関する基本的な機能を提供している。

ウィンドウマネージャでは、基本的にパネルプレーン上のフローティングパネル ( ウィンドウ ) の管理のみを行ない、ダイアログタイプのパネルやシステムパネルは独立したパネルマネージャにより管理されることになる。

ウィンドウには必ず対応するアプリケーションプロセスが存在し、そのアプリケーションプロセスがウィンドウマネージャで提供している機能を一定のルールに従って使用することにより、実際のウィンドウの表示 / 操作が行なわれ、全体としてのウィンドウシステムが実現されることになる。従って、ウィンドウを使用するすべてのアプリケーションは、アプリケーションテンプレートとして規定されたプログラム構造を持たなくてはならない。

ウィンドウマネージャは、タイトル、スクロールバーを含んだウィンドウ枠部分のみの表示を行ない、ウィンドウ内部の表示は、対応するアプリケーションプロセスが自分で行なうことになる。このための描画環境はウィンドウマネージャにより用意される。

ウィンドウシステムの実現方法には幾つかの方式があるが、ここでは基本的に特殊ハードウェアサポートが無いことを前提としており、またウィンドウが重なって隠れた部分の表示イメージは、原則的に保存されないことを前提としている。

これにより、最小限のハードウェアと、メモリによるウィンドウシステムの実現を可能としているが、実際のインプリメントにおいて、特殊なハードウェアを使用したり、隠れたイメージを自動的に保存 / 再現することを禁止しているわけではない。

なお、ウィンドウマネージャが管理する表示画面は、最下部のシステムメッセージパネルを除いた部分となる。ただし、フルスクリーンモードの場合は、システムメッセージパネルを含めたスクリーン全部に、ウィンドウが表示される。

#### 3.1.1.2 ウィンドウ

##### ウィンドウのタイプ

ウィンドウはユーザがアプリケーションを操作する上での環境としての、画面上の 1 つの独立した矩形領域である。この矩形領域は基本的に自由に移動 / 変形が可能であり、前後関係を持って互いにオーバーラップされて表示される。

ウィンドウには、以下に示す 4 種類があり、単にウィンドウと言った場合は、通常ウィンドウを意味する。

##### 通常ウィンドウ

一般のアプリケーションが使用する通常のウィンドウ。

## 前面ウィンドウ

フロントエンド機能を提供する特殊なアプリケーションが使用するウィンドウであり、常に最前面に置かれる。

## パネルウィンドウ

ダイアログタイプのパネルとして使用される特殊なウィンドウで通常ウィンドウの上で前面ウィンドウの下に表示される。このタイプのウィンドウはパネルマネージャによって使用され、一般のアプリケーションが直接使用することはない。また、適用可能な操作も非常に限定される。

## 内部ウィンドウ

独立した描画環境をウィンドウとして取り扱うために内部的に使用される特殊な擬似的なウィンドウである。このタイプのウィンドウは、実身/仮身マネージャにより使用され、一般のアプリケーションが直接使用することはない。また、適用可能な操作も非常に限定される。

通常ウィンドウは、さらに以下の2種類に分類される。

## 主ウィンドウ

アプリケーションの表示/操作の中心となるウィンドウ。

## 従属ウィンドウ

主ウィンドウの表示/操作の補助のために使用されるウィンドウであり、論理的に主ウィンドウと同一のウィンドウとして取り扱われる。従属ウィンドウは常に主ウィンドウの前面に表示される。

なお、前面ウィンドウは、存在する全ての主ウィンドウに共有される従属ウィンドウとして取り扱われ、現在入力受付状態のウィンドウを主ウィンドウとした場合の従属ウィンドウと同様に取

## 入力受付状態

ある時点で、ユーザからのインタラクティブな操作が可能なウィンドウを入力受付状態のウィンドウと呼び、入力受付状態でないウィンドウを入力不可状態のウィンドウと呼ぶ。

入力受付状態は主ウィンドウ間で任意に切り換えることができ、ある時点で1つの主ウィンドウのみが入力受付状態となりうる。主ウィンドウを新たにオープンした時点では、その主ウィンドウに入力受付状態が切り換わる。

主ウィンドウが入力受付状態の場合は、それに従属するすべての従属ウィンドウも入力受付状態となる。主ウィンドウが入力不可状態の場合は、それに従属するすべての従属ウィンドウの表示は消去され見えない状態となる。この状態を不可視状態と呼ぶ。従って、従属ウィンドウは、対応する主ウィンドウが入力受付状態の場合のみ表示されて「見える」存在となる。

前面ウィンドウは、ユーザからインタラクティブな操作に対してフロントエンドとしての一種の前処理を行なうため、その意味で常に入力受付状態を主ウィンドウと共有していると言える。即ち、ユーザからインタラクティブな操作は、常に前面ウィンドウを経由して、入力受付状態の主ウィンドウに送られることになる。

パネルウィンドウは、ダイアログタイプのパネルとして使用され、パネルウィンドウがオープンされている間は、入力受付状態を主ウィンドウから横取りして占有することになる。パネルウィンドウがクローズされた時点で、入力受付状態は元の主ウィンドウに戻される。複数のパネルウィンドウがオープンされた場合は、最後にオープンされたパネルウィンドウが入力受付状態を占有し、パネルウィンドウのクローズにより順番に元のウィンドウに入力受付状態が戻るこ



になる。

内部ウィンドウに対しては、入力受付状態は定義されない。

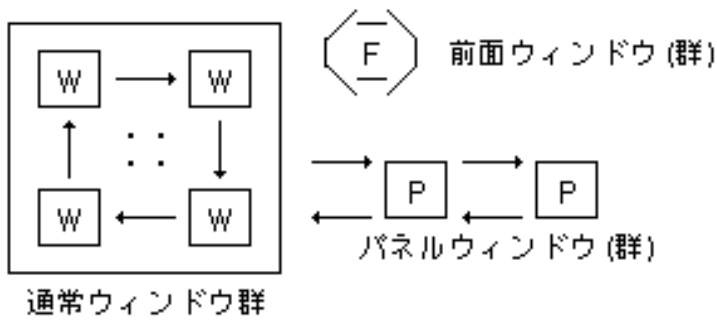


図 45 : 入力受付状態の遷移

ウィンドウの前後関係

ウィンドウの表示上の前後関係は、最前面から順番に以下のようになる。

- 最前面
- (1) 前面ウィンドウ
  - | (2) パネルウィンドウ
  - | (3) 入力受付状態の従属ウィンドウ
  - | (4) 入力受付状態の主ウィンドウ
  - | (5) 入力不可状態の主ウィンドウ

パネルウィンドウが複数オープンされている場合は、オープンされた順番に最前面に表示され、最前面に表示されているパネルウィンドウが入力受付状態を占有する。パネルウィンドウ間の前後関係の変更はできない。

前面ウィンドウが複数オープンされている場合の前後関係は以下のようになる。

- オープンした時には一番前面に表示される。
- 移動 / 変形を行なった場合、その対象とした前面ウィンドウが一番前面に表示される。この場合、他の前面ウィンドウの前後関係は変化しない。

入力受付状態の従属ウィンドウが複数オープンされている場合の前後関係は前面ウィンドウと同様である。なお、入力不可状態、即ち不可視状態になった場合は、従属ウィンドウ間の前後関係は保持され、再び入力受付状態となった場合に、以前の前後関係が再現される。

- オープンした時には一番前面に表示される。
- 移動 / 変形を行なった場合、その対象とした従属ウィンドウが一番前面に表示される。この場合、他の従属ウィンドウの前後関係は変化しない。

主ウィンドウの間では、入力受付状態の主ウィンドウが一番前面に表示される。即ち、新たに入力受付状態となった主ウィンドウが一番前面に表示される。この場合、他の主ウィンドウの前後関係は変化しない。

なお、内部ウィンドウに対しては、前後関係は定義されない。

ウィンドウの親子関係

通常、主ウィンドウは仮身に対してアプリケーションを実行することにより生成されることが多い。この場合、元となった仮身が存在していた主ウィンドウを生成された主ウィンドウの親ウィンドウと呼ぶ。

従って、主ウィンドウ全体はその親子関係により木構造となっていることになる。主ウィンドウの親子関係は画面上の表示には一切反映されないが、主ウィンドウを閉じた場合は、その元となった仮身に戻ることにになり、その結果、親ウィンドウに入力受付状態が切替わることになる。

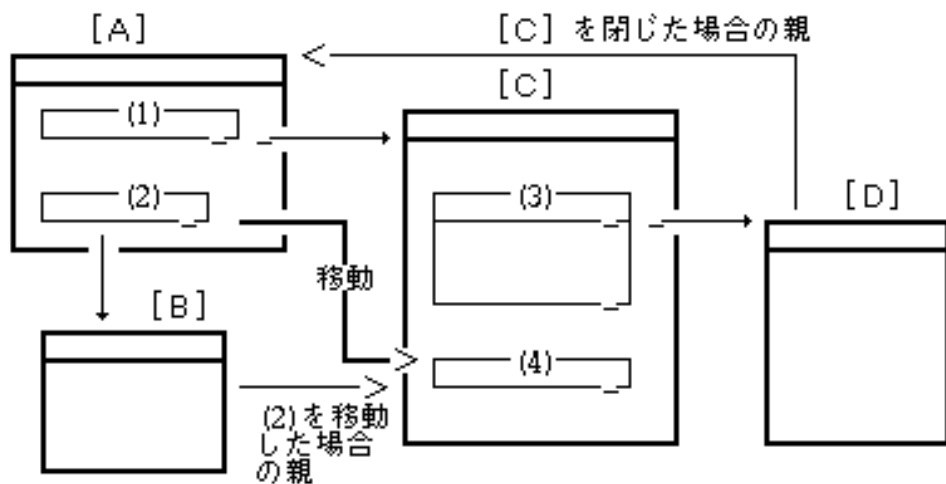
主ウィンドウの親ウィンドウは、ウィンドウのオープン時に指定されるが、オープン後も変更することが可能である。また、ウィンドウが閉じられた場合、そのすべての子ウィンドウの親ウィンドウは閉じたウィンドウの親ウィンドウに自動的に変更される。

主ウィンドウ以外のウィンドウに対しては、親ウィンドウは定義されない。

親を持つウィンドウにはその生成の元となった仮身の表示上の領域が定義され、この領域を生成元と呼ぶ。生成元は、親ウィンドウ内の相対座標により定義される。

生成元も、ウィンドウのオープン時に指定されるが、オープン後も変更することが可能である。また、ウィンドウが閉じられた場合、そのすべての子ウィンドウの生成元は閉じたウィンドウの生成元と同じものに自動的に変更される。

なお、仮身の移動等に伴う、親ウィンドウおよび生成元の変更は、実身/仮身マネージャにより行なわれる。



- ・ [C] の親は [A] であり、生成元は (1) となる。
- ・ [D] の親は [C] であり、生成元は (3) となる。 [C] が閉じられた場合は、親は [A] となり、生成元は (1) となる。
- ・ [B] の親は [A] であり、生成元は (2) であるが、(2) の仮身を [C] の (4) に移動した場合は、親を [C]、生成元を (4) と変更する必要がある。

図 46 : ウィンドウの親子関係

## ウィンドウのモード

主ウィンドウには、全面モードと通常モードが定義され、交互に切り換えが可能となる。

全面モードでは、主ウィンドウは画面一杯に拡がって表示され、通常モードに戻すことにより、元の位置、大きさの表示に戻る。

全面モードで表示されている状態で、ウィンドウの移動/変形を行なった場合は、そのウィンドウの全面モード状態は解除され、通常モードの表示と見なされることになる。

主ウィンドウ以外のウィンドウに対しては、このモードは定義されない。

全面モード、通常モードの他に、システムメッセージパネルを含めたスクリーン全部に主ウィンドウを表示することができるフルスクリーンモードがある。この場合、システムメッセージパネルは隠れて見えなくなる。

### 3.1.1.3 ウィンドウの表示

#### ウィンドウの標準形状

ここでは、通常ウィンドウ、および前面ウィンドウの標準的な表示形状に関して説明しているが、表示形状の詳細はインプリメントに依存する。内部ウィンドウ、パネルウィンドウに関しては、ウィンドウマネージャでは表示を行なわないため表示形状は定義されない。

ウィンドウは、標準的に下図のように右下に影が付いた境界線により囲まれた矩形領域として表示される。



図 47 : ウィンドウの標準的な境界線

ウィンドウには、通常、最上部にタイトルバーが存在する。タイトルバーはピクトグラムと任意のタイトル文字列からなるが、ピクトグラムは無い場合もある。このタイトルバーの表示は虚身ウィンドウの状態では灰色化表示となる。

ウィンドウの大きさがアプリケーションにより固定されていて変形不可の場合は、作業領域の境界は上辺はタイトルバーのすぐ下であり、タイトルバーとの間に細線で作業領域の境界線が引かれる。下、左、右辺はウィンドウの境界線が作業領域の境界線となる。

これに対し、変形可能なウィンドウでは、作業領域の境界は下、左、右辺においてウィンドウの境界線から半文字幅程度内側の部分となり、細線で作業領域の境界線が引かれる。

タイトルバーを含む、作業領域の境界線とウィンドウの境界線間の部分を総称してウィンドウの枠と呼ぶ。

変形可能なウィンドウの場合は、枠の4隅の部分はハンドルでありウィンドウ変形の手掛かりとなる。また、スクロールバーが存在する場合は、右、下、(さらに3方向の場合は左)の枠上に表示され、ハンドルとスクロールバーを除く枠の他の部分は、ウィンドウ移動の手掛かりとなる。

入力受付状態のウィンドウでは、枠部分は灰色パターンで埋められ、スクロールバーは、正常に表示される。この時タイトルバーは、角の丸い矩形で囲まれ、パターンで埋められない。

入力不可状態のウィンドウでは、枠部分は白くなり、スクロールバーは不能状態表示となり、灰色部分は境界線を残して白くなりトンボは表示されない。また作業領域内の各パーツ類の表示も不能状態表示となる。

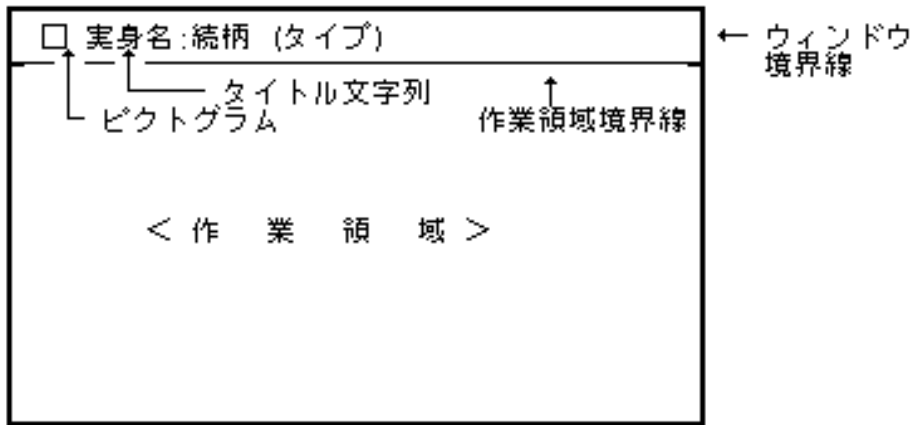


図 48 : 変形不可のウィンドウの標準的な形状

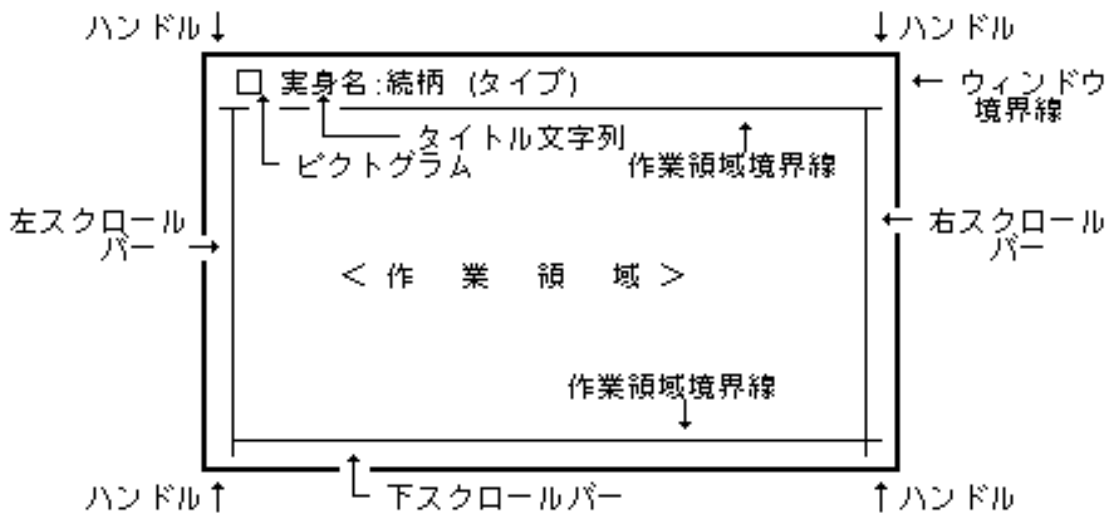


図 49 : 変形可能なウィンドウの標準的な形状

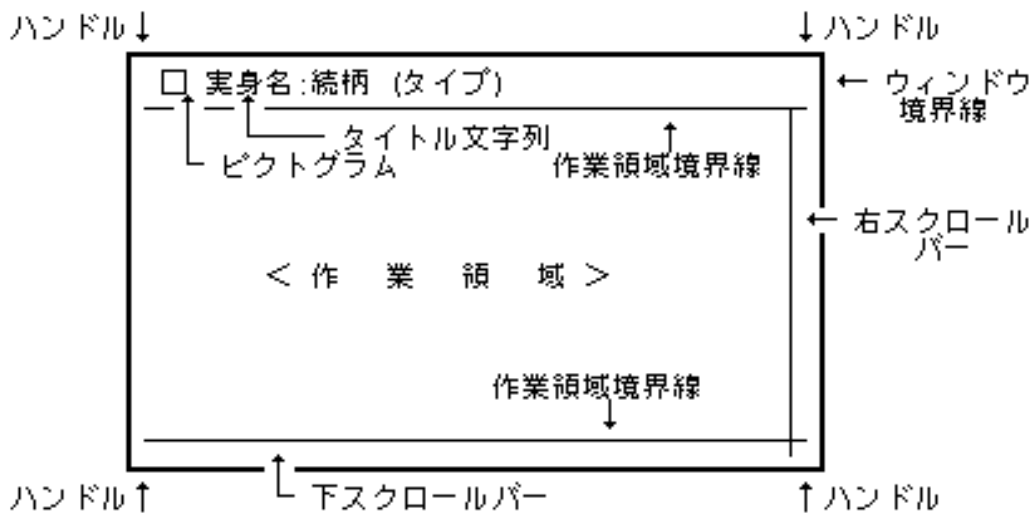


図 50 : 変形可能で左スクロールバーのないウィンドウの標準的な形状

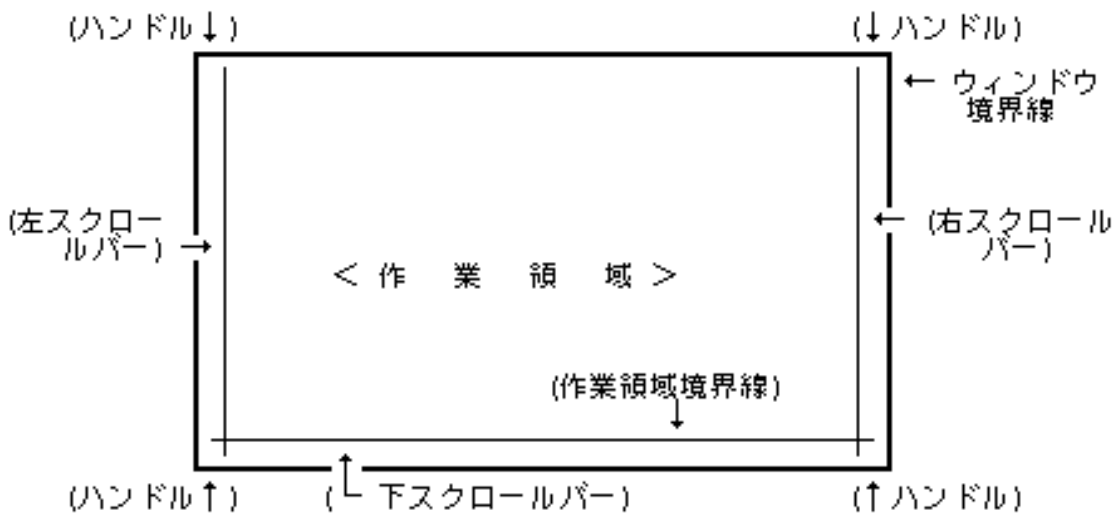


図 51 : タイトルバーのないウィンドウの標準的な形状

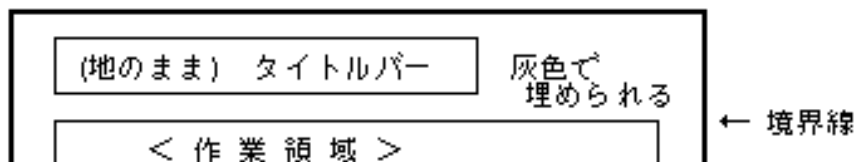


図 52 : 入力受付状態の標準的なタイトルバー表示

### ウィンドウの属性

ウィンドウには、そのタイプ、表示形状を表わすための以下に示す属性があり、ウィンドウの生成時に指定することができる。ウィンドウの属性は生成後は変更することはできない。内部ウィンドウ、パネルウィンドウに対してはウィンドウ属性は定義されない。

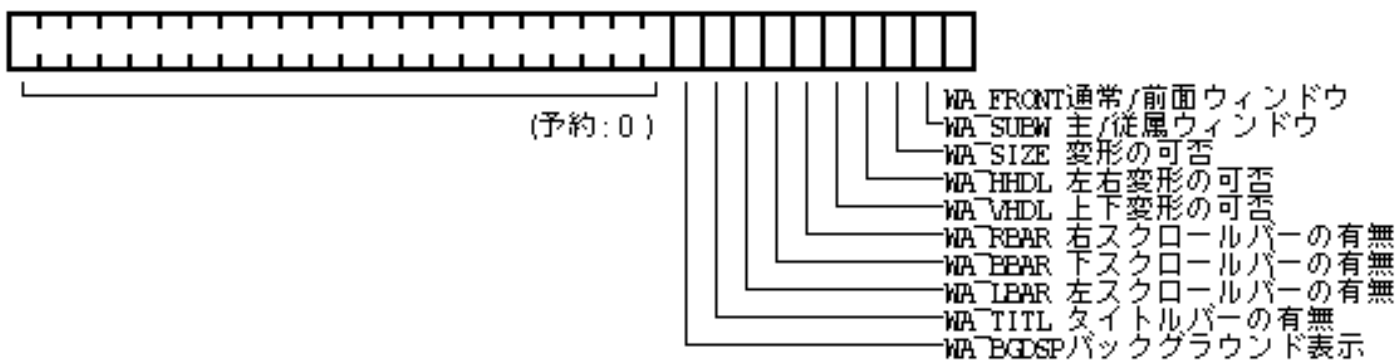


図 53 : ウィンドウの属性

|          |                    |                      |
|----------|--------------------|----------------------|
| WA_FRONT | 0 = 通常ウィンドウ        |                      |
|          | 1 = 前面ウィンドウ        |                      |
| WA_SUBW  | 0 = 主ウィンドウ         | (WA_FRONT = 0の時のみ有効) |
|          | 1 = 従属ウィンドウ        |                      |
| WA_SIZE  | 0 = ドラッグ変形不可のウィンドウ |                      |
|          | 1 = ドラッグ変形可能のウィンドウ |                      |
|          | 0 = 左右ドラッグ変形不可     |                      |

|          |                       |                     |
|----------|-----------------------|---------------------|
| WA_HHDL  | 1 = 左右ドラッグ変形可能        | (WA_SIZE = 1の時のみ有効) |
| WA_VHDL  | 0 = 上下ドラッグ変形不可        |                     |
|          | 1 = 上下ドラッグ変形可能        |                     |
| WA_RBAR  | 0 = 右スクロールバーの無        |                     |
|          | 1 = 右スクロールバーの有        |                     |
| WA_BBAR  | 0 = 下スクロールバーの無        |                     |
|          | 1 = 下スクロールバーの有        |                     |
| WA_LBAR  | 0 = 左スクロールバーの無        |                     |
|          | 1 = 左スクロールバーの有        |                     |
| WA_TITL  | 0 = タイトルバー有のウィンドウ     |                     |
|          | 1 = タイトルバー無のウィンドウ     |                     |
| WA_BGDSP | 0 = 入力不可状態の時は表示を更新しない |                     |
|          | 1 = 入力不可状態の時も表示を更新する  |                     |

WA\_SIZE, WA\_TITL によりウィンドウの形状は決定される。

WA\_HHDL, WA\_VHDL が両方とも 1 の場合は任意方向へのドラッグ変形が可能なことを意味する。両方とも 0 の場合はハンドルが存在しないことになる。

WA\_BGDSP 指定のウィンドウにのみ、パネルの消去時に再表示要求が送られる。

#### ウィンドウの表示属性

ウィンドウの生成時に以下に示す表示属性を指定することができる。ウィンドウの表示属性は生成後は変更することはできない。内部ウィンドウ、パネルウィンドウに対しては表示属性は定義されない。なお、カラーの実際の適用、および解釈方法はインプリメントに依存する。

```
typedef struct {
    UW    frame;      /* 境界線幅/パターンデータ番号 */
    UW    tlbfg;     /* タイトル背景パターンデータ番号 */
    UW    barpat;    /* スクロールバーのパターン部のパターン番号 */
    UW    barbg;     /* スクロールバーの白部分のパターン番号 */
    COLOR ticol;     /* タイトル文字色(負のときは default) */
} WDDISP;
```

frame の内容は以下の通りである。

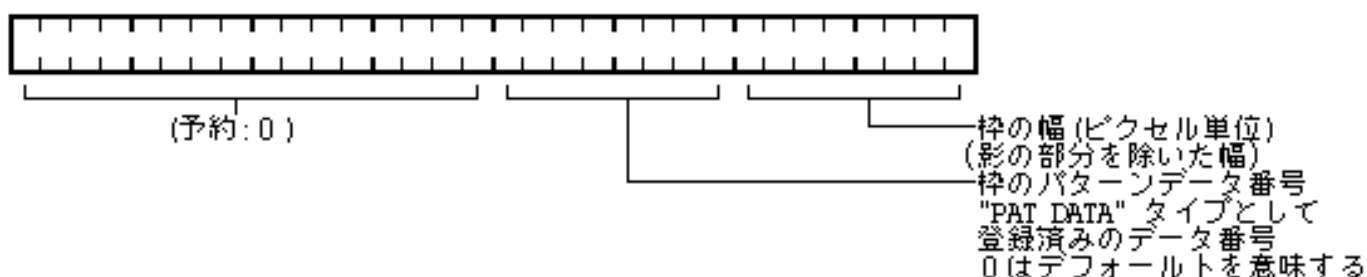


図 54 : frame の内容

tlbg, barpat, barbg はそれぞれ、"PAT\_DATA" タイプとして登録してあるデータ番号で指定し、0 はデフォルトを意味する。

tlcol は、タイトル表示の色を指定し、<0の場合は、デフォルトを意味する。

デフォルト値はウィンドウマネージャにより適当な値が規定されているが、デフォルト値自体も変更可能である。デフォルト値を変更した場合は、変更以後に生成された全てのデフォルト属性を持つウィンドウに反映される。

さらに、ウィンドウの作業領域の背景パターンを、ウィンドウの生成時に指定することができる。背景パターンは、基本的にディスプレイプリミティブで定義されるパターンデータの形式で指定される。

なお、背景パターンはウィンドウの生成後も変更することができる。

#### ポインタの形状

PD がプレスされていない状態の場合、ポインタの形状はPD の位置により、その位置での可能な操作を意味する以下の形状に自動的に変更される。ただしPD がプレスされている状態では、ポインタの形状はその位置により変更はされない。

「命令」キーが押された状態では、ポインタの形状は、その位置に無関係に常に「プリメニュー」の形状に変更される。

「命令」キーが押されていない状態では、ポインタの位置に応じて、以下に示すようにポインタ形状が変更される。

|                   |                     |                   |         |   |
|-------------------|---------------------|-------------------|---------|---|
| 入力受付状態の<br>ウィンドウ内 | ハンドル部分              | ウィンドウ属性として左右変形のみ可 | 横方向の変形手 | PD をプレスしてウィンドウのドラッグ変形を行っている間は、「つまみ」の形状となる。  |
|                   |                     | ウィンドウ属性として上下変形のみ可 | 縦方向の変形手 |   |
|                   |                     | ウィンドウ属性として任意の変形が可 | 変形手     |   |
|                   | スクロールバーの<br>見えている部分 | 右/左のスクロールバー       | 縦方向の移動手 | PD をプレスしてジャンプスクロールのドラッグを行っている間は、「握り」の形状となる。 |
|                   |                     | 下のスクロールバー         | 横方向の移動手 |   |
|                   | スクロールバーの見えていない灰色部分  |                   | 選択指     |   |
|                   | ピクトグラムの部分           |                   | 選択指     | PD をプレスしてウィンドウのドラッグ移動を行っている間は、「握り」の形状となる。   |
|                   | 上記以外のウィンドウの枠部分      |                   | 移動手     |   |
| ウィンドウの作業領域内       |                     | 形状は変更しない。         |         |   |
| 入力受付状態のウィンドウ外     |                     |                   | 選択指     |   |

入力不可状態のウィンドウで、PD をプレスした場合はそのウィンドウに入力受付状態が移行するが、プレス位置が上記の場合は、ポインタの形状も入力受付状態の移行と同時に変化する。

このポインタ形状の変更は、ウィンドウマネージャがウィンドウイベントの処理の一貫として自動的に行なわれるため、アプリケーションは、作業領域外にポインタが存在する場合は、そのポインタ形状を変更してはいけない。逆に、アプリケーションは、入力受付状態のウィンドウの作業領域内にポインタが存在する場合、そのポインタ形状を常に自分で設定しなくてはならない(外側からポインタが作業領域内に移動してきた場合、そのポインタ形状は保証されない)。

#### 3.1.1.4 ウィンドウの操作

以下に示すウィンドウの操作はアプリケーションを含めた全体として実現すべきウィンドウの操作であり、アプリケーションはウィンドウマネージャで用意されている機能を使用してウィンドウ操作を実現することになる。

ウィンドウの入力受付状態とウィンドウの表示戦略はインプリメントに依存する。下記は、インプリメントの一例である。

##### ウィンドウの生成

- 以下の操作により、選択した仮身がウィンドウに開かれ、その仮身の位置にウィンドウは生成されて最前面に表示される。
  - 実行メニューによりアプリケーションを起動する。
  - ピクトグラムをダブルクリックする。
- 小物メニューによりある種の小物アプリケーションを起動することにより、小物ウィンドウが開かれて最前面に表示される。小物によっては、前面ウィンドウが開かれる場合がある。
- 従属ウィンドウは、既に開かれている主ウィンドウのアプリケーションにより直接開かれる。
- 生成されたウィンドウは、常に入力受付状態となる。

##### 入力受付状態の変更

- ウィンドウ内でクリック(プレス)によりそのウィンドウは入力受付状態となり、最前面に表示される。
- ウィンドウの生成元である仮身(影付き)に対しウィンドウに開く操作を行なった場合、新たなウィンドウは生成されず、既に存在している対応するウィンドウが入力受付状態となり、最前面に表示される。
- 表示管理メニューにより指定することにより、そのウィンドウは入力受付状態となり、最前面に表示される。
- 他のウィンドウが入力受付状態となることにより、そのウィンドウは入力受付状態でなくなる。

##### 移動

- ウィンドウ枠のハンドルと、スクロールバー以外の部分をドラッグすることによりウィンドウは移動できる。



- 枠をプレスした時点で、ポインタは一瞬「移動手」となる。ウィンドウが既に入力受付状態であった場合は、枠上にポインタが移動した時点で、「移動手」となる。
- ドラッグ中はポインタは「握り」となり、ウィンドウ枠の影がドラッグに合わせて移動する。
- ポインタをシステムメッセージパネル上に移動すると、影は消去される。ポインタをシステムメッセージパネル以外に移動すると、再び影が表示される。
- リリースにより、ポインタは「移動手」となり、その位置にウィンドウが移動し、入力受付状態となり最前面に表示される。システムメッセージパネル上でリリースした場合は、ウィンドウの移動は取り消される。
- なお、移動によりウィンドウの一部が画面からはみ出ることもある。

## 変形

- ウィンドウ枠のハンドルをドラッグすることによりウィンドウは変形できる。4つの変形ハンドルは、それぞれの対角点を固定して得られる矩形にウィンドウを変形する。
- ハンドルをプレスした時点で、ポインタは一瞬「変形手」となる。ウィンドウが既に入力受付状態であった場合は、枠上にポインタが移動した時点で、「変形手」となる。
- ドラッグ中はポインタは「つまみ」となり、ウィンドウ枠の影がドラッグに合わせて変形する。
- 変形の最大の大きさを越えた場合は、その大きさを影は追従しなくなる。
- ポインタをシステムメッセージパネル上に移動すると、影は消去される。ポインタをシステムメッセージパネル以外に移動すると、再び影が表示される。
- リリースにより、ポインタは「変形手」となり、その位置にウィンドウが変形され、入力受付状態となり最前面に表示される。システムメッセージパネル上でリリースした場合は、ウィンドウの変形は取り消される。
- 表示メニューの「全面モード」、およびハンドルのダブルクリックにより、全面モードと通常モードの交互の切り換えが行なわれる。

## 消去

- 以下の操作で、ウィンドウは消去される。
  - 「終了」メニューを選択する。
  - ピクトグラムをダブルクリックする。
- 仮身から開かれたウィンドウの場合は、ウィンドウが消去された後、その仮身を含むウィンドウが入力受付状態となる。

## ウィンドウ属性

```
#define WA_FRONT 0x0001 /* 前面ウィンドウ */
#define WA_SUBW 0x0002 /* 従属ウィンドウ */
#define WA_SIZE 0x0004 /* 変形可 */
#define WA_HHDL 0x0008 /* 左右変形可 */
#define WA_VHDL 0x0010 /* 上下変形可 */
#define WA_RBAR 0x0020 /* 右スクロールバー有 */
```

```

#define WA_BBAR    0x0040    /* 下スクロールバー有 */
#define WA_LBAR    0x0080    /* 左スクロールバー有 */
#define WA_TITL    0x0100    /* タイトルバー無 */
#define WA_BGDSP   0x0200    /* 入力不可状態の時も表示を更新 */
#define WA_FULL    0x0400    /* 全面モード */
#define WA_SFULL   0x0800    /* フルスクリーン */
#define WA_STD     0         /* 正規座標でウィンドウ枠を指定 */
#define WA_WORK    0x1000    /* 作業領域座標でウィンドウ枠を指定 */
#define WA_FRAME   0x2000    /* 外枠座標でウィンドウ枠を指定 */
#define WA_FMOVE   0x4000    /* オープン時、画面内に強制移動 */
#define WA_NORMAL  0x007c    /* 通常 (変形可、右/下スクロールバー) */

```

WA\_SIZE を指定していない場合は、WA\_HHDL で左右のウィンドウ枠が広くなり、WA\_VHDL で上下のウィンドウ枠が広がる。

WA\_FULL の指定により、はじめから全面モードでウィンドウを開くことができる。

WA\_STD、WA\_WORK、WA\_FRAME の指定により、入出力に使用する枠座標を指定することができる。これは、wopn\_wnd, wget\_sts, wmov\_wnd, wmov\_drg, wrsz\_wnd, wrsz\_drg に影響する。

WA\_FMOVE の指定により、wopn\_wnd 時にウィンドウが完全に画面からはみ出していた場合 (実際には 8~16 ドット程度の余裕を持つ)、最低でもウィンドウの一部が画面に表示されるように移動される。

WA\_SFULL の指定によるフルスクリーンのウィンドウを、複数開くことができるが、新しく開いたウィンドウが常に上に位置し、ウィンドウの配置を変更することはできない。

#### ウィンドウの表示属性

ウィンドウの生成時に以下に示す表示属性を指定することができる。ウィンドウの表示属性は生成後は変更することはできない。内部ウィンドウ、パネルウィンドウに対しては表示属性は定義されない。なお、カラーの実際の適用、および解釈方法はインプリメントに依存する。

```

typedef struct {
    UW    frame;    /* 境界線幅/パターンデータ番号 */
    UW    tlb;      /* タイトル背景パターンデータ番号 */
    UW    barpat;   /* スクロールバーのパターン部のパターン番号 */
    UW    barbg;    /* スクロールバーの白部分のパターン番号 */
    COLOR ticol;   /* タイトル文字色(負のときは default) */
} WDDISP;

```

## 3.1.2 ウィンドウの管理

### 3.1.2.1 ウィンドウとプロセス

ウィンドウには必ずその管理プロセスが存在し、管理プロセスによりそのウィンドウの表示 / 操作処理が行なわれる。ウィンドウを生成したプロセスがそのウィンドウの管理プロセスとなる。

ウィンドウに対する以下に示すような各種の操作は、すべてウィンドウの管理プロセスが自分で行なうことになる。

- ウィンドウの生成 / 削除
- ウィンドウの移動 / 変形 / 全面モード切り換え
- ウィンドウの作業領域の表示 / 操作

- スクロールバーの表示値の設定 / 変更

1つのプロセスは、複数のウィンドウの管理プロセスとなることもできる。また、主ウィンドウとその従属ウィンドウは必ず同一の管理プロセスにより管理される。

入力受付状態の主ウィンドウまたはパネルウィンドウの管理プロセスを、アクティブプロセスと呼び、ある時点で唯一存在することになる。アクティブプロセスはKB、PDからのイベントを順次取り出し、その処理を行なうことになる。

入力受付状態でないウィンドウの管理プロセスは、通常は待機状態であり、入力受付状態の切り換え待ちの状態となっている。この場合、待機状態とならず、ウィンドウ内の表示を続行しても構わない。ただし、この場合は表示のみであり、ユーザからの入力を受け取ることはできない。

従って、全体としてウィンドウを処理しているプロセス群は、入力受付状態をお互いにパスしあい、アクティブプロセスを切り換えながら動作することになる。入力受付状態は、通常PDの操作により切り換えられるが、それ以外にもプログラムで切り換えることができる。

前面ウィンドウは、常に入力受付状態であるが、イベントの取り扱いに関してはフロントエンドとしての動作をするため特殊な構造となっている。

前面ウィンドウの管理プロセスは前面ウィンドウを生成したプロセスであるが、実際の管理プロセスは現在のアクティブプロセスと見なされて処理される。即ち、前面ウィンドウに対する要求イベント等は、生成した管理プロセスではなく、現在のアクティブプロセスに対して送られる。ウィンドウは基本的にそのウィンドウを生成したプロセスにより管理され、基本的にはプロセス依存であるが、自分が生成した以外のウィンドウの管理情報等を参照する場合があるため、ウィンドウID自体はプロセスグローバルであり、またウィンドウの管理情報もすべてのプロセスからアクセスできる必要がある。

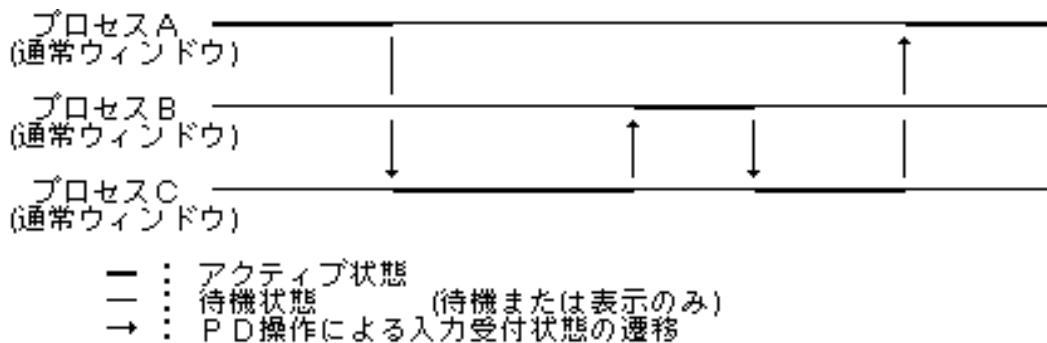


図 55 : アクティブプロセスの遷移

### 3.1.2.2 ウィンドウイベント

#### ウィンドウイベント

ウィンドウを使用するアプリケーションは、KBやPD等の操作に伴うイベントを順次取り出して対応する処理を行なうが、イベントの取り出しは、核のイベント管理機能を直接使用するのではなく、ウィンドウマネージャで提供しているウィンドウイベント機能を使用する。

ウィンドウイベント機能では、核のイベント管理機能に加えて以下のような機能が追加されている。

- PDの絶対座標位置から対応するウィンドウの相対座標への変換
- ウィンドウ操作関連の拡張イベントのサポート
- フロントエンド機能のサポート
- アクティブプロセスの切り換え機能

● ポインタ形状の変更

ウィンドウイベントの内容を以下に示す。

| type            | time   | pos<br>(r)       | data  |                                 | stat<br>(src)                                   |
|-----------------|--|------------------|---|---------------------------------|---|
|                 |  |                  | (cmd)   | (wid)                           |   |
| EV_NULL (0)     | 発生時刻   | P D 相対座標         | 位置コード   | WID                             | キー状態  |
| EV_BUTDWN (1)   | 発生時刻   | P D 相対座標         | 位置コード   | WID                             | キー状態  |
| EV_BUTUP (2)    | 発生時刻   | P D 相対座標         | 位置コード   | WID                             | キー状態  |
| EV_KEYDWN (3)   | 発生時刻   | P D 絶対座標         | キーコード   | 文字コード                           | キー状態  |
| EV_KEYUP (4)    | 発生時刻   | P D 絶対座標         | キーコード   | 文字コード                           | キー状態  |
| EV_AUTKEY (5)   | 発生時刻   | P D 絶対座標         | キーコード   | 文字コード                           | キー状態  |
| EV_DEVICE (6)   | 発生時刻   | P D 絶対座標         | 種別  | デバイスNO                          | キー状態  |
| EV_SWITCH (8)   | 発生時刻<br>未使用 (0)<br>生成元                           | P D 相対座標<br>矩形領域 | 位置コード<br>(W_SWITCH)<br>(W_CLOSED)                                 | WID<br>WID<br>WID               | キー状態<br>タイプ<br>ソース WID                          |
| EV_REQUEST (9)  | 未使用 (0)<br>発生時刻<br>未使用 (0)<br>未使用 (0)<br>未使用 (0) | P D 相対座標<br>仮身情報 | (W_REDISP)<br>(W_PASTE)<br>(W_FINISH)<br>(W_DELETE)<br>(W_OBJREQ) | WID<br>WID<br>WID<br>WID<br>WID | タイプ<br>ソース PID<br>ソース PID<br>ソース PID<br>ソース PID |
| EV_MENU (10)    | 発生時刻   | P D 絶対座標         | 位置コード   | WID                             | キー状態  |
| EV_APPL4 (11)   | ---  | ---              | ---   | ---                             | ---   |
| EV_APPL5 (12)   | ---  | ---              | ---   | ---                             | ---   |
| EV_APPL6 (13)   | ---  | ---              | ---   | ---                             | ---   |
| EV_APPL7 (14)   | ---  | ---              | ---   | ---                             | ---   |
| EV_APPL8 (15)   | ---  | ---              | ---   | ---                             | ---   |
| EV_RSWITCH (16) | 擬似イベント： EV_SWITCH と同じ                            |                  |   |                                 |   |
| EV_INACT (17)   | 擬似イベント： 未定義                                      |                  |   | WID                             | タイプ   |
| EV_MSG (18)     | 擬似イベント： プロセス間メッセージの先頭 4 バイト                      |                  |   |                                 |   |
| EV_NOMSG (19)   | 擬似イベント： 未定義                                      |                  |   |                                 |   |

※ EV\_NULL ~ EV\_BUTUP の pos は PD 絶対座標を取り出すことも可能  
 ※ --- : 元の内容のままを示す  
 WID : 対象ウィンドウ ID  
 ソース WID : クローズしたウィンドウ ID  
 ソース PID : EV\_REQUEST を発行したプロセス ID

図 56: ウィンドウイベント一覧

EV\_SWITCH、EV\_REQUEST、および EV\_MENU は、ウィンドウイベントとして新たに追加されたイベントである。これらのイベントはイベントタイプとして EV\_APPL1~3 を使用しているため、アプリケーションは、このタイプを使用してはいけない。また、EV\_RSWITCH ~ EV\_NOMSG は擬似イベントとして追加されているものである。

ウィンドウイベントタイプとして以下のようなタイプが定義されており、イベントのタイプに応じて、EVENT、SEVENT、REVENT の 4 つの形式を使い分けて使用する。

```
typedef struct SEVENT { /* 切換イベント形式 */
    W      type;
    UW    time;
    PNT   pos;
    H     cmd; /* 位置コードコマンド */
    H     wid;
    UW    stat;
```

```
} SEVENT;
```

```
typedef struct REVENT { /* 要求イベント形式 */
    W    type;
    RECT r; /* 対象矩形領域 (相対座標) */
    H    cmd;
    H    wid; /* 対象ウィンドウID */
    W    src; /* ソースPID/他 */
} REVENT;
```

```
typedef struct GEVENT { /* 一般イベント形式 */
    W    type;
    H    data[4]; /* 任意のデータ */
    H    cmd;
    H    wid;
    W    src;
} GEVENT;
```

```
typedef union _WEVENT {
    EVENT e; /* 生イベント(核イベント) */
    SEVENT s; /* 切換イベント形式 */
    REVENT r; /* 要求イベント形式 */
    GEVENT g; /* 一般イベント形式 */
} WEVENT;
```

EV\_NULL

有効なイベントが何も存在していない場合に得られるイベントであり、PDの位置するウィンドウID、位置コード、およびPDの現在位置(対応するウィンドウの相対座標)が設定されている。PDの現在位置は、イベントの取り出し時の指定により絶対座標のままとすることも可能である。PDの位置にウィンドウが存在しない場合はウィンドウID=0、システムメッセージパネル上であれば-1となる。

位置コードはPDの位置がウィンドウのどの位置であることを示す以下のコードである。

|             |            |          |                                     |
|-------------|------------|----------|-------------------------------------|
| 位置コード       | W_WORK (0) | 作業領域内    | 対応する部分が存在しないウィンドウの場合は、すべてW_FRAMとなる。 |
|             | W_FRAM (1) | ウィンドウ枠   |                                     |
|             | W_PICT (2) | ピクトグラム   |                                     |
|             | W_TITL (3) | タイトル文字列  |                                     |
|             | W_LTHD (4) | ハンドル(左上) |                                     |
|             | W_RTHD (5) | ハンドル(右上) |                                     |
|             | W_LBHD (6) | ハンドル(左下) |                                     |
|             | W_RBHD (7) | ハンドル(右下) |                                     |
|             | W_RBAR (8) | 右スクロールバー |                                     |
|             | W_BBAR (9) | 下スクロールバー |                                     |
| W_LBAR (10) | 左スクロールバー   |          |                                     |

このイベントは、アクティブプロセスのみが得ることができる。

EV\_BUTDWN

PD のプレスにより発生するイベントであり、PD の位置するウィンドウ ID、位置コード、および PD のプレス位置 ( 対応するウィンドウの相対座標 ) が設定されている。PD のプレス位置は、イベントの取り出し時の指定により絶対座標のままとすることも可能である。

このイベントは、アクティブプロセスのみが得ることができる。

PD のプレスは、その状態 / 位置により、EV\_MENU、EV\_SWITCH または EV\_BUTDWN のイベントとして取り出される。

#### EV\_MENU

メニューボタン、または命令キーが同時に押された時に得られる。

#### EV\_BUTDWN

対応するウィンドウが現在入力受付状態の主ウィンドウまたは従属ウィンドウである場合に得られる。

#### EV\_SWITCH

対応するウィンドウが現在入力不可状態のウィンドウである場合に、入力受付状態の切換えを行ない、新たにアクティブプロセスとなったプロセスが EV\_SWITCH を得る。イベント取り出し時の指定により、入力受付状態の切換えを行わず、EV\_BUTDWN として取り出すこともできる。

#### EV\_BUTUP

PD のリリースにより発生するイベントであり、PD の位置するウィンドウ ID、位置コード、および PD のリリース位置 ( 対応するウィンドウの相対座標 ) が設定されている。PD のリリース位置は、イベントの取り出し時の指定により絶対座標のままとすることも可能である。

このイベントは、アクティブプロセスのみが得ることができる。

#### EV\_KEYDWN/EV\_KEYUP/EV\_AUTKEY

KB のプレス、リリース、連続プレスにより得られるイベントであり、核のイベントマネージャで得られたイベントそのものである。従って、PD 位置は絶対座標となる。

このイベントは、アクティブプロセスのみが得ることができる。

#### EV\_DEVICE/EV\_RSV

#### EV\_APPL4 ~ 8

核のイベントマネージャで得られたイベントそのものである。このイベントは、アクティブプロセスのみが得ることができる。

#### EV\_SWITCH

入力受付状態のウィンドウの切換えが発生したことを示すイベントであり、切換えの結果、新たにアクティブプロセスとなったプロセスのみが得ることのできるイベントである。

このイベントを取り出した時点では、既に wid で示されるウィンドウに入力受付状態は切り換わっており、アプリケーションはそのウィンドウに対する通常の実行を行なうことになる。イベント内の cmd は以下の内容であり、切り換えの状態または原因を示す。

cmd : 位置コード (W\_WORK (0) ~ W\_LBAR (10))  
W\_SWITCH (128)  
W\_CLOSED (129)

位置コードの場合は、PD のプレスにより入力受付状態の切換えが発生したことを意味し、プレスした位置の位置コード、および相対座標位置が入っている。アプリケーションはこのイベント

を EV\_BUTDWN として処理する必要がある。

W\_SWITCH の場合は、アプリケーションからの要求による入力受付状態の切り換え、または、パネルのクローズによる入力受付状態の切り換えが発生したことを意味し、PD の位置に関する情報は存在しない。

src は以下の内容であり、切り換えの原因を示す。

```
src : 0    ウィンドウの切り換え
      1    パネルのクローズ
```

W\_CLOSED の場合は、子ウィンドウがクローズされたため親ウィンドウに入力受付状態が切替わったことを意味し、この時点では、既に子ウィンドウはクローズングアニメーションとともに削除されている。src には、クローズした子ウィンドウのウィンドウIDが入り、r には、生成元の矩形領域が相対座標で入っている。

## EV\_REQUEST

ウィンドウマネージャ、または他のプロセスからの各種の処理の要求およびそれに対する応答を示すイベントであり、対象とするウィンドウの管理プロセスが得るものであり、アクティブプロセスと非アクティブプロセスの両方から取り出される。

cmd は要求する処理内容を示す以下のものである。

```
cmd : W_REDISP (0)  -- ウィンドウの再表示要求
      W_PASTE  (1)  -- データの貼り込み要求
      W_DELETE (2)  -- ウィンドウのクローズ要求
      W_FINISH (3)  -- 処理の終了要求
      W_VOBJREQ(4)  -- 仮身の再表示要求
      W_OPENED (5)  -- ウィンドウのオープン通知
      :           :
```

```
W_ACK (0x40)+(要求コード) -- 要求に対するACK応答
```

```
W_NAK (0x80)+(要求コード) -- 要求に対するNACK応答
```

W\_REDISP は、ウィンドウの再表示を要求するもので、ウィンドウの操作に伴って、ウィンドウの再表示が必要になった場合にウィンドウマネージャが発生するイベントである。このイベントを得た場合は、アプリケーションは対応するウィンドウの再表示を速やかに行なわなくてはならない。

```
src =0 : 通常再表示要求
      =1 : パネル再表示要求
```

W\_REDISP 以外のものは、ウィンドウマネージャとしては特別な処理を行わず、要求イベントの送付、および応答イベントの送付を行なう。要求イベントの src にはウィンドウマネージャにより要求したプロセス ID が設定され、応答イベントの宛先として使用される。

各要求イベントは、以下の意味を持つ。

W\_PASTE:

ドラッグによるデータの移動 / 複写のために使用され、移動 / 複写先のウィンドウに対して移動 / 複写元のプロセスがこのイベントを送信する。  
このイベントを受信したプロセスは速やかに要求された貼り込み処理を行ない、必ず、W\_ACK または W\_NAK の応答を戻さなくてはならない。

W\_DELETE:

ウィンドウのクローズ要求のために使用される。このイベントを受信したプロセスは速やかにウィンドウのクローズ処理を行ない、必ず W\_ACK (または W\_NAK) の応答を戻さなくてはならない。

W\_FINISH:

処理の終了要求のために使用される。このイベントを受信したプロセスは即座に処理を終了し、必ず W\_ACK (または W\_NAK) の応答を戻さなくてはならない。

W\_VOBJREQ:

仮身に対する各種の操作 / 表示要求を行なうために使用され、実身 / 仮身マネージャが送信する。このイベントを受信したプロセスは、速やかに要求された処理を行なわなくてはならない。応答は不要である。詳細は、「[3.8 実身 / 仮身マネージャ](#)」を参照のこと。

W\_OPENED:

子プロセスが親プロセスに対して、新たにオープンしたウィンドウ ID を通知するために使用する。一般に応答は不要である。

EV\_MENU

メニューボタンがプレスされた場合、もしくは PD のプレス時に命令キーがプレスされていた場合に得られるイベントであり、メニューの表示を行なうことを意味する。PD 位置は絶対座標であるが、対応するウィンドウ ID および位置コードが設定されている。

このイベントは、アクティブプロセスのみが得ることができる。

EV\_RSWITCH

擬似イベントであり、EV\_SWITCH と同様の意味を持ち、得られるイベントの内容も、type 以外は EV\_SWITCH の場合と同一であるが、入力受付状態が切り換わったことにより、対応するウィンドウの再表示が必要であることを意味している。

EV\_INACT

擬似イベントであり、対応するウィンドウが入力受付状態から入力不可状態に切り換わった場合に得られる。イベントの内容には、入力不可状態に切り換わったウィンドウ ID が (wid) に設定されている。

src は以下の内容であり、切り換えの原因を示す。

|       |   |            |
|-------|---|------------|
| src : | 0 | ウィンドウの切り換え |
|       | 1 | パネルのオープン   |

このイベントは、EV\_SWITCH と必ず対になって得られる (ただし、EV\_SWITCH の W\_CLOSED コマンドの場合は、EV\_INACT は発生しない)。

このイベントは、アクティブプロセスのみが得ることができ、このイベントを得た以降は通常、アクティブプロセスでなくなる。

EV\_MSG

擬似イベントであり、通常のプロセス間メッセージが得られたことを意味する。イベントの内容には、type の後にメッセージの先頭4バイトが設定されている。メッセージ自体はメッセージキューに残っているので、rcv\_msg() によりメッセージを取り出す必要がある。このイベントは、アクティブプロセスと非アクティブプロセスの両方から取り出される。

EV\_NOMSG



擬似イベントであり、イベント、またはメッセージが発生していない場合に得られ、イベントの内容には type 以外は何も設定されない。このイベントは、非アクティブプロセスのみが得ることができ、EV\_NULL に相当する。

## ウィンドウイベントの処理

ウィンドウイベント処理は、要求したプロセスがアクティブプロセスか否かで大きくその内部処理が異なる。

アクティブプロセスの場合は、核イベントを取り出し、対応するウィンドウのサーチ、相対座標への変換、メニュー指定のチェック、ポインタ形状の変更、入力受付状態の切り換え等の処理を行なったのち、必要に応じて後述するフロントエンド処理を行ない、ウィンドウイベントとして戻す。

非アクティブプロセスの場合は、メッセージを取り出し、内部イベントメッセージとして得られた EV\_SWITCH、EV\_REQUEST をウィンドウイベントとして戻す。この場合、他のメッセージも受信するため、内部イベントメッセージ以外のメッセージを受信した場合は、その旨を戻すことになる。またメッセージ受信を待つか否かの指定を行なうことが可能で、待たない指定を行なった場合でメッセージが何も無かった場合は、その旨を戻す (EV\_NOMSG)。

入力受付状態でない (即ち非アクティブプロセス) 状態で、ウィンドウの表示を更新する場合は、待たない指定を行なう必要がある。

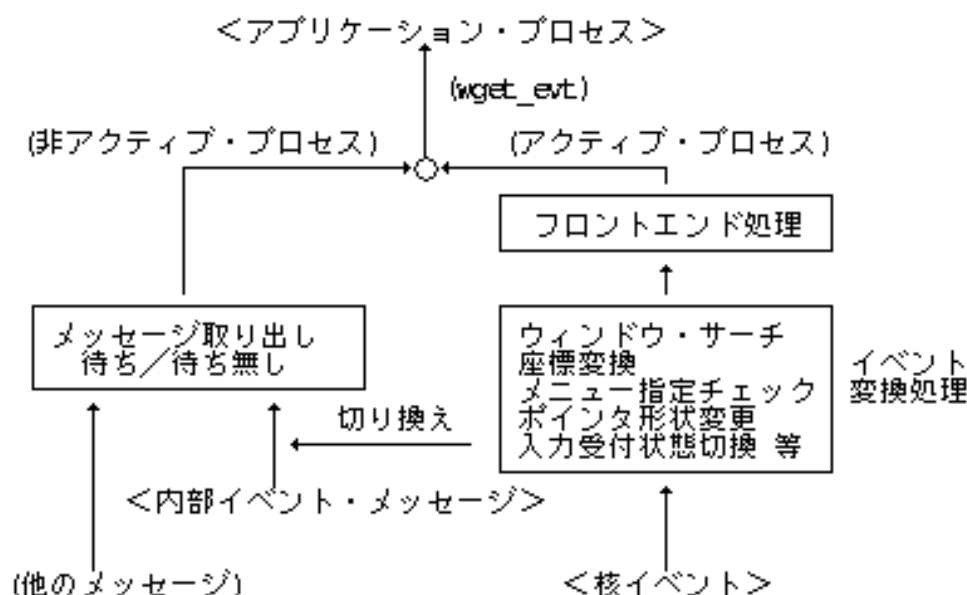


図 57: ウィンドウイベントの処理

内部イベントメッセージは入力受付状態の切り換え、ウィンドウの再表示要求、フロントエンド処理等の際に、ウィンドウイベントをウィンドウの管理プロセス間で受け渡すために使用されるメッセージであり、その詳細はウィンドウマネージャのインプリメントに依存する。

内部イベントメッセージ用に、MS\_MNG1 と MS\_MNG2 の 2 つのメッセージタイプが使用されるため、アプリケーションはこの 2 つのメッセージタイプを使用してはいけない。

## フロントエンド機能

フロントエンド機能は、アクティブプロセスが取り出す、ウィンドウイベントを横取りして、イベントの前処理等を行なう機能であり、「キーボード表小物」等の特殊なアプリケーションにより使用される。フロントエンド機能を使用する特殊なプロセスをフロントエンドプロセスと呼ぶ。

フロントエンドプロセスは、画面表示を必要とする場合、前面ウィンドウを使用することができ、各種のウィンドウ操作関数を特別な制限なしで使用することができる。逆に前面ウィンドウ以外のウィンドウを使用することはできない。

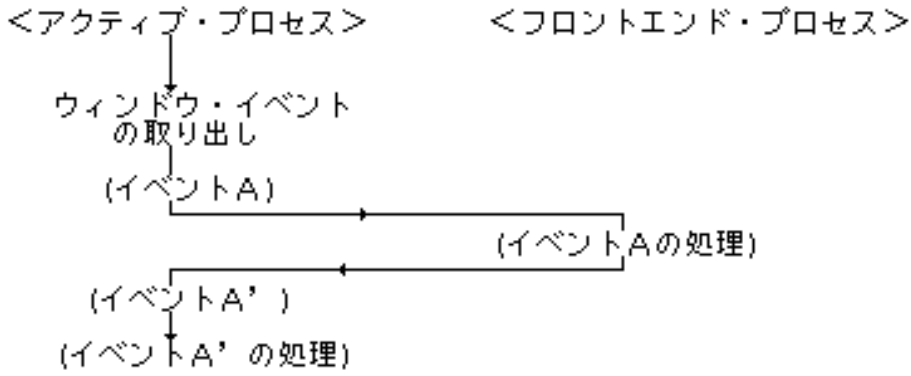


図 58 : フロントエンド機能の基本構造

フロントエンドプロセスは wdef\_fep(1) により定義され、同時に最大 4 個まで定義できる。フロントエンドプロセスは、すべての処理を終了した時点で wdef\_fep(0) により、フロントエンドプロセスでなくなるが、そのプロセスが終了した場合は自動的にフロントエンドプロセスとしての定義から外される。

フロントエンドプロセスが定義されている状態では、フロントエンドプロセスが実質的なアクティブプロセスとして取り扱われ、元のアクティブプロセスは擬似アクティブプロセスとなり、非アクティブプロセスと同様にメッセージ待ち状態となる。なお、フロントエンドプロセスの存在はウィンドウの表示には一切影響しない。

複数のフロントエンドプロセスが定義されている場合は、最後に定義されたフロントエンドプロセスが実質的なアクティブプロセスとして取り扱われ、他はすべて擬似アクティブプロセスとなり、メッセージ待ち状態となる。

フロントエンドプロセスが定義されている場合のウィンドウイベントの流れの詳細を以下に示す。

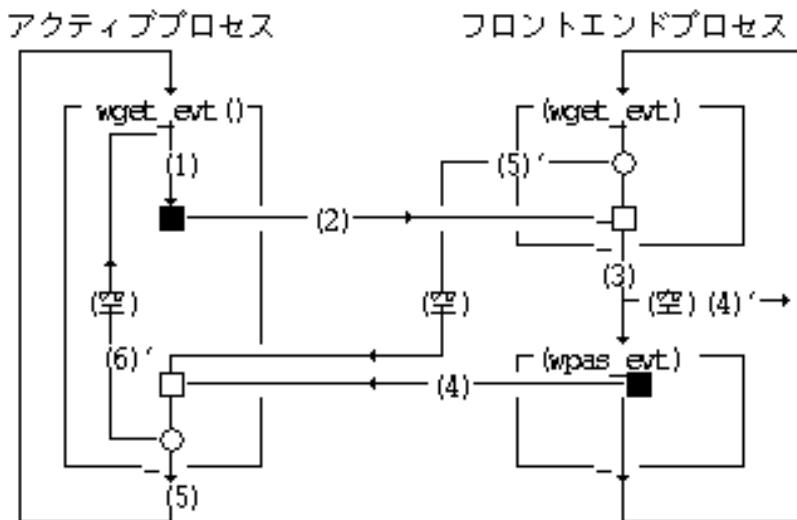


図 59 : フロントエンドプロセスへのイベント流れ(1)

- (1) アクティブプロセスがウィンドウイベントを取り出す。
- (2) フロントエンドプロセスに取り出したイベントを渡す。
- (3) フロントエンドプロセスは渡されたイベントを取り出して処理する。

- (4) フロントエンドプロセスは処理後のイベントを戻す。
- (5) アクティブプロセスはフロントエンドプロセスから戻された処理後のイベントを取り出す。

- (4)' フロントエンドプロセスが、イベントを処理して戻す必要が無い場合。
- (5)' アクティブプロセスに「空」イベントを戻す。
- (6)' アクティブプロセスは「空」イベントを受け取ると、再度ウィンドウイベントを取り出す。( (1)に戻る)

複数のフロントエンドプロセスが定義されている場合は、最後に定義されたフロントエンドプロセスに最初にイベントが渡される。この場合のウィンドウイベントの流れの詳細を以下に示す((4)'(5)'のパスは省略している)。

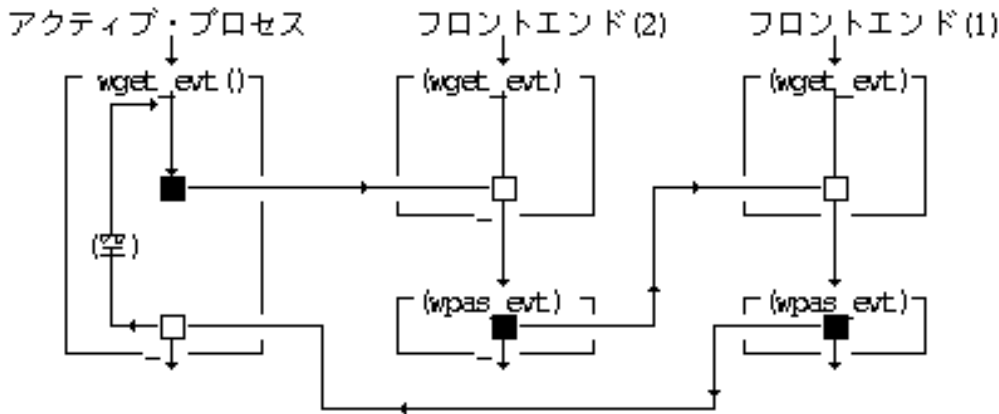


図 60 : フロントエンドプロセスへのイベント流れ(2)

フロントエンドプロセスの標準的な動作は以下に示すものとなる。

1. wdef\_fep(1) によりフロントエンドプロセスとなる。
2. 通常のプロセスと同様に、wget\_evt (\*evt, WAIT) により、イベントを取り出して、その処理を行なうイベントループの構造となる。wpas\_evt() によりイベントを戻す点のみが通常のプロセスと異なることになる。
3. 得られたイベントの処理は、基本的に以下の3種類のいずれかとなる。

1. そのまま何もせずに wpas\_evt() で戻す

自分の前面ウィンドウ以外に対する EV\_REQUEST、自分の前面ウィンドウ以外での EV\_BUTDWN 等

2. 対応する処理を行なった後、得られたイベントをそのまま、または変換して wpas\_evt() で戻す

EV\_NULL

ポインタ形状の変更、リンク / ちらつき処理等を行なった後、そのまま EV\_NULL を戻す。

キーボード小物の場合の例 :

EV\_KEYDOWN

対応するキー表示を反転し、必要ならキーコードを変換して戻す。

EV\_BUTDWN

表示されているキー上の場合、PD をトラックし、リリース時点のキー表示に対応する EV\_KEYDOWN を生成して戻す。

### 3. 対応する処理を行ない、イベントを戻さない( (空)イベントを戻す)

自分の前面ウィンドウに対する再表示要求等の EV\_REQUEST、自分の前面ウィンドウに対する PD プレスで、移動、変形操作等

4. wget\_evt()では、メッセージキューのチェックは行なわれないため、イベントループ内で自分でチェックする必要がある。
5. 処理を終了する場合は、wdef\_fep(0)により、フロントエンドプロセスであることをやめる。

フロントエンドプロセスには、wdef\_fep(1)で定義される一般フロントエンドプロセスと、wdef\_fep(-1)で定義される特殊フロントエンドプロセスの2つがある。

一般フロントエンドプロセスに渡されるイベントは、アクティブプロセスが、RAW EVT / DRGEVT 以外の指定で、wget\_evt()を実行して取り出したイベントのみであり、他の関数 (mov\_drg, wrsz\_drg, wget\_drg, wchk\_dck) で内部的に取り出されたイベント、および RAW EVT / DRGEVT 指定での wget\_evt() で取り出されたイベントは渡されない。

特殊フロントエンドプロセスには、他の関数 (wmov\_drg, wrsz\_drg, wget\_drg, wchk\_dck) で内部的に取り出されたイベントも含めて、すべてのイベントが渡される。これは、イベントのレコーディングやプレイバックの特殊用途に使用される。

#### 3.1.2.3 ウィンドウの再表示処理

ウィンドウの入力受付状態の切り換え、移動、変形、削除、全面モードの解除等の操作を行なった場合、他のウィンドウにオーバーラップされて隠れていた部分が見えてきた場合のウィンドウ内容の再表示を行なう必要がある。

この場合、実際の再表示は、そのウィンドウの管理プロセスが行なうことになるが、再表示が必要なウィンドウ、およびそのウィンドウ内の再表示が必要な領域の管理はウィンドウマネージャが行ない、ウィンドウの管理プロセスに対して、再表示要求イベントを発行する。

ウィンドウマネージャのインプリメントによっては隠れた部分のイメージを保持し、再表示を自動的に行なうことも考えられるが、この場合は管理プロセス側に再表示要求イベントが送られないことになるが、基本的にすべてのアプリケーションは再表示要求イベントに対する再表示処理を行なう必要がある。

各ウィンドウは、再表示が必要な領域を矩形の集合 (矩形リスト) として保持しており、ウィンドウの新たな再表示が必要になった時点で更新され、ウィンドウに対しての再表示要求イベントが発行される。この場合、操作の対象となったウィンドウ (通常は入力受付状態のウィンドウ) に対しては、再表示領域は更新されるが、再表示要求イベントは発行されず、ウィンドウ操作関数の関数値として再表示の必要性の有無が戻ることになる。

再表示要求イベントは、src フィールドの値により以下のタイプに分類される。

src = 0: 通常再表示要求

他のウィンドウの移動/変形等により再表示が必要になった場合に発生する。

src = 1: パネル再表示要求

パネルウィンドウが削除された場合に、パネルに隠された部分の描画をパネル表示中に行なった可能性があるウィンドウに対して発生する。

この場合は、パネルに隠された部分の描画を実際に行なわなかった場合は、再表示を行なう必要はないが、後述する wsta\_dsp() ~ wend\_dsp() は実行する必要がある。

アプリケーションは再表示要求イベントを受け取った場合、またはウィンドウ操作の結果として、対象ウィンドウの再表示が必要となった場合に、以下の処理を行なう必要がある。

- ウィンドウの描画環境の前置長方形リストは新しい配置のものが設定されており、他のウィンドウにはみだで描画できないことが保証されている。
- `wsta_dsp()` により再表示が必要な矩形領域を取り出すと共に、再表示処理の開始を宣言する。
- `wsta_dsp()` で取り出した矩形領域の再描画を行なう。この場合、作業領域全体の再描画を行なっても問題ないが、処理の高速化のためには最低限の描画が望ましい。
- 再表示が完了した場合は、`wend_dsp()` を実行し、再表示の完了を宣言する。

再表示領域の更新処理、および `wsta_dsp()`、`wend_dsp()` に対する処理は、ウィンドウマネージャにより以下のように行なわれる。

- 再表示領域が空の場合は、新たな再表示領域を設定し、再表示要求イベントを発行する。
- 再表示領域が空でなく、再表示処理中状態でない場合は、以前の再表示領域と新たな再表示領域をマージする。再表示要求イベントは発行しない。
- 再表示領域が空でなく、再表示処理中状態の場合は、再表示領域をペンディングとする。既にペンディングの再表示領域が存在する場合は、その領域と新たな再表示領域をマージする。再表示要求イベントは発行しない。
- `wsta_dsp()` が実行された場合は、再表示領域を戻し、その領域が空でない場合は、再表示処理中状態とする。
- `wend_dsp()` が実行された場合で、再表示処理中状態の場合は、再表示領域を空とする。この時、ペンディングの再表示領域が存在する場合は、その領域を再表示領域として設定し、`wend_dsp()` の関数値として "1" を戻す。ペンディングの再表示領域が存在しない場合は、関数値として "0" を戻す。再表示処理中状態でない場合は、何もせずに関数値 "0" を戻す。

複数のウィンドウの再表示が必要な場合、再表示要求イベントの送信順番は特に規定しないが、実際の画面上で自然に見えることが望ましい。

ウィンドウに対するある種の操作を行なった場合は、他のウィンドウに再表示要求イベントが送信される可能性があるが、アプリケーションは特にそのことを意識する必要はなく、また他ウィンドウの再表示の完了を待つ必要もないが、`wchk_dsp()` により、他ウィンドウの再表示領域の有無をチェックすることができる。

内部ウィンドウに対しては再表示要求イベントは送られず、内部ウィンドウを削除した場合には再表示要求イベントは発生しない。

パネルウィンドウに対しては再表示要求イベントは送られないが、パネルウィンドウを削除した場合には、以下に示す再表示要求イベントが発生する。

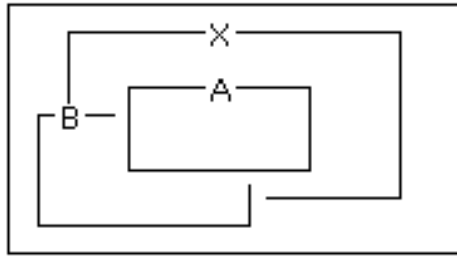
- CLR 指定でパネルウィンドウが閉じられた場合：  
パネルで隠れていたウィンドウに対して通常の再表示要求イベントが発生する。
- NOCLR 指定でパネルウィンドウが閉じられた場合：  
パネルで隠れていたウィンドウで、`WA_BG DSP` 属性を指定したウィンドウに対してパネル再表示要求イベントが発生する。

パネルウィンドウの消去時にも通常の再表示要求イベントが発生する。

前面ウィンドウに対しては、通常のウィンドウと同様に再表示要求イベントが送られ、前面ウィンドウの移動 / 変形 / 削除により、再表示要求イベントが発生する。ただし、前面ウィンドウに対する再表示要求イベントは前面ウィンドウの管理プロセスではなく、その時にアクティブプロ

セスに対して送られる点が通常ウィンドウの場合と異なる。

前面ウィンドウの再表示の動作例を以下に示す。



X : 通常ウィンドウ  
 A : 前面ウィンドウ  
 B : 前面ウィンドウ  
 A, Bの順に定義されている

Aを移動させた場合の再表示処理の流れを以下に示します。

□ : wget\_evt のコール      ■ : wget\_evt からのリターン  
 ○ : wpas\_evt のコール      ● : wpas\_evt からのリターン

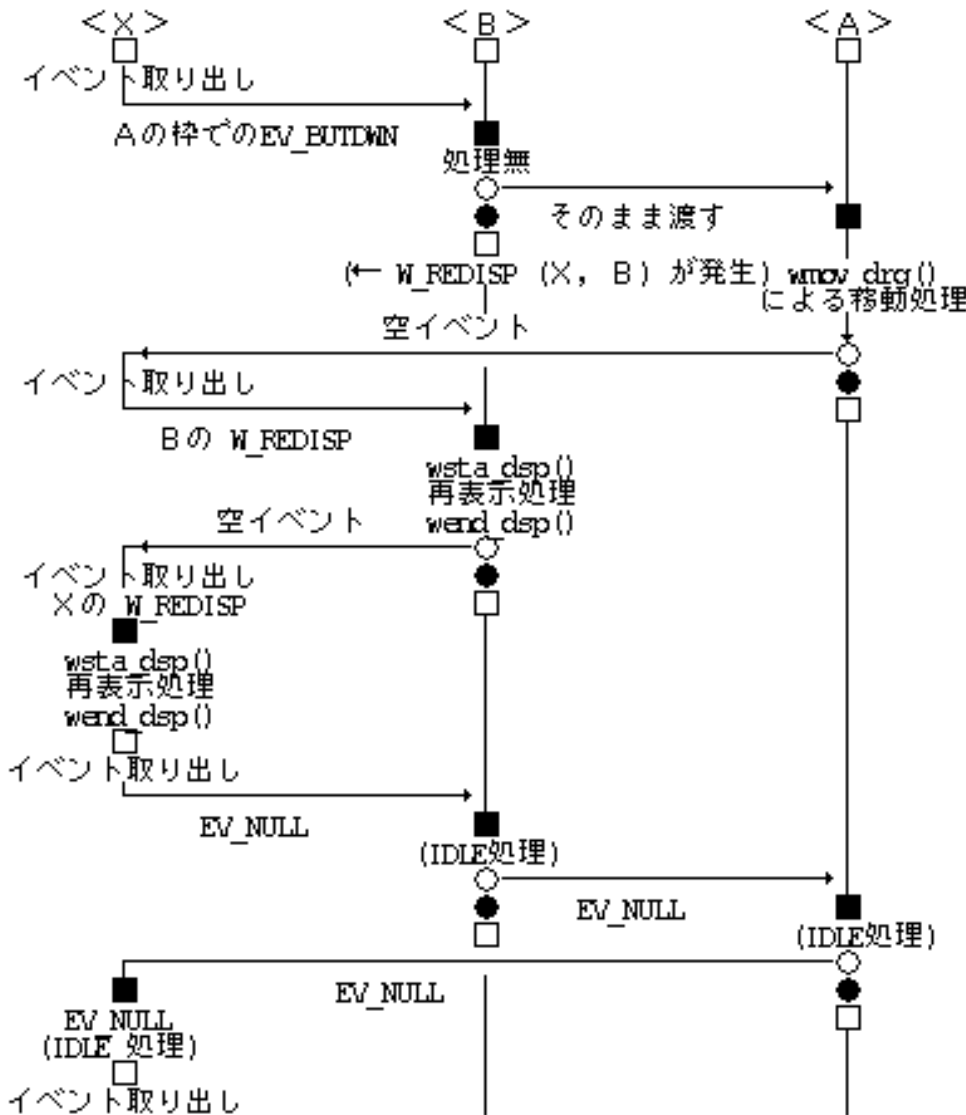


図 61 : 前面ウィンドウの再表示処理

### 3.1.2.4 ウィンドウの表示管理

ウィンドウの表示管理のための機能として、「表示管理メニュー」によるウィンドウの操作機能が提供される。

|       |                               |  |
|-------|-------------------------------|--|
| ウィンドウ | 一番後に                          | 積み重ねの一番後にし、二番目を一番前にする。   |
|       | =ウィンドウ # 1<br>≠ウィンドウ # 2<br>⋮ | 現在存在する主ウィンドウのタイトルのリストであり<br>入力受付状態にあるウィンドウが選択状態となってい<br>る。新たに選択したウィンドウに入力受付状態に切り<br>換わる。 |

図 62 : 表示管理メニュー

入力受付状態の切り換えでは、`wswi_wnd()` を実行した場合と同様の処理となり、切り換わった先のウィンドウに `W_SWICHTH` イベントが送られ、切り換わった元のウィンドウには `EV_INACT` が送られる。

### 3.1.2.5 ウィンドウの描画環境

#### ウィンドウの描画環境

アプリケーションプログラムはウィンドウマネージャにより生成されたウィンドウ内で表示を行なう必要があるが、アプリケーションプログラムからは直接ディスプレイ関数を使用することにより実際の描画を行なう。

ウィンドウマネージャはウィンドウ毎に独立した描画環境を生成するので、アプリケーションはウィンドウに対応する描画環境 ID を取り出して、その描画環境上でディスプレイ関数を実行して実際の描画を行なう。

ウィンドウ生成時に同時に生成される描画環境には、以下の設定が行なわれている。

- ビットマップ

画面メモリを示すビットマップ  
座標系は作業領域の左上が (0, 0) となる

- フレーム長方形

作業領域の矩形

- 表示長方形

作業領域の矩形

- 前置長方形リスト

ウィンドウの配置に対応した矩形リスト

- その他

描画環境の生成時の初期値

上記のうち、ビットマップ、フレーム長方形、前置長方形リストは、ウィンドウマネージャが管理しており、ウィンドウの移動/変形等に伴って変更されるため、アプリケーションでは変更してはならない。表示長方形、文字属性等のその他のデータはアプリケーションで自由に設定することができ、ウィンドウマネージャにより変更されることはない。

不可視状態の従属ウィンドウに対応する描画環境では、フレーム長方形が空 (0, 0, 0, 0) に設定され、描画を行なっても、実際には無視されることになる。

ウィンドウのオープン時には、作業領域の左上が (0, 0) となるように描画環境の座標系が設定され、これを相対座標と呼ぶ。

作業領域内は、オープン時に指定した背景パターンで塗り潰される。

ウィンドウの移動では作業領域の左上の座標値は変化しないが、ウィンドウの右下方向への変形以外のウィンドウの変形およびスクロール処理では、作業領域の左上の点はウィンドウの変化に対応して変化することになる。例えば、上方向へ20ドットスクロールした場合は、作業領域の点(0,0)は(0,20)となる。

また、ウィンドウを左上方向に(20,20)拡大した場合は、以下の図のようになる。

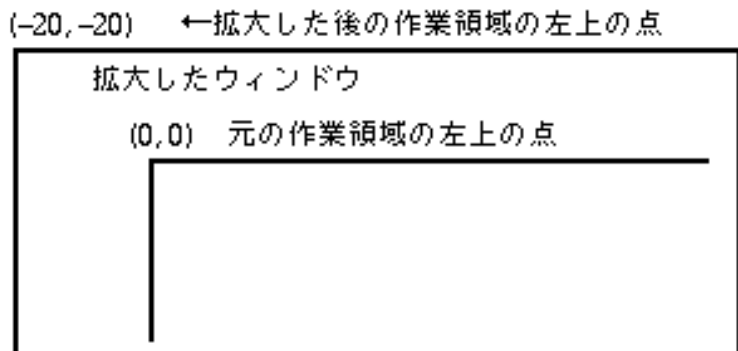


図 63 : ウィンドウ拡大時の作業領域の座標

また、ウィンドウの全面モードの切り換えの場合は、元の作業領域内のイメージの移動方法を含めて、座標系の設定方法を指定できるようになっている。

#### ドラッグの描画環境

アプリケーションによる描画は、ウィンドウに1対1対応した描画環境を使用するが、選択領域をドラッグした場合の「影」の描画は、ウィンドウの作業領域をはみ出て描画する必要もあるため、以下のような特別の方法を用いる。

1. `wsta_drg()` 関数により、ドラッグ用の描画環境を得る。この描画環境は、元のウィンドウの描画環境と、以下の設定が異なっている。
  - フレーム長方形 -- 画面全体(システムメッセージパネルは除く)
  - 表示長方形 -- 元のウィンドウの作業領域の矩形(相対座標)
  - 前置長方形リスト -- 元のウィンドウの前置長方形リストと同じ

`wsta_drg()` 関数のパラメータとしてロック指定を行なった場合は、他のウィンドウの同時描画は禁止される。元のウィンドウをはみ出て描画する必要がある場合は、ロック指定を必ず行なう必要がある。

2. アプリケーションは、得られた描画環境上で、ドラッグ影の描画を行なう。この描画環境上では、基本的にドラッグ影以外の描画を行なってはいけない。ドラッグ描画のためのPDの位置を取り出すために `wget_drg()` 関数が用意されており、描画環境レコードの座標系でのPDの座標位置を戻す。

この場合、元のウィンドウをはみ出た描画を行なう必要がある場合は、表示長方形を変更する必要がある。なお、描画環境の各種のパラメータは変更してもよい。

3. ドラッグ影の描画を終了した場合は、`wend_drg()` 関数を実行する。ロック指定を行なっていた場合は、ロックは解除される。また `wend_drg()` 実行後は、`wsta_drg()` で得られた描画環境を使用した描画は保証されない。

`wsta_drg()` により得られた描画環境の座標系は、元になったウィンドウの座標系が `wscr_wnd()` 関数、または `wset_wrk()` 関数により相対的に移動した場合には、自動的に座標系が一致するようにトラックされる。即ち、元のウィンドウのビットマップの座標系 (`bounds`) の移動量に従っ



て、ドラッグ用の描画環境のビットマップの座標系 ( bounds )、 フレーム長方形、 および表示長方形が平行移動することになる。

これによりドラッグ影の描画中に元のウィンドウの内容をスクロールしたり、座標系を変更した場合にも、正しいドラッグ影の描画が保証されることになる。ただし、サイズ変更、移動、および、全面モードの切り換えに関してはトラックされない。

### 3.1.2.6 ウィンドウ情報レコード

ウィンドウ情報レコードは、ウィンドウに関連する各種のデータを特定のシステムアプリケーションからアクセスするために用意されており、以下に標準的なデータを示すが、その詳細はインプリメントに依存する。

一般項目 ( WI\_GENERAL = 200 )

WI\_FRAMESIZE ( WI\_GENERAL + 0 )

スクリーン全体の領域。 フルスクリーンモードの場合は、これが作業領域となる。(取得のみ)

WI\_WORKSIZE ( WI\_GENERAL + 1 )

スクリーンからシステムメッセージパネルの領域を除いた領域。(取得のみ)

WI\_WINLIST ( WI\_GENERAL + 2 )

表示されている全ウィンドウのウィンドウIDの配列。手前に位置するウィンドウから順に並ぶ。(取得のみ)

WI\_FRONTLIST ( WI\_GENERAL + 3 )

存在する全フロントエンドプロセスのプロセスIDの配列。(取得のみ)

WI\_MSGSTR ( WI\_GENERAL + 4 )

システムメッセージパネルの文字列 ( 表示できる範囲のみ )。

WI\_MENUSTR ( WI\_GENERAL + 5 )

ウィンドウの基本メニュー文字列。

イネーブルウェア項目 ( WI\_ENABLEWARE = 300 )

WI\_TITLEFONT ( WI\_ENABLEWARE + 0 )

タイトルバーのフォントサイズ。

WI\_MENUFONT ( WI\_ENABLEWARE + 1 )

メニューのフォントサイズ。

WI\_SBARWIDTH ( WI\_ENABLEWARE + 2 )

スクロールバーの幅。

WI\_DRAGWIDTH ( WI\_ENABLEWARE + 3 )

移動 / 変形時のラバーバンド幅。

WI\_DCLICKT ( WI\_ENABLEWARE + 4 )

ダブルクリック間隔。

WI\_DCLICKW ( WI\_ENABLEWARE + 5 )

ダブルクリック位置許容度。

WI\_CARETT ( WI\_ENABLEWARE + 6 )

カレット点滅間隔。

WI\_CARETW ( WI\_ENABLEWARE + 7 )

カレット幅の倍率。

WI\_SELECTT (WI\_ENABLEWARE + 8)

選択枠点滅間隔。

WI\_SELECTW (WI\_ENABLEWARE + 9)

選択枠線幅の倍率。

修飾項目 (WI\_ATTRIBUTE = 400)

WI\_WORKBACK (WI\_ATTRIBUTE + 0)

背景画面のパターン構造体。

WI\_PANELBACK (WI\_ATTRIBUTE + 1)

パネル背景のパターン構造体。

その他 (WI\_OTHER = 500)

WI\_LASTCARPOS (WI\_OTHER + 0)

最後にカレットが表示された絶対座標。(取得のみ)

WI\_LASTCARTIME (WI\_OTHER + 1)

最後にカレットが表示されたシステム時刻。(取得のみ)

### 3.1.3 データ/定数の定義

#### ウィンドウイベント

```
typedef struct SEVENT { /* 切換イベント形式 */
```

```
    W    type;
```

```
    UW   time;
```

```
    PNT  pos;
```

```
    H    cmd;          /* 位置コード / コマンド */
```

```
    H    wid;
```

```
    UW   stat;
```

```
} SEVENT;
```

```
typedef struct REVENT { /* 要求イベント形式 */
```

```
    W    type;
```

```
    RECT r;          /* 対象矩形領域 (相対座標) */
```

```
    H    cmd;
```

```
    H    wid;        /* 対象ウィンドウID */
```

```
    W    src;        /* ソースPID/他 */
```

```
} REVENT;
```

```
typedef struct GEVENT { /* 一般イベント形式 */
```

```
    W    type;
```

```
    H    data[4];    /* 任意のデータ */
```

```
    H    cmd;
```

```
    H    wid;
```

```
    W    src;
```

```
} GEVENT;
```

```
typedef union _WEVENT {
```

```
    EVENT e;        /* 生イベント(核イベント) */
```

```

SEVENT s;          /* 切換イベント形式 */
REVENT r;          /* 要求イベント形式 */
GEVENT g;          /* 一般イベント形式 */
} WEVENT;

```

## ウィンドウイベントタイプ

```

#define EV_SWITCH    8 /* 切換イベント */
#define EV_REQUEST   9 /* 要求/応答イベント */
#define EV_MENU      10 /* メニューイベント */
#define EV_RSWITCH   16 /* 切換イベント(再表示要) */
#define EV_INACT     17 /* 入力不可状態に切換わった */
#define EV_MSG       18 /* 他のメッセージが得られた */
#define EV_NOMSG     19 /* メッセージは得られなかった */

```

## 位置コード / コマンド

```

#define W_WORK       0 /* 作業領域内 */
#define W_FRAM       1 /* ウィンドウ枠 */
#define W_PICT       2 /* ピクトグラム */
#define W_TITL       3 /* タイトル部 */
#define W_LTHD       4 /* ハンドル (左上) */
#define W_RTHD       5 /* ハンドル (右上) */
#define W_LBHD       6 /* ハンドル (左下) */
#define W_RBHD       7 /* ハンドル (右下) */
#define W_RBAR       8 /* 右スクロールバー */
#define W_BBAR       9 /* 下スクロールバー */
#define W_LBAR       10 /* 左スクロールバー */

#define W_SWITCH     128 /* 入力受付状態の移行 */
#define W_CLOSED     129 /* ウィンドウのクローズ */

```

## 要求イベント

```

#define W_REDISP     0 /* ウィンドウの再表示要求 */
#define W_PASTE      1 /* データの貼り込み要求 */
#define W_DELETE     2 /* ウィンドウのクローズ要求 */
#define W_FINISH     3 /* 処理の終了要求 */
#define W_OPENED     5 /* 子ウィンドウのオープン通知 */

#define W_ACK        0x40 /* 要求に対する ACK 応答 */
#define W_NAK        0x80 /* 要求に対する NACK 応答 */

```

## ウィンドウ削除

```

#define CLR          0 /* 表示をクリア */
#define NOCLR        8 /* 表示をクリアしない */

```

## ウィンドウイベント取得モード

```

#define WAIT          0 /* ウィンドウイベント待ち */
#define NOWAIT       1 /* ウィンドウイベント待たない */
#define NOMSG       0x0010 /* メッセージは取り出さない */
#define RAWEVT      0x0020 /* メニュー用(絶対座標) */
#define DRGEVT      0x0030 /* パーツ用 (相対座標) */

```

### 全面ウィンドウ/スクロール

```

#define W_NORM       0 /* 通常モード */
#define W_FULL      1 /* 全面モード */

#define W_MOVE      0 /* 座標系を固定する(イメージを移動) */
#define W_MOVEEC   1 /* 座標系を固定する(イメージを移動) */
#define W_HOLD     2 /* 座標系を移動する(イメージを固定) */

#define W_SCTL      1 /* 座標系を移動する */

#define W_VALID     1 /* スクロール表示は正当 */
#define W_NOOVL    2 /* オーバーラップ時はスクロールしない */
#define W_RDSET    4 /* 再表示領域を設定する */

```

### ウィンドウの状態定義

```

typedef struct wdstat {
    UW    attr; /* ウィンドウの属性 */
    W     parent; /* 親ウィンドウID/主ウィンドウID */
    W     pid; /* 管理プロセスID (0:無し) */
    RECT  r; /* ウィンドウの表示位置 (絶対座標) */
    RECT  wr; /* 作業領域の位置 (相対座標) */
    RECT  org; /* 生成元の矩形領域 ((0,0,0,0):無し) */
} WDSTAT;

```

### PD

```

#define W_PRESS     0
#define W_QPRESS   1
#define W_CLICK    2
#define W_DCLICK   3

```

## 3.1.4 ウィンドウマネージャの関数

wopn\_wnd

ウィンドウの生成

## 【形式】

WID wopn\_wnd(UW attr, UW par, RECT \*r, RECT \*org, W pict, TC \*tit, PAT \*bgpat,  
WDDISP \* atr)

## 【パラメータ】

|        |        |  |
|--------|--------|--|
| UW     | attr   | ウィンドウ属性  |
| UW     | par    | 主ウィンドウの場合には、親のウィンドウIDを示し、0は親が存在しないことを意味する。また従属ウィンドウの場合には、主ウィンドウIDを示す。主ウィンドウは同一プロセスにより管理されていなければならない、そうでない場合はエラーとなる。  |
| RECT   | *r     | ウィンドウの外枠の領域を指定した矩形領域であり、絶対座標で指定される。実行後には、実際に生成されたウィンドウの作業領域の相対座標が格納される(左上は常に(0,0)となっている)。attrによって、WA_STD 指定時には正規座標を、WA_WORK 指定時には作業領域座標を、WA_FRAME 指定時には外枠座標を、指定する。 |
| RECT   | *org   | ウィンドウの生成元の矩形領域であり、親ウィンドウの相対座標で指定される。存在しない場合はNULLとする。主ウィンドウでない場合、およびpar = 0の場合はこの指定は無視され、生成元は存在しないものと見なされる。   |
| W      | pict   | タイトルバーのピクトグラムのデータ番号を示すもので、0の場合はピクトグラムが存在しないことを意味する。  |
| TC     | *tit   | タイトルバーのタイトル文字列。  |
| PAT    | *bgpat | 作業領域の背景パターンの指定であり、生成されたウィンドウの作業領域は指定した背景パターンで塗り潰される。NULLまたは内容が不正な場合は、デフォルトの背景パターンとなる。  |
| WDDISP | *atr   | ウィンドウの表示属性の指定であり、NULLまたは内容が不正な場合は、デフォルトの表示属性となる。   |

## 【リターン値】

0 正常(ウィンドウID)  
< 0 エラー(エラーコード)

## 【解説】

指定したウィンドウ属性を持った、通常ウィンドウまたは前面ウィンドウを新たに生成し、そのウィンドウID(>0)を戻り値として戻す。このウィンドウIDは以後そのウィンドウの操作の際に使用する。

生成したウィンドウは、必ず入力受付状態となり前面に表示される。以前に入力受付状態であったウィンドウは入力不可状態となり、その管理プロセスに対してEV\_INACTが送られる。管理プロセスが自プロセスであった場合は、自分自身に対してEV\_INACTが送られることになる。ただし、別のプロセスがパネルを表示している場合は、パネルが削除されるのを待ってウィンドウがオープンされる。同じプロセスがパネルを表示している場合は、EX\_WNDのエラーとなる。アクティブプロセスのみが、従属ウィンドウを生成することが可能であり、そうでない場合は、EX\_WPRCのエラーとなる。従属ウィンドウの生成では、入力受付状態は変化しないため、EV\_INACTは発生しない。

また、前面ウィンドウの生成でも入力受付状態は変化しないため、EV\_INACTは発生しない。

フロントエンドプロセスのみが前面ウィンドウを生成することが可能であり、それ以外は EX\_WPRC のエラーとなる。また、フロントエンドプロセスで前面ウィンドウ以外をオープンした場合も、エラーとなる。

スクロールバーのあるウィンドウを生成した場合、スクロールバーの設定値を表示状況に合わせて適宜変更する必要がある。

atr の指定は意味を持たない。

### 【エラーコード】

- EX\_ADR : アドレス(r,org,title,bgpat,atr)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_WID : ウィンドウ(wid)は存在していない  
(指定した親/主ウィンドウは存在していない、  
主ウィンドウの管理プロセスが異なる)。  
EX\_WPRC : ウィンドウ(wid)の管理プロセスでない  
(従属ウィンドウの生成でアクティブプロセスでない、  
フロントエンドプロセスで前面ウィンドウ以外をオープンした、  
フロントエンドプロセス以外で前面ウィンドウをオープンした)。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない  
(同じプロセスがパネルを表示している、  
フルスクリーンのウィンドウが開かれている)。

## wopn\_iwd

内部ウィンドウのオープン

### 【形式】

WID wopn\_iwd(W gid)

### 【パラメータ】

W gid 描画環境

### 【リターン値】

- 0 正常(ウィンドウID)  
< 0 エラー(エラーコード)

### 【解説】

gid で指定した描画環境を内部ウィンドウとしてオープンし、そのウィンドウ ID を関数値として戻す。内部ウィンドウは、gid で指定した描画環境を 1 つのウィンドウとして割り当てたものであり、ウィンドウ枠等の描画、領域の管理、入力受付状態の切換え等は一切行なわれない。

内部ウィンドウに対しては、以下の関数のみ適用可能であり、他はすべて EX\_WND のエラーとなる。また、wget\_evt()、wfnwnd() での位置のサーチの対象とはならない。

wcls\_wnd() 内部ウィンドウのクローズ  
wget\_gid() 内部ウィンドウの描画環境IDの取り出し  
wset\_dat() 内部ウィンドウのユーザデータの設定  
wget\_dat() 内部ウィンドウのユーザデータの取り出し

## 【エラーコード】

EX\_NOSPC : システムのメモリ領域が不足した。  
EG\_GID : gid で指定した描画環境は存在しない。  
EX\_WPRC : ウィンドウ(wid)の管理プロセスでない  
(フロントエンドプロセスである)。

## wopn\_pwd

パネルウィンドウのオープン

## 【形式】

WID wopn\_pwd(RECT \*r)

## 【パラメータ】

RECT \*r 外枠の絶対座標

## 【リターン値】

0 正常(ウィンドウID)  
< 0 エラー(エラーコード)

## 【解説】

\*r で指定した絶対座標の領域を外枠とするパネルウィンドウをオープンし、そのウィンドウ ID を戻り値として返す。パネルウィンドウは、枠を含めた描画は一切行なわれず、その存在の管理のみを行なうことになる。なお、パネルウィンドウの移動 / 変形はできない。

パネルウィンドウはオープンされると入力受付状態を横取りした形で占有し、パネルウィンドウをオープンしたプロセスがアクティブプロセスとなる。パネルウィンドウをオープンしたプロセスがアクティブプロセスでなかった場合は、元の入力受付状態のウィンドウに対して EV\_INACT が送られ、EV\_INACT が受信された後にパネルウィンドウがオープンされる。パネルウィンドウをオープンしたプロセスが既にアクティブプロセスであった場合は、EV\_INACT は送られない。パネルウィンドウをクローズした場合、オープンした時に入力受付状態であったウィンドウに入力受付状態が戻り、EV\_INACT を送った場合は、EV\_SWITCH (W\_SWITCH コマンド) が送られる。

## 【エラーコード】

EX\_ADR : アドレス(r,bgpat)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。

EX\_ADR : ウィンドウ(wid)の管理プロセスでない  
(フロントエンドプロセスである)。

## wcls\_wnd

ウィンドウの削除

### 【形式】

ERR wcls\_wnd(W wid, W opt)

### 【パラメータ】

W wid ウィンドウID

W opt ::= ( CLR NOCLR )

CLR (=0):

削除したウィンドウ(群)により隠されていた他ウィンドウの再表示要求イベントが発行され、その結果ウィンドウの表示は消去される。

NOCLR ( 0):

削除したウィンドウ(群)の表示はそのまま残り、再表示要求イベントは一切発行されない。

### 【リターン値】

0 正常

< 0 エラー(エラーコード)

### 【解説】

widで指定したウィンドウ、および指定したウィンドウに従属する全ての従属ウィンドウを削除する。削除したウィンドウに属する全てのパーツも同時に削除される。内部ウィンドウの削除の場合を除いて、ウィンドウに対応する描画環境も削除される。

入力受付状態の主ウィンドウを削除した場合、親ウィンドウに対してEV\_SWITCH(W\_CLOSEDコマンド)が送られ、親ウィンドウに入力受付状態が切替わる。この時、生成元が存在する場合には生成元に対してクロージングアニメーションが表示される。

親ウィンドウが存在しない入力受付状態のウィンドウをクローズした場合は、直前に入力受付状態であったウィンドウにEV\_SWITCHが送られて入力受付状態が移行する。直前に入力受付状態であったウィンドウが既に存在しない場合は、さらにその直前に入力受付状態であったウィンドウというように、入力受付状態の遷移を逆にたどってウィンドウに入力受付状態が戻る。なお、この場合は生成元は存在しないものとみなされる。

パネルウィンドウを削除した場合は、パネルウィンドウをオープンした時に入力受付状態であったウィンドウに入力受付状態が切替わり、その管理プロセスが異なるプロセスの場合には、EV\_SWITCH(W\_SWITCHコマンド)が送られる(同一プロセスの場合には、EV\_SWITCHは送られない)。

削除したウィンドウが、入力受付状態でない主ウィンドウ、従属ウィンドウ、内部ウィンドウ、および前面ウィンドウの場合は、入力受付状態の切替えは発生しない。



削除したウィンドウに子ウィンドウが存在する場合は、その子ウィンドウの親ウィンドウおよび生成元は、削除したウィンドウの親ウィンドウおよび生成元にそれぞれ変更される。

削除するウィンドウは入力受付状態でなくてもよい。削除するウィンドウの管理プロセスでない場合は、EX\_WPRC のエラーとなる。また、プロセスが終了した場合は、そのプロセスがオープンした全てのウィンドウは自動的に CLR 指定でクローズされる。

### 【エラーコード】

EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_ADR : ウィンドウ(wid)の管理プロセスでない。

## wchg\_wnd

ウィンドウの全面/通常モード切換

### 【形式】

W wchg\_wnd(W wid, RECT \*r, W mode)

### 【パラメータ】

W wid ウィンドウID

RECT \*r NULL でない場合は、切換えた新しい作業領域(相対座標)が r で指定した領域に格納される。

W mode ::= ( W\_MOVE W\_MOVEC W\_HOLD )

W\_MOVE:

新作業領域の左上の点の座標値は、元の作業領域の左上の点の座標値と同じ値となる(ビットマップの境界が変更され、フレーム長方形は右下に拡大/縮小される。表示長方形は一切変更されない)。全面モードに切り換えた場合は、元の作業領域の表示内容を新作業領域の左上隅に移動し右下の追加領域が背景パターンで塗り潰され、通常モードに切り換えた場合は、元の作業領域の左上隅の表示内容が新作業領域の表示内容となる。

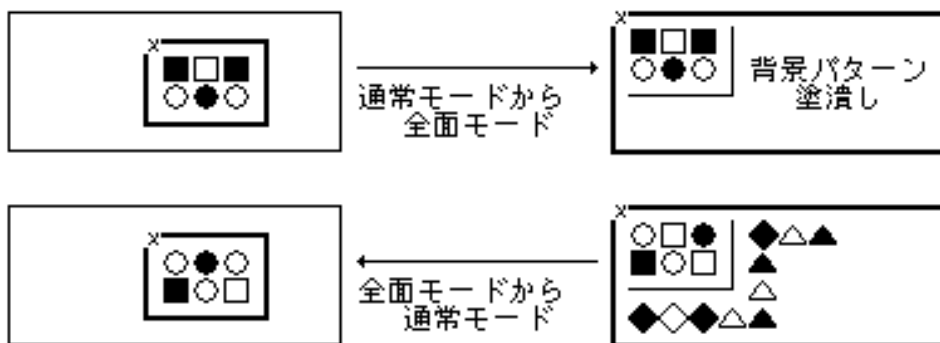


図 64 : W\_MOVE

W\_MOVEC:

新作業領域の左上の点の座標値は、元の作業領域の左上の点の座標値と同じ値となる(ビットマップの境界が変更され、フレーム長方形は右下に拡大/縮小される。表示長方形は一切

変更されない)。全面モードに切り換えた場合は、新作業領域全体はウィンドウの背景パターンで塗り潰され、通常モードに切り換えた場合は、元の作業領域の左上隅の表示内容が新作業領域の表示内容となる(通常モードへの切り換えの場合はW\_MOVEの場合と同じ)。

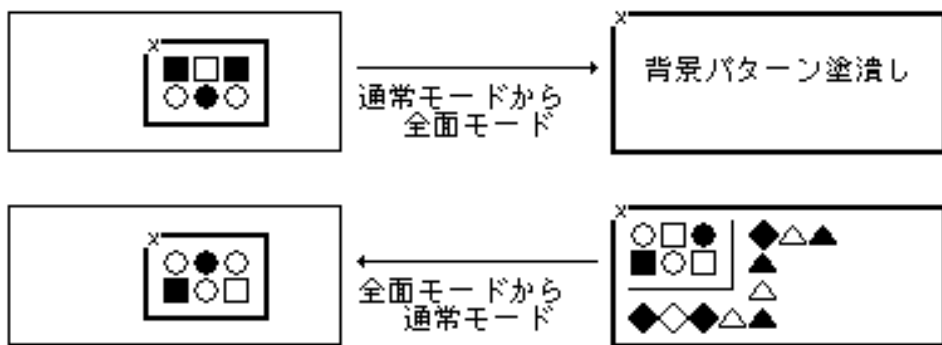


図 65 : W\_MOVEC

W\_HOLD:

新作業領域の左上の点の座標値は、元の作業領域の左上の点の座標値が変化しない値になる。即ち、座標系は変更されない(ビットマップの境界は変更されず、フレーム長方形は4隅が拡大/縮小される。表示長方形は一切変更されない)。全面モードに切り換えた場合は、元の作業領域の表示内容はそのまま残り、周囲の追加領域がウィンドウの背景パターンで塗り潰され、通常モードに切り換えた場合は、元の作業領域の対応する位置の表示内容が新作業領域の表示内容となる。

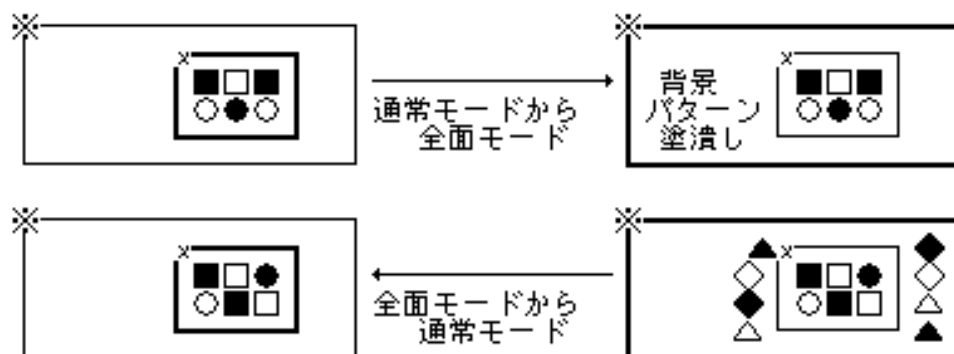


図 66 : W\_HOLD

なお、全面モードのウィンドウを移動/変形した場合は、通常モードとなる。

### 【リターン値】

- 0 関数値 = 0 : 通常モードに切替わった(対象ウィンドウの再表示は不要)。
- 1 : 全面モードに切替わった(対象ウィンドウの再表示が必要)。
- 2 : 通常モードに切替わった(対象ウィンドウの再表示が必要)。

< 0 エラー(エラーコード)

### 【解説】

widで指定したウィンドウの全面モード/通常モードを切替える。現在通常モードの場合は全面モードに切替え、全面モードの場合は通常モードに切替える。

全面モードから通常モードに切り換えた場合で、他のウィンドウの再表示が必要な時は、そのウィンドウに対して再表示要求イベントが送られる。通常、対象ウィンドウの再表示は不要であ

るが、必要な場合もあり、その場合は再表示領域が設定されるので、wsta\_dsp() ~ wend\_dsp()により再表示を行なう必要がある。

通常モードから全面モードに切り換えた場合は、他ウィンドウの再表示は不要であるが、対象ウィンドウの再表示は自分で行なう必要がある。再表示要求イベントは送られないが、再表示領域は設定されるので、wsta\_dsp() ~ wend\_dsp()により再表示を行なう必要がある。全面モードにした場合でもその従属ウィンドウ、および前面ウィンドウはオーバーラップして表示されることになる。

### 【エラーコード】

- EX\_ADR : アドレス(r)のアクセスは許されていない。
- EX\_PAR : パラメータが不正である(mode が不正)。
- EX\_WID : ウィンドウ(wid)は存在していない。
- EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない  
(入力受付状態の主ウィンドウでない)。
- EX\_ADR : ウィンドウ(wid)の管理プロセスでない。

## wmov\_wnd

ウィンドウの移動

### 【形式】

W wmov\_wnd(W wid, RECT \*newr)

### 【パラメータ】

W wid ウィンドウID  
RECT \*newr newr->lefttop で移動後の左上の点を指定

### 【リターン値】

0 正常 0(再表示不要)  
1(再表示必要)  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウの外枠の左上の点が newr->lefttop で指定した絶対座標位置に一致するようにウィンドウ全体を移動する。移動により作業領域の表示イメージも移動し、最終的な新しいウィンドウの外枠が newr で指定した領域に絶対座標値で格納される。

ウィンドウ属性によって、WA\_STD 指定時には正規座標が、WA\_WORK 指定時には作業領域座標が、WA\_FRAME 指定時には外枠座標が、格納される。

移動の結果、他ウィンドウの再表示が必要になった場合は、再表示要求イベントが発行される。また移動したウィンドウ自体の再表示が必要な場合は、戻り値として "1" が戻り、不要な場合は "0" が戻る。対象ウィンドウへの再表示要求イベントは発行されないが、再表示領域は設定され

るため、関数値が "1" の場合は、wsta\_dsp() ~ wend\_dsp() による再表示を行なう必要がある。

移動したウィンドウが全面モードの場合は、そのウィンドウは通常モードとなる。

ウィンドウの移動では、ビットマップの境界のみが変更され、フレーム長方形、表示長方形は一切変更されない。

対象ウィンドウが入力受付状態でない場合は、EX\_WND のエラーとなり、管理プロセスでない場合は、EX\_WPRC のエラーとなる。

### 【エラーコード】

EX\_ADR : アドレス(newr)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない  
(入力受付状態でない)。  
EX\_ADR : ウィンドウ(wid)の管理プロセスでない。

## wmov\_drg

ウィンドウのドラッグ移動

### 【形式】

W wmov\_drg(WEVENT \*evt, RECT \*newr)

### 【パラメータ】

WEVENT \*evt ドラッグ移動のきっかけとなったウィンドウイベント。

RECT \*newr NULL でない場合は、移動したウィンドウの新しい外枠の絶対座標が newr で指定した領域に格納される。

ウィンドウ属性によって、WA\_STD 指定時には外枠座標が、WA\_WORK 指定時には作業領域座標が、WA\_FRAME 指定時には外枠座標が、格納される。WA\_STD 指定時に外枠座標が格納されるのは、従来仕様との互換性のためである。

### 【リターン値】

0 正常      0(再表示不要)  
            1(再表示必要)  
< 0 エラー(エラーコード)

### 【解説】

evt->s.wid で指定したウィンドウをドラッグにより移動する。\*evt には、ドラッグの契機となった EV\_SWITCH または EV\_BUTDWN または EV\_RSWITCH のウィンドウイベントを指定する。このイベントの位置コードは、W\_FRAM(1)、W\_LTHD(4) ~ W\_RBHD(7) のいずれかでなくてはならない。イベントのタイプや、位置コードが不正の場合は EX\_PAR のエラーとなる。

この関数の実行により、ポインタの形状は「握り」となり、PD のボタンが押されている間、ウィンドウ枠の影が PD の移動に伴って移動する。ボタンをリリースした位置にウィンドウが移

動し、ポインタの形状はもとの「移動手」に戻る。

ポインタをシステムメッセージパネル上に移動した場合は、影は消去される。ポインタをシステムメッセージパネル以外に移動すると、影は再び表示される。ポインタをシステムメッセージパネル上でリリースした場合は、ウィンドウの移動は取り消される。

ドラッグ中、シフトキーが押されている間は、移動の方向が縦方向、または横方向に拘束され、ドラッグの影は、縦方向または横方向にのみ移動する。拘束の方向は、PDの最初の位置から最初に移動した変位の長辺の方向となる。縦方向に拘束されている状態では、ポインタの形状は縦方向のものとなり、横方向に拘束されている状態では、ポインタの形状は横方向のものとなる。

シフトキーを離れた場合は、拘束は解除される。従って、シフトキーを押している状態でPDをリリースすることにより、縦方向または横方向の移動を行なうことができる。なお、移動の方向が拘束されている状態でも、ポインタ自体の動きは拘束されない。

ドラッグ中に発生したEV\_KEYDOWN, EV\_KEYUP, EV\_AUTKEY イベントは取り除かれて無視されるが、その他のイベントが発生した場合はイベントキューに残される。PDのリリース時により発生したEV\_BUTTON イベントは取り除かれる。

移動の結果、他ウィンドウの再表示が必要になった場合は、再表示要求イベントが発行される。また移動したウィンドウ自体の再表示が必要な場合は、関数値として"1"が戻り、不要な場合は"0"が戻る。対象ウィンドウへの再表示要求イベントは発行されないが、再表示領域は設定されるため、関数値が"1"の場合は、wsta\_dsp() ~ wend\_dsp()による再表示を行なう必要がある。

移動したウィンドウが全面モードの場合は、そのウィンドウは通常モードとなる。

ウィンドウの移動では、ビットマップの境界のみが変更され、フレーム長方形、表示長方形は一切変更されない。

対象ウィンドウが入力受付状態でない場合は、EX\_WNDのエラーとなり、管理プロセスでない場合は、EX\_WPRCのエラーとなる。

## 【エラーコード】

- EX\_WID : ウィンドウ(wid)は存在していない。
- EX\_PAR : パラメータが不正である  
(イベントのタイプ、位置コードが不正)。
- EX\_ADR : アドレスのアクセスは許されていない(newr)。
- EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない  
(入力受付状態でない)。
- EX\_ADR : ウィンドウ(wid)の管理プロセスでない。

## wrsz\_wnd

ウィンドウの変形

## 【形式】

W wrsz\_wnd(W wid, RECT \*newr)

## 【パラメータ】

W wid ウィンドウID  
REC \*newr 変形する大きさ

### 【リターン値】

0 正常 0(再表示不要)  
1(再表示必要)  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウの外枠を \*newr で指定した大きさに変形する。\*newr は絶対座標で指定され、4 隅のいずれか 1 つは現在のウィンドウの外枠の 4 隅と一致していなくてはならない。\*newr がシステムで規定した最小サイズ以下の場合は、最小サイズで実行される。変形後のウィンドウの外枠が \*newr に格納される。

ウィンドウ属性によって、WA\_STD 指定時には外枠座標が、WA\_WORK 指定時には作業領域座標が、WA\_FRAME 指定時には外枠座標が、格納される。WA\_STD 指定時に外枠座標が格納されるのは、従来仕様との互換性のためである。

変形の結果、他ウィンドウの再表示が必要になった場合は、再表示要求イベントが発行される。また変形したウィンドウ自体の再表示が必要な場合は、関数値として "1" が戻り、不要な場合は "0" が戻る。対象ウィンドウへの再表示要求イベントは発行されないが、再表示領域は設定されるため、関数値が "1" の場合は、wsta\_dsp() ~ wend\_dsp() による再表示を行なう必要がある。

変形により拡張された作業領域の追加領域はウィンドウの背景パターンで塗りつぶされる。また元の作業領域に表示されていた部分の相対座標値は一切変化しない。

変形したウィンドウが全面モードの場合は、そのウィンドウは通常モードとなる。

ウィンドウの変形では、ビットマップの境界は変更されず、フレーム長方形の 4 隅のいずれかが拡大 / 縮小されることになる。表示長方形は一切変更されない。

対象ウィンドウが入力受付状態でない場合は、EX\_WND のエラーとなり、管理プロセスでない場合は、EX\_WPRC のエラーとなる。

### 【エラーコード】

EX\_ADR : アドレス(newr)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である  
(newr の4隅が全て一致していない,\*newr が範囲外)。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない  
(入力受付状態でない)。  
EX\_ADR : ウィンドウ(wid)の管理プロセスでない。

## wrsz\_drg

ウィンドウのドラッグ変形

### 【形式】

W wsz\_drg(WEVENT \*evt, RECT \*limit, RECT \*newr)

## 【パラメータ】

WEVENT \*evt ドラッグ変形のきっかけとなったウィンドウイベント。

RECT \*limit limit->lefttop はウィンドウの最小サイズ ( lefttop.c.v が縦、 lefttop.c.h が横の大きさ ) を示し、 limit->rightbot は同様に最大サイズを示す。 limit が NULL の場合は最小サイズはシステムの規定値、最大サイズは特に制限無しと見なされる。

RECT \*newr newr が NULL でない場合は、変形したウィンドウの新しい外枠の絶対座標が newr で指定した領域に格納される。ウィンドウ属性によって、WA\_STD 指定時には正規座標が、WA\_WORK 指定時には作業領域座標が、WA\_FRAME 指定時には外枠座標が、格納される。

## 【リターン値】

0 正常 0(再表示不要)  
1(再表示必要)  
< 0 エラー(エラーコード)

## 【解説】

evt->s.wid で指定したウィンドウをドラッグにより変形する。 \*evt には、ドラッグの契機となった EV\_SWITCH, EV\_BUTDOWN または EV\_RSWITCH のウィンドウイベントを指定する。このイベントの位置コードは、W\_LTHD(4) ~ W\_RBHD(7) のいずれかでなくてはならない。イベントのタイプや、位置コードが不正の場合は EX\_PAR のエラーとなる。

この関数の実行により、ポインタの形状は「つまみ」となり、PD のボタンが押されている間、ウィンドウ枠の影が PD の移動に伴って変形する。変形のしかたは、 \*evt で指定したイベントの位置コード、およびウィンドウの属性 ( WA\_HHDL , WA\_VHDL ) によって決まり、さらに \*limit で指定した最小サイズと最大サイズにより制限される。

ドラッグ中に最大サイズ以上になった場合は、ウィンドウ枠の影は、縦または横の最大サイズでクランプし、PD の位置に追従しなくなる。最小サイズ以下になった場合も同様にウィンドウ枠の影は、縦または横の最小サイズでクランプし、PD の位置に追従しなくなる。

ポインタをシステムメッセージパネル上に移動した場合は、影は消去される。ポインタをシステムメッセージパネル以外に移動すると、影は再び表示される。

PD のボタンをリリースした時点のウィンドウ枠の影の大きさにウィンドウは変形し、ポインタの形状はもとの「変形手」に戻る。

ポインタをシステムメッセージパネル上でリリースした場合、ウィンドウの変形は取り消される。

ドラッグ中、シフトキーが押されている間は、変形の方法が縦方向、または横方向に拘束され、ドラッグの影は、縦方向または横方向にのみ変形する。拘束の方法は、PD の最初の位置から最初に移動した変位の長辺の方法となる。縦方向に拘束されている状態では、ポインタの形状は縦方向のものとなり、横方向に拘束されている状態では、ポインタの形状は横方向のものとなる。ただし、上下左右ともに変形可のウィンドウに対してのみ有効である。

シフトキーを離した場合は、拘束は解除される。従って、シフトキーを押している状態で PD を

リリースすることにより、縦方向または横方向の変形を行なうことができる。なお、変形の方法が拘束されている状態でも、ポインタ自体の動きは拘束されない。

ドラッグ中に発生した EV\_KEYDOWN, EV\_KEYUP, EV\_AUTKEY イベントは取り除かれて無視されるが、その他のイベントが発生した場合はイベントキューに残される。PD のリリース時により発生した EV\_BUTTON イベントは取り除かれる。

変形の結果、他ウィンドウの再表示が必要になった場合は、再表示要求イベントが発行される。また変形したウィンドウ自体の再表示が必要な場合は、関数値として "1" が戻り、不要な場合は "0" が戻る。対象ウィンドウへの再表示要求イベントは発行されないが、再表示領域は設定されるため、関数値が "1" の場合は、wsta\_dsp() ~ wend\_dsp() による再表示を行なう必要がある。

変形により拡張された作業領域の追加領域はウィンドウの背景パターンで塗りつぶされる。また元の作業領域に表示されていた部分の相対座標値は一切変化しない。

変形したウィンドウが全面モードの場合は、そのウィンドウは通常モードとなる。

ウィンドウの変形では、ビットマップの境界は変更されず、フレーム長方形の4隅のいずれかが拡大 / 縮小されることになる。表示長方形は一切変更されない。

対象ウィンドウが入力受付状態でない場合は、EX\_WND のエラーとなり、管理プロセスでない場合は、EX\_WPRC のエラーとなる。

### 【エラーコード】

- EX\_ADR : アドレス(evt,newr,limit)のアクセスは許されていない。
- EX\_PAR : パラメータが不正である(イベントのタイプ、位置コードが不正)。
- EX\_WID : ウィンドウ(wid)は存在していない。
- EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない(入力受付状態でない、ドラッグ変形不可)。
- EX\_ADR : ウィンドウ(wid)の管理プロセスでない。

## wscr\_wnd

ウィンドウのスクロール

### 【形式】

W wscr\_wnd(W wid, RECT \*r, W dh, W dv, W mode)

### 【パラメータ】

- W wid ウィンドウID
- RECT \*r スクロールする矩形領域
- W dh スクロールする水平距離(負の場合は左方向)
- W dv スクロールする垂直距離(負の場合は上方向)
- W mode ::= ( W\_MOVE W\_SCRL ) | ( W\_NOOVL ) | ( W\_RDSET )

W\_MOVE :  
作業領域の枠の座標は変化しない(ビットマップ境界、フレーム長方形、および表示長方形は変更されない)。



W\_SCRL :

作業領域の枠の座標は変化する(ビットマップ境界、フレーム長方形、および表示長方形は、縦方向に -dv、横方向に -dh 移動する)。

W\_NOOVL :

スクロールによりオーバーラップしていたウィンドウに隠されていた部分が出てくるような場合は、イメージのスクロールは行なわない。この場合でも、W\_SCRL を指定した場合は座標系は移動する。

W\_RDSET :

スクロールにより発生した再表示が必要な領域をウィンドウの再表示領域として設定し、wsta\_dsp() ~ wend\_dsp() により再表示が必要な領域を取り出せるようにする。この指定を行なわない場合は、wsta\_dsp() により再表示領域を取り出すことはできない。W\_NOOVL 指定で実際にイメージのスクロールが行なわれなかった場合は、W\_RDSET を指定しても再表示領域は設定されない。W\_SCRL を指定した場合の再表示領域は移動後の座標系で設定される。

### 【リターン値】

0 正常 ::= ( W\_VALID ) | ( W\_NOOVL ) | ( W\_RDSET )

W\_VALID: 0=オーバーラップされて隠れていた部分がでてきた。  
(スクロールしたイメージは正しくない)  
1=オーバーラップされて隠れていた部分はでてきていない。  
(スクロールしたイメージは正しい)

W\_NOOVL: 0=スクロールした。  
1=スクロールしなかった。(W\_NOOVL 指定時のみ)

W\_RDSET: 0=再表示領域は設定されていない。  
1=再表示領域が設定された。(W\_RDSET 指定時のみ)

< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウの作業領域内の \*r で指定した矩形領域の表示内容を dh だけ右方向 ( 負の場合は左方向 )、dv だけ下方向 ( 負の場合は上方向 ) に移動する。r = NULL の場合は、作業領域全体の移動を意味する。

スクロールにより新たに表示される領域は、ウィンドウの背景パターンで塗り潰されるが、アプリケーションが再表示を行なう必要がある。また、スクロールバーの再表示も行なう必要がある。

対象ウィンドウは入力受付状態でなくてもよい。管理プロセスでない場合は、EX\_WPRC のエラーとなる。

### 【エラーコード】

EX\_ADR : アドレス(r)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(modeが不正)。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。  
EX\_ADR : ウィンドウ(wid)の管理プロセスでない。

# wsta\_dsp

ウィンドウの再表示開始

## 【形式】

W wsta\_dsp(W wid, RECT \*r, RLIST \*r1st)

## 【パラメータ】

W wid ウィンドウID  
RECT \*r 再表示を必要とする領域を包含する最小の矩形領域が格納される。  
RLIST \*r1st 個々の矩形リストが格納される。

## 【リターン値】

0 正常(関数値は再表示を必要とする矩形領域の数)  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウの再表示処理の開始を宣言し、再表示を必要とする領域を包含する最小の矩形領域を r で指定した領域に、個々の矩形リストを r1st で指定した領域に格納する。r1st の領域はあらかじめリストになっている必要があり、実際に設定された矩形の数が関数値として戻る。設定される矩形はそのウィンドウの相対座標で示される。

r = NULL、r1st=NULL の場合は、それぞれ対応するデータは格納されないが、再表示が必要な矩形領域の数が関数値として戻る。

再表示が必要な矩形領域が存在しない場合は、関数値として 0 が戻り、r には (0, 0, 0, 0) が格納され、r1st には何も格納されない。また、r1st の要素数が足りない場合は、その要素数分の矩形のみが格納される。

ウィンドウの操作に伴い再表示が必要になった場合、および再表示要求イベントを受け取った場合は、必ず wsta\_dsp() を呼び、関数値 1 以上の時は、再表示処理を行なわなくてはならない。再表示処理が終了した後は、必ず wend\_dsp() を実行しなくてはならない。

対象ウィンドウの管理プロセスでない場合は、EX\_WPRC のエラーとなる。

## 【エラーコード】

EX\_ADR : アドレス(r, r1st)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。  
EX\_ADR : ウィンドウ(wid)の管理プロセスでない。

# wend\_dsp

ウィンドウの再表示終了

## 【形式】

W wend\_dsp(W wid)

## 【パラメータ】

W wid ウィンドウID

## 【リターン値】

0 正常(関数値は再表示要求の有無(0:無、1:有))  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウの再表示処理の終了を宣言し、ウィンドウの再表示領域をリセットする。

wsta\_dsp() により再表示処理を開始した場合は、必ず、この関数を実行する必要がある。ただし、wsta\_dsp() の関数値が 0 で、再表示が不要な場合は、特にこの関数を実行する必要はない。また、wsta\_dsp() による再表示処理が開始されていない場合は何も行なわない。

wend\_dsp() を実行した時点で、新たな再表示要求が発生していた場合は、関数値として "1" が戻る。この場合は、再度、wsta\_dsp() ~ wend\_dsp() による再表示処理を行なう必要がある。新たな再表示要求が発生していない場合は、関数値として "0" が戻る。

対象ウィンドウの管理プロセスでない場合は、EX\_WPRC のエラーとなる。

## 【エラーコード】

EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。  
EX\_ADR : ウィンドウ(wid)の管理プロセスでない。

# wchk\_dsp

ウィンドウの再表示状態の取出し

## 【形式】

W wchk\_dsp(W wid)

## 【パラメータ】

W wid ウィンドウID

## 【リターン値】

0 正常(関数値は再表示が必要なウィンドウ数)  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウの再表示が必要な場合は関数値として "1" を戻し、不要な場合は "0" を戻す。wid = 0 の場合は、再表示が必要なウィンドウの個数を関数値として戻す。

## 【エラーコード】

EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

wsta\_drg

ドラッグ処理の開始

## 【形式】

GID wsta\_drg(W wid, W lock)

## 【パラメータ】

W wid ウィンドウID  
W lock lock 0 の場合は、他のウィンドウの同時描画を禁止する。

## 【リターン値】

0 正常(関数値はドラッグ描画用の描画環境ID)  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウに対するドラッグ「影」の描画用の描画環境を生成し、ドラッグ「影」の描画を開始する。関数値として生成したドラッグ描画用の描画環境 ID が戻る。

lock はロック指定パラメータであり、lock 0 の場合は、他のウィンドウの同時描画は禁止される。元のウィンドウをはみ出て描画する必要がある場合は、ロック指定を必ず行なう必要がある。

この関数は排他的に実行され、既に他のプロセスがドラッグ描画を実行していた場合は、

EX\_DRAG のエラーとなる。

対象ウィンドウが入力受付状態でない場合は、EX\_WND のエラーとなり、管理プロセスでない場合は、EX\_WPRC のエラーとなる。

### 【エラーコード】

EX\_DRAG : ドラッグエラー(既に他のプロセスが実行中である)。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない  
(入力受付状態ではない)。  
EX\_ADR : ウィンドウ(wid)の管理プロセスでない。

## wend\_drg

ドラッグ処理の終了

### 【形式】

ERR wend\_drg(void)

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

wsta\_drag() により開始したドラッグ「影」の描画処理を終了し、対応する描画環境を解放する。ロック指定を行なっていた場合は、他のウィンドウの同時描画の禁止は解除される。ドラッグ描画が行なわれていなかった場合は何もしない。

ドラッグ描画を終了した場合は、必ず wend\_drg() を実行しなくてはならない。なお、ドラッグ描画対象のウィンドウが削除された場合、または wsta\_drag() を実行したプロセスが終了した場合は、自動的に wend\_drg() の処理が行なわれる。

### 【エラーコード】

EX\_DRAG : ドラッグエラー(ドラッグ描画を開始したプロセスでない)。

## wget\_drg

PDドラッグ位置の取出し

### 【形式】

W wget\_drg(PNT \*pos, WEVENT \*evt)

### 【パラメータ】

PNT \*pos 現在のPDの位置を、ドラッグ描画環境の相対座標値で取り出す。  
WEVENT \*evt ドラッグのきっかけとなったウィンドウイベント

### 【リターン値】

0 正常(関数値は発生したウィンドウイベントのタイプ)  
< 0 エラー(エラーコード)

### 【解説】

wsta\_drag() により開始したドラッグ「影」の描画処理において、現在のPDの位置を、ドラッグ描画環境の相対座標値で取り出し、posで指定した領域に格納する。

EV\_REQUEST、EV\_SWITCHを除くウィンドウイベントが発生していた場合は、そのイベントを取り出し、evtで指定した領域に格納して、関数値として発生したウィンドウイベントのタイプを戻す。

この場合、発生したイベントのタイプによっては、PD位置は、ウィンドウの相対座標となるため、posに戻される値と同一の値になるとは限らない。

EV\_REQUEST、EV\_SWITCHのみが発生していた場合、またはウィンドウイベントが発生していない場合は、関数値として0(EV\_NULL)を戻し、evtの領域にはEV\_NULLが格納される。格納されるイベントの、ウィンドウID、位置コード、および発生時刻の値は設定されず、PD位置とメタキー状態のみ設定され、PD位置はposに戻される値と同一となる。

ドラッグ描画が開始されていない場合、あるいは、ドラッグ描画を開始したプロセスでない場合は、EX\_DRAGのエラーとなる。

### 【エラーコード】

EX\_ADR : アドレス(pos,evt)のアクセスは許されていない。  
EX\_DRAG : ドラッグエラー(ドラッグ描画を開始したプロセスでない、ドラッグ描画は開始されていない)。

## wget\_evt

ウィンドウイベントの取出し

### 【形式】

W wget\_evt(WEVENT \*evt, W mode)

### 【パラメータ】

WVENT \*evt 取り出されたウィンドウイベント  
W mode ::= ( WAIT NOMSG RAWEVT DRGEVT ) | [ NOWAIT ]

|        |        |                       |
|--------|--------|-----------------------|
| WAIT   | 0x0000 | イベント発生まで待つ            |
| NOWAIT | 0x0001 | イベントが発生していなくても即座にリターン |
| NOMSG  | 0x0010 | メッセージは取り出さない          |
| RAWEVT | 0x0020 | 絶対座標でイベントを取り出す        |
| DRGEVT | 0x0030 | 相対座標でイベントを取り出す        |

## 【リターン値】

0 正常(関数値はウィンドウイベントのタイプ)  
< 0 エラー(エラーコード)

## 【解説】

ウィンドウイベントを取り出し、evt で指定した領域に格納する。関数値として取り出したイベントのタイプを戻す。

mode により以下のように振る舞う。

WAIT:

アクティブプロセスの時:

イベントが発生するまで待つ、イベントを取りだし、対応するウィンドウの相対座標に変換する。PD のプレスが入力受付状態でないウィンドウで発生した場合は、EV\_SWITCH を発生し、入力受付状態の切り換えを行なう。

フロントエンドプロセスが存在する時は、以下の処理を行なう。

取り出したウィンドウイベントのタイプに応じて以下の処理を行なう(取り出したウィンドウイベントには、存在する全ての前面ウィンドウに対する要求イベント等も含まれる)。

EV\_NULL ~ EV\_RSV, EV\_MENU, EV\_APPL4 ~ 8:

最後に定義されたフロントエンドプロセスに取り出したイベントを渡し、返答待ち状態となる。

EV\_SWITCH, EV\_RSWITCH, EV\_INACT:

そのままリターンする。

(前面ウィンドウに対しては、これらのイベントは発生しない(常に入力受付状態であるため))

EV\_REQUEST:

対象ウィンドウの管理プロセスが、フロントエンドプロセスの場合は、そのフロントエンドプロセスに取り出した要求イベントを送り、返答待ち状態となる。そうでない場合は、そのままリターンする。

返答待ち状態になった場合は、返答がフロントエンドプロセスから得られた時点で、返答のイベントを得られたイベントとしてリターンする。返答のイベントが(空)の場合は、リターンせずに、再度ウィンドウイベントを取り出し、上記の処理を繰り返す。

フロントエンドプロセスの時:

前回取り出したイベントに対する wpas\_evt() がまだ実行されていない場合は、(空)イベントをアクティブプロセスに戻す(wpas\_evt(NULL) の処理)。待ち状態に入り、アクティブプロセス、または直前のフロントエンドプロセスから渡されるイベントを待つ。イベントが渡されたら、そのイベントを得られたイベントとしてリターンする。

((空) イベントが得られることはない)。

非アクティブプロセスの時:

メッセージが既に存在している場合は、そのメッセージを取り出し、存在していない場合はメッセージ待ち状態となる。

NOWAIT: (WAIT|NOWAIT としても同じ)

WAIT 指定に対して、以下の点が異なる。

- アクティブプロセスの時でイベントが発生していない場合には、関数値として EV\_NULL が返る。
- 非アクティブプロセスの時でメッセージが存在していない場合には、関数値として EV\_NOMSG が返る。

NOMSG:

アクティブプロセスの時:

イベントが発生するまで待って、イベントを取りだし、対応するウィンドウの相対座標に変換する。PDのプレスが入力受付状態でないウィンドウで発生した場合でも、EV\_SWITCH を発生せず、EV\_BUTDWN として戻す(入力受付状態の切換は行なわない)。W\_REDISP 以外の EV\_REQUEST 及び、EV\_MSG は取り出さない。

フロントエンドプロセスが存在する時は、WAIT 指定と同じ処理を行なう。

フロントエンドプロセスの時:

WAIT 指定の場合と同じ。

非アクティブプロセスの時:

待ち状態となる。

NOMSG|NOWAIT:

NOMSG 指定に対して、以下の点が異なる。

- アクティブプロセスの時で、イベントが発生していない場合には、関数値として EV\_NULL が返る。
- 非アクティブプロセスの時は、関数値として EV\_NOMSG が返る。

RAWEVT:

アクティブプロセスの時

フロントエンドプロセスの時

EV\_BUTDWN, EV\_BUTUP, EV\_KEYDOWN, EV\_KEYUP, EV\_AUTKEY のいずれかのイベントが発生するまで待って、そのイベントをとりだし、その座標は絶対座標のままとする。wid は常に 0 となる。PD のプレスが入力受付状態でないウィンドウで発生した場合でも、EV\_SWITCH を発生せず、EV\_BUTDWN として戻す(入力受付状態の切換は行なわない)。

非アクティブプロセスの時:

何もせずに、EV\_NOMSG を関数値として戻す。

RAWEVT|NOWAIT:

RAWEVT 指定に対して、以下の点が異なる。

- アクティブプロセスの時で、イベントが発生していない場合には、関数値として EV\_NULL が返る。



DRGEVT :

座標が対応するウィンドウの相対座標となる以外は、RAW EVT と同じ。  
NOMSG および RAW EVT、DRGEVT 指定はパネル、メニュー、パーツ等の内部処理に使用され、通常のアプリケーションでは使用しない。

DRGEVT|NOWAIT :

DRGEVT 指定に対して、以下の点が異なる。

- アクティブプロセスの時で、イベントが発生していない場合には、関数値として EV\_NULL が返る。

WAIT, NOMSG, RAW EVT, DRGEVT を単独で指定した場合、イベントがまったく発生していない場合には、イベントが発生するかまたは表示の更新が必要になるまで待つ (カレットの点滅等が必要な場合は、EV\_NULL が発生する)。一方、WAIT, NOMSG, RAW EVT, DRGEVT を NOWAIT とともに指定した場合は、イベントがまったく発生していない場合でも EV\_NULL (アクティブプロセスの場合) または EV\_NOMSG (非アクティブプロセスの場合) が返り、待つことはない。

NOWAIT 指定は、フロントエンドプロセスでは無視される。

フロントエンドプロセスでなく mode が WAIT の場合には、内部イベントメッセージ (ウィンドウイベント) を取り出すが、この場合、内部イベントメッセージ以外のすべてのタイプのメッセージもチェックされ、内部イベントメッセージ以外のメッセージを受信していた場合には、関数値として EV\_MSG が戻り、evt で指定した領域には type の後にメッセージの先頭 8 バイト (タイプとサイズ) が格納される。

### 【エラーコード】

EX\_ADR : アドレス (evt) のアクセスは許されていない。  
EX\_ADR : ウィンドウ (wid) の管理プロセスでない (ウィンドウが1つも存在しない)。

## wugt\_evt

ウィンドウイベントの戻し

### 【形式】

ERR wugt\_evt (WEVENT \*evt)

### 【パラメータ】

WEVENT \*evt ウィンドウイベント

### 【リターン値】

0 正常  
< 0 エラー (エラーコード)

### 【解説】

\*evt に格納されているウィンドウイベントを戻して、次に WAIT または INWAIT のモードで実行される、wget\_evt() 関数で取り出すウィンドウイベントとする。他のモードで実行した場合は取り出されない。

戻したイベントは、通常のイベントに優先して取り出される。また、取り出される順番は戻した順番である。イベントの内容については一切チェックされない。

1 プロセスで最大16個のウィンドウイベントをイベントキューに戻すことができる。同時に使用できるプロセスはウィンドウ数個までであり、それ以上はエラー (EX\_NOSPC) となる。

wugt\_evt() で戻したイベントが、wget\_evt() で取り出された場合、フロントエンドプロセスには渡されない。

この処理は1つのプロセス内で行なわれるため、戻したイベントは、他のプロセスからの wget\_evt() では取り出されない点に注意が必要である。

プロセスが終了した場合は、そのプロセスによって戻されたイベントは自動的に捨てられる。

この関数は、パーツやパネルの処理で期待外のイベントが得られた場合、そのイベントを一旦戻して、通常のイベントループに戻った後、wget\_evt() で取り出して処理するために使用される。

wugt\_evt() は、特殊な用途に使用される関数であり、通常アプリケーションは使用しない。

#### 【エラーコード】

EX\_ADR : アドレス(evt)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した  
(戻したイベントが16個以上)。

## wpas\_evt

ウィンドウイベントの回送

#### 【形式】

ERR wpas\_evt(WEVENT \*evt)

#### 【パラメータ】

WEVENT \*evt ウィンドウイベント

#### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

#### 【解説】

\*evt で指定したフロントエンド処理後のウィンドウイベントを次のフロントエンドプロセスまた

は、アクティブプロセスに戻す。 evt = NULL の場合は、戻すべきイベントが(空)であることを意味し、次のフロントエンドプロセスが存在していても、必ずアクティブプロセスに(空)イベントに戻す。

イベントのタイプは0 ~ 15 でなくてはならないが、その内容は一切チェックされない。

この関数は、フロントエンドプロセスのみが、実行可能である。

### 【エラーコード】

EX\_ADR : アドレス(evt)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(イベントタイプが0 ~ 15以外)。  
EX\_ADR : ウィンドウ(wid)の管理プロセスでない  
(フロントエンドプロセスでない)。

## wswi\_wnd

ウィンドウの入力受付状態の切換

### 【形式】

ERR wswi\_wnd(W wid, WEVENT \*evt)

### 【パラメータ】

W wid ウィンドウID  
WEVENT \*evt ウィンドウイベント

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウに EV\_SWITCH を送信して入力受付状態に切り換える。

evt = NULL の場合は、EV\_SWITCH イベントの W\_SWITCH コマンドが wid で指定したウィンドウに対して送られる。

evt NULL の場合は、\*evt で指定した EV\_BUTDWN イベントを EV\_SWITCH に変換して、wid で指定したウィンドウに対して送られる。 evt->type EV\_BUTDWN または、 evt->wid wid の時は EX\_PAR のエラーとなる。

現在入力受付状態であったウィンドウは入力不可状態となり、続くウィンドウイベントの取り出しでは、EV\_INACT が得られることになる。

アクティブプロセスでない場合、または指定したウィンドウが既に入力受付状態の場合は、EX\_WND のエラーとなる。

## 【エラーコード】

- EX\_ADR : アドレス(evt)のアクセスは許されていない。
- EX\_PAR : パラメータが不正である(イベントのタイプが不正)。
- EX\_WID : ウィンドウ(wid)は存在していない。
- EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない  
(アクティブプロセスでない、既に入力受付状態である)。

## wsnd\_evt

要求イベントの送信

## 【形式】

W wsnd\_evt(WEVENT \*evt)

## 【パラメータ】

WEVENT \*evt 要求イベント

## 【リターン値】

- 0 正常(関数値は送付した要求イベントの数)
- < 0 エラー(エラーコード)

## 【解説】

evt で指定した要求イベントを evt->r.wid で指定したウィンドウに対して送付する。evt->r.type は EV\_REQUEST でなくてはならない。また evt->r.cmd は W\_REDISP であってはならない。送信時に evt->r.src に送信元のプロセス ID が設定される。送付先のウィンドウ ID = 0 の場合は、すべてのウィンドウ(内部ウィンドウ、パネルウィンドウを除く)に対して要求イベントが送付される。ウィンドウ ID < 0 の場合は、- wid をプロセス ID として、指定したプロセスに対し要求イベントを送付する。

関数値として、要求イベントを送付したウィンドウの数を返す。従って、wid = 0 以外の場合は常に関数値は "1" となる。

ウィンドウ間のデータの貼り込みは、wsnd\_evt() により W\_PASTE イベントを送付して行なわれる。

## 【エラーコード】

- EX\_ADR : アドレス(evt)のアクセスは許されていない。
- EX\_PAR : パラメータが不正である(イベントのタイプ、内容が不正)。
- EX\_WID : ウィンドウ(wid)は存在していない。
- EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

## wrsp\_evt

要求イベントへの応答

### 【形式】

ERR wrsp\_evt(WEVENT \*evt, W nak)

### 【パラメータ】

WEVENT \*evt 要求イベント

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

evt で指定した要求イベントに対する応答を evt->r.src で指定したプロセスに送信する。evt->r.type は EV\_REQUEST でなくてはならない。また evt->r.cmd は W\_REDISP であってはならない。

nak = 0 の時は、ACK レスポンスとして evt->r.cmd に W\_ACK が + されるが、nak < 0 の場合は、NACK レスポンスとして evt->r.cmd に W\_NAK が + される。

### 【エラーコード】

EX\_ADR : アドレス(evt)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(イベントのタイプが不正、応答先PIDが不正)。

## wwai\_rsp

要求イベントへの応答待ち

### 【形式】

W wwai\_rsp(WEVENT \*evt, W cmd, UW tmout)

### 【パラメータ】

WEVENT \*evt ウィンドウイベント  
W cmd 要求コマンド。W\_REDISP であってはならない。  
UW tmout 応答待ちのタイムアウトを msec 単位で指定するものであり、=0の場合は、タイムアウト無しを意味する。

## 【リターン値】

0 正常(関数値は(W\_ACK/W\_NAK/0))  
< 0 エラー(エラーコード)

## 【解説】

cmd で指定した要求イベントに対する ACK 応答、または NAK 応答を待ち、得られた応答イベントを evt で指定した領域に格納する。evt = NULL の場合は、得られた応答イベントは格納されない。

関数値として、ACK 応答が得られた場合は W\_ACK、NAK 応答が得られた場合は W\_NAK、タイムアウトとなった場合は 0 が戻る。

## 【エラーコード】

EX\_ADR : アドレス(evt)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(cmdが不正)。

wreq\_dsp

ウィンドウの再表示要求

## 【形式】

ERR wreq\_dsp(W wid)

## 【パラメータ】

W wid ウィンドウID

## 【リターン値】

0 正常  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウの再表示領域を作業領域全体とし、再表示要求イベントを送る。ウィンドウの枠部分も再表示される。

wid = 0 の場合は、すべてのウィンドウ(内部ウィンドウを除く)に対して再表示要求を行ない、これは、画面全体を書き直す場合に使用される。

# wfnd\_wnd

## ウィンドウの位置サーチ

### 【形式】

W wfnd\_wnd(PNT \*gpos, PNT \*lpos, W \*wid)

### 【パラメータ】

PNT \*gpos 絶対座標の点。  
PNT \*lpos 相対座標位置が格納される。  
W \*wid ウィンドウIDが格納される。

### 【リターン値】

#### 0 正常(位置コード)

W\_WORK (0) -- 作業領域内  
W\_FRAM (1) -- ウィンドウ枠  
W\_PICT (2) -- ピクトグラム (\*)  
W\_TITL (3) -- タイトル文字列 (\*)  
W\_LTHD (4) -- ハンドル (左上) (\*)  
W\_RTHD (5) -- ハンドル (右上) (\*)  
W\_LBHD (6) -- ハンドル (左下) (\*)  
W\_RBHD (7) -- ハンドル (右下) (\*)  
W\_RBAR (8) -- 右スクロールバー (\*)  
W\_BBAR (9) -- 下スクロールバー (\*)  
W\_LBAR (10) -- 左スクロールバー (\*)

(\*) 対応する部分が存在しないウィンドウの場合は、すべてW\_FRAM となる。

#### < 0 エラー(エラーコード)

### 【解説】

gpos で指定した絶対座標の点が、どのウィンドウのどの位置であるかを見付け、wid で指定した領域に対応するウィンドウIDを、lpos で指定した領域に、そのウィンドウでの相対座標位置を格納する。関数値として以下の位置コードを戻す。lpos = NULL の場合は、相対座標位置は格納されない。

指定した点がウィンドウにも属していない場合はwid には0、lpos には指定した gpos そのものが格納され、関数値は0が戻る。パネルウィンドウもサーチの対象となり、パネルウィンドウの領域の位置コードはW\_WORK となる。

### 【エラーコード】

EX\_ADR : アドレス(gpos, lpos, wid)のアクセスは許されていない。

## wget\_act

入力受付状態のウィンドウの取出し

### 【形式】

WID wget\_act(W \*pid)

### 【パラメータ】

W \*pid プロセスIDが格納される

### 【リターン値】

0 正常(関数値は入力受付状態のウィンドウID)  
< 0 エラー(エラーコード)

### 【解説】

現在、入力受付状態である主ウィンドウのウィンドウ ID を関数値として戻す。pid が NULL でない場合は、その管理プロセス、即ちアクティブプロセスのプロセス ID を pid で指定した領域に格納する。自プロセスが管理プロセスの場合は \*pid には 0 が格納される。

パネルウィンドウも対象となるため、パネルウィンドウが存在する場合は、必ず、パネルウィンドウのIDが戻ることになる。

### 【エラーコード】

EX\_ADR : アドレス(pid)のアクセスは許されていない。  
EX\_ADR : 入力受付状態のウィンドウがない、又は確定していない。

## wget\_tit

ウィンドウタイトルの取出し

### 【形式】

W wget\_tit(W wid, W \*pict, TC \*title)

### 【パラメータ】

W wid ウィンドウID  
W \*pict ピクトグラムデータ番号が格納される  
TC \*title タイトル文字列が格納される



## 【リターン値】

0 正常(関数値は虚身ウィンドウ状態(1:虚身ウィンドウ,0:正常))  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウのピクトグラムの変換データ番号、タイトル文字列を取り出し、それぞれ、pict、title で指定した領域に格納する。

title としては 96 + 1 文字分の領域を確保しておく必要がある。

関数値として、虚身ウィンドウの場合は "1"、そうでない場合は "0" が戻る。title = NULL または pict = NULL の場合は対応するデータは格納されない。

## 【エラーコード】

EX\_ADR : アドレス(pict,title)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

## wset\_tit

ウィンドウタイトルの変更

## 【形式】

ERR wset\_tit(W wid, W pict, TC \*title, W mode)

## 【パラメータ】

W wid ウィンドウID  
W pict ピクトグラムのデータ番号  
TC \*title タイトル文字列  
W mode !=0 の場合は虚身ウィンドウ

## 【リターン値】

0 正常  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウのピクトグラムを pict で指定したデータ番号に、タイトル文字列を title で指定した文字列に変更して、再表示を行なう。

mode = 0 の場合は、通常の表示を行ない、0 の場合は、虚身ウィンドウとして表示する。

pict < 0 の場合は、ピクトグラムの変更なし、title = NULL の場合はタイトル文字列の変更なし

しを意味する。

### 【エラーコード】

EX\_ADR : アドレス(title)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

## wget\_sts

ウィンドウ状態の取出し

### 【形式】

W wget\_sts(W wid, WDSTAT \*stat, WDDISP \*atr)

### 【パラメータ】

|        |       |         |
|--------|-------|---------|
| W      | wid   | ウィンドウID |
| WDSTAT | *stat | ウィンドウ状態 |
| WDDISP | *atr  | 表示属性    |

### 【リターン値】

0 正常(関数値は入力受付状態(1:入力受付状態、0:入力不可状態))  
< 0 エラー(エラーコード)

### 【解説】

widで指定したウィンドウの現在の状態、および表示属性を取り出し、それぞれstat、atrで指定した領域に格納する。関数値として指定したウィンドウが入力受付状態の時は1、入力不可状態の時は0を戻す。statまたはatrがNULLの場合は、対応するデータは格納されない。

全面モードの場合にstat->atrにWA\_FULLがセットされる。またstat->atrによって、WA\_STD指定時には正規座標が、WA\_WORK指定時には作業領域座標が、WA\_FRAME指定時には外枠座標が、stat->rに格納される。

### 【エラーコード】

EX\_ADR : アドレス(stat,atr)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

## wget\_gid

ウィンドウの描画環境IDの取出し

## 【形式】

GID wget\_gid(W wid)

## 【パラメータ】

W wid ウィンドウID

## 【リターン値】

0 正常(関数値は描画環境ID)  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウの描画環境 ID を関数値として戻す。

## 【エラーコード】

EX\_WID : ウィンドウ(wid)は存在していない。

# wset\_bgp

ウィンドウの背景パターンの設定

## 【形式】

ERR wset\_bgp(W wid, PAT \*pat)

## 【パラメータ】

W wid ウィンドウID  
PAT \*pat 設定する背景パターン

## 【リターン値】

0 正常  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウの背景パターンを、 pat で指定したパターンに設定する。設定するパターンは、その時点のウィンドウの描画環境のパターンサイズに合ったものと見なされる。

## 【エラーコード】

- EX\_ADR : アドレス(pat)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である(pat の内容が不正)。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

## wget\_bgp

ウィンドウの背景パターンの取出し

### 【形式】

W wget\_bgp(W wid, PAT \*pat, W size)

### 【パラメータ】

|     |      |                |
|-----|------|----------------|
| W   | wid  | ウィンドウID        |
| PAT | *pat | 取り出す背景パターン格納場所 |
| W   | size | pat の領域サイズ     |

### 【リターン値】

- 0 正常(関数値は背景パターンのサイズ)  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウの背景パターンを、 pat で指定した領域に格納する。 size は pat で指定した領域のバイトサイズであり、この領域が小さくて背景パターンを格納できない場合は、EX\_PAR のエラーとなる。関数値として、実際に格納した背景パターンのサイズ(バイト数)が戻る。 pat = NULL の場合は、背景パターンの格納は行なわずに格納に必要なサイズが関数値として戻る。

### 【エラーコード】

- EX\_ADR : アドレス(pat)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(sizeが小さすぎる)。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

## wera\_wnd

ウィンドウの背景パターンでの塗り潰し

### 【形式】

ERR wera\_wnd(W wid, RECT \*r)

### 【パラメータ】

W wid ウィンドウID  
RECT \*r 矩形領域

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウの \*r で指定した矩形領域をウィンドウの背景パターンで塗り潰す。  
r=NULL の場合は、ウィンドウの作業領域全体を背景パターンで塗り潰す。

### 【エラーコード】

EX\_ADR : アドレス(r)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

## wchg\_dsp

ウィンドウのデフォルト表示属性の変更

### 【形式】

ERR wchg\_dsp(WDDISP \*atr, PAT \*bgpat)

### 【パラメータ】

WDDISP \*atr 表示属性  
PAT \*bgpat 背景パターン

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

ウィンドウのデフォルト表示属性を atr で指定した内容に変更し、デフォルトの背景パターンを bgpat で指定した内容に変更する。atr = NULL、または bgpat = NULL の場合はそれぞれ対応する属性を変更しないことを意味する。

## 【エラーコード】

EX\_ADR : アドレス(atr, bgpat)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である。

## wget\_bar

スクロールバーのパーツIDの取出し

## 【形式】

ERR wget\_bar(W wid, W \*rbar, W \*bbar, W \*lbar)

## 【パラメータ】

W wid ウィンドウID  
W \*rbar 右スクロールバーのパーツIDの格納場所  
W \*bbar 下スクロールバーのパーツIDの格納場所  
W \*lbar 左スクロールバーのパーツIDの格納場所

## 【リターン値】

0 正常  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウに付属する 3 種類のスクロールバーのパーツIDを取り出して、 rbar, bbar, lbar で指定した領域にそれぞれ格納する。 rbar, bbar, lbar がそれぞれ NULL の場合は対応するパーツIDは格納されない。

ウィンドウにスクロールバーが存在する場合は、この関数により、そのパーツIDを取り出して、スクロールバーの設定値を表示状況に合わせて適宜変更する必要がある。

## 【エラーコード】

EX\_ADR : アドレス(rbar, bbar, lbar)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

## wget\_dat

ユーザデータの取出し

## 【形式】

ERR wget\_dat(W wid, W \*dat)

### 【パラメータ】

W wid ウィンドウID  
W \*dat ユーザデータの格納場所

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウのユーザデータを取り出し、 dat で指定した領域に格納する。

ユーザデータはウィンドウの管理情報の一部としてアプリケーションが任意に使用することができる1つの W データである。

### 【エラーコード】

EX\_ADR : アドレス(dat)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。

## wset\_dat

ユーザデータの設定

### 【形式】

ERR wset\_dat(W wid, W dat)

### 【パラメータ】

W wid ウィンドウID  
W dat ユーザデータ

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウのユーザデータを dat の値に設定する。

ユーザデータはウィンドウの管理情報の一部としてアプリケーションが任意に使用することができる1つのWデータである。

### 【エラーコード】

EX\_WID : ウィンドウ(wid)は存在していない。

## wget\_wrk

ウィンドウ作業領域の取出し

### 【形式】

ERR wget\_wrk(W wid, RECT \*r)

### 【パラメータ】

W wid ウィンドウID  
RECT \*r 作業領域の格納場所

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウの作業領域の相対座標値を取り出し、r で指定した領域に格納する。

### 【エラーコード】

EX\_ADR : アドレス(r)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

## wset\_wrk

ウィンドウ作業領域の変更

### 【形式】

ERR wset\_wrk(W wid, RECT \*r)

### 【パラメータ】



W wid ウィンドウID  
RECT \*r 作業領域矩形

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウの作業領域の左上の点の相対座標を、r で指定した矩形の左上の点の座標値と一致するように変更し、変更後の作業領域の相対座標値を r で指定した領域に格納する。即ち、r->lefttop はそのまま、r->rightbot のみが設定される。

作業領域座標の変更により、対応する描画環境のビットマップ境界、フレーム長方形、および表示長方形の座標値が指定した位置になるように移動する。

ウィンドウの作業領域の表示は一切変化しない。

### 【エラーコード】

EX\_ADR : アドレス(r)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない。

## wget\_org

ウィンドウの生成元の取出し

### 【形式】

ERR wget\_org(W wid, W \*parent, RECT \*org)

### 【パラメータ】

W wid ウィンドウID  
W \*parent 親ウィンドウIDの格納場所  
RECT \*org 生成元矩形の格納場所

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウの親ウィンドウ ID、生成元と取り出し、それぞれ parent、org で指定

した領域に格納する。生成元は、親ウィンドウの相対座標で表わされる。

主ウィンドウ以外の場合はエラーとなる。

### 【エラーコード】

EX\_ADR : アドレスのアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない  
(主ウィンドウでない)。

## wset\_org

ウィンドウの生成元の設定

### 【形式】

ERR wset\_org(W wid, W parent, RECT \*org)

### 【パラメータ】

|      |        |          |
|------|--------|----------|
| W    | wid    | ウィンドウID  |
| W    | parent | 親ウィンドウID |
| RECT | *org   | 生成元矩形    |

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウの親ウィンドウ ID を、parent の値に、生成元を、\*org の値に設定する。parent < 0 の場合には、parent の変更なし、org = NULL の場合は、生成元の変更なしを意味する。生成元は、親ウィンドウの相対座標で設定する。parent = 0 または parent = wid の場合は org の値に無関係に、生成元を無くすことを意味する。

主ウィンドウ以外の場合はエラーとなる。

### 【エラーコード】

EX\_ADR : アドレス(org)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_WND : ウィンドウ(wid)のタイプ/状態では適用されない  
(主ウィンドウでない)。

# wlst\_wnd

ウィンドウのリストアップ

## 【形式】

W wlst\_wnd(W wid, W size, W \*wids)

## 【パラメータ】

W wid ウィンドウID  
W size wids が示す領域のワード数  
W \*wids wid 格納領域

## 【リターン値】

0 正常  
< 0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウの子ウィンドウのウィンドウ ID を取り出して、wids で指定した領域に格納する。wid = 0 の場合は、現在存在しているすべてのウィンドウを、wid < 0 の場合は、-wid で指定した値をプロセス ID として、そのプロセスの管理するウィンドウのウィンドウ ID を取り出す。wids は size で示されるワード数の配列領域である。関数値として存在するウィンドウの個数が戻る。

取り出されるウィンドウ ID には、内部ウィンドウ、パネルウィンドウも含まれる。

ウィンドウの個数よりも size が小さい場合は、size 個のウィンドウ ID のみが格納される。size 0 の場合または、wids = NULL の場合は、単に存在するウィンドウの数が戻るだけとなる。

## 【エラーコード】

EX\_ADR : アドレス(wids)のアクセスは許されていない。

# wget\_dmn

表示管理メニューの取り出し

## 【形式】

ERR wget\_dmn(TC \*\*dmenu)

## 【パラメータ】

TC \*\*dmenu 表示管理メニューの格納領域

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

dmenuで指定した領域に「表示管理メニュー」項目リストへのポインタを設定する。実際の項目リストは共有メモリ領域に作成され、そのポインタがdmenuに設定される。不能項目フラグ、選択フラグ等もその時点の「表示管理メニュー」の状態に合わせて設定される。

共有メモリ領域に作成されたメニュー項目は、次にwget\_dmn()が実行されるまで存在が保証される。

### 【エラーコード】

EX\_ADR : アドレス(dmenu)のアクセスは許されていない。

wexe\_dmn

表示管理メニューの実行

### 【形式】

W wexe\_dmn(W item)

### 【パラメータ】

W item メニュー項目番号

### 【リターン値】

0 正常(関数値は切り換えた先のウィンドウID/0)  
< 0 エラー(エラーコード)

### 【解説】

wget\_dmn()で得られた、「表示管理メニュー」のitemで指定した項目を実行する。itemの上位バイトは無視され、下位バイトが1～の項目番号を示す。項目番号が不正の場合はEX\_PARのエラーとなる。

関数値として、入力受付状態のウィンドウが切り換わった場合は、切り換えた先のウィンドウIDが戻り、切り換わらない場合は、0が戻る。

### 【エラーコード】

EX\_PAR : パラメータが不正である(item)。  
EX\_WPRC : アクティブプロセスでない。

## wdef\_fep

フロントエンドプロセスの登録

### 【形式】

ERR wdef\_fep(W onoff)

### 【パラメータ】

W onoff フロントエンドプロセスのオン/オフ

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

自プロセスをフロントエンドプロセスとして登録、またはフロントエンドプロセスとしての登録を解消する。onoff > 0 の場合は一般フロントエンドプロセスとして、onoff < 0 の場合は特殊フロントエンドプロセスとして登録、onoff = 0 の場合は解消となる。登録していない状態で、解消しようとした場合は、何も行なわれない。

フロントエンドプロセスは最大4つまで同時に登録可能であり、登録した逆順にイベントが渡されてシーケンシャルに処理されいく。

フロントエンドプロセスは終了すると、自動的にフロントエンドプロセスとしての登録は解消される。

### 【エラーコード】

EX\_LIMIT : システムの制限を超えた  
(既に最大個数のフロントエンドプロセスが登録されている)。  
EX\_WPRC : ウィンドウ(wid)の管理プロセスでない  
(通常ウィンドウを開いたまま登録した、  
前面ウィンドウを開いたまま登録解消した)。

## wchk\_dck

ダブルクリック操作のチェック

### 【形式】

W wchk\_dck(UW first)

### 【パラメータ】

UW first 最初のPDのプレスで得られたイベントの発生時刻(time)

### 【リターン値】

0 正常(関数値はPD操作タイプ)  
< 0 エラー(エラーコード)

### 【解説】

PD の操作がダブルクリック操作か否かをチェックし、その結果を関数値として戻す。

関数値として以下のいずれかの値が戻る。

W\_PRESS : プレス操作 :

イベントは何も取り除かれない。

W\_QPRESS : クイックプレス操作 :

EV\_BUTUP および、続く EV\_BUTDWN イベントは取り除かれる。

W\_CLICK : クリック操作 :

EV\_BUTUP イベントは取り除かれる。

W\_DCLICK : ダブルクリック操作 :

EV\_BUTUP、EV\_BUTDWN、および2回目の EV\_BUTUP イベントは取り除かれる。

### 【エラーコード】

EX\_WPRC : ウィンドウ(wid)の管理プロセスでない  
(アクティブプロセスでない)。

wchg\_dck

ダブルクリック間隔の変更

### 【形式】

W wchg\_dck(W time)

### 【パラメータ】

W time ダブルクリック間隔(秒単位)

### 【リターン値】

0 正常(変更以前のダブルクリック間隔)  
< 0 エラー(エラーコード)

#### 【解説】

wchk\_dck() で使用するダブルクリック間隔を time で指定した値に変更し、変更以前の値を関数値として取り出す。time 0 の場合は変更せずに現在値を取り出す。

なお、システムスタートアップ時にはダブルクリック間隔として適当なデフォルト値が設定されているものとする。

#### 【エラーコード】

発生しない。

wchg\_ful

フルスクリーンモードの設定

#### 【形式】

ERR wchg\_ful(void)

#### 【パラメータ】

なし

#### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

#### 【解説】

この関数を実行した後、同一プロセスから最初に開かれた主ウィンドウが、自動的にフルスクリーンモードとなり、システムメッセージパネル領域を含めた全画面に表示される。システムメッセージパネルは隠れて見えなくなる。

フルスクリーンモードで開くことの出来るのは、主ウィンドウのみである。

開かれたウィンドウの属性は、変形不可、枠、タイトルバー、スクロールバーなしとなる。指定した表示位置は無視され、画面全体が作業領域となり、移動、拡大、縮小は出来ない。

フルスクリーンモードのウィンドウが開いている間は、同一プロセスからも、別の主ウィンドウを開くことは出来ない。フルスクリーンウィンドウの従属ウィンドウ及び、パネルは開くことが出来る。また、フルスクリーンモードのウィンドウから、他のウィンドウへ入力受付状態を移すことは出来ない。

ウィンドウが閉じるかプロセスが終了すると、自動的にフルスクリーンモードは解除される。

フルスクリーンモードで、下記の関数を実行した場合、エラーとなる場合がある。

wopn\_wnd()

EX\_WND : フルスクリーンモードのウィンドウが開かれている。  
(別の主ウィンドウを開こうとした場合)

wswi\_wnd()

EX\_WND : フルスクリーンモードのウィンドウが開かれている。

wexe\_dmn()

EX\_PAR : フルスクリーンモードのウィンドウが開かれている。  
(フルスクリーンモードでは、表示管理メニューの項目は、  
全て不能状態になる)

wmov\_wnd(), wmov\_drg(), wrsz\_wnd(), wrsz\_drg(), wchg\_wnd()

EX\_WND : ウィンドウのタイプ/状態では実行出来ない。  
(フルスクリーンモードのウィンドウに対して実行した場合)

ウィンドウ属性に WA\_SFULL を指定することでフルスクリーンウィンドウを開く。また、WA\_SFULL によるフルスクリーンウィンドウは複数開くことができるが、wchg\_ful によるフルスクリーンウィンドウは1つしか開くことができない。

### 【エラーコード】

EX\_LIMIT : システムの制限を超えた(既に別のプロセスが実行中)。

## wcnv\_rec

枠座標の変換

### 【形式】

ERR wcnv\_rec(UW attr, RECT \*r, UW opt)

### 【パラメータ】

|      |      |           |
|------|------|-----------|
| UW   | attr | 変換元の属性    |
| RECT | *r   | 枠座標へのポインタ |
| UW   | opt  | 変換先の枠座標指定 |

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

attr のウィンドウ属性を持った、r で示されるウィンドウ枠を、opt で指定した枠座標に変換する。結果は r に直接戻される。opt には、WA\_STD, WA\_WORK, WA\_FRAME のいずれか1つが指定できる。たとえば wcnv\_rec(WA\_SIZE|WA\_WORK, r, WA\_FRAME) とすると、WA\_SIZE 属性を持つウィンドウにおいて、r は作業領域座標から外枠座標に変換される。



なこのシステムコールはウィンドウを開いていない状態でも実行することができる。また、イネーブルウェアの設定によって実行結果は変化する。

## 【エラーコード】

EX\_ADR : アドレス(r)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(attr, optが不正)。

# wget\_inf

ウィンドウ情報レコードの取り出し

## 【形式】

ERR wget\_inf(W item, VP buf, W size)

## 【パラメータ】

W item ウィンドウ情報レコード番号  
VP buf 格納領域へのポインタ  
W size 格納領域のバイトサイズ

## 【リターン値】

0 正常(レコードのバイトサイズ)  
< 0 エラー(エラーコード)

## 【解説】

itemで示されるウィンドウ情報レコードを、bufで示される領域に取り出し、レコード全体のバイトサイズを返す。bufからsizeバイトがアクセスできる必要がある。bufがNULLの場合やsize 0の場合はレコードの内容は取り出されないが、レコードの全体サイズを得ることができる(エラーにはならない)。

領域が不足した場合(関数値 > sizeの場合)はsizeバイトまでが格納されるが、この場合はレコードの種類(ポインタを含んでいる構造体など)によっては取り出される内容は保証されない。

## 【エラーコード】

EX\_ADR : アドレス(buf)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(itemが不正)。  
EX\_NOSPT : itemのウィンドウ情報レコードはサポートされていない。  
EX\_NOSPC : システムのメモリが不足した。

# wget\_inf

ウィンドウ情報レコードの取り出し

## 【形式】

ERR wget\_inf(W item, VP buf, W size)

## 【パラメータ】

W item      ウィンドウ情報レコード番号  
VP buf      格納領域へのポインタ  
W size      格納領域のバイトサイズ

## 【リターン値】

0 正常(実際に書き込まれたバイトサイズ)  
< 0 エラー(エラーコード)

## 【解説】

item で示されるウィンドウ情報レコードを、buf で示される内容で更新し、実際に格納されたバイトサイズを返す。buf から size バイトがアクセスできる必要がある。buf が NULL の場合や size 0 の場合は、レコードの内容をシステムのデフォルト値に戻す。

buf で示される内容が不正な場合には、EX\_PAR のエラーが返る。

## 【エラーコード】

EX\_ADR      : アドレス(buf)のアクセスは許されていない。  
EX\_PAR      : パラメータが不正である(item, bufが不正)。  
EX\_NOSPT    : itemのウィンドウ情報レコードはサポートされていない。  
             または、bufで示されるデータ形式はサポートされていない。  
EX\_NOSPC    : システムのメモリが不足した。

---

[この章の目次にもどる](#)

[前頁:第3章 外殻にもどる](#)

[次頁:3.2 メニューマネージャにすすむ](#)

[この章の目次にもどる](#)

[前頁:3.1 ウィンドウマネージャにもどる](#)

[次頁:3.3 パーツマネージャにすすむ](#)

---

## 3.2 メニューマネージャ

### 3.2.1 メニューマネージャの機能

#### 3.2.1.1 概要

メニューマネージャは外殻の HMI 機能の 1 つとして位置付けられ、標準メニュー、および汎用メニューの表示 / 操作 / 登録 / 削除等の機能を提供している。アプリケーションプログラムは、このメニューマネージャを通して容易に標準メニューおよび汎用メニューを使用することができる。

本仕様書では、メニューの標準的な形状に関して説明しているが、表示形状の詳細はインプリメントに依存する。

#### 3.2.1.2 標準メニュー

1 レベルのポップアップメニューを標準メニューとしているが、特殊な場合として 1 レベルのポップアップメニューも存在する。

最初に表示されるメニューをメインメニューと呼び、項目を選択することにより表示される 2 段目のメニューをサブメニューと呼ぶ。サブメニューを開くための項目を親項目 ( 代表項目 ) と呼び、サブメニューを開かずに最終的な指示を与える項目を子項目 ( 末端項目 ) と呼ぶ。

通常、メインメニューの項目は親項目であるが、子項目をメインメニューの項目とすることも可能である。

メニューは項目名を縦方向に並べた矩形であり。パネルと同様の逆 L 字型に引かれた陰影を持っている。項目名の文字はシステム標準の文字が使用され、項目の位置にポインタが行くと項目名が反転表示される。

また、メニュー項目が多くあった場合に、煩雑になるのを避け視認性を上げるために、項目をグループ分けするための点線を項目間に挿むことができる。

キーマクロで起動するように設定されているメニューの子項目については、その項目と同じ行に右詰めで、設定されているマクロ文字があらわれる。

項目が多すぎて、画面に収まりきらない場合は、左右に 2 分割されて表示される。

メニューの項目は大きく以下のように分類される。

#### 状態項目

選択されたことにより、一つの状態を設定するようなコマンドでは、その項目の前にインジケータが付く。状態が設定されている場合インジケータが ON 状態になる。

#### 動作項目

選択されたことにより即何らかの動作を引き起こす一般の項目を動作項目と呼ぶ。

## 不能項目

ある時点で使用不可能な選択できない項目を不能項目と呼び、その項目の文字は灰色化表示される。また、対応するサブメニューの全ての項目が不能項目の場合、その親項目も不能項目となり、対応するサブメニューは表示されることはない。灰色化表示とは、使用できないことを明確に見分けることができるようにした「弱い」表示である。

また、親項目が不能項目の場合は、選択できないため、対応するサブメニューは表示されることはない。

## 強調項目

強調するために、下線が引かれた項目を強調項目と呼ぶ。これは、文字属性を設定するメニューの設定項目において、その属性の文字データが存在する場合と、存在しない場合とがソフトウェアによって生成できる場合を表示上見分ける場合等に使用する。この場合データが存在する項目を強調項目とする。

メニューはポインタの下に表示され、最初にポインタの位置に表示される項目を起点項目と呼ぶ。起点項目の表示方法として以下のいずれかを指定することができる。

- 起点項目は、常にメインメニューの上から2番目の項目
- 起点項目は、最後に使用したメインメニューの項目

表示する位置をポインタに揃えたため、メニューがスクリーンの上、下端にぶつかるような場合は、スクリーンの端にメニューの端が揃うように移動して表示される。従って、この場合はポインタの位置は上記の指定した位置とは異なることになる。

サブメニューは反転状態にあるメインメニューの親項目の右横に、出来る限り親項目と上端を揃えるようにして現れる。このときメインメニューに対し左に8ドット程度オーバーラップして現れる。

親項目の右横に、上端を揃えたためスクリーンの下端にぶつかるような場合は、スクリーンの下端に下端を揃えた形で現れる。

メインメニューまたはサブメニューがスクリーンの右端にぶつかるような場合は、左にづらして表示される。サブメニューが、スクリーンの右端にはみ出してしまう場合は、サブメニューはメインメニューの右側ではなく、左側に表示される。親項目が1つのみの場合は、1レベルのメニューとなり、サブメニューがメインメニューのごとく直接表示される。表示の位置は、メインメニューの場合と同様である。

メインメニュー、サブメニューの項目の数は一画面に表示できる数を最大とする。なお、画面に表示しきれない項目を、スクロールや段組みによって表示するインプリメントも考えられる。また、各項目の文字数はインプリメント依存であるが、なるべく短いことが望ましい。

標準メニューの表示に関する属性として以下の項目が定義されており、個々のメニューに対して指定可能である。また、表示属性を指定せずにデフォルトとすることも可能である。

- メインメニューの枠の幅
- メインメニューの枠のパターン
- メインメニューの背景パターン
- メインメニューの項目文字の色

- サブメニューの枠の幅
- サブメニューの枠のパターン
- サブメニューの背景パターン
- サブメニューの項目文字の色
- インジケータの色
- キーマクロ文字の色

なお、メニューが表示されている間は、原則的にスクリーン上の他の全ての描画は停止することになる。

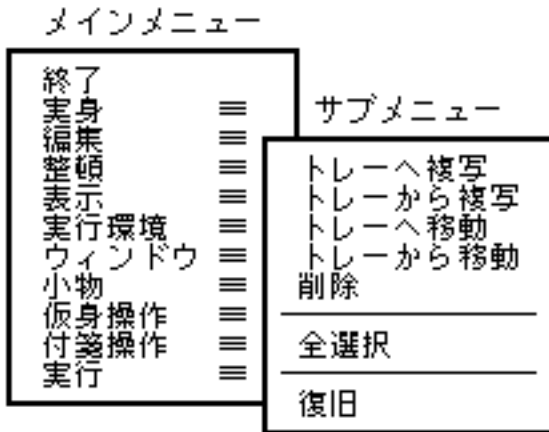


図 67 : 標準メニューの例

### 3.2.1.3 標準メニューの操作

#### メインメニューの表示

メニューボタンをプレスするか、または[命令]キーを押した状態でPDをプレスすると、メインメニューが表示され、ポインタの位置の起点項目は反転表示される。

#### 項目の反転

項目にポインタを合わせることにより、その項目が反転表示される。ただし、不能項目や区切り線は反転しない。

#### サブメニューの表示

親項目内でプレスまたはクリックするか、親項目内にある一定時間ポインタが滞在していると、サブメニューが表示される。

サブメニューが表示され、ポインタを親項目からサブメニューの子項目に移動するとき、斜めにポインタを移動すると、サブメニューを表示していない親項目上を通過することがある。このとき、通過途中の親項目より別のサブメニューが現われないよう、ある一定時間親項目の反転が保持され、サブメニューは表示され続ける。

サブメニューが表示されるまでの時間、及び親項目の反転が保持される時間をディレイタイムとよび、任意に設定することができる。

#### サブメニューの消去

サブメニューが表示されている親項目以外のメインメニューの項目をプレスまたはクリックするか、そのような項目にポインタがディレイタイム以上留まると、サブメニューは消去される。

また、サブメニューが表示された時点のポインタ位置からサブメニューとは反対方向に、親項目の領域内を1文字分水平移動するか、もしくは親項目の領域外へ移動するとサブメニューは消える。

### 項目の選択

子項目をクリックすることで項目が選択され、メニューは終了し消える。サブメニューの不能項目や区切り線が選択された場合は、メニュー動作は継続する。

ポインタがメニューを外れると、子項目の反転表示はなくなる。表示されているサブメニューは、そのまま表示されメインメニューの親項目の反転表示もそのまま残る。この状態でクリックすると、メニューの表示が消えるだけで何も起らない。

メニューボタンをクリックするとメニューは消え、何も起らない。

### サブメニューのロック

反転した親項目上でクリックすると、その時点で表示されているサブメニュー以外のサブメニューは表示できなくなる。この状態をロック状態という。つまりロック状態ではポインタを移動したり、別の親項目上にポインタをもってきてても、サブメニューが消えたり、新たなサブメニューが表示されたりはしない。

ロックされていない他の親項目上でクリックすることで、ロック状態は切り換わり、新しいサブメニューがロックされる。

ロック状態はロックされている親項目か、不能項目になっている親項目をクリックすることで解除される。

### 1レベルのメニュー

1レベルのメニューの場合は、直接(サブ)メニューが表示され、ポインタの位置の子項目は反転表示される。

ポインタが(サブ)メニューを外れると、反転した子項目がなくなるが、表示は残る。この状態でリリースすると何も起らない。

### マクロキー

マクロ文字のついた子項目は[命令]キーとともに指定された文字キーをプレスすることで、選択することができる。この場合は、メニューの表示は一切行なわれない。

### 3.2.1.4 標準メニューのデータ構造

#### メニューの表示属性

標準メニューは以下に示す表示属性を持ち、この属性に従って表示が行なわれる。

```
typedef struct menudisp {
```

```

UW      m_frame;    /* メインメニューの枠の幅/パターン */
UW      s_frame;    /* サブメニューの枠の幅/パターン */
UW      m_bgpat;    /* メインメニューの背景パターン */
UW      s_bgpat;    /* サブメニューの背景パターン */
UW      s_indpat;    /* インジケータのパターン */
COLOR   m_chcol;    /* メインメニューの項目文字色 */
COLOR   s_chcol;    /* サブメニューの項目文字色 */
COLOR   s_keycol;   /* サブメニューのキーマクロ文字色 */
} MENU_DISP;

```

m\_frame / s\_frame の内容は以下の通りである。

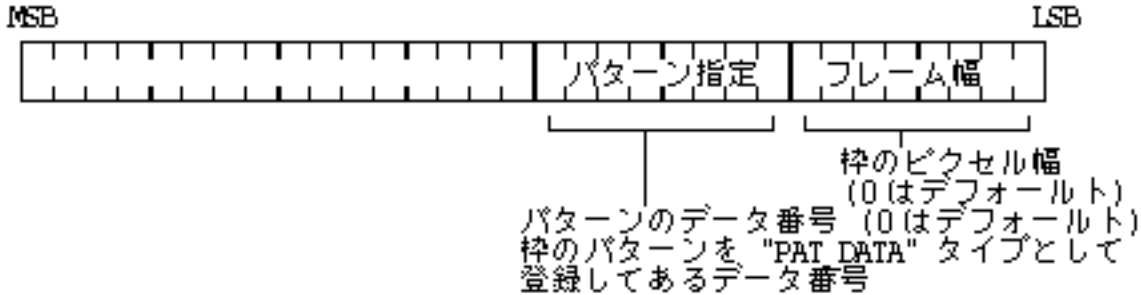


図 68 : 標準メニューのフレーム属性

枠の幅は最大 8 ピクセルであり、それ以上指定しても 8 ピクセル幅となる。

m\_bgpat / s\_bgpat は、それぞれ背景パターンを "PAT\_DATA" タイプとして登録してあるデータ番号で指定する。0 はデフォルトを意味する。

s\_indpat は、インジケータのパターンを "PAT\_DATA" タイプとして登録してあるデータ番号で指定する。0 はデフォルトを意味する。

m\_chcol / s\_chcol は、メニュー項目の文字および区切線の色を指定するが、メニュー項目毎に色の指定も可能となる。

COLORVAL で指定した値が < 0 の場合は、デフォルト値とすることを意味する。

デフォルト値はメニューマネージャにより適当な値が規定されているが、デフォルト値自体も変更可能である。デフォルト値を変更した場合は、変更以後に表示される全てのデフォルト属性を持つ標準メニューに反映される。

また、以下の表示位置に関する指定を行なうことができる。この指定は、すべてのメニューに共通であり、個々のメニューごとに指定することはできない。

起点位置指定:

M\_FIXPOS

起点項目は、常にメインメニューの上から 2 番目の項目

M\_LASTPOS

起点項目は、最後に使用したメインメニューの項目

項目リストの構造

メニューは複数の項目リストの順番を持った集まりとして定義される。1 つの項目リストは親項目と、それに対応する複数の子項目から構成され、1 つの文字列として以下の形で定

義される。項目の最大数は 128 個までであり、先頭の項目が親項目となり、続いて 127 個の子項目が続く。項目が 1 つのみの場合は親項目は存在せず、その項目は子項目と見なされる (実際に登録できる項目数は画面の大きさに依存する)。

ここで、[.]は省略可能、| は選択 (OR)、... は繰り返しを意味する。

<親項目指定>[<子項目指定>...]<TNULL>

<親項目指定> および <子項目指定> は、以下の形式となる。

<属性コード>[<文字属性指定> | <図形指定>][<マクロ文字>...]<項目文字列>

属性コード (メタコード) は、区切り文字としての意味と、続く項目の属性指定の意味を持つ。また <TNULL> は文字列の終端としての 0 である。

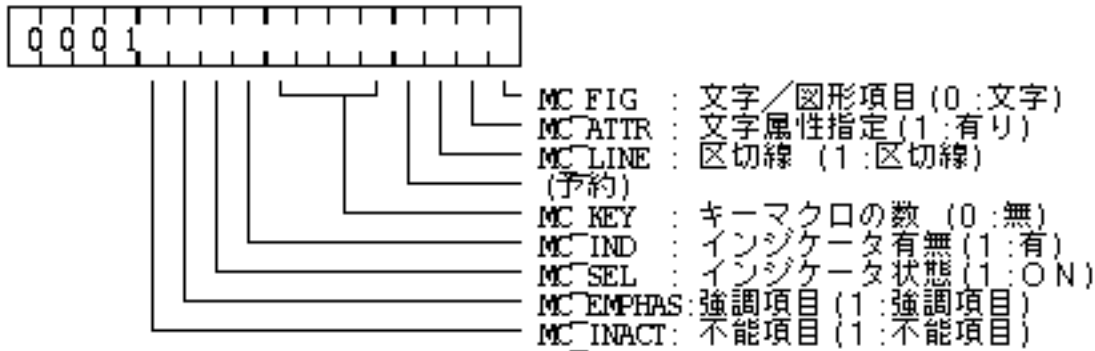


図 69: 属性コード

MC FIG:

- 0 -- 文字項目
- 1 -- 図形項目

MC\_ATTR:

- 0 -- 標準属性の文字列 ( MC FIG = 0 の時のみ有効 )
- 1 -- 指定属性の文字列 ( MC FIG = 0 の時のみ有効 )

文字項目の場合は、メニュー項目は文字列であり、MC\_ATTR = 1 の場合は直後に 12 バイトの文字属性指定コード列が存在し、その後項目としての文字列がくる。

MC\_ATTR = 0 の場合は文字属性指定コードは存在せず、直後に項目としての文字列がくる。文字列は属性コードまたは TNULL コード (=0) で終了し、後者の場合は全体の終了を意味する。文字列としては表示可能な任意の文字が許され、それ以外のものは無視される。

文字属性指定コードは以下に示す 12 バイト固定のデータである。

```
COLORVAL chcol -- 文字色指定 (-1は文字色指定無し)
WORD      class -- フォントクラス指定 (-1はデフォルトフォント)
UWORD     attr  -- フォント属性指定 (0は通常体)
PNT       csize -- フォントサイズ指定
           (csize.c.h または csize.c.v が 0のときデフォルトサイズ)
           システム標準文字サイズより大きいサイズは指定できない。
```

MC\_ATTR = 0は、文字色指定無し、デフォルトフォント、通常体を意味する。文字色指定無しの場合は、メニューの表示属性として定義された文字色が適用される。



図形項目の場合は、メニュー項目は図形の矩形表示となり、MC\_ATTRは無視される。  
この場合、直後に以下の2ワードの図形指定コードがくる。

```
TTTT DDDD DDDD DDDD  
0000 0000 000L LLLL
```

T:図形のデータタイプ ID

0:ポインタイメージ ("PTR\_DATA" )

1:ピクトグラムイメージ ("PICT\_DATA")

2:パターンデータ ("PAT\_DATA" )

3:ビットマップデータ ("BMAP\_DATA")

4~:予約

D:図形のデータ番号 (0の場合、または存在しない  
データの場合は何も表示されない)

L:図形の横幅文字数 (1~16)

図形項目は、Lで指定した1~16文字分の矩形領域に表示される。

MC\_LINE:

0 -- 通常項目

1 -- 区切線 (全体の矩形の横幅一杯の点線が表示される)

区切線の色は、メニューの文字色と同じである。MC\_LINE = 1の場合は、他のすべての属性は無視され、常に選択不可となる。親項目でMC\_LINE = 1の場合は、親項目、子項目は実際には表示されず無視される。

MC\_KEY:

指定されているキーマクロの個数を示す (0はキーマクロ無し)。最大15個までのキーマクロが1つの項目に対して指定可能であり、属性コードの直後に個数分のキーマクロ文字コードが続き、その先頭の1文字が項目の後ろに表示される。キーマクロ文字の色はメニューの表示属性として定義された色が適用される。

文字属性指定や図形指定の場合は、文字属性指定コードや図形指定コードの次にキーマクロ文字コードがくる。

キーマクロ文字の言語はシステムスクリプトとして解釈される。

MC\_IND:

0 -- 動作項目

1 -- 状態項目 (項目の前にインジケータが表示される。)

親項目の場合、MC\_INDは無視される。インジケータの色はメニューの表示属性として定義された色が適用される。

MC\_SEL:

0 -- 非選択状態 (インジケータはOFF表示)

1 -- 選択状態 (インジケータはON表示)

親項目の場合、または、MC\_IND = 0の場合は無視される。

MC\_EMPHAS :

- 0 -- 通常項目
- 1 -- 強調項目 (項目の表示に下線がつけられる)

MC\_INACT :

- 0 -- 正常項目
- 1 -- 不能項目 (灰色化表示され一切選択することはできない。親項目の場合はその子項目は表示されない)

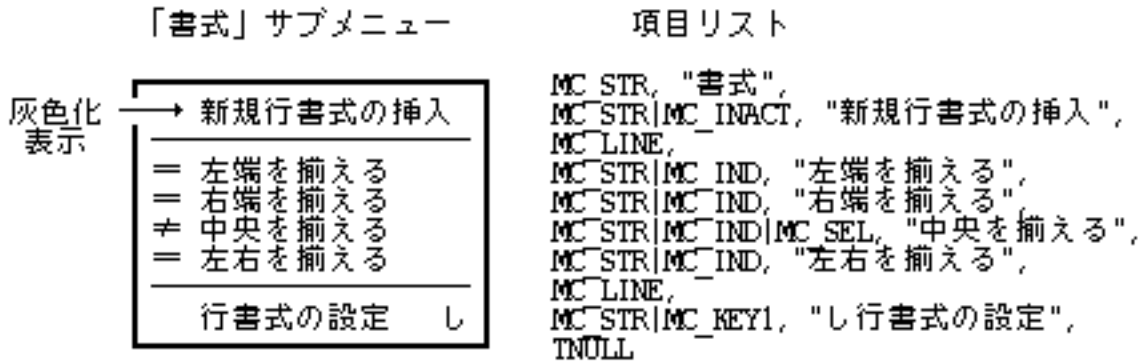


図 70 : メニュー項目の例

### メニュー項目の構造

1つのメニュー項目 (親項目と、子項目の組) は以下の形で定義される。

```
typedef struct menuitem {
    UW  inact;      /* 不能項目フラグ */
    UW  select;    /* 選択フラグ */
    W   desc;      /* 内部ディスクリプタ */
    W   dnum;      /* 項目リストのデータ番号 */
    TC  *ptr;      /* 項目リストへのポインタ */
} MENUITEM;
```

不能項目フラグ ( inact ) は以下に示す形で各項目にビット対応する。このフラグは、属性コードの MC\_INACT と同じ効果を持つが、属性コードを変更せずに一時的に不能項目とする場合に使用される。このフラグが 1 の場合、または属性コードの MC\_INACT が 1 の場合に不能項目と見なされる。このフラグは項目番号 0 ~ 31 に対応する項目にしか有効でないが、mchg\_atr() を使用することで項目番号 32 ~ 127 に対応する項目を設定することができる。

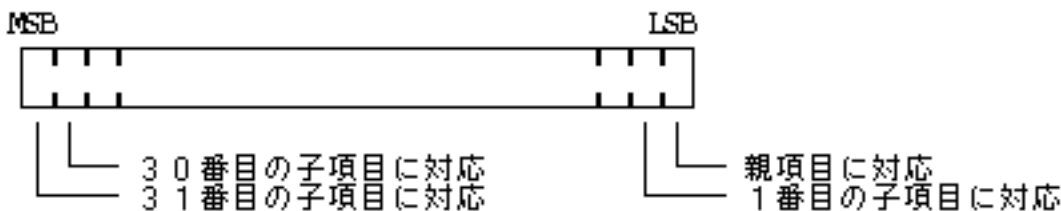


図 71 : 不能項目 / 選択フラグ

選択フラグ ( select ) も不能項目フラグと同様に各項目にビット対応しており、属性コードの MC\_SEL と同様の効果を持つものである。属性コードの MC\_IND = 1 の場合で、MC\_SEL =

1、またはこのフラグが1の場合に、インジケータがON表示となる。このフラグも項目番号0～31に対応する項目にしか有効でないが、mchg\_attr()を使用することで項目番号32～127に対応する項目を設定することができる。

動的に不能項目、選択項目が変化する場合は、通常、属性コードのMC\_INACT, MC\_SELは設定せず、不能項目フラグ、選択フラグを使用する。

項目リストは、メモリ上のデータとして直接、そのポインタで指定する場合と、データマネージャにより"TEXT\_DATA"タイプのデータとして管理されているものを指定する場合の2つがある。

ポインタで指定する場合は、dnum=0とし、ptrに項目リストへのポインタを設定する。ptr=NULLの場合は、メニューとしては何も表示されないが、メニューの登録後に親項目を所定の位置に追加したい場合には、その位置にあらかじめ、ptr=NULLのメニュー項目を定義しておく必要がある。

データマネージャ管理のデータの場合には、dnumに"TEXT\_DATA"タイプのデータ番号(>0)を設定する。この場合、ptrは無視される。

内部ディスクリプタ(desc)は、dnumで項目リストが指定されていた場合、その共有メモリのアドレスを入れる内部領域であり、定義/設定時には無視される。

#### メニュー全体の構造

メニュー全体は、メニュー項目の配列の形で定義される。

親項目の数が1の場合は、1レベルのメニューとなり、親項目は表示されないことになるが、メニュー項目としてはやはり親項目がなくてはいけない。ただし、この場合、親項目はその属性も含めて無視されるので、空の文字列であってもよい(不能項目であってはならない)。

メニューでの子項目は、以下の選択番号で表わされる。

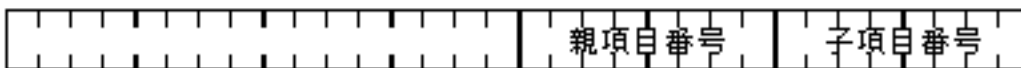


図 72 : 項目の選択番号

<親項目番号> はメニュー項目の定義順の番号で、先頭は0である。

<子項目番号> は1つの項目リスト内の項目順で、先頭は0であるが、先頭の親項目は選択されることがないので、実際には1からとなる(区切りの横線も1つの項目として数えられる)。子項目のみの項目リストの場合は、子項目番号は常に1となる。1レベルのメニューの場合は、親項目番号は常に0となり、子項目番号(1～)のみとなる。

#### 3.2.1.5 汎用メニュー

汎用メニューは、1レベルのポップアップメニューであり、標準メニューのサブメニューと似ているが、項目の配置を2次元的に任意に指定可能である。

通常、汎用メニューはコントロールパーツのスイッチと組み合わせて、ポップアップセレクトクを実現するために使用される。

なお、汎用メニューも標準メニューと同様に表示されている間は、原則的にスクリーン上の他の全ての描画は停止することになる。

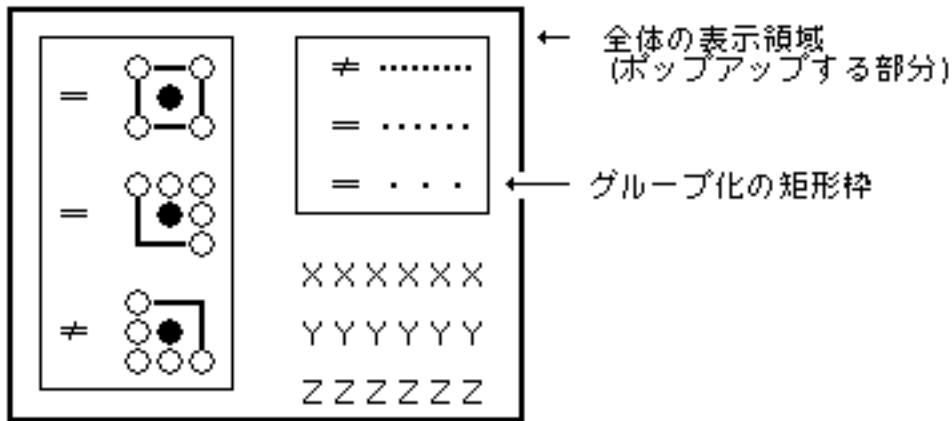


図 73 : 汎用メニューの例

### 3.2.1.6 汎用メニューの操作

- PD のドラッグにつれて、表示領域中の項目が仮選択状態になり、リリースした時点でその項目が選択され、表示領域は消える。
- ドラッグ中に表示領域外にポインタが出ると仮選択状態の項目はなくなり、リリースしても表示領域が消えるだけで何も起らない。
- 表示領域中の項目の矩形領域内にポインタが入っている場合に、その項目が仮選択されたことになる。どの項目の矩形領域にも入っていない場合、または不能状態の項目の矩形領域の場合は、仮選択の表示はなく、そこでリリースしても何も起らない。

### 3.2.1.7 汎用メニューのデータ構造

汎用メニューは、標準メニューのメニュー項目と良く似た以下の構造により定義される。メニューの項目は最大 128 個までである。

```
typedef struct gmenu {
    UW    frame;        /* フレーム属性 */
    UW    bgpat;        /* 背景パターン */
    UW    indpat;       /* インジケータのパターン */
    COLOR chcol;        /* 項目文字 / 区切枠の色 */
    RECT  area;         /* 全体の表示領域 */
    UW    inact;        /* 不能項目フラグ */
    UW    select;       /* 選択フラグ */
    W     desc;         /* 内部ディスクリプタ */
    W     dnum;         /* 項目リストのデータ番号 */
    TC    *ptr;         /* 項目リストへのポインタ */
    W     nitem;        /* 項目数 */
    RECT  r[32];        /* 項目の表示領域(nitem個:32より大きい場合もある) */
} GMENU;
```

frame の内容は以下の通りである。

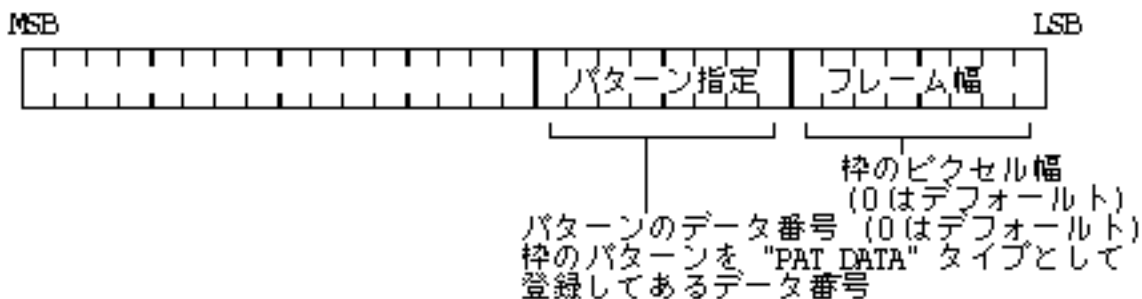


図 74 : 汎用メニューのフレーム属性

bgpat, indpat は、それぞれ背景パターンとインジケータのパターンを、"PAT\_DATA" タイプとして登録してあるデータ番号で指定したもので、0の場合は、デフォルト値を意味する。

chcol は、項目文字 / 区切枠の色を示し、<0の場合は、それぞれデフォルト値を意味する。

area は、ポップアップする全体の矩形領域を絶対座標で指定するものであるが、実際は表示時に位置を指定するため大きさのみ意味を持つ。従って、通常は左上の点を(0,0)と指定する。

inact, select, desc, dnum, ptr は、それぞれ標準メニューと同じ意味を持つ。ただし、親項目は存在せず、不能項目フラグ、選択フラグの親項目に対応するビット(LSB)は最初の(子)項目に対応することになり、項目リストの先頭は最初の(子)項目となる。メニューの項目は、1 ~ 128の番号で区別される。

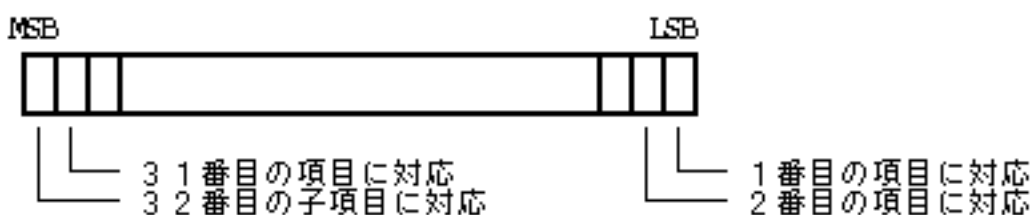


図 75 : 汎用メニューの不能項目 / 選択フラグ

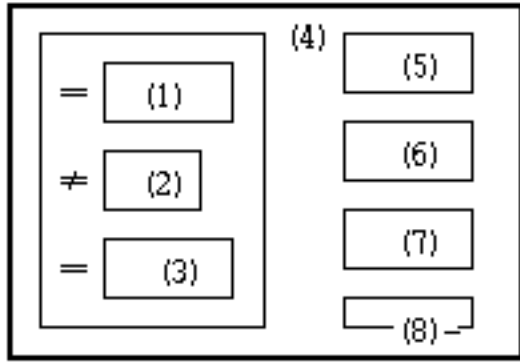
nitem は項目数であり、最大 128 である。

r は、各項目が表示される矩形領域の nitem 個の配列であり、全体の表示領域の左上を(0,0)とした座標で定義される。

項目リストの内容は、親項目が存在しない点を除いて、基本的に標準メニューの場合と同一であるが、属性コードの意味は以下の点異なる。

- MC\_FIG = 1 (図形項目) の場合、続く図形指定データの 2 ワード目の文字幅は必要であるが、意味を持たない。
- MC\_FIG = 0 (文字項目) の場合、表示領域内で左上詰めに表示される。
- MC\_LINE = 1 (区切線) の場合、横線ではなく、表示領域を囲む矩形枠となる。
- MC\_KEY は使用されず、無視される。
- MC\_IND = 1 の場合、インジケータは表示領域の直前に描画されるため、その領域を考慮する必要がある。

(0,0)



← 全体の表示領域  
(1)～(8) 各項目の表示領域  
(4) は MC\_LINE=1 であり  
矩形棒を表示する。

図 76 : 汎用メニューの項目領域

### 3.2.2 データ / 定数の定義

#### メニューID

```
typedef W      MNID;
```

#### メニュー項目

```
#define MC_LINE      0x1004 /* 区切線 */  
#define MC_KEY      0x10f0 /* キーマクロの個数マスク */  
#define MC_IND      0x1100 /* インジケータ有無 */  
#define MC_SEL      0x1200 /* インジケータ状態 */  
  
#define MC_KEY1     0x1010 /* キーマクロ 1 個 */  
#define MC_KEY2     0x1020 /* キーマクロ 2 個 */  
~  
#define MC_KEY14    0x10e0 /* キーマクロ 14 個 */  
#define MC_KEY15    0x10f0 /* キーマクロ 15 個 */
```

#### 標準メニュー定義

```
typedef struct menuitem {  
    UW  inact;      /* 不能項目フラグ */  
    UW  select;    /* 選択フラグ */  
    W   desc;      /* 内部ディスクリプタ */  
    W   dnum;      /* 項目リストのデータ番号 */  
    TC  *ptr;      /* 項目リストへのポインタ */  
} MENUITEM;
```

#### 標準メニュー表示属性定義

```
typedef struct menudisp {
```

```

UW      m_frame;      /* メインメニューの枠の幅/パターン */
UW      s_frame;      /* サブメニューの枠の幅/パターン */
UW      m_bgpat;      /* メインメニューの背景パターン */
UW      s_bgpat;      /* サブメニューの背景パターン */
UW      s_indpat;     /* インジケータのパターン */
COLOR   m_chcol;     /* メインメニューの項目文字色 */
COLOR   s_chcol;     /* サブメニューの項目文字色 */
COLOR   s_keycol;    /* サブメニューのキーマクロ文字色 */
} MENUISP;

```

## 汎用メニュー定義

```

typedef struct gmenu {
    UW      frame;      /* フレーム属性 */
    UW      bgpat;      /* 背景パターン */
    UW      indpat;     /* インジケータのパターン */
    COLOR   chcol;     /* 項目文字/区切枠の色 */
    RECT    area;      /* 全体の表示領域 */
    UW      inact;     /* 不能項目フラグ */
    UW      select;    /* 選択フラグ */
    W       desc;      /* 内部ディスクリプタ */
    W       dnum;      /* 項目リストのデータ番号 */
    TC      *ptr;      /* 項目リストへのポインタ */
    W       nitem;     /* 項目数 */
    RECT    r[32];     /* 項目の表示領域(nitem個) */
} GMENU;

```

## メニュー項目属性

```

#define M_STAT      0 /* 変更しない */
#define M_SEL       1 /* 選択状態とする */
#define M_NOSEL     4 /* 選択状態としない */
#define M_ACT       8 /* 不能項目としない */
#define M_INACT    2 /* 不能項目とする */

```

## メニュー表示位置指定

```

#define M_LASTPOS  0x0000 /* 起点項目は最後に使用したメイン項目 */
#define M_FIXPOS   0x0002 /* 起点項目は2番目のメイン項目 */

```

### 3.2.3 メニューマネージャの関数

#### mcre\_men

標準メニューの登録

## 【形式】

MNID      mcre\_men(W nitem, MENUITEM \*item, MENUDISP \*attr)

## 【パラメータ】

|          |       |          |
|----------|-------|----------|
| W        | nitem | 親項目数     |
| MENUITEM | *item | メニュー項目配列 |
| MENUDISP | *attr | メニュー表示属性 |

## 【リターン値】

0      正常      (関数値は登録されたメニュー ID)  
< 0      エラー      (エラーコード)

## 【解説】

nitem で指定した親項目数を持つ標準メニューを登録する。 item は nitem 個の要素数を持つメニュー項目配列へのポインタである。即ち、

item[n]      n 番目のメニュー項目 (n=0 ~ nitem-1)

nitem = 1 の場合は、1 レベルのメニューとなり item[0] で指定されるメニュー項目の親項目は実際には表示されない。 nitem が画面に表示できる最大項目数を越える場合は、この最大項目数が nitem として登録される。

attr はメニューの表示属性を指定するもので、 attr = NULL の場合は、デフォルト属性を意味する。表示属性で指定されたパターンデータ番号が不正な場合は、デフォルトパターンで登録される。

登録後も、 item で指定したメニュー項目内の ptr で示された項目リストは参照されるため、保存しておかなければいけない。なお、項目リストの内容を直接変更した場合は、必ず mset\_itm() により再設定しない限り表示は保証されない。

nitem と同様に項目リストの子項目数が画面に表示できる最大項目数を越える場合は、この最大項目数までが参照される。項目の最大文字数はインプリメントに依存し、長すぎる場合は途中までが参照される。

この関数ではメニューの登録のみを行ない、メニューの実際の表示は msel\_men() により行なわれる。

関数値としてメニュー ID (>0) が戻され、以後そのメニューの操作の際に使用する。メニュー ID はプロセス依存であるため登録したプロセスでのみ有効であり、メニュー ID を他のプロセスに渡してメニューの操作を行なうことはできない。

## 【エラーコード】



EX\_ADR : アドレス(item,attr)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である(nitem,メニュー項目が不正)。

メモリ領域が不足して登録できない場合、通常は核エラーコードが返される。

## mopn\_men

標準メニューのデータボックスからの登録

### 【形式】

MNID mopn\_men(W dnum)

### 【パラメータ】

W dnum MENU\_DATA タイプのデータ番号

### 【リターン値】

0 正常(関数値は登録されたメニューID)  
< 0 エラー(エラーコード)

### 【解説】

dnum で指定したデータ番号を持つ、"MENU\_DATA" タイプのデータボックス定義データにより標準メニューを登録する。

この関数は、メニューの定義データがデータボックスとして定義されている点を除いて、その動作は、mcre\_men() と全く同一である。

### 【エラーコード】

EX\_DNUM : データ(dnum等)はデータマネージャに登録されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である(nitem,メニュー項目が不正)。

メモリが不足して登録できない場合、通常は核エラーコードが返される。

## mdel\_men

標準メニューの削除

### 【形式】

ERR mdel\_men(W mid)

### 【パラメータ】

W mid メニューID

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

mid で指定した登録済み標準メニューを削除する。

登録したプロセスが終了した場合、メニューは自動的に削除されるが、基本的に不要になった時点で明示的に削除することが望ましい。

### 【エラーコード】

EX\_MID : メニュー(mid) は存在していない  
(標準メニューでない場合も含む)。

## mset\_men

標準メニューの選択動作

### 【形式】

W mset\_men(W mid, PNT pos)

### 【パラメータ】

W mid メニューID  
PNT pos 画面上の絶対座標位置

### 【リターン値】

0 正常(関数値は選択された項目の選択番号(選択されなかった場合は0))  
< 0 エラー(エラーコード)

### 【解説】

mid で指定した登録済み標準メニューを pos で指定したスクリーン上の絶対座標位置に表示し、PD の移動に従った選択動作を行ない、PD のクリックにより表示を消して選択された項目の選択番号を関数値として戻す。何も選択されなかった場合は、関数値は 0 となる。

pos は、メニューイベントが発生した位置を絶対座標を指定する。

メニュー動作に発生した EV\_BUTDOWN, EV\_BUTUP, EV\_KEYDOWN, EV\_KEYUP, EV\_AUTKEY は、イベントキューから取り除かれて捨てられる。それ以外のイベントはイベントキューに残される。

なお、この関数内ではポインタ形状変更は行なわないため、通常、実行前に「選択指」のポインタ形状にしておく必要がある。

イメージ退避領域確保できない場合は、メニュー動作終了後すべてのウィンドウに再表示要求が送られる。

### 【エラーコード】

EX\_MID : メニュー(mid)は存在していない  
(標準メニューでない場合も含む)。

mfnd\_key

キーマクロの検索

### 【形式】

W mfnd\_key(W mid, TC ch)

### 【パラメータ】

W mid メニューID

TC ch 文字コード

### 【リターン値】

- 0 正常 (関数値はキーマクロ項目の選択番号  
(定義されていなかった場合は0))
- < 0 エラー (エラーコード)

### 【解説】

mid で指定した標準メニュー内に、ch で指定した文字がキーマクロとして定義されているか否かを調べ、定義されていた場合は、関数値として対応する項目の選択番号を戻し、定義されていなかった場合は関数値として 0 を戻す。

この関数は、通常 [命令] キーと同時にキーが押された場合に呼ばれ、その時のキーに対応

する文字コードを ch として渡すことになる。システムメッセージパネルのクイック起動パネルは、命令 + 特殊文字として入力される。アプリケーションは、命令 + 文字を受け取ったらこの関数を実行してイベントを解釈する必要がある。

文字コードの言語はシステムスクリプトとして解釈される。

### 【エラーコード】

EX\_MID : メニュー (mid) は存在していない  
(標準メニューでない場合も含む)。

## mget\_itm

標準メニュー項目の取出し

### 【形式】

W mget\_itm(W mid, W pnum, MENUITEM \*item)

### 【パラメータ】

|          |       |        |
|----------|-------|--------|
| W        | mid   | メニューID |
| W        | pnum  | 親項目番号  |
| MENUITEM | *item | メニュー項目 |

### 【リターン値】

|     |             |
|-----|-------------|
| 0   | 正常          |
| < 0 | エラー(エラーコード) |

### 【解説】

mid で指定した標準メニューの pnum で指定した親項目番号 (0 ~ ) のメニュー項目の内容を取り出し、item で指定した領域に格納する。関数値は0が戻る。

### 【エラーコード】

EX\_ADR : アドレス(item)のアクセスは許されていない。  
EX\_MID : メニュー(mid)は存在していない  
(標準メニューでない場合も含む)。  
EX\_PAR : パラメータが不正である(pnum が不正)。

# mset\_itm

標準メニュー項目の設定

## 【形式】

ERR mset\_itm(W mid, W pnum, MENUITEM \*item)

## 【パラメータ】

|          |       |        |
|----------|-------|--------|
| W        | mid   | メニューID |
| W        | pnum  | 親項目番号  |
| MENUITEM | *item | メニュー項目 |

## 【リターン値】

|     |             |
|-----|-------------|
| 0   | 正常          |
| < 0 | エラー(エラーコード) |

## 【解説】

mid で指定した標準メニューの pnum で指定した親項目番号 (0 ~ ) のメニュー項目の内容を、 item で指定した領域に格納してある内容に変更する。

項目リストの内容を直接変更した場合は、メニュー項目の内容自体は変更なくても、mset\_itm() を行なう必要がある。

## 【エラーコード】

|         |                                       |
|---------|---------------------------------------|
| EX_ADR  | : アドレス(item)のアクセスは許されていない。            |
| EX_MID  | : メニュー(mid)は存在していない(標準メニューでない場合も含む)。  |
| EX_PAR  | : パラメータが不正である(pnum が不正、メニュー項目の内容が不正)。 |
| EX_DNUM | : データ番号で指定したデータがデータマネージャに登録されていない。    |

# mchg\_atr

標準メニューの項目属性の変更

## 【形式】

W mchg\_atr(W mid, W selnum, UW mode)

## 【パラメータ】

W mid メニューID  
W selnum 選択番号  
UW mode ::= ( M\_STAT ) ( ( M\_SEL M\_NOSEL ) | ( M\_ACT M\_INACT ) )

M\_STAT : 変更しない。(現在の属性を取り出す)  
M\_SEL : 選択状態とする。(選択フラグをセット)  
M\_NOSEL : 選択状態としない。(選択フラグをリセット)  
M\_ACT : 不能項目としない。(不能項目フラグをリセット)  
M\_INACT : 不能項目とする。(不能項目フラグをセット)

## 【リターン値】

0 正常(関数値は項目の現在の属性)  
< 0 エラー(エラーコード)

## 【解説】

mid で指定した標準メニューの selnum で指定した選択番号を持つメニュー項目の不能項目フラグ、選択フラグを変更して、その結果としての現在の属性を関数値として戻す。メニュー項目リスト内の属性コード自体は変更されない。

関数値として戻る現在の属性は以下に示す内容となる。この属性は項目リスト内の属性コードも考慮した実際の属性であるため、M\_NOSEL、または M\_ACT を指定した場合でも属性コードの値によっては、変化しない場合もある。

動作項目の場合 (MC\_IND=0) は、M\_SEL / M\_NOSEL の指定は無視され、関数値の M\_SEL は常に 0 となる。



M\_SEL 選択状態 (1 : 選択状態)  
M\_INACT 不能項目 (1 : 不能項目)

図 77 : mchg\_atr

## 【エラーコード】

EX\_MID : メニュー(mid)は存在していない(標準メニューでない場合も含む)。  
EX\_PAR : パラメータが不正である(selnum が不正)。

## mchg\_dsp

標準メニューのデフォルト表示属性の変更

## 【形式】

W mchg\_dsp(MENUDISP \*attr, W posattr)

### 【パラメータ】

MENUDISP \*attr メニュー表示属性  
W posattr 表示位置属性

### 【リターン値】

0 正常(関数値は変更後の表示位置属性)  
< 0 エラー(エラーコード)

### 【解説】

標準メニューのデフォルト表示属性を、attr で指定した内容に変更し、表示位置属性を posattr で指定した値に変更する。変更した内容は、以後の全てのメニュー表示に反映される。

attr = NULL の場合は表示属性は変更せず、posattr < 0 の場合は表示位置属性は変更しない。関数値として変更後の表示位置属性が戻る。

### 【エラーコード】

EX\_ADR : アドレス(attr)のアクセスは許されていない。  
EX\_DNUM : データ番号で指定したデータがデータマネージャに登録されていない。

mcre\_gmn

汎用メニューの登録

### 【形式】

MNID mcre\_gmn(GMENU \*gm)

### 【パラメータ】

GMENU \*gm 汎用メニュー

### 【リターン値】

0 正常(関数値は登録されたメニューID)  
< 0 エラー(エラーコード)

## 【解説】

gm で指定した汎用メニューを登録する。

登録後も、gm で指定したメニュー項目内の ptr で示された項目リストは参照されるため、保存しておかなければいけない。項目リストの内容は登録後は変更することはできない。

項目数が 32 を越える場合は、32 として登録する。

表示属性で指定されたパターンデータ番号が不正な場合は、デフォルトパターンで登録される。

この関数では、メニューの登録のみで、メニューの実際の表示は msel\_gmn() により行なわれる。

関数値としてメニュー ID (> 0) が戻され、以後そのメニューの操作の際に使用する。メニュー ID はプロセス依存であるため登録したプロセスでのみ有効であり、メニュー ID を他のプロセスに渡してメニューの操作を行なうことはできない。

## 【エラーコード】

- EX\_ADR : アドレス(gm)のアクセスは許されていない。
- EX\_NOSPC : システムのメモリ領域が不足した。
- EX\_PAR : パラメータが不正である(メニューの内容が不正)。

メモリ領域が不足して登録できない場合、通常は核エラーコードが返される。

# mopn\_gmn

汎用メニューのデータボックスからの登録

## 【形式】

MNID mopn\_gmn(W dnum)

## 【パラメータ】

W dnum GMENU\_DATA タイプのデータ番号

## 【リターン値】

- 0 正常(関数値は登録されたメニューID)
- < 0 エラー(エラーコード)

## 【解説】



dnum で指定したデータ番号を持つ、"GMENU\_DATA" タイプのデータボックス定義データにより汎用メニューを登録する。

この関数は、メニューの定義データがデータボックスとして定義されている点を除いて、その動作は、mcre\_gmn() と全く同一である。

### 【エラーコード】

EX\_DNUM : データ(dnum等)はデータマネージャに登録されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である(メニューの内容が不正)。

メモリ領域が不足して登録できない場合、通常は核エラーコードが返される。

## mdel\_gmn

汎用メニューの削除

### 【形式】

ERR mdel\_gmn(W mid)

### 【パラメータ】

W mid メニューID

### 【リターン値】

0 正常  
< 0 エラー(エラーコード)

### 【解説】

mid で示された登録済み汎用メニューを削除する。

登録したプロセスが終了した場合、メニューは自動的に削除されるが、基本的に不要になった時点で明示的に削除することが望ましい。

### 【エラーコード】

EX\_MID : メニュー(mid)は存在していない(汎用メニューでない場合も含む)

# mset\_gmn

汎用メニューの選択動作

## 【形式】

W mset\_gmn(W mid, PNT pos)

## 【パラメータ】

W mid メニューID  
PNT pos 画面上の絶対座標

## 【リターン値】

0 正常(関数値は選択された項目の番号(選択されなかった場合は0))  
< 0 エラー(エラーコード)

## 【解説】

mid で指定した登録済み汎用メニューを pos で指定したスクリーン上の絶対座標位置に汎用メニューの矩形の左上の点を合わせて表示し、PD の移動に従った選択動作を行ない、PD の動作に応じて選択された項目の番号(1 ~ 128) を関数値として戻す。何も選択されなかった場合は、関数値は 0 となる。

pos は、通常メニューボタンのプレスが行なわれた位置を絶対座標で指定する。pos が不正な場合は、適当な位置から動作する。

メニュー動作中に発生した EV\_BUTDWN, EV\_BUTUP, EV\_KEYDWN, EV\_KEYUP, EV\_AUTKEY は、イベントキューから取り除かれて捨てられる。それ以外のイベントはイベントキューに残される。

なお、この関数内ではポインタ形状変更は行なわないため、通常、実行前に「選択指」のポインタ形状にしておく必要がある。イメージ退避領域が確保できない場合は、メニュー動作終了後すべてのウィンドウに再表示要求が送られる。

## 【エラーコード】

EX\_MID : メニュー(mid)は存在していない(汎用メニューでない場合も含む)

# mchg\_gat

汎用メニューの項目属性の変更

## 【形式】

W mchg\_gat(W mid, W num, UW mode)

## 【パラメータ】

W mid       メニューID  
W num       メニュー項目番号  
UW mode     ::= ( M\_STAT)     (( M\_SEL     M\_NOSEL) | ( M\_ACT     M\_INACT) )

M\_STAT : 変更しない。       (現在の属性を取り出す)  
M\_SEL  : 選択状態とする。   (選択フラグをセット)  
M\_NOSEL: 選択状態としない。   (選択フラグをリセット)  
M\_ACT  : 不能項目としない。   (不能項目フラグをリセット)  
M\_INACT: 不能項目とする。   (不能項目フラグをセット)

## 【リターン値】

0     正常(関数値は項目の現在の属性)  
< 0   エラー(エラーコード)

## 【解説】

mid で指定した汎用メニューの num で指定したメニュー項目の不能項目フラグ、選択フラグを変更して、その結果としての現在の属性を関数値として戻す。項目リスト内の属性コード自体は変更されない。

関数値として戻る現在の属性は以下に示す内容となる。この属性は項目リスト内の属性コードも考慮した実際の属性であるため、M\_NOSEL、または M\_ACT を指定した場合でも属性コードの値によっては、変化しない場合もある。

動作項目の場合 (MC\_IND = 0) は、M\_SEL / M\_NOSEL の指定は無視され、関数値の M\_SEL は常に 0 となる。

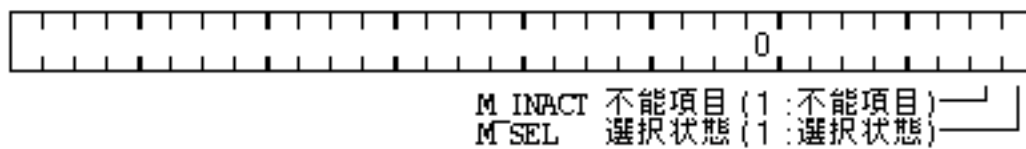


図 78 : mchg\_gat

## 【エラーコード】

EX\_MID     : メニュー(mid)は存在していない(汎用メニューでない場合も含む)。  
EX\_PAR     : パラメータが不正である(numが不正)。

# mchg\_dtm

## 標準メニューのディレイタイムの変更

### 【形式】

W mchg\_dtm(W time)

### 【パラメータ】

W time ミリ秒単位のディレイタイム

### 【リターン値】

0 正常(変更以前のディレイタイム)  
< 0 エラー(エラーコード)

### 【解説】

標準メニュー選択時の操作性を上げるために設定されているメインメニュー内での反応の遅れ(ディレイタイム)を、timeで指定した値に変更し、変更以前の値を関数値として取り出す。time < 0の場合は変更せずに現在値を取り出す。なお、システムスタートアップ時には、ディレイタイムとして適当なデフォルト値が設定されているものとする。

### 【エラーコード】

発生しない

---

[この章の目次にもどる](#)

[前頁:3.1 ウィンドウマネージャにもどる](#)

[次頁:3.3 パーツマネージャにすすむ](#)

[この章の目次にもどる](#)

[前頁:3.2 メニューマネージャにもどる](#)

[次頁:3.4 パネルマネージャにすすむ](#)

---

## 3.3 パーツマネージャ

### 3.3.1 パーツマネージャの機能

#### 3.3.1.1 概要

パーツマネージャは、外殻のHMI機能の1つとして位置付けられ、各種の標準コントロールパーツの表示/操作/登録/削除等の機能を提供している。アプリケーションプログラムは、このパーツマネージャを通して容易に標準コントロールパーツを使用することができる。

パーツマネージャは、各パーツの表示/操作を行なう基本的な関数群を提供しているが、パーツとしての総合的な動作を実現するためには、アプリケーションプログラムがコントロールパーツのルールに従ってパーツマネージャの関数群を使用する必要がある。

本仕様書では、パーツの標準的な形状に関して説明しているが、表示形状の詳細はインプリメントに依存する。

#### 3.3.1.2 パーツの種類

パーツマネージャでは標準的に以下の種類のパーツをサポートしている。

|                  |      |
|------------------|------|
| テキストボックス         | (TB) |
| シークレットテキストボックス   | (XB) |
| 数値ボックス           | (NB) |
| シリアルボックス         | (SB) |
| テキストオルタネートスイッチ   | (AS) |
| テキストモーメンタリスイッチ   | (MS) |
| ピクトグラムオルタネートスイッチ | (PA) |
| ピクトグラムモーメンタリスイッチ | (PM) |
| スイッチセレクト         | (WS) |
| スクロールセレクト        | (SS) |
| ボリューム            | (VL) |

アプリケーションプロセスは、使用するパーツを登録し、登録時に得られるパーツIDを使用して実際の表示/操作等を行なう。

登録の方法としては大きく2種類あり、メモリ上に存在するパーツデータ構造を直接指定して登録する場合と、データマネージャにより管理されているパーツデータ構造を示すデータ番号を指定して登録する場合がある。パーツは必ず1つの独立したウィンドウまたはパネルに属する形で登録され使用される。即ち、パーツの登録の際にそのパーツを使用するウィンドウID/パネルIDを指定することになる。

パーツは基本的に登録したウィンドウ/パネルに属しており、登録したウィンドウ/パネルが削除されると、パーツも自動的に削除される。さらに、登録したプロセスが終了した場合にもパーツの登録は自動的に削除される。

パーツは基本的にそれを登録したプロセスに属しており、登録したプロセスが終了すると自動的に

に登録も削除される。同様に登録したウィンドウ/パネルにも属しており、登録したウィンドウ/パネルが削除されると、パーツも削除される。

パーツは基本的にそのパーツを登録したプロセスにより使用されるが、他のプロセスからでも同一のパーツIDによりそのパーツを使用することが可能である。従って、パーツID自体はプロセスグローバルであり、またパーツの登録データもすべてのプロセスからアクセスできる。ただし、パーツによっては登録データとして文字列等への参照ポインタを持つ場合があり、そのポインタがプロセスローカルメモリ領域を示している場合は、そのパーツは他のプロセスでは、使用できないことになる。ただし、他プロセスからの描画時等においては、プロセス間通信や、セマフォ等により、排他的制御を行なう必要がある。

### 3.3.1.3 パーツのデータ構造

#### パーツの定義構造

パーツはそれぞれのタイプにより異なる定義構造を持ち、また構成要素により可変サイズであるが、これらをすべてまとめた以下の構造により取り扱われる。

```
typedef UNION {
    TEXTBOX  tb;  -- (シークレット)テキストボックス
    NUMBOX   nb;  -- 数値ボックス
    SERBOX   sb;  -- シリアルボックス
    SWSEL    ss;  -- テキストオルターネート / モーメンタリスイッチ、
                  スイッチ / スクロールセレクタ
    PICTSW   pw;  -- ピクトグラムオルターネート / モーメンタリスイッチ
    VOLUME   vl;  -- ボリューム
} PARTS;
```

各タイプに対応した定義構造は、基本的に以下の構造を持っている。

```
struct {
    UW  type;  -- パーツのタイプ / 属性 / 状態
    RECT  r;  -- パーツ表示矩形領域
    . . . . . -- 個々のパーツにより異なる
}
```

type は以下の形式でパーツのタイプ / 属性 / 状態を示している。

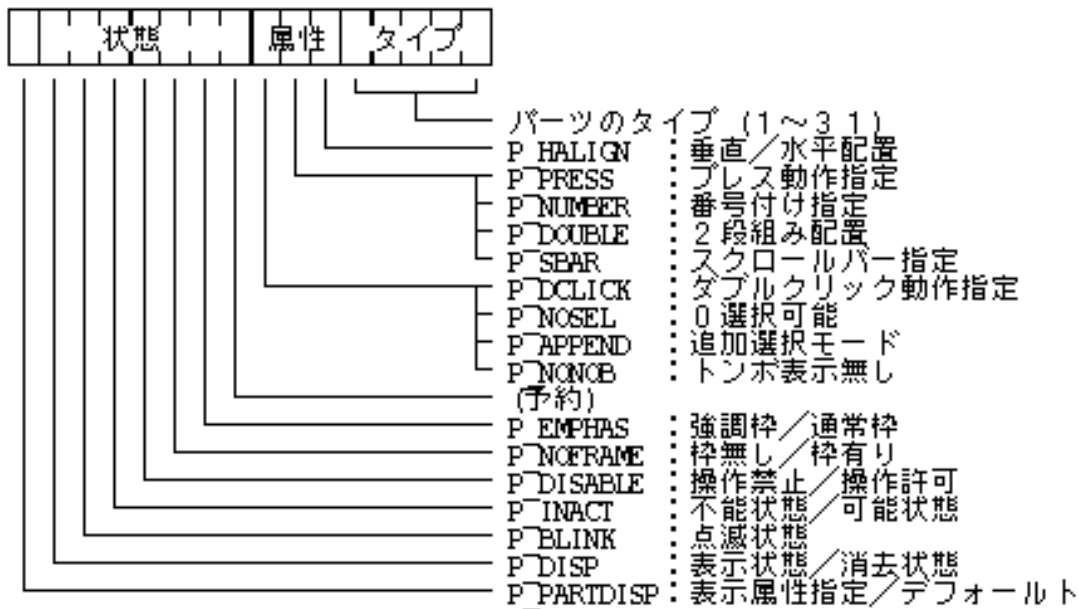


図 79: パーツのタイプ / 属性 / 状態

- タイプとしては以下の値が使用される。

TB\_PARTS 1 -- テキストボックス  
 XB\_PARTS 2 -- シークレットテキストボックス  
 NB\_PARTS 3 -- 数値ボックス  
 SB\_PARTS 4 -- シリアルボックス  
 AS\_PARTS 5 -- テキストオルタネートスイッチ  
 MS\_PARTS 6 -- テキストモーメンタリスイッチ  
 PA\_PARTS 7 -- ピクトグラムオルタネートスイッチ  
 PM\_PARTS 8 -- ピクトグラムモーメンタリスイッチ  
 WS\_PARTS 9 -- スイッチセレクタ  
 SS\_PARTS 10 -- スクロールセレクタ  
 VL\_PARTS 11 -- ボリューム

- 属性はパーツの動作の属性を示すもので登録後の変更はできない。状態はパーツの表示 / 操作に関する状態を示すもので、登録後も変更可能である。

r はパーツの表示矩形領域であり、パーツ全体を囲む最小の矩形であり、属するウィンドウ / パネルの相対座標で指定される。登録後も移動 / 変形が可能である。

パーツの登録時には、以下に示す表示属性を指定することが可能である。この表示属性は登録後は変更することはできない。表示属性を指定せずに登録したパーツに対しては、デフォルトの表示属性が適用される。また、表示属性のビットは定義時のみ有効であり、cget\_sts() では、常に 0 となって得られる。

```
typedef struct {
    W frpat; /* 枠のパターン(0:デフォルト) */
    W bgpat; /* 背景パターン(0:デフォルト) */
    COLOR chcol; /* 文字色 (<0:デフォルト) */
    W misc; /* 各種のパターン / 色 */
} PARTDISP;
```

frpat, bgpat はそれぞれ、枠と背景のパターンを示し、"PAT\_DATA" タイプとしてデータボックス

に登録されているデータのデータ番号で指定される。0はデフォルトを意味する。また、データ番号で指定されたデータがデータボックスにない場合は、デフォルトパターンを用いて描画する。chcolは、文字の色を表わす。これは、パーツの種類によっては使用されない(ただし、後述するメタコードによる色指定の方が優先される)。miscは、パーツの種類によって異なる意味に使用されるデータで、COLORVALとして使用される場合と、2個のデータ番号として使用される場合がある。

### 文字列の属性コード

パーツの定義において、通常、文字列はその属性指定のために先頭文字として特殊な意味を持つ属性コード(メタコード)が置かれる場合がある。この属性コードは文字列等の区切りの意味も持つ。

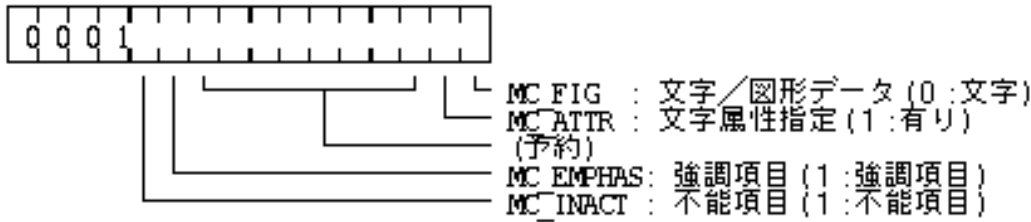


図 80 : 属性コード

#### MC\_FIG:

- 0 -- 文字データ
- 1 -- 図形データ

#### MC\_ATTR:

- 0 -- 標準属性の文字列 ( MC\_FIG = 0 の時のみ有効)
- 1 -- 指定属性の文字列 ( MC\_FIG = 0 の時のみ有効)

文字データの時、MC\_ATTR = 1 の場合は直後に 6 ワードの文字属性指定コード列が存在し、その後項目としての文字列がくる。MC\_ATTR = 0 の場合は文字属性指定コードは存在せず、直後にデータとしての文字列がくる。文字列は属性コードまたは TNULLコード (= 0) で終了し、後者の場合は全体の終了を意味する。文字列としては表示可能な任意の文字が許され、それ以外のものは無視される。

文字属性指定コードは以下に示すものであり、ワードの値が 0 となる場合もあることに注意が必要である。

- COLOR chcol -- 文字色指定 (-1は文字色指定無し)
- W class -- フォントクラス指定 (-1はデフォルトフォント)
- UW attr -- フォント属性指定 (0は通常体)
- PNT csize -- フォントサイズ指定  
 (csize.c.h または csize.c.v が0以下はデフォルトサイズ)

MC\_ATTR = 0 は、文字色指定無し、デフォルトフォント、通常体を意味する。文字色指定無しの場合は、パーツの表示属性として定義された文字色が適用される。

図形データの場合は、図形の矩形表示となり、MC\_ATTRは無視される。この場合、直後に以下の2ワードの図形指定コードがくる。

```
TTTT DDDD DDDD DDDD
0000 0000 LLLL LLLL
```



T: 図形のデータタイプID

0: ポインタイメージ ("PTR\_DATA")

1: ピクトグラムイメージ ("PICT\_DATA")

2: パターンデータ ("PAT\_DATA")

3: ビットマップデータ ("BMAP\_DATA")

4~: 予約

D: 図形のデータ番号 (0の場合、または存在しないデータの場合は何も表示されない)

L: 図形の横幅文字数

図形項目は、Lで指定した文字分の矩形領域に左上詰めで表示される。

MC\_EMPHAS:

0 -- 通常項目

1 -- 強調項目 (項目の表示に下線がつけられる。)

MC\_INACT:

0-- 正常項目

1 -- 不能項目 (灰色化表示され一切選択することはできない)

### 3.3.1.4 パーツの状態

登録されたパーツは、以下の状態を保持している。

- タイプ / 属性:

登録時に指定した定義データそのものであり、登録後は変更できない。

- 表示 / 操作状態:

登録時に指定した定義データを初期状態とするが、登録後も変更可能である。

表示/消去状態 :P\_DISP =1: パーツは表示状態

0 : パーツは消去状態

点滅状態 :P\_BLINK =1: パーツは点滅状態である

0 : パーツは点滅状態でない

(P\_DISP=0の場合は常に0となる)

不能 / 可能状態 :P\_INACT =1: パーツは不能状態

0 : パーツは可能状態

禁止 / 許可状態 :P\_DISABLE =1: パーツは操作禁止(表示のみ)

0 : パーツの操作許可

枠の有無状態 :P\_NOFRAME =1: パーツの矩形枠なし

0 : パーツの矩形枠あり

枠の強調状態 :P\_EMPHAS =1: 強調枠(3ドット程度)

0 :: 通常枠(1ドット程度)

不能 / 可能状態は、パーツが登録されているウィンドウ / パネルの状態変化により自動的に変更されるもので、ウィンドウ / パネルが入力受付状態に移行した場合に、P\_INACT = 0 に設定され、入力不可状態に移行した場合に、P\_INACT = 1 に設定される。この設定はウィンドウマネージャ / パネルマネージャによりそのウィンドウ / パネルに登録されているすべてのパーツに対して一括して行なわれる (但し、P\_SBAR = 1 のボリュームは対象外となる)。

- 登録されたウィンドウ / パネルID:

パーツの登録時に指定されたそのパーツの属するウィンドウ/パネルID。パーツの描画環境は属するウィンドウ/パネルの描画環境に等しく、座標系、クリッピング領域、背景等の各種パターン等は属するウィンドウ/パネルの描画環境が適用される。このIDは、指定したウィンドウ/パネルに属するすべてのパーツに対するグループ処理にも使用される。ただし、P\_SBAR = 1 のボリュームは特別に扱われ、登録したウィンドウ/パネルの作業領域(フレーム長方形のクリップ領域)をはみ出して描かれることになる。

また、後述する cidl\_par() を除いては、ウィンドウ/パネルに対するグループ処理の対象外とされる。

- 表示矩形領域：

パーツが現在表示されている矩形領域であり登録時に指定するが、登録後の変更(移動、大きさ変更)も可能である。属するウィンドウ/パネルの相対座標により示される。

- 入力状態：

各種ボックスパーツ群では、キー入力の受け付けが可能か否かの状態が定義される。ボックス以外のパーツではこの状態は定義されない。「入力状態」では、ボックス内の選択領域(部分文字列)が「ちらつき枠」で囲まれて表示され、ヌル選択の場合は選択ギャップが点滅して表示される。さらにカレットも表示され、キー入力を受け付け可能であることを示す。「非入力状態」では、「ちらつき枠」やカレットは表示されず、キー入力を受け付け不可能であることを示す。「入力状態」のパーツはシステムに0または1つであり、あるボックスが「入力状態」になった場合、いままで「入力状態」であったボックスは自動的に「非入力状態」となる。

- 現在値：

各パーツは、パーツとしての現在値を保持しており、その初期値は登録時に指定した値となり、パーツの操作により変更される。値の内容/意味はパーツのタイプに依存する。

- 選択領域：

各種ボックスパーツ群では、選択領域の情報を保持している。ボックス以外のパーツでは適用されない。登録時にはテキストボックスではデフォルト選択(P\_APPEND)に従って選択領域が設定され、シークレットテキストボックスでは常に全体が選択領域となり、シリアルボックスでは先頭フィールドが選択領域となる。

- 表示パラメータ：

各パーツは、現在値の他に登録時に指定した表示用の文字列等の各種のパラメータを保持している。このパラメータは登録後も変更可能であり、パラメータの内容/意味はパーツのタイプに依存する。

## 3.3.2 パーツの種類

### 3.3.2.1 テキストボックス

テキストボックスは1行の文字列を表示し、文字列の入力/編集を行なうパーツである。編集はボックス内部で、通常のリプレースエディタとして行なうことができ、部分文字列の選択、ドラッグ&リリースによる移動/複写、トレイ経由のカット&貼り込み操作も可能である。日本語の入力に関しては仮名漢字変換が行なわれる。

表示

- 矩形枠の中に左詰めで文字列が表示され、「入力状態」、かつ「可能状態」の場合、内部にカレットと選択領域(「ちらつき枠」)または点滅する文字カーソルが存在する。矩形枠より長い文字列の場合はキー入力や、選択動作により自動的に水平スクロールが行なわれ

る。

- 「禁止状態」の場合は矩形枠ではなく下線が引かれる。また「入力状態」には成り得ない。

操作許可文字列

操作禁止文字列

図 81 : テキストボックス

#### 状態 / 属性

|            |               |                                  |
|------------|---------------|----------------------------------|
| P_DISP     | : 0 (消去状態)    | : パーツは表示されていない                   |
|            | : 1 (表示状態)    | : パーツは表示されている                    |
| P_BLINK    | : 適用されない      |                                  |
| P_INACT    | : 0 (可能状態)    | : 入力状態の時、選択領域、カレットの表示有り          |
|            | : 1 (不能状態)    | : 選択領域、カレットの表示無し                 |
| P_DISABLE  | : 0 (許可状態)    | : 矩形枠の表示 ( P_NOFRAME = 0 の時のみ有効) |
|            | : 1 (禁止状態)    | : 下線の表示 ( P_NOFRAME = 0 の時のみ有効)  |
| P_NOFRAME  | : 0 (枠有り)     | : 矩形枠/下線の表示有り                    |
|            | : 1 (枠無し)     | : 矩形枠/下線の表示無し                    |
| P_EMPHAS   | : 0 (通常状態)    | : 通常枠/下線 ( P_NOFRAME = 0 の時のみ有効) |
|            | : 1 (強調状態)    | : 強調枠/下線 ( P_NOFRAME = 0 の時のみ有効) |
| P_APPEND   | : 0 (置換)      | : デフォルト選択時、文字列全体が選択される           |
|            | : 1 (追加)      | : デフォルト選択時、文字列の最後がヌル選択される        |
| P_PARTDISP | : 0 (デフォルト表示) | : デフォルトの表示属性で表示                  |
|            | : 1 (表示属性指定)  | : 指定された表示属性で表示                   |

#### 動作

- キー入力により、リプレースエディタとしての動作を行ない、入力文字がボックス内に表示される。ボックスの幅を超えた場合は、表示は自動的に左スクロールして、入力は可能となる。
- 指定した最大文字数を超えた場合は、入力は捨てられ、ビープが鳴る。最大文字数はボックスの幅とは無関係である。
- ボックス内のプレスにより、リリース位置までを選択状態とすることができる。プレスしたまま、ボックスの両端を超えた場合は、その方向にある見えない部分が見えるように、全体が横スクロールする。ドラッグしている間は、そこでリリースした場合に選択される部分文字列の枠の影がドラッグに追従する。
- ボックス内の任意の位置での、ダブルクリックにより、文字列全体を選択状態とすることができる。
- 選択領域上にポインタを持ってくるとポインタは、「移動手」となり、プレスすることにより、「握り」となり、選択された部分文字列を、ドラッグして移動/複写できる。リリース場所が、ボックス内部の場合は、移動/複写処理が行なわれ、入力処理は続行するが、リリース場所が、ボックス外の場合は、入力処理は終了する。リリースした時点で、ポインタは元の形状に戻る。

- [タブ], [入力終], [改段落 / 改行] キー、ボックス外での PD のプレス、ドラッグ後のボックス外での PD のリリース、およびメニュー指定により入力処理は終了する。但し、[入力終] キーは、仮名漢字変換の未確定文字列が存在する場合は、その確定を行なうため、未確定文字列が存在していない場合にのみ入力処理の終了となる。[タブ][改段落 / 改行] キーは未確定文字列の確定とともに入力処理を終了する。

一般にパネル上で使用される場合は、以下のようにする。

[タブ]            -- 次のボックスに移動  
 [入力終]        -- 入力の終了(評価)、ボックスの再選択  
 [改段落 / 改行] -- パネルを抜ける

- [タブ], [入力終], [改段落 / 改行] キーにより終了した場合は、非入力状態となり、選択領域とカレットの表示は消え、選択領域はデフォルト選択 (P\_APPEND) に従って変更される。その他の場合で終了した場合は、入力状態のままとなり、選択領域とカレットの表示は終了時の状態のままとなる。
- [命令] + [タブ], [入力終], [改段落 / 改行] キーにより、それぞれを文字として文字列中に入力することができる。
- 入力処理終了時には仮名漢字変換の未確定文字列は強制的に確定状態とされる。

#### 定義構造

```
typedef struct {
    UW  type;      -- タイプ(TB_PARTS) / 属性 / 状態
    RECT r;       -- パーツの矩形領域
    W   txsize;   -- 最大文字数
    TC* text;     -- 初期表示データ文字列へのポインタ
    PARTDISP atr; -- 表示属性(type の MSB が1の時のみ)
} TEXTBOX;
```

- text は初期の表示文字列へのポインタであり、編集結果が格納される領域ではない。text = NULL の場合は、何も表示されない。また、先頭の属性コードは、MC\_STR, MC\_ATTR のみ有効で、省略しても構わない。従って、text の先頭が MC\_ATTR で始まらない場合は標準フォントとして扱われる。また、文字属性の指定は登録時のみ有効で登録後の変更は一切受け付けない。
- txsize は、入力可能な最大の文字数である (TNULL を含む)。内部で使用されるバッファのサイズを意味する。指定できる最大値はインプリメント依存(80文字以上)とする。
- パーツとして登録した後は、text のメモリ内容は参照されないため、変更してもそれは反映されない。
- 登録したパーツは常にグローバルに使用可能である。

#### 表示属性

frpat : 枠、または下線のパターン (P\_NOFRAME = 0 の時のみ有効)  
 bgpat : 矩形領域の背景パターン  
 chcol : 文字の色  
 misc : (COLORVAL) カレット、および「ちらつき枠」の色

### 3.3.2.2 シークレットテキストボックス

シークレットテキストボックスは、基本的にテキストボックスと同様の機能を持つが、隠し名や合言葉等の入力に使用される、外から「読みにくい」ようにした特殊なテキストボックスである。シークレットテキストボックスでは、文字列の表示は荒い斜線パターンでORされ、読みにくくなっており、また編集は、「一字消」のみであり、部分文字列の選択、ドラッグ & リリースによる移動/複製、トレイ経由のカット & 貼り込み操作等はできない。日本語の入力に関しては仮名入力のみである。

#### 表示

- 初期文字列が存在する場合は、矩形枠の中全体が荒い斜線パターンで埋められ実際に文字列は表示されない。初期文字列が存在しない場合は、矩形枠の中は空白となる。
- 「入力状態」で、かつ「可能状態」の場合、初期文字列が存在する場合は、斜線パターンが「ちらつき枠」で囲まれ、その直後にカレットが表示される。初期文字列がない場合は、先頭に点滅する文字カーソルが表示される。
- 「禁止状態」の場合は矩形枠ではなく下線が引かれる。また「入力状態」には成り得ない。



図 82 : シークレットテキストボックス

#### 状態 / 属性

|            |               |                                  |
|------------|---------------|----------------------------------|
| P_DISP     | : 0 (消去状態)    | : パーツは表示されていない                   |
|            | : 1 (表示状態)    | : パーツは表示されている                    |
| P_BLINK    | : 適用されない      |                                  |
| P_INACT    | : 0 (可能状態)    | : 入力状態の時、選択領域、カレットの表示有り          |
|            | : 1 (不能状態)    | : 選択領域、カレットの表示無し                 |
| P_DISABLE  | : 0 (許可状態)    | : 矩形枠の表示 ( P_NOFRAME = 0 の時のみ有効) |
|            | : 1 (禁止状態)    | : 下線の表示 ( P_NOFRAME = 0 の時のみ有効)  |
| P_NOFRAME  | : 0 (枠有り)     | : 矩形枠/下線の表示有り                    |
|            | : 1 (枠無し)     | : 矩形枠/下線の表示無し                    |
| P_EMPHAS   | : 0 (通常状態)    | : 通常枠/下線 ( P_NOFRAME = 0 の時のみ有効) |
|            | : 1 (強調状態)    | : 強調枠/下線 ( P_NOFRAME = 0 の時のみ有効) |
| P_APPEND   | : 0 (置換)      | : デフォルト選択時、文字列全体が選択される           |
|            | : 1 (追加)      | : デフォルト選択時、文字列の最後がヌル選択される        |
| P_PARTDISP | : 0 (デフォルト表示) | : デフォルトの表示属性で表示                  |
|            | : 1 (表示属性指定)  | : 指定された表示属性で表示                   |

#### 動作

- キー入力により、リプレースエディタとしての動作を行ない、斜線パターンでORされた

入力文字がボックス内に表示される。ボックスの幅を超えた場合は、表示は自動的に左スクロールして、入力は可能となる。

- 指定した最大文字数を超えた場合は、入力は捨てられ、ビープが鳴る。最大文字数はボックスの幅とは無関係である。
- ボックス内のシングルクリックにより、文字列全体を選択状態とすることができる。
- [タブ], [入力終], [改段落 / 改行] キー ボックス外での PD のプレス、およびメニュー指定により入力処理は終了する。入力処理が終了した場合、最終結果として文字列が存在する場合は、文字の表示が消え、ボックスの内部全体が斜線で覆われる。文字列が空となった場合は空白となる。

一般にパネル上で使用される場合は、以下のようにする。

[タブ] -- 次のボックスに移動

[入力終] -- 入力の終了(評価)、ボックスの再選択

[改段落 / 改行] -- パネルを抜ける

- [タブ], [入力終], [改段落 / 改行] キー 終了した場合は、非入力状態となり、選択領域とカレットの表示は消えるが、その他の場合で終了した場合は、入力状態のままとなり、選択領域とカレットの表示は終了時の状態のままとなる。

#### 定義構造

```
typedef struct {
    UW type;          -- タイプ(XB_PARTS) / 属性 / 状態
    RECT r;          -- パーツの矩形領域
    W txsize;        -- 最大文字数
    TC* text;        -- 初期表示データ文字列へのポインタ
    PARTDISP atr;    -- 表示属性(type の MSB が1の時のみ)
} TEXTBOX;
```

- textは初期の表示文字列へのポインタであり、編集結果が格納される領域ではない。text = NULL の場合は何も表示されない。また、先頭の属性コードは、MC\_STR, MC\_ATTR のみ有効で省略しても構わない。従って、text の先頭が MC\_ATTR で始まらない場合は標準フォントとして扱われる。また、文字属性の指定は登録時のみ有効で、登録後の変更は一切受け付けない。
- txsize は、入力可能な最大の文字数である ( TNULL を含む)。内部で使用されるバッファのサイズを意味する。指定できる最大値はインプリメント依存 ( 80 文字以上 ) とする。
- 登録したパーツは常にグローバルに使用可能である。

#### 表示属性

frpat : 枠、または下線のパターン ( P\_NOFRAME = 0 の時のみ有効 )

bgpat : 矩形領域の背景パターン

chcol : 文字の色、斜線のパターンの色

misc : ( COLORVAL ) カレット、および「ちらつき枠」の色

### 3.3.2.3 数値ボックス

数値ボックスは、1つの数値を指定した形式で表示し、数値の入力 / 編集を行なうパーツであ

る。編集はボックス内部で、通常のリプレースエディタとして行なうことができ、部分数値列の選択、ドラッグ & リリース による移動 / 複写、トレイ経由のカット & 貼り込み操作も可能である。但し、入力は数値に制限され、仮名漢字変換は行なわれない。

## 表示

- 矩形枠の中に数値が右詰めで表示され、「入力状態」、かつ「可能状態」の場合、内部に caret と選択領域（「ちらつき枠」）または点滅する文字カーソルが存在する。矩形枠より長い文字列の場合はキー入力や、選択動作により自動的に水平スクロールが行なわれる。
- 「禁止状態」の場合は矩形枠ではなく下線が引かれる。また「入力状態」には成り得ない。



図 83 : 数値ボックス

## 状態 / 属性

|            |               |                                    |
|------------|---------------|------------------------------------|
| P_DISP     | : 0 (消去状態)    | : パーツは表示されていない                     |
|            | : 1 (表示状態)    | : パーツは表示されている                      |
| P_BLINK    | : 適用されない      |                                    |
| P_INACT    | : 0 (可能状態)    | : 入力状態の時、選択領域、caret の表示あり          |
|            | : 1 (不能状態)    | : 選択領域、caret の表示なし                 |
| P_DISABLE  | : 0 (許可状態)    | : 矩形枠の表示 ( P_NOFRAME = 0 の時のみ有効)   |
|            | : 1 (禁止状態)    | : 下線の表示 ( P_NOFRAME = 0 の時のみ有効)    |
| P_NOFRAME  | : 0 (枠有り)     | : 矩形枠 / 下線の表示あり                    |
|            | : 1 (枠無し)     | : 矩形枠 / 下線の表示なし                    |
| P_EMPHAS   | : 0 (通常状態)    | : 通常枠 / 下線 ( P_NOFRAME = 0 の時のみ有効) |
|            | : 1 (強調状態)    | : 強調枠 / 下線 ( P_NOFRAME = 0 の時のみ有効) |
| P_APPEND   | : 0 (置換)      | : デフォルト選択時、文字列全体が選択される             |
|            | : 1 (追加)      | : デフォルト選択時、文字列の最後がヌル選択される          |
| P_PARTDISP | : 0 (デフォルト表示) | : デフォルトの表示属性で表示                    |
|            | : 1 (表示属性指定)  | : 指定された表示属性で表示                     |

## 動作

- 数値キー入力により、リプレースエディタとしての動作を行ない、入力数値がボックス内に表示される。ボックスの幅を超えた場合は、表示は自動的に左スクロールして、入力は可能となる。キー入力は、符号、数値、小数点(実数の場合)のみ可能で、かつその結果としての文字列が指定したタイプの数値としての正しい形式をとる場合のみ可能で、そうでないものはビープが鳴り、入力されない。
- ボックス内のプレスにより、リリース位置までを選択状態とすることができる。プレスしたまま、ボックスの両端を超えた場合は、その方向にある見えない部分が見えるように、全体が横スクロールする。ドラッグしている間は、そこでリリースした場合に選択される部分数値列の枠の影がドラッグに追従する(ただし、ドラッグ中の横スクロールは禁止される)。

- ボックス内の任意の位置での、ダブルクリックにより、数値列全体を選択状態とすることができる。
- 選択領域上にポインタを持ってくるとポインタは、「移動手」となり、プレスすることにより、「握り」となり、選択された部分数値列を、ドラッグして移動/複写できる。リリース場所が、ボックス内部の場合は、移動/複写処理が行なわれ、入力処理は続行するが、リリース場所が、ボックス外の場合は、入力処理は終了する。リリースした時点で、ポインタは元の形状に戻る。
- [タブ], [入力終], [改段落 / 改行] キー、ボックス外でのPDのプレス、ドラッグ後のボックス外でのPDのリリース、およびメニュー指定により入力処理は終了する。

一般にパネル上で使用される場合は、以下のようにする。

[タブ]            -- 次のボックスに移動  
 [入力終]         -- 入力の終了(評価)、ボックスの再選択  
 [改段落 / 改行] -- パネルを抜ける

- [タブ], [入力終], [改段落 / 改行] キーにより終了した場合は、非入力状態となり、選択領域とカレットの表示は消え、選択領域はデフォルト選択 (P\_APPEND) に従って変更される。その他の場合で終了した場合は、入力状態のままとなり、選択領域とカレットの表示は終了時の状態のままとなる。

#### 定義構造

```
typedef struct {
    UW type;            -- タイプ(NB_PARTS) / 属性 / 状態
    RECT r;            -- パーツの矩形領域
    UW fmt;            -- 数値フォーマット指定
    union {            -- 現在値
        W i;            -- 整数の場合
        DOUBLE d;       -- 実数の場合
    } cv;
    PARTDISP atr;      -- 表示属性(type の MSB が1の時のみ)
} NUMBOX;
```

- fmt は数値の形式を規定するもので、以下の内容を持つ。数値表示がボックスに入りきらない場合は、数値文字列の先頭(左側)のみが表示される。

fmt = SSSS SSSS TPxF DDDD

T:

数値データタイプ

0:

整数 (32ビット整数型)

1:

実数 (64ビット浮動小数点型)

D:

小数点以下の表示桁数 (0 ~ 15)

F:

実数の表示方法



0:

Dで指定された小数点以下の桁数を常に表示する。  
また、D=0の場合は小数点のみを表示する。

1:

Dで指定された小数点以下の桁数を最大として、後ろの0を表示しない。  
また、D=0の場合は小数点は表示しない。

S:

縦横のフォントサイズ

P:

0:

全角指定

1:

半角指定

x:

予約 (0)

数値の表示は、以下のように行なわれる。

T = 0 ( 整数 ) の場合 :

Dに無関係に整数として表示する。

T = 1 ( 実数 ) の場合 :

D、Fによりに以下のように表示する。

|     | D=0     | D=2         | D=4             |
|-----|---------|-------------|-----------------|
| F=0 | 1 2 3 . | 1 2 3 . 5 0 | 1 2 3 . 5 0 0 0 |
| =1  | 1 2 3   | 1 2 3 . 5   | 1 2 3 . 5       |

図 84 : 数値の形式

- cv は現在値であり、L\_UNDEF ( 負の最大値 ) または D\_UNDEF ( NaN, または ± ) の場合は表示は空白となる。
- 登録したパーツは常にグローバルに使用可能である。

表示属性

frpat : 枠、または下線のパターン ( P\_NOFRAME = 0 の時のみ有効 )

bgpat : 矩形領域の背景パターン

chcol : 文字の色

misc : ( COLORVAL ) カレット、および「ちらつき枠」の色

### 3.3.2.4 シリアルボックス

シリアルボックスは、複数のフィールドから構成される特定の形式を持ったパラメータを表示し変更を可能とするものである。各フィールドに入る内容は、ある連続範囲内の整数値、もしくは順序付けされた文字列の集合であり、各フィールド間は任意の固定文字列により分離される。

整数値フィールドの値の変更は1つつの値の増加 / 減少、またはキーによるの直接入力により行なわれ、文字列フィールドの変更は、順序付けされた文字列の正方向 / 逆方向への切換えにより

行なわれる。

なお、フィールドの一部分の選択、ドラッグ & リリース による移動 / 複写、トレイ経由のカット & 貼り込み操作等の編集機能は持たない。

#### 表示

- 矩形枠の内部には適当な固定文字列により分割された複数のフィールドの連鎖が存在する (一つでもよい)。
- 「入力状態」かつ「可能状態」では、選択されたフィールド全体が反転表示され、内容を変更できることをあらわす。さらに、全体を囲む矩形の後端に " " と " " の記号が縦に並んだ矩形 (操作ボックス) が現れる。ただし、記号の表示方法はインプリメントに依存し、選択されたフィールドの直下に " " と " " の記号が並んで現れるというようなバリエーションも考えられる。ただし横表示の場合、" " (減少方向) は左、" " (増加方向) は右に規定される。
- 「禁止状態」の場合は矩形枠ではなく下線が引かれる。また「入力状態」には成り得ない。

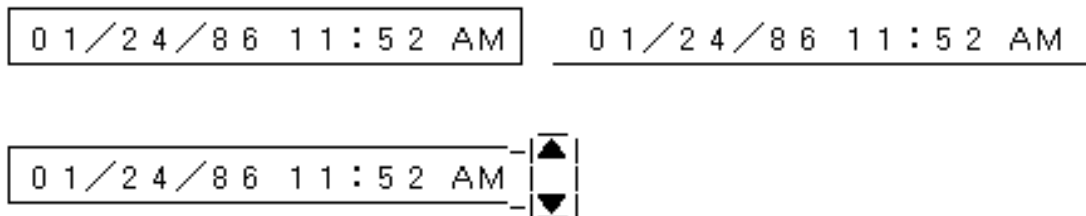


図 85 : シリアルボックス

#### 状態 / 属性

|            |              |                                   |
|------------|--------------|-----------------------------------|
| P_DISP     | :0 (消去状態)    | :パーツは表示されていない                     |
|            | :1 (表示状態)    | :パーツは表示されている                      |
| P_BLINK    | :適用されない      |                                   |
| P_INACT    | :0 (可能状態)    | :入力状態の時、選択領域表示あり                  |
|            | :1 (不能状態)    | :選択領域の表示なし                        |
| P_DISABLE  | :0 (許可状態)    | :矩形枠の表示 ( P_NOFRAME = 0 の時のみ有効)   |
|            | :1 (禁止状態)    | :下線の表示 ( P_NOFRAME = 0 の時のみ有効)    |
| P_NOFRAME  | :0 (枠有り)     | :矩形枠 / 下線の表示あり                    |
|            | :1 (枠無し)     | :矩形枠 / 下線の表示なし                    |
| P_EMPHAS   | :0 (通常状態)    | :通常枠 / 下線 ( P_NOFRAME = 0 の時のみ有効) |
|            | :1 (強調状態)    | :強調枠 / 下線 ( P_NOFRAME = 0 の時のみ有効) |
| P_PARTDISP | :0 (デフォルト表示) | :デフォルトの表示属性で表示                    |
|            | :1 (表示属性指定)  | :指定された表示属性で表示                     |

#### 動作

- ボックス内のプレスにより、その位置に最も近いフィールドが選択され、反転表示となる。選択は1つのフィールド全体であり、フィールド内の部分選択はできない。選択状態では、後端に操作ボックスが表示される。なお操作ボックス下のイメージは保存される。
- PDで後端の操作ボックスをプレスすると " " または " " の記号を囲む矩形内 (即ち操作

ボックスの上または下半分) が反転表示し、選択されたフィールドの現在の値が正方向または逆方向に1つつ変化する。プレスし続ける事により適当な間隔で連続的に変化していき、リリースした時点の値が新たな値となる。

プレス中に操作ボックスを離れると、操作ボックスの反転表示は解除され、値の変化も停止する。そのままリリースするとその時点の値が新しい値となる。操作ボックスに戻ると値の変化は継続する。プレスしたまま操作ボックス内の逆の記号側に移動すると、反転表示は切り換わり変化の方向も逆転する。

- 最大値の次の正方向は最小値、最小値の次の逆方向は最大値と見なす。また、フィールドが空白(未定義)の場合、現在値は最大値と最小値の間と考え、最初の正方向の値は最小値、逆方向の値は最大値となる。
- [変換]と[逆変換]キーは、後端の操作ボックスをPD プレスすることと同様の機能を持ち、押し続けることで連続的に値が変化する。[変換]が" "に対応し、[逆変換]が" "に対応する。
- 整数値フィールドが選択されている場合は、選択後の最初の数値キー入力により、反転表示されている現在の内容がリプレースされ、その後順次右詰めで入力が入っていく。この場合、フィールドは常に反転表示されたままである。数値キー入力はその結果として指定範囲内の数値になるもののみ可能で、そうでないものはビープが鳴り、入力を受け付けない。入力中の文字の変更は[一字消]によるのみである。
- 選択されているフィールドが文字列の場合は、キー入力は無視される。
- [タブ]キーにより次のフィールドが選択されるが、最後のフィールドの場合は入力処理が終了する。
- [入力終],[改段落/改行]キー、ボックス外でのPDのプレス、およびメニュー指定により入力処理は終了する。

一般にパネル上で使用される場合は、以下のようにする。

[タブ] -- 次のボックスに移動  
[入力終] -- 入力の終了(評価)、ボックスの再選択  
[改段落/改行] -- パネルを抜ける

- [タブ],[入力終],[改段落/改行]キーにより終了した場合は、非入力状態となり、選択領域の表示は消え、選択フィールドは先頭フィールドに変更される。その他の場合で終了した場合は、入力状態のままとなり、選択領域の表示は終了時の状態のままとなる。

## 定義構造

```
typedef struct {
    UW type;          -- タイプ(SB_PARTS) / 属性 / 状態
    RECT r;          -- パーツの矩形領域
    TC* fmt;         -- 表示フォーマット指定
    W nfld;          -- データフィールド数
    W *cv;           -- 初期フィールド値配列へのポインタ
    PARTDISP atr;   -- 表示属性(type の MSB が1の時のみ)
} SERBOX;
```

- fmt は表示フォーマットを指定するワードデータ列であり、先頭に属性コード、次に以下に示す要素を並べ、最後に終端コード(1ワードの0)を置いたものである。ただし、先頭の属性コードは、MC\_STR, MC\_ATTR のみ有効で省略しても構わない。従って、fmt の先頭が

MC\_ATTR で始まらない場合は、標準フォントとして扱われる。  
また、先頭の属性コードは整数フィールド、固定文字列に有効で項目フィールドでは、項目リスト中の属性コードを優先する。

- <整数フィールド指定> -- 指定した範囲の整数値の選択表示(可変)
- <項目フィールド指定> -- 指定した項目リストの選択表示(可変)
- <固定文字列> -- 固定文字列表示

<固定文字列> は、固定的に表示される文字列であり、偶数バイトでなくてはならないため、必要によっては、最後に1バイトの0をパッドしたものである。

<整数フィールド指定> は以下の3ワード固定のデータとなる。

```
0000 11ZA SSSS SSSS -- 数値フィールド指定
NNNN NNNN NNNN NNNN -- 最小値
NNNN NNNN NNNN NNNN -- 最大値
```

S:  
    フィールドの文字数

Z:  
    0= 前ゼロサプレス  
    1= 前ゼロ表示

A:  
    0= 右詰め  
    1= 左詰め

<項目フィールド指定> は以下の形式の可変長のデータとなる。候補項目リストは候補となる表示項目を項目データの形式で並べたものであり終端コード(1ワードの0)で終了する。なお、fmt の最後に<項目フィールド指定>がある場合は、終端コードが2つ連続することになる。

```
0000 1000 SSSS SSSS -- 項目フィールド指定
          :      :      -- 候補項目リスト
          :      :
0000 0000 0000 0000 -- 終端コード
```

S:  
    フィールドの文字数

(例) 表示例は以下のように定義される([n] は16進数 n をコード化したものを表わしている)。

```
[1000][e02][1][c]/[e02][1][15]/[e02][50][63] [e02][0][b]:
[e02][0][3b] [801][1000]A[1000]P[0]M[0]
```

- nFld はフィールドの数で、fmt 中含まれるフィールド指定の数と一致していなくてはならない。  
フィールドは最大16個までとする。
- cv は各フィールドの初期値配列へのポインタであり、フィールドが文字列の場合は何番目かを示している(先頭は0)。cv[i] が L\_UNDEF (負の最大値) の場合、i + 1 番目のフィールドの表示は空白となる。この値は、初期値としてのみ使用され、値の変更は反映されない。
- fmt で示されるメモリ内容は、パーツとして登録した後も参照されるので領域は保持しておく必要がある。従って、fmt がローカルメモリ領域にある場合は、登録したパーツはグ

ローバルに使用できない。

表示属性

frpat : 枠、または下線のパターン ( P\_NOFRAME = 0 の時のみ有効 )

bgpat : 矩形領域の背景パターン

chcol : 文字の色

misc : ( COLORVAL ) 操作ボックスの " " と " " の記号の色

操作ボックスの枠は frpat、背景は、bgpat を適用するものとする。

### 3.3.2.5 テキストオルタネートスイッチ

テキストオルタネートスイッチは、文字列で表記されたスイッチ名と、ON/OFF 状態を示す矩形インジケータから構成されるスイッチであり、ON/OFF 状態の表示/切り換えに使用する。

表示

- 角を丸めた矩形枠の中にインジケータとスイッチ名が左詰めで表示される。インジケータは "=" が OFF 状態、" ≠ " が ON 状態を表わす。また、表示属性の指定がない場合、ON 状態のインジケータをデフォルト色で強調するインプリメントも考えられる。
- 「禁止状態」または「不能状態」では、スイッチ名、およびインジケータは灰色化表示される。
- 「点滅状態」では、インジケータ部分のみが点滅表示する。

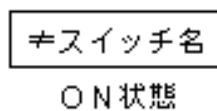
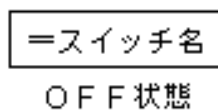


図 86 : テキストオルタネートスイッチ

状態 / 属性

|            |               |                                |
|------------|---------------|--------------------------------|
| P_DISP     | : 0 (消去状態)    | : パーツは表示されていない                 |
|            | : 1 (表示状態)    | : パーツは表示されている                  |
| P_BLINK    | : 0 (通常状態)    | : 正常表示                         |
|            | : 1 (点滅状態)    | : インジケータ部分が点滅表示                |
| P_INACT    | : 0 (可能状態)    | : 正常表示                         |
|            | : 1 (不能状態)    | : 灰色化表示                        |
| P_DISABLE  | : 0 (許可状態)    | : 正常表示                         |
|            | : 1 (禁止状態)    | : 灰色化表示                        |
| P_NOFRAME  | : 0 (枠有り)     | : 矩形枠の表示あり                     |
|            | : 1 (枠無し)     | : 矩形枠の表示なし                     |
| P_EMPHAS   | : 0 (通常状態)    | : 通常枠 ( P_NOFRAME = 0 の時のみ有効 ) |
|            | : 1 (強調状態)    | : 強調枠 ( P_NOFRAME = 0 の時のみ有効 ) |
| P_PARTDISP | : 0 (デフォルト表示) | : デフォルトの表示属性で表示                |
|            | : 1 (表示属性指定)  | : 指定された表示属性で表示                 |

## 動作

- 矩形枠内でプレスすると、インジケータをマージン1ドットで囲む矩形領域が反転表示される。そのままリリースするとスイッチの状態は逆転し (ON → OFF / OFF → ON)、インジケータの反転表示は解除されて、逆転した状態の表示となる。
- プレスしたままドラッグしてスイッチの枠外へ出ると反転表示は解除され、そのまま枠外でリリースした場合は何の変化も起らない。最初にプレスした枠内に戻ると反転表示は復活し、リリースすることによりスイッチの状態が逆転し、インジケータの表示も逆転する。

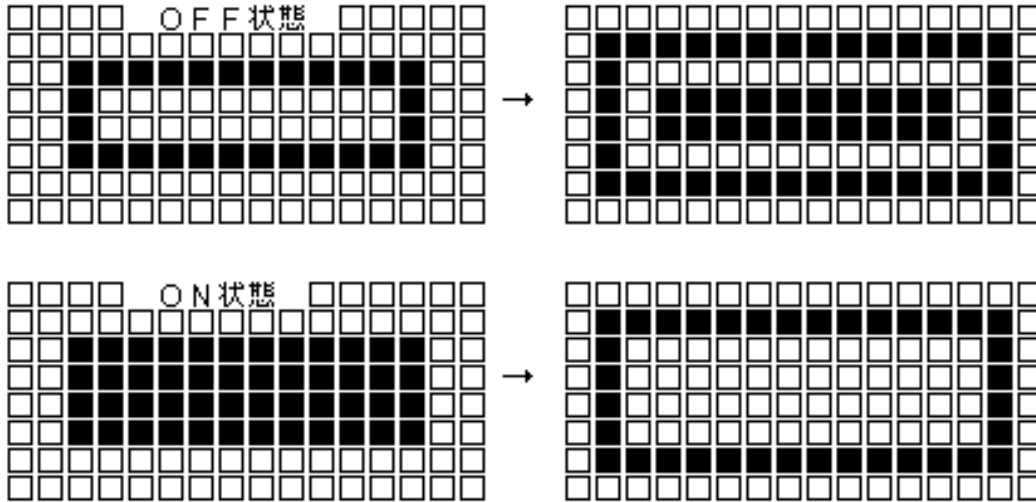


図 87 : インジケータの反転表示

## 定義構造

```
typedef struct {
    UW type;          -- タイプ(AS_PARTS) / 属性 / 状態
    RECT r;          -- パーツの矩形領域
    W cv;            -- 現在値 (0:OFF / 1:ON)
    TC* name;        -- スイッチ名へのポインタ
    PARTDISP atr;    -- 表示属性(type の MSB が1の時のみ)
} SWSEL;
```

- name はスイッチ名文字列へのポインタであり、先頭文字は属性コードとして特殊な意味を持つ(MC\_EMPHAS, MC\_INACT は無視される)。
- name で示されるメモリ内容は、パーツとして登録した後も参照されるので領域は保持しておく必要がある。従って、name がローカルメモリ領域にある場合は、登録したパーツはグローバルに使用できない。

## 表示属性

frpat : 枠、または下線のパターン ( P\_NOFRAME = 0 の時のみ有効 )  
bgpat : 矩形領域の背景パターン  
chcol : 文字の色  
: 下位ワード -- インジケータ描画用のデータ番号  
misc ("PAT\_DATA" タイプで矩形の大きさは、 [図 87 : インジケータの反転表示](#) 固定)  
上位ワード -- 未使用

### 3.3.2.6 テキストモーメンタリスイッチ

テキストモーメンタリスイッチは、文字列で表記されたスイッチ名のみで構成されるスイッチであり、ON/OFF状態を保持する機能はなく、何らかの動作の起動、または押している間だけ何らかの動作をさせる場合に使用される。

このスイッチに対する操作としては、プレス、リリース、ダブルクリックがある。

表示

- 角を丸めた矩形枠の中にスイッチ名が中央揃えで書かれている。特に頻度の多いデフォルトと考えられるスイッチは太枠で強調表示される (但し矩形枠内にスイッチ名が入りきらない場合は左詰めに表示する)。
- 「禁止状態」または「不能状態」では、スイッチ名は灰色化表示される。
- 「点滅状態」では、スイッチ名のみが点滅表示する。



図 88 :テキストモーメンタリスイッチ

状態 / 属性

|            |               |                               |
|------------|---------------|-------------------------------|
| P_DISP     | : 0 (消去状態)    | : パーツは表示されていない                |
|            | : 1 (表示状態)    | : パーツは表示されている                 |
| P_BLINK    | : 0 (通常状態)    | : 正常表示                        |
|            | : 1 (点滅状態)    | : スイッチ名部分が点滅表示                |
| P_INACT    | : 0 (可能状態)    | : 正常表示                        |
|            | : 1 (不能状態)    | : 灰色化表示                       |
| P_DISABLE  | : 0 (許可状態)    | : 正常表示                        |
|            | : 1 (禁止状態)    | : 灰色化表示                       |
| P_NOFRAME  | : 0 (枠有り)     | : 矩形枠の表示あり                    |
|            | : 1 (枠無し)     | : 矩形枠の表示なし                    |
| P_EMPHAS   | : 0 (通常状態)    | : 通常枠 ( P_NOFRAME = 0 の時のみ有効) |
|            | : 1 (強調状態)    | : 強調枠 ( P_NOFRAME = 0 の時のみ有効) |
| P_DCLICK   | : 0           | : ダブルクリックは検出しない               |
|            | : 1           | : ダブルクリックを検出する                |
| P_PRESS    | : 0           | : プレス中の動作を行なわない               |
|            | : 1           | : プレス中の動作を行なう                 |
| P_PARTDISP | : 0 (デフォルト表示) | : デフォルトの表示属性で表示               |
|            | : 1 (表示属性指定)  | : 指定された表示属性で表示                |

動作

- 矩形枠内でプレスすると、枠の内部が反転表示される。ここでリリースすると反転表示は解除され、スイッチをONにしたことになる。
- プレスしたままドラッグして枠外へ出ると反転表示が解除され、そのまま枠外でリリース

した場合は何も起らない。最初にプレスした枠内に戻ると反転表示は復活し、リリースすることによりスイッチを ON したことになる。

- P\_DCLICK = 1 の場合は、ダブルクリック動作を検出し、シングルクリックかダブルクリックかの情報をアプリケーションに戻す。0 の場合は、ダブルクリック動作は検出されず、最初のリリース状態が即時にアプリケーションに戻される。
- P\_PRESS = 1 の場合は、矩形枠内でプレス中 (反転表示されている状態) の状態が検出され、アプリケーションに戻される。0 の場合は、アプリケーションには戻されない。

#### 定義構造

テキストオルタネートスイッチと同じ構造となる。

```
typedef struct {
    UW      type;      -- タイプ(MS_PARTS) / 属性 / 状態
    RECT    r;        -- パーツの矩形領域
    W       cv;        -- 現在値 (未使用)
    TC*     name;      -- スイッチ名へのポインタ
    PARTDISP atr;     -- 表示属性(type の MSB が1の時のみ)
} SWSEL;
```

- cv は意味を持たない (現在値を取り出すと常に 0 となる)。
- name はスイッチ名文字列へのポインタであり、先頭文字は属性コードとして特殊な意味を持つ (MC\_EMPHAS, MC\_INACT は無視される)。
- name で示されるメモリ内容は、パーツとして登録した後も参照されるので領域は保持しておく必要がある。従って、name がローカルメモリ領域にある場合は、登録したパーツはグローバルに使用できない。

#### 表示属性

frpat : 枠、または下線のパターン (P\_NOFRAME = 0 の時のみ有効)

bgpat : 矩形領域の背景パターン

chcol : 文字の色

misc : 未使用

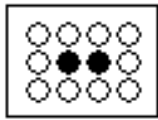
#### 3.3.2.7 ピクトグラムオルタネートスイッチ

ピクトグラムオルタネートスイッチはテキストオルタネートスイッチと同様の機能を持つが、スイッチ名 / インジケータの代わりに ON / OFF 状態を示す 2 つの任意の図形 (ビットマップデータ) が表示される。

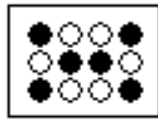
#### 表示

- 角を丸めた矩形枠の中に ON / OFF 状態のそれぞれに対応した指定された図形が中央揃えで書かれる。また、拡大 / 縮小は行なわず、指定された図形が矩形より大きい場合はクリッピングされる。
- 「禁止状態」または「不能状態」では、図形は灰色化表示される。
- 「点滅状態」では、図形のみが点滅表示する。





OFF状態



ON状態

図 89 : ピクトグラムオルタネートスイッチ

## 状態 / 属性

|            |               |                               |
|------------|---------------|-------------------------------|
| P_DISP     | : 0 (消去状態)    | : パーツは表示されていない                |
|            | : 1 (表示状態)    | : パーツは表示されている                 |
| P_BLINK    | : 0 (通常状態)    | : 正常表示                        |
|            | : 1 (点滅状態)    | : 図形部分が点滅表示                   |
| P_INACT    | : 0 (可能状態)    | : 正常表示                        |
|            | : 1 (不能状態)    | : 灰色化表示                       |
| P_DISABLE  | : 0 (許可状態)    | : 正常表示                        |
|            | : 1 (禁止状態)    | : 灰色化表示                       |
| P_NOFRAME  | : 0 (枠有り)     | : 矩形枠の表示あり                    |
|            | : 1 (枠無し)     | : 矩形枠の表示なし                    |
| P_EMPHAS   | : 0 (通常状態)    | : 通常枠 ( P_NOFRAME = 0 の時のみ有効) |
|            | : 1 (強調状態)    | : 強調枠 ( P_NOFRAME = 0 の時のみ有効) |
| P_PARTDISP | : 0 (デフォルト表示) | : デフォルトの表示属性で表示               |
|            | : 1 (表示属性指定)  | : 指定された表示属性で表示                |

## 動作

- 矩形枠内でプレスすると、図形パターンが反転表示される。ここでリリースするとスイッチの状態は逆転し ( ON OFF / OFF ON )、図形の反転表示は解除されて、逆転した状態の図形の表示となる。
- プレスしたままドラッグしてスイッチの枠外へ出ると反転表示は解除され、そのまま枠外でリリースした場合は何の変化も起らない。最初にプレスした枠内に戻ると反転表示は復活し、リリースすることによりスイッチの状態が逆転し、図形の表示も逆転する。

## 定義構造

```
typedef struct {
    UW      type;      -- タイプ(PA_PARTS) / 属性 / 状態
    RECT    r;        -- パーツの矩形領域
    W       cv;        -- 現在値 (0:OFF / 1:ON)
    W       onpat;     -- ON状態の「BMAP_DATA」タイプの番号
    W       offpat;    -- OFF状態の「BMAP_DATA」タイプの番号
    PARTDISP atr;     -- 表示属性(type の MSB が1の時のみ)
} PICTSW;
```

- onpat, offpat はそれぞれ、データマネージャが管理するビットマップデータのデータ番号で、データタイプは「BMAP\_DATA」タイプに固定される。offpat = 0 の場合は、offpat は onpat の反転であると見なされる。また onpat = 0 の場合は何も表示されない(空白となる)。

- 登録したパーツはグローバルに使用可能である。

表示属性

frpat : 枠、または下線のパターン ( P\_NOFRAME = 0 の時のみ有効 )

bgpat : 矩形領域の背景パターン

chcol : 未使用

misc : 未使用

### 3.3.2.8 ピクトグラムモーメンタリスイッチ

ピクトグラムモーメンタリスイッチはテキストモーメンタリスイッチと同様の機能を持つが、スイッチ名の代わりに任意の図形(ビットマップデータ)が表示される。

表示

- 角を丸めた矩形枠の中に指定された図形が中央揃えで書かれる。また、ピクトグラムオルタネートスイッチ同様、拡大 / 縮小は行なわない。
- 「禁止状態」または「不能状態」では、図形は灰色化表示される。
- 「点滅状態」では、図形のみが点滅表示する。



図 90 : ピクトグラムモーメンタリスイッチ

状態 / 属性

|            |               |                               |
|------------|---------------|-------------------------------|
| P_DISP     | : 0 (消去状態)    | : パーツは表示されていない                |
|            | : 1 (表示状態)    | : パーツは表示されている                 |
| P_BLINK    | : 0 (通常状態)    | : 正常表示                        |
|            | : 1 (点滅状態)    | : 図形部分が点滅表示                   |
| P_INACT    | : 0 (可能状態)    | : 正常表示                        |
|            | : 1 (不能状態)    | : 灰色化表示                       |
| P_DISABLE  | : 0 (許可状態)    | : 正常表示                        |
|            | : 1 (禁止状態)    | : 灰色化表示                       |
| P_NOFRAME  | : 0 (枠有り)     | : 矩形枠の表示あり                    |
|            | : 1 (枠無し)     | : 矩形枠の表示なし                    |
| P_EMPHAS   | : 0 (通常状態)    | : 通常枠 ( P_NOFRAME = 0 の時のみ有効) |
|            | : 1 (強調状態)    | : 強調枠 ( P_NOFRAME = 0 の時のみ有効) |
| P_DCLICK   | : 0           | : ダブルクリックは検出しない               |
|            | : 1           | : ダブルクリックを検出する                |
| P_PRESS    | : 0           | : プレス中の動作を行なわない               |
|            | : 1           | : プレス中の動作を行なう                 |
| P_PARTDISP | : 0 (デフォルト表示) | : デフォルトの表示属性で表示               |
|            | : 1 (表示属性指定)  | : 指定された表示属性で表示                |

## 動作

- 矩形枠内でプレスすると、図形は反転表示される。ここでリリースすると反転表示は解除され、スイッチを ON にしたことになる。
- プレスしたままドラッグして枠外へ出ると反転表示が解除され、そのまま枠外でリリースした場合は何も起らない。最初にプレスした枠内に戻ると反転表示は復活し、リリースすることによりスイッチを ON したことになる。
- P\_DCLICK = 1 の場合は、ダブルクリック動作を検出し、シングルクリックかダブルクリックかの情報をアプリケーションに戻す。0 の場合は、ダブルクリック動作は検出されず、最初のリリース状態が即時にアプリケーションに戻される。
- P\_PRESS = 1 の場合は、矩形枠内でプレス中 (反転表示されている状態) の状態が検出され、アプリケーションに戻される。0 の場合は、アプリケーションには戻されない。

## 定義構造

ピクトグラムオルタネートスイッチと同じ構造であるが、offpat は使用しない。

```
typedef struct {
    UW      type;    -- タイプ(PM_PARTS) / 属性 / 状態
    RECT    r;      -- パーツの矩形領域
    W       cv;     -- 現在値 (未使用)
    W       onpat;  -- 「BMAP_DATA」タイプのデータ番号
    W       offpat; -- 未使用
    PARTDISP atr;   -- 表示属性(type の MSB が1の時のみ)
} PICTSW;
```

- onpat はデータマネージャが管理するビットマップデータのデータ番号で、データタイプは「BMAP\_DATA」タイプに固定される。また onpat = 0 の場合は何も表示されない(空白となる)。
- cv は意味を持たない(現在値を取り出すと常に0となる)。
- 登録したパーツはグローバルに使用可能である。

## 表示属性

frpat : 枠、または下線のパターン (P\_NOFRAME = 0 の時のみ有効)

bgpat : 矩形領域の背景パターン

chcol : 未使用

misc : 未使用

### 3.3.2.9 スイッチセレクト

スイッチセレクトは複数の選択肢のうち、1 個ないしは 0 個の選択肢を選択するために使用され、すべての選択肢の名称と、その状態を示すインジケータから構成される。

## 表示

- すべての選択肢を角の丸い矩形で囲んで表示される。選択肢の前にはインジケータがあり、"=" で OFF、" " で ON を表わす。(インジケータに関しては、[テキストオルタネートスイッチ](#)を参照)

- 状態の変更が禁止されている、「不能項目」の選択肢は、選択肢名と、インジケータが灰色化表示される。
- 「禁止状態」または「不能状態」の場合は、全体が灰色化表示される。
- 配置は、縦、横または2段組のいずれかである。

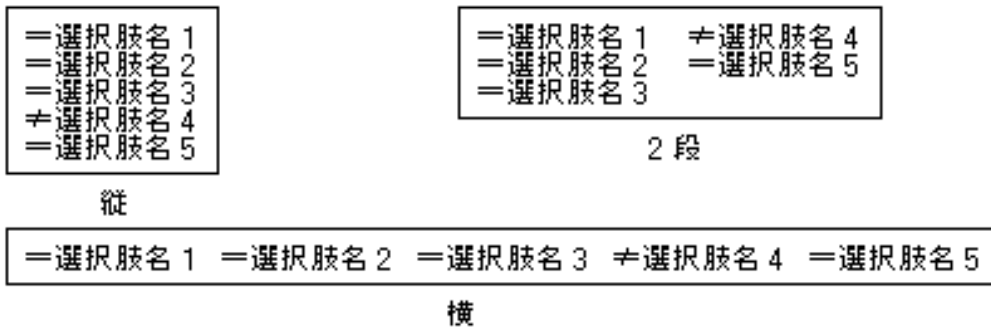


図 91 : スイッチセレクタ

状態 / 属性

|            |               |                               |
|------------|---------------|-------------------------------|
| P_DISP     | : 0 (消去状態)    | : パーツは表示されていない                |
|            | : 1 (表示状態)    | : パーツは表示されている                 |
| P_BLINK    | 適用されない        |                               |
| P_INACT    | : 0 (可能状態)    | : 正常表示                        |
|            | : 1 (不能状態)    | : 全体が灰色化表示                    |
| P_DISABLE  | : 0 (許可状態)    | : 正常表示                        |
|            | : 1 (禁止状態)    | : 全体が灰色化表示                    |
| P_NOFRAME  | : 0 (枠有り)     | : 矩形枠の表示あり                    |
|            | : 1 (枠無し)     | : 矩形枠の表示なし                    |
| P_EMPHAS   | : 0 (通常状態)    | : 通常枠 ( P_NOFRAME = 0 の時のみ有効) |
|            | : 1 (強調状態)    | : 強調枠 ( P_NOFRAME = 0 の時のみ有効) |
| P_NOSEL    | : 0           | : 必ず1つは選択される                  |
|            | : 1           | : 1つも選択されなくてもよい               |
| P_DOUBLE   | : 0           | : 縦または横一列                     |
|            | : 1           | : 2 段組み                       |
| P_PARTDISP | : 0           | : 縦一列 ( P_DOUBLE = 0 の時のみ有効)  |
|            | : 1           | : 横一列 ( P_DOUBLE = 0 の時のみ有効)  |
| P_PARTDISP | : 0 (デフォルト表示) | : デフォルトの表示属性で表示               |
|            | : 1 (表示属性指定)  | : 指定された表示属性で表示                |

動作

- OFF 状態の選択肢を囲む仮想的な矩形を考え、この矩形の中でプレスした場合、インジケータをマージン1ドットで囲む矩形領域が反転表示される。ここで、リリースすると、インジケータの表示が " " になり状態が ON に変わったことを示す。同時に他の ON 状態であった選択肢のインジケータの状態は OFF となり " = " の表示に変化する。(インジケータの反転表示に関しては、[テキストオルタネートスイッチ](#)を参照)
- ドラッグして最初にプレスした選択肢を囲む仮想的な矩形外へ出ると、インジケータの反

転表示は解除され、そのまま外でリリースした場合は何の変化も起らない。

- 最初にプレスした選択肢内に戻った場合は反転表示は復活し、リリースすることによってその選択肢の状態を ON にできる。
- ON 状態の選択肢を囲む矩形中でプレスした場合も、同様にインジケータは反転表示となり、P\_NOSEL = 0 の時は、その矩形領域内でリリースしても状態は ON のままで変化しないが、P\_NOSEL = 1 の時は、状態は OFF に変化する。
- 不能項目の選択肢を囲む矩形中でプレスした場合は、何も起らない。

#### 定義構造

```
typedef struct {
    UW      type;  -- タイプ(WS_PARTS) / 属性 / 状態
    RECT    r;    -- パーツの矩形領域
    W       cv;   -- 現在値 (選択番号 0 ~)
    TC*     name; -- 選択肢名列へのポインタ
    PARTDISP atr; -- 表示属性(type の MSB が1の時のみ)
} SWSEL;
```

- cv は現在選択されている選択肢の番号 (1 ~ ) を示す。何も選択されていない場合は 0 となる。
- name は各選択肢の文字列を属性コードで区切って繋げたものである。属性コードが区切りと同時に次の文字列の属性を指定している。選択肢は最大 32 個までとする。

<属性コード><第1選択肢名><属性コード ><第2選択肢名> . . . . <TNULL>

<属性コード>が MC\_INACT の場合は、対応する選択肢は不能項目であり、灰色化表示される。

- name で示されるメモリ内容は、パーツとして登録した後も参照されるので領域は保持しておく必要がある。従って、name がローカルメモリ領域にある場合は、登録したパーツはグローバルに使用できない。

#### 表示属性

frpat : 枠、または下線のパターン (P\_NOFRAME = 0 の時のみ有効)  
bgpat : 矩形領域の背景パターン  
chcol : 文字の色  
      : 下位ワード -- インジケータのデータ番号  
misc ("PAT\_DATA" タイプで矩形の大きさは、[図 87: インジケータの反転表示固定](#))  
      上位ワード -- 未使用

#### 3.3.2.10 スクロールセレクト

スクロールセレクトはスイッチセレクトと同様に複数の選択肢のうち、1 個ないしは 0 個の選択肢を選択するために使用されるが、選択肢の一部のみ表示され全体はスクロールすることにより見ることができる。

これは、選択肢が多く全てを表示すると場所を取りすぎる場合や、アプリケーションの実行中に選択肢を動的に変化させる場合に使用する。

スクロールセレクトは、キーボードからの入力によっても操作できる。

表示

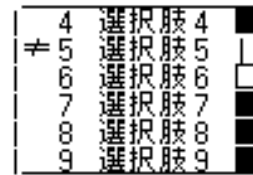
- 右辺にスクロールバーを持った矩形のなかに選択肢が表示される。現在選択されている選択肢にはインジケータ (  ) が付けられている。選択肢の文字列の幅が表示領域より大きくて全体が表示できない場合は最後に「...」を付加し、後半が省略されていることが明白に示される。
- 番号付けが指定されている場合には、選択肢名の直前に1～の連続番号が付けられて表示される。
- スクロールバーは、中央にトンボがあるノブ ( 白 ) 部分と灰色パターン部により表示される。
- 状態の変更が禁止されている、「不能項目」の選択肢は、選択肢名が灰色化表示される。
- 「禁止状態」または「不能状態」の場合は、選択肢全体が灰色化表示される。また、「不能状態」の場合は、スクロールバーのトンボは表示されず、灰色パターンの部分は、白部分との境界線を残して白くなる(不能表示)。
- 配置は、縦のみである。



可能状態



不能状態



番号付け指定の場合



選択肢表示に必要な表示領域が  
パーツ矩形より小さい場合

図 92 : スクロールセレクト

状態 / 属性

|           |            |                               |
|-----------|------------|-------------------------------|
| P_DISP    | : 0 (消去状態) | : パーツは表示されていない                |
|           | : 1 (表示状態) | : パーツは表示されている                 |
| P_BLINK   | 適用されない     |                               |
| P_INACT   | : 0 (可能状態) | : 正常表示                        |
|           | : 1 (不能状態) | : 選択肢全体が灰色化表示、スクロールバーが不能表示    |
| P_DISABLE | : 0 (許可状態) | : 正常表示                        |
|           | : 1 (禁止状態) | : 選択肢全体が灰色化表示                 |
| P_NOFRAME | : 0 (枠有り)  | : 矩形枠の表示あり                    |
|           | : 1 (枠無し)  | : 矩形枠の表示なし                    |
| P_EMPHAS  | : 0 (通常状態) | : 通常枠 ( P_NOFRAME = 0 の時のみ有効) |
|           | : 1 (強調状態) | : 強調枠 ( P_NOFRAME = 0 の時のみ有効) |
| P_NOSEL   | : 0        | : 必ず1つは選択される                  |

:1 :1つも選択されなくてもよい  
P\_NUMBER :0 :番号付けは行なわない  
:1 :番号付けを行なう  
P\_PARTDISP:0 (デフォルト表示):デフォルトの表示属性で表示  
:1 (表示属性指定) :指定された表示属性で表示

## 動作

- インジケータの付かない状態の選択肢を囲む仮想的な矩形を考え、この中でプレスすると、矩形内部が反転表示される。リリースすると、選択肢の先頭にインジケータが現れ選択されたことを示す。同時に今まで選択されていた選択肢のインジケータが消える（見えていない位置にあった場合はその変化は見えない）。
- スクロールエリア内に見えていない選択肢を扱う場合はスクロールバー（ボリュームと同様の操作、ジャンプ移動、スムーズ移動、エリア移動）により選択肢をスクロールさせる。また、スクロールバーに伴う処理はパーツマネージャ側が行ない、エリア移動の場合、表示されている選択肢数 - 1 個ずつスクロールする。
- インジケータの付いた選択肢を囲む矩形中でプレスした場合も、反転表示されるが、リリースした場合、選択状態のまま変化しないか、選択状態でなくなるかは P\_NOSEL に依存する。
- ドラッグにより最初にプレスした選択肢外へ出て他の選択肢の位置にいった場合、それにつれて反転表示も移動する。そのままスクロールエリアの左右の境界を超えずに上下の境界だけを超えて外に出た場合、ドラッグしている間はその方向と逆方向へのスクロールが起り、その方向にある見えていない選択肢が見えるようになる（同時にスクロールバーの表示も変化する）。スクロールエリアのその方向の端にある選択肢が反転状態になっている。選択肢の列が跡切れた段階でスクロールは停止し、末端の選択肢が反転表示され、リリースすると、末端の選択肢の選択状態が反転する。スクロールエリアの左右の境界を超えた場合反転表示が消え、そのままリリースした場合は何の変化も起らない。
- 選択肢の数が少なく必要な行数がスクロールエリアの縦より小さい場合、スクロールバー内部に灰色の部分は見えなくなり、最後の選択肢とスクロールエリアの下辺との間に選択肢のない隙間ができる。この部分でプレスした場合は何もおこらない。
- スクロールセレクトタによっては、[変換]、[逆変換]キーによって操作を行なうことが可能な場合もあり、その場合の操作法は基本的にアプリケーションに依存するが、標準的には以下の操作となる。
  - [変換]、[逆変換]キーをプレスすると、現在インジケータが付いている選択肢から下（[変換]）または上（[逆変換]）にある選択肢が選択され、インジケータが移動する。キーのプレスを続けると、連続的に選択した選択肢が移動し、スクロールも行なわれる。
  - [シフト]キーと同時に[変換]、[逆変換]キーをプレスした場合も、同様に選択肢の選択が移動していくが、1 個ずつではなく適当な個数（通常は表示されている選択肢の数 - 1）ずつ移動する。
  - P\_NUMBER = 1 の場合は、選択肢の直前に 1 ~ の連続番号が表示され、数字をキーボードから入力し [変換] キーを押すことで、指定した番号の選択肢を直接選択することができる。

## 定義構造

スイッチセレクトタと同じ構造である。



```
typedef struct {
    UW      type;    -- タイプ(SS_PARTS) / 属性 / 状態
    RECT    r;      -- パーツの矩形領域
    W       cv;     -- 現在値 (選択番号 0 ~)
    TC*     name;   -- 選択肢名列へのポインタ
    PARTDISP atr;   -- 表示属性(type の MSB が1の時のみ)
} SWSEL;
```

- cv は現在選択されている選択肢の番号 ( 0 ~ ) を示す。何も選択されていない場合は0となる。
- name は各選択肢の文字列を属性コードで区切って繋げたものである。属性コードが区切りと同時に次の文字列の属性を指定している。

<属性コード><第1選択肢名><属性コード ><第2 選択肢名> . . . . <TNULL>

<属性コード>が MC\_INACT の場合は、対応する選択肢は不能項目であり、灰色化表示される。

- name で示されるメモリ内容は、パーツとして登録した後も参照されるので領域は保持しておく必要がある。従って、name がローカルメモリ領域にある場合は、登録したパーツはグローバルに使用できない。

#### 表示属性

frpat : 枠、または下線のパターン ( P\_NOFRAME = 0 の時のみ有効 )

bgpap : 矩形領域の背景パターン

chcol : 文字の色

: 下位ワード -- インジケータのデータ番号

misc ("PAT\_DATA" タイプで矩形の大きさは、[図 87: インジケータの反転表示固定](#))

上位ワード -- スクロールバーのパターン部のデータ番号 (" PAT\_DATA"タイプ)

スクロールバーのトンボは frpat、ノブ部分は bgpat が適用されるものとする。

#### 3.3.2.11 ボリューム

ボリュームは数値パラメータをアナログ的に表示し、同時にパラメータを設定する手段を与えるものである。

これは、通常はスクロールバーとして表示領域の制御に使用するが、ユーザからは操作できないようにして達成度の表示等にも使用される。

#### 表示

- 細長い帯状の矩形エリアで、通常 ( デフォルト表示 ) では白の部分とパターンに埋められた部分に分かれている。白の部分は全体との相対位置で設定されている範囲を示しノブと呼ぶ。ノブの中心にはバーの方向に垂直な線 ( トンボ ) がある ( デフォルト色によってトンボを強調するインプリメントも考えられる )。
- 見えている部分がごくわずかの場合は、ノブは以下ようになり、これ以上圧縮されることはない。



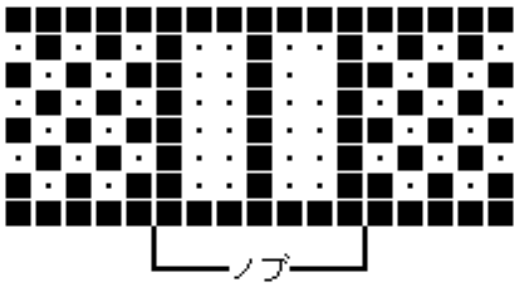


図 93: ノブの最小幅

- 基本的には水平のボリュームでは基準レベルは左端とし、右方向を増加方向とする。垂直のボリュームでは基準レベルは下端とし、上方向を増加方向とする。
- ボリュームの目盛り等の外側の表示はパーツとしてはサポートしていない。
- 「不能状態」の場合は、パターンはノブとの境界線を残して白くなり、トンボも表示されない(不能表示)。
- 「禁止状態」のボリュームでは「不能状態」に無関係に表示は常に正常となる。この場合は、通常トンボは表示しないようにする。

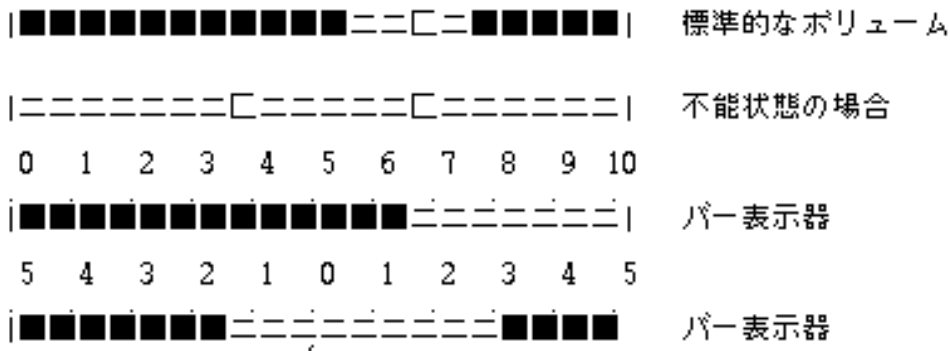


図 94: ボリューム

状態 / 属性

|           |            |                              |
|-----------|------------|------------------------------|
| P_DISP    | : 0 (消去状態) | : パーツは表示されていない               |
|           | : 1 (表示状態) | : パーツは表示されている                |
| P_BLINK   | 適用されない     |                              |
| P_INACT   | : 0 (可能状態) | : 正常表示                       |
|           | : 1 (不能状態) | : 許可状態の場合、不能表示となる            |
| P_DISABLE | : 0 (許可状態) | : 正常表示                       |
|           | : 1 (禁止状態) | : 正常表示 (通常、P_NONOB = 1 とする)  |
| P_NOFRAME | : 0 (枠有り)  | : 矩形枠の表示あり                   |
|           | : 1 (枠無し)  | : 矩形枠の表示なし                   |
| P_EMPHAS  | : 0 (通常状態) | : 通常枠 (P_NOFRAME = 0 の時のみ有効) |
|           | : 1 (強調状態) | : 強調枠 (P_NOFRAME = 0 の時のみ有効) |
| P_NONOB   | : 0        | : トンボを表示する                   |
|           | : 1        | : トンボを表示しない                  |
| P_SBAR    | : 0        | : 常のボリュームとして登録する             |
|           | : 1        | : ウィンドウのスクロールバーとして登録する       |

P\_HALIGN : 0 : (縦) 垂直ボリューム  
: 1 : (横) 水平ボリューム  
P\_PARTDISP : 0 (デフォルト表示) : デフォルトの表示属性で表示  
: 1 (表示属性指定) : 指定された表示属性で表示

## 動作

### ● 【ジャンプ移動】

可能状態のボリュームのノブの部分にポインタを持ってくると、ポインタは縦または横の「移動手」となり、そこでプレスすると「握り」となりドラッグが可能になる。不能状態の場合は、プレスした時点で可能状態に変化し、ポインタは、一瞬「移動手」となり、その後「握り」となりドラッグが可能になる。

ドラッグに従い影 ( ノブと同じ大きさの XOR で描かれた矩形 ) が現れ移動するので、移動比率に見合った望みの位置に持っていくリリースする。この時点で、影の位置にノブが移動し、それに従って数値パラメータの値が変化する(ポインタは「移動手」に戻る)。

ドラッグの途中でポインタをボリュームの外に出した場合は、ボリューム中心から 12 ~ 16 ドット程度離れるまでは影のドラッグは継続し、それ以上離れると影は消えてしまいリリースしても何も起らない。ボリューム中心よりに戻すと、再び影が現れ動作が継続される。影が消えた場合は、ポインタは「移動手」となる。

### ● 【スムーズ移動】

可能状態のボリュームのパターンの部分をプレスすると、パターンの部分がプレスされている間、その方向へノブを近づけるように数値パラメータが徐々に変化する。この場合の変化の最小単位はアプリケーションに依存する。ノブが移動してその中心がプレスした位置に来た時点、またはパターン部分が無くなった時点で変化が停止する。また、ドラッグするとその時点の位置により上記動作が連続して行なわれる。

プレスした位置と、トンボの距離に比例した速度で変化するようにすることが望ましい。不能状態の場合は、プレスした時点で可能状態となり同様の処理が行なわれる。

### ● 【エリア移動】

可能状態のボリュームのパターンの部分をダブルクリックすると、その方向へ ( ノブの幅 - 最小移動幅 ) 分だけノブを近づけるように数値パラメータが変化する。最小移動幅は、アプリケーションに依存する。

不能状態の場合は、プレスした時点で可能状態となり同様の処理が行なわれる。

- 全ての変更は、ノブが全体の幅からはみ出さない範囲で行なわれ、ノブの端が全体の端にたどり着いた時点でその方向への設定の変更は停止する。

## 定義構造

```
typedef struct {  
    UW      type;      -- タイプ(VL_PARTS) / 属性 / 状態  
    RECT    r;         -- パーツの矩形領域  
    W       clo,chi;   -- ノブの最小値、最大値  
    W       lo,hi;     -- 全体の最小値、最大値  
    W       patnum;    -- パターン指定のデータ番号  
    PARTDISP atr;     -- 表示属性(type の MSB が1の時のみ)  
} VOLUME;
```

- lo, hi はそれぞれ右端 ( 上端 ) と左端 ( 下端 ) の値を示し、 clo, chi はノブの右端 ( 上端 ) と左端 ( 下端 ) の値を示す。 大小関係は特に意味はない。

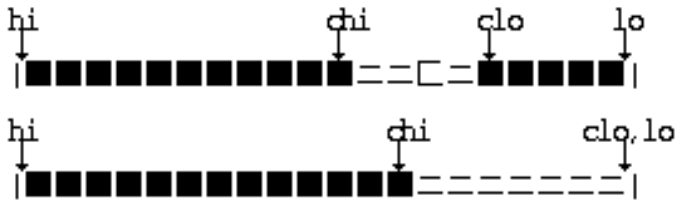


図 95 : ボリュームの値

- patnum は、「PAT\_DATA」タイプのデータ番号を指定する。 0 の場合は標準の灰色パターンとなる。
- 登録したパーツはグローバルに使用可能である。

表示属性

frpat : 枠、または下線のパターン ( P\_NOFRAME = 0 の時のみ有効 )

bgpat : スクロールバーのノブ ( 白 ) 部分のパターン

chcol : 未使用

misc : 未使用

### 3.3.3 データ / 定数の定義

パーツID

```
typedef W PAID; /* パーツID */
```

パーツの表示属性

```
typedef struct {
    W frpat; /* 枠のパターン(0:デフォルト) */
    W bgpat; /* 背景パターン(0:デフォルト) */
    COLOR chcol; /* 文字色 (<0:デフォルト) */
    W misc; /* 各種のパターン/色 */
} PARTDISP;
```

テキストボックス

```
typedef struct textbox { /* TB_PARTS, XB_PARTS */
    UW type;
    RECT r;
    W txsize;
    TC *text;
    PARTDISP atr;
} TEXTBOX;
```

数値ボックス

```

typedef struct numbox {          /* NB_PARTS */
    UW      type;
    RECT    r;
    UW      fmt;                /* 数値フォーマット */
    union {
        W    l;
        double d;
    } cv;
    PARTDISP atr;
} NUMBOX;

```

## シリアルボックス

```

typedef struct serbox {         /* SB_PARTS */
    UW      type;
    RECT    r;
    TC      *fmt;                /* 表示フォーマット指定 */
    W       nfld;                /* データフィールド数 */
    W       *cv;                 /* 初期フィールド値配列 */
    PARTDISP atr;
} SERBOX;

```

## セレクタ

```

typedef struct swsel {         /* AS/MS_PARTS, WS_PARTS, SS_PARTS */
    UW      type;
    RECT    r;
    W       cv;                  /* 1:on, 0:off / 現在値 (選択番号 0..) */
    TC      *name;              /* スイッチ名/選択肢名列へのポインタ */
    PARTDISP atr;
} SWSEL;

```

## ピクトグラムスイッチ

```

typedef struct pictsw {        /* PA_PARTS, PM_PARTS */
    UW      type;
    RECT    r;
    W       cv;                  /* 現在値 0:off, 1:on / 未使用 */
    W       onpat;               /* on状態の BMAP_DATA タイプの番号 */
    W       offpat;              /* off状態の BMAP_DATA タイプの番号 / 未使用 */
    PARTDISP atr;
} PICTSW;

```

## ボリューム

```

typedef struct volume {       /* VL_PARTS */
    UW      type;
    RECT    r;

```

```

W      clo, chi;    /* ノブの最小値、最大値 */
W      lo, hi;     /* 全体の最小値、最大値 */
W      patnum;     /* パターン指定のデータ番号 */
PARTDISP atr;
} VOLUME;

```

## パーツ

```

typedef union parts {
    TEXTBOX tb;    /* TB_PARTS, XB_PARTS */
    NUMBOX nb;    /* NB_PARTS */
    SERBOX sb;    /* SB_PARTS */
    SWSEL ss;    /* AS_PARTS, MS_PARTS, WS_PARTS, SS_PARTS */
    PICTSW pw;    /* PA_PARTS, PM_PARTS */
    VOLUME vl;    /* VL_PARTS */
} PARTS;

```

## パーツタイプ

```

#define TB_PARTS      1    /* テキストボックス */
#define XB_PARTS      2    /* シークレットテキストボックス */
#define NB_PARTS      3    /* 数値ボックス */
#define SB_PARTS      4    /* シリアルボックス */
#define AS_PARTS      5    /* テキストオルタネートスイッチ */
#define MS_PARTS      6    /* テキストモーメンタリスイッチ */
#define PA_PARTS      7    /* ピクトグラムオルタネートスイッチ */
#define PM_PARTS      8    /* ピクトグラムモーメンタリスイッチ */
#define WS_PARTS      9    /* スイッチセレクタ */
#define SS_PARTS     10    /* スクロールセレクタ */
#define VL_PARTS     11    /* ボリューム */

#define P_TYPE        0x001f /* タイプマスク */

#define P_HALIGN      0x0020 /* 垂直 / 水平配置 */
#define P_PRESS       0x0040 /* プレス動作指定 */
#define P_NUMBER      0x0040 /* 番号付け指定 */
#define P_DOUBLE      0x0040 /* 2 段組み配置 */

#define P_SBAR        0x0040 /* スクロールバー指定 */
#define P_DCLICK      0x0080 /* ダブルクリック動作指定 */
#define P_NOSEL       0x0080 /* 0 選択可能 */
#define P_APPEND      0x0080 /* 追加選択モード */
#define P_NONOBB      0x0080 /* トンボ表示無し */

#define P_EMPHAS      0x0200 /* 強調枠 / 通常枠 */
#define P_NOFRAME     0x0400 /* 枠無し / 枠有り */
#define P_DISABLE     0x0800 /* 操作禁止 / 操作許可 */
#define P_INACT       0x1000 /* 不能状態 / 可能状態 */
#define P_BLINK       0x2000 /* 点滅状態 */

```

```
#define P_DISP      0x4000 /* 表示状態 / 消去状態 */
```

```
#define P_PARTDISP  0x8000 /* 表示属性指定 */
```

### 属性コード(メタコード)

```
#define MC_STR      0x1000 /* 文字列 */
```

```
#define MC_FIG      0x1001 /* 図形データ */
```

```
#define MC_ATTR     0x1002 /* 文字属性指定 */
```

```
#define MC_EMPHAS   0x1400 /* 強調項目 */
```

```
#define MC_INACT    0x1800 /* 不能項目 */
```

```
#define MC_SFLD     0x0800 /* 文字列フィールド指定 */
```

```
#define MC_NFLD     0x0c00 /* 数値フィールド指定 */
```

```
#define MC_NZERO    0x0200 /* 前ゼロ表示 */
```

```
#define MC_NLEFT    0x0100 /* 左揃え */
```

### 数値形式

```
#define P_LNUM      0x0000 /* 整数 (32ビット) */
```

```
#define P_DNUM      0x0080 /* 実数 (64ビット) */
```

```
#define P_FIX0      0x0000 /* 固定小数点表現 - 0 */
```

```
#define P_FIX1      0x0010 /* 固定小数点表現 - 1 */
```

```
#define S_UNDEF     0x8000 /* 負の最大値 */
```

```
#define L_UNDEF     0x80000000 /* 負の最大値 */
```

```
#define D_UNDEF     1.797693e308 /* NaN */
```

### 終了コード

```
#define P_CHANGE    0x1000 /* 変化あり */
```

```
#define P_BREAK     0x2000 /* 処理の中断 */
```

```
#define P_SMASK     0x0fff /* 終了 / 中断状態のマスク */
```

```
#define P_MENU      0x6010 /* メニュー指定で中断 */
```

```
#define P_EVENT     0x6020 /* 他のイベント発生で中断 */
```

```
#define P_SGL       1 /* 有効なシングルクリック */
```

```
#define P_DBL       2 /* 有効なダブルクリック */
```

```
#define P_TAB       0 /* タブで終了 */
```

```
#define P_NL        1 /* 改段落 / 改行で終了 */
```

```
#define P_END       2 /* 入力終で終了 */
```

```
#define P_BUT       3 /* ボックス外でのプレス */
```

```
#define P_MOVE      4 /* ボックス外への移動 */
```

```
#define P_COPY      5 /* ボックス外への複写 */
```

```
#define P_UP        0 /* 上方向 への移動 */
```

```
#define P_DOWN      1 /* 下方向 への移動 */
```

```
#define P_LEFT      2 /* 左方向 への移動 */
```

```
#define P_RIGHT      3   /* 右方向 への移動 */
#define P_SMOOTH    0   /* スムース移動 */
#define P_AREA      4   /* エリア移動 */
#define P_JUMP      8   /* ジャンプ移動 */
```

#### 状態変更コマンド

```
#define P_ERASE      0x0000 /* 消去 */
#define P_NODISP    0x1000 /* 消去状態に */
#define P_RDISP     0x8000 /* 再表示 */

#define P_NORMAL    0x0002 /* 通常枠 */
#define P_FRAME     0x0004 /* 枠有り */
#define P_ENABLE    0x0008 /* 操作許可 */
#define P_ACT       0x0010 /* 可能状態 */
```

### 3.3.4 パーツマネージャの関数

#### ccre\_par

パーツの登録

#### 【形式】

```
PAID    ccre_par(W wid, PARTS *parts)
```

#### 【パラメータ】

W wid ウィンドウID  
PARTS \*parts パーツデータ

#### 【リターン値】

0 正常(パーツID > 0)  
<0 エラー(エラーコード)

#### 【解説】

parts により定義したコントロールパーツを wid で指定したウィンドウ / パネルに属するパーツとして登録し、関数値として登録されたパーツのパーツID (pid > 0) を戻す。パーツの定義データの状態が「表示状態」の場合はパーツの表示が行なわれるが、「消去状態」の場合は、登録のみで表示は行なわれない。

#### 【エラーコード】

EX\_ADR : アドレス(parts等)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である。  
EX\_WID : ウィンドウ(wid)は存在していない。  
EX\_DNUM : データ(onpat , ofpat , patnum)はデータマネージャに登録されていない。

## ccre\_tbx

パーツの登録

### 【形式】

PAID ccre\_tbx(W wid, UW ty, RECT \*r, W txsize, TC \*text, PARTDISP \*atr)

### 【パラメータ】

|          |        |                                  |
|----------|--------|----------------------------------|
| W        | wid    | ウィンドウID                          |
| UW       | ty     | パーツ属性(タイプは無視されて TB_PARTS に固定される) |
| RECT     | *r     | パーツ領域                            |
| W        | txsize | 最大文字数                            |
| TC       | *text  | 初期文字列                            |
| PARTDISP | *atr   | 表示属性                             |

### 【リターン値】

0 正常(パーツID > 0)  
<0 エラー(エラーコード)

### 【解説】

テキストボックスパーツを作成しそのパーツIDを戻す。

### 【エラーコード】

## ccre\_xbx

パーツの登録

### 【形式】

PAID ccre\_xbx(W wid, UW ty, RECT \*r, W txsize, TC \*text, PARTDISP \*atr)

### 【パラメータ】

|   |     |         |
|---|-----|---------|
| W | wid | ウィンドウID |
|---|-----|---------|



|          |        |                                  |
|----------|--------|----------------------------------|
| UW       | ty     | パーツ属性(タイプは無視されて XB_PARTS に固定される) |
| RECT     | *r     | パーツ領域                            |
| W        | txsize | 最大文字数                            |
| TC       | *text  | 初期文字列                            |
| PARTDISP | *atr   | 表示属性                             |

### 【リターン値】

0 正常(パーツID > 0)  
<0 エラー(エラーコード)

### 【解説】

シークレットテキストボックスパーツを作成しそのパーツIDを戻す。

### 【エラーコード】

ccre\_nbx

パーツの登録

### 【形式】

PAID ccre\_nbx(W wid, UW ty, RECT \*r, UW fmt, W \*val, PARTDISP \*atr)

### 【パラメータ】

|          |      |                                  |
|----------|------|----------------------------------|
| W        | wid  | ウィンドウID                          |
| UW       | ty   | パーツ属性(タイプは無視されて NB_PARTS に固定される) |
| RECT     | *r   | パーツ領域                            |
| UW       | fmt  | 書式                               |
| W        | *val | 初期値                              |
| PARTDISP | *atr | 表示属性                             |

### 【リターン値】

0 正常(パーツID > 0)  
<0 エラー(エラーコード)

### 【解説】

数値ボックスパーツを作成しそのパーツIDを戻す。

### 【エラーコード】

# ccre\_sbx

## パーツの登録

### 【形式】

PAID ccre\_sbx(W wid, UW ty, RECT \*r, TC \*fmt, W nfld, W \*cv, PARTDISP \*atr)

### 【パラメータ】

|          |      |                                  |
|----------|------|----------------------------------|
| W        | wid  | ウィンドウID                          |
| UW       | ty   | パーツ属性(タイプは無視されて SB_PARTS に固定される) |
| RECT     | *r   | パーツ領域                            |
| TC       | *fmt | 書式                               |
| W        | nfld | フィールド数                           |
| W        | *cv  | 初期値配列                            |
| PARTDISP | *atr | 表示属性                             |

### 【リターン値】

0 正常(パーツID > 0)  
<0 エラー(エラーコード)

### 【解説】

シリアルボックスパーツを作成しそのパーツIDを戻す。

### 【エラーコード】

# ccre\_asw

## パーツの登録

### 【形式】

PAID ccre\_asw(W wid, UW ty, RECT \*r, W cv, TC \*name, PARTDISP \*atr)

### 【パラメータ】

|      |       |                                  |
|------|-------|----------------------------------|
| W    | wid   | ウィンドウID                          |
| UW   | ty    | パーツ属性(タイプは無視されて AS_PARTS に固定される) |
| RECT | *r    | パーツ領域                            |
| W    | cv    | 初期値                              |
| TC   | *name | スイッチ名文字列                         |

PARTDISP \*atr 表示属性

【リターン値】

0 正常(パーツID > 0)  
<0 エラー(エラーコード)

【解説】

テキストオルタネートスイッチパーツを作成しそのパーツIDを戻す。

【エラーコード】

ccre\_msw

パーツの登録

【形式】

PAID ccre\_msw(W wid, UW ty, RECT \*r, TC \*name, PARTDISP \*atr)

【パラメータ】

|          |       |                                  |
|----------|-------|----------------------------------|
| W        | wid   | ウィンドウID                          |
| UW       | ty    | パーツ属性(タイプは無視されて MS_PARTS に固定される) |
| RECT     | *r    | パーツ領域                            |
| TC       | *name | スイッチ名文字列                         |
| PARTDISP | *atr  | 表示属性                             |

【リターン値】

0 正常(パーツID > 0)  
<0 エラー(エラーコード)

【解説】

テキストモーメンタリスイッチパーツを作成しそのパーツIDを戻す。

【エラーコード】

ccre\_pas

パーツの登録

【形式】

PAID ccre\_pas(W wid, UW ty, RECT \*r, W cv, W onp, W offp, PARTDISP \*atr)

### 【パラメータ】

|          |      |                                  |
|----------|------|----------------------------------|
| W        | wid  | ウィンドウID                          |
| UW       | ty   | パーツ属性(タイプは無視されて PA_PARTS に固定される) |
| RECT     | *r   | パーツ領域                            |
| W        | cv   | 初期値                              |
| W        | onp  | オン時の BMAP_DATA タイプのデータ番号         |
| W        | offp | オフ時の BMAP_DATA タイプのデータ番号         |
| PARTDISP | *atr | 表示属性                             |

### 【リターン値】

0 正常(パーツID > 0)  
< 0 エラー(エラーコード)

### 【解説】

ピクトグラムオルタネートスイッチパーツを作成しそのパーツIDを戻す。

### 【エラーコード】

ccre\_pms

パーツの登録

### 【形式】

PAID ccre\_pms(W wid, UW ty, RECT \*r, W onp, PARTDISP \*atr)

### 【パラメータ】

|          |      |                                  |
|----------|------|----------------------------------|
| W        | wid  | ウィンドウID                          |
| UW       | ty   | パーツ属性(タイプは無視されて PM_PARTS に固定される) |
| RECT     | *r   | パーツ領域                            |
| W        | onp  | オン時の BMAP_DATA タイプのデータ番号         |
| PARTDISP | *atr | 表示属性                             |

### 【リターン値】

0 正常(パーツID > 0)  
< 0 エラー(エラーコード)

### 【解説】

ピクトグラムモーメンタリスイッチパーツを作成しそのパーツIDを戻す。

### 【エラーコード】

ccre\_sel

パーツの登録

### 【形式】

PAID ccre\_sel(W wid, UW ty, RECT \*r, W cv, TC \*name, PARTDISP \*atr)

### 【パラメータ】

|          |       |                                  |
|----------|-------|----------------------------------|
| W        | wid   | ウィンドウID                          |
| UW       | ty    | パーツ属性(タイプは無視されて WS_PARTS に固定される) |
| RECT     | *r    | パーツ領域                            |
| W        | cv    | 初期値                              |
| TC       | *name | スイッチセクタ文字列                       |
| PARTDISP | *atr  | 表示属性                             |

### 【リターン値】

0 正常(パーツID > 0)  
<0 エラー(エラーコード)

### 【解説】

スイッチセクタパーツを作成しそのパーツIDを戻す。

### 【エラーコード】

ccre\_scr

パーツの登録

### 【形式】

PAID ccre\_scr(W wid, UW ty, RECT \*r, W cv, TC \*name, PARTDISP \*atr)

### 【パラメータ】

|   |     |         |
|---|-----|---------|
| W | wid | ウィンドウID |
|---|-----|---------|

|          |       |                                  |
|----------|-------|----------------------------------|
| UW       | ty    | パーツ属性(タイプは無視されて SS_PARTS に固定される) |
| RECT     | *r    | パーツ領域                            |
| W        | cv    | 初期値                              |
| TC       | *name | スクロールセクタ文字列                      |
| PARTDISP | *atr  | 表示属性                             |

### 【リターン値】

0 正常(パーツID > 0)  
<0 エラー(エラーコード)

### 【解説】

スクロールセクタパーツを作成しそのパーツIDを戻す。

### 【エラーコード】

ccre\_vol

パーツの登録

### 【形式】

PAID ccre\_vol(W wid, UW ty, RECT \*r, W \*val, W patnum, PARTDISP \*atr)

### 【パラメータ】

|          |        |                                  |
|----------|--------|----------------------------------|
| W        | wid    | ウィンドウID                          |
| UW       | ty     | パーツ属性(タイプは無視されて VL_PARTS に固定される) |
| RECT     | *r     | パーツ領域                            |
| W        | *val   | 初期値配列                            |
| W        | patnum | ノブの PAT_DATA タイプのデータ番号           |
| PARTDISP | *atr   | 表示属性                             |

### 【リターン値】

0 正常(パーツID > 0)  
<0 エラー(エラーコード)

### 【解説】

ボリュームパーツを作成しそのパーツIDを戻す。

### 【エラーコード】

## copn\_par

パーツのデータボックスからの登録

### 【形式】

PAID copn\_par(W wid, W dnum, PNT \*pos)

### 【パラメータ】

W wid ウィンドウID  
W dnum PARTS\_DATA タイプのデータ番号  
PNT \*pos パーツの矩形領域の左上の点(相対座標)

### 【リターン値】

0 正常(パーツID > 0)  
< 0 エラー(エラーコード)

### 【解説】

dnum で指定したデータ番号の "PARTS\_DATA" タイプのデータにより定義されるコントロールパーツを wid で指定したウィンドウ / パネルに属するパーツとして登録する。

\*pos はパーツの矩形領域の左上の点を指定したウィンドウ / パネルの相対座標で指定するもので、パーツ定義データで定義された位置を移動して登録する。pos = NULL の場合は、パーツ定義データ上で定義された位置で登録することを意味する。

パーツの定義データの状態が「表示状態」の場合はパーツの表示が行なわれるが、「消去状態」の場合は、登録のみで表示は行なわれない。

### 【エラーコード】

EX\_ADR : アドレス(pos)のアクセスは許されていない。  
EX\_DNUM : データ(dnum等)はデータマネージャに登録されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である。  
EX\_WID : ウィンドウ(wid)は存在していない。

## cdel\_par

パーツの削除

### 【形式】

ERR cdel\_par(W pid, W opt)

## 【パラメータ】

W opt ::= ( CLR NOCLR )

CLR (=0) :

削除したパーツが表示されている時は表示を消去する。

NOCLR( 0) :

削除したパーツが表示されている時は表示はそのまま残る(何もしない)。

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

pid で指定した登録済みパーツを削除する。

opt により、削除したパーツの表示を消去するか否かを指定する。パーツを登録したプロセスが終了した場合、またはパーツが属するウィンドウ/パネルが削除された場合には、自動的にパーツは削除されるが、原則的には不要になった時点で明示的に削除すべきである。

## 【エラーコード】

EX\_PID : パーツ(pid)は存在していない。

**cdel\_pwd**

ウィンドウ内パーツの削除

## 【形式】

ERR cdel\_pwd(W wid, W opt)

## 【パラメータ】

W wid ウィンドウID  
W opt cdel\_par と同じ

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】



wid で指定したウィンドウ / パネルに属する登録済みパーツをすべて削除する。但し、P\_SBAR = 1 の属性を持つボリュームは対象外となり削除されない。

opt により、削除したパーツの表示を消去するか否かを指定し、その内容は cdel\_par() の場合と同一である。

パーツを登録したプロセスが終了した場合、またはパーツが属するウィンドウ / パネルが削除された場合には、自動的にパーツは削除されるが、原則的には不要になった時点で明示的に削除すべきである。

## 【エラーコード】

EX\_WID : ウィンドウ(wid)は存在していない。

# cdsp\_par

パーツの表示制御

## 【形式】

ERR cdsp\_par(W pid, UW vis)

## 【パラメータ】

W pid パーツID

UW vis 以下のいずれかの値を取り、消去 / 表示 / 点滅 / 再表示の指定を行なう。

P\_ERASE (消去):

パーツが「表示状態」の場合、背景パターンでパーツの矩形領域を塗り潰し「消去状態」とし「点滅状態」をリセットする。それ以外の場合は何も行なわない。結果としての状態は P\_DISP = 0, P\_BLINK = 0 となる。

P\_NODISP (消去状態に):

状態のみを消去状態とする。結果としての状態は P\_DISP = 0, P\_BLINK = 0 となる。

P\_DISP (表示):

パーツが「消去状態」、または「点滅状態」の場合、表示を行ない「表示状態」とし「点滅状態」をリセットする。それ以外の場合は何も行なわない。結果としての状態は P\_DISP = 1, P\_BLINK = 0 となる。

P\_BLINK (点滅):

パーツが「点滅状態」でない場合、「表示状態」かつ「点滅状態」とする。実際の点滅表示は cidl\_par() により行なわれる。それ以外の場合は何も行なわない。「点滅状態」が定義されていないパーツの場合は、P\_DISP と同じ動作となる。結果としての状態は P\_DISP = 1, P\_BLINK = 1 となる。

P\_RDISP (再表示):

パーツの状態に関係なく表示を行ない「表示状態」とし、「点滅状態」をリセットする。結果としての状態は P\_DISP = 1, P\_BLINK = 0 となる。

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

pid で指定したパーツの消去 / 表示 / 点滅 / 再表示を行なう。

## 【エラーコード】

EX\_PAR : パラメータが不正である(vis が不正)。

EX\_PID : パーツ(pid)は存在していない。

# cdsp\_pwd

ウィンドウ内パーツの表示制御

## 【形式】

ERR cdsp\_pwd(W wid, RECT \*r, UW vis)

## 【パラメータ】

|      |     |  |
|------|-----|--|
| W    | wid | ウィンドウID  |
| RECT | *r  | 表示したいウィンドウ内の矩形領域   |
| UW   | vis | 消去 / 表示 / 点滅 / 再表示の指定を行なうもので、その内容は cdsp_par() の場合と同一である。 |

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウ / パネルに属している登録済みのパーツのうち、\*r で指定した矩形領域に重なるパーツの消去 / 表示 / 点滅 / 再表示を行なう。但し、P\_SBAR = 1 の属性を持つ、ボリュームは対象外となる。消去 / 表示 / 点滅 / 再表示の処理は登録された順番に行なわれる。

\*r は指定したウィンドウ / パネルの相対座標で指定され、r = NULL の場合は、ウィンドウ / パネル全体の領域を意味する。

## 【エラーコード】

EX\_ADR : アドレス(r)のアクセスは許されていない。

EX\_PAR : パラメータが不正である(vis が不正)。

EX\_WID : ウィンドウ(wid)は存在していない。

## cchk\_par

パーツの位置チェック

### 【形式】

W cchk\_par(W pid, PNT pos)

### 【パラメータ】

W pid パーツID  
PNT pos 位置座標

### 【リターン値】

0 正常(関数値はパーツのタイプ(含まれていない場合は0))  
<0 エラー(エラーコード)

### 【解説】

pos で指定した位置の点が pid で指定したパーツの矩形領域に含まれているか否かをチェックし、含まれていない場合は0、含まれていた場合はそのパーツのタイプコード(1 ~ )を関数値として戻す。

ただし、禁止状態 (P\_DISABLE = 1) のパーツ、消去状態 (P\_DISP = 0) のパーツはチェックの対象外となり関数値は常に0となる。

pos はウィンドウ内の相対座標で指定される。

### 【エラーコード】

EX\_PID : パーツ(pid)は存在していない。

## cfnd\_par

ウィンドウ内パーツの位置サーチ

### 【形式】

W cfnd\_par(W wid, PNT pos, W \*pid)

### 【パラメータ】

W wid ウィンドウID

PNT pos 位置座標  
W \*pid パーツID格納領域

### 【リターン値】

0 正常(関数値はパーツのタイプ(含まれていない場合は0))  
<0 エラー(エラーコード)

### 【解説】

wid で指定したウィンドウ / パネルに属する登録済みパーツに対して、 pos で指定した位置の点が含まれているか否かを登録された順番にサーチし、含まれていた場合は、そのパーツIDを \*pid に戻し、含まれていない場合は 0 を \*pid に戻す。

ただし、禁止状態 (P\_DISABLE = 1) のパーツ、消去状態 (P\_DISP=0) のパーツ、および P\_SBAR = 1 の属性を持つボリュームはサーチの対象外となる。

関数値としては、含まれていない場合は 0 が戻り、含まれていた場合はそのパーツのタイプコード (1 ~ ) が戻る。

### 【エラーコード】

EX\_ADR : アドレス(pid)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。

cact\_par

パーツの動作実行

### 【形式】

W cact\_par(W pid, WEVENT \*ev)

### 【パラメータ】

W pid パーツID  
WEVENT \*ev パーツ動作のきっかけとなったイベント

### 【リターン値】

0 正常(関数値下位16bit): 01PC SSSS SSSS SSSS  
P = 0: 処理の終了  
= 1: 処理の中断  
C = 0: 現在値/状態の変化無し  
= 1: 現在値/状態の変化有り  
S : 各種のコード  
<0 エラー(エラーコード)

## 【解説】

pid で指定したパーツの固有の動作を開始する。\*ev はパーツの動作の起動とする、ウィンドウイベントであり、パーツの動作終了、中断時には、終了、中断の契機となったイベントが設定されて戻る。

この関数の動作はパーツのタイプによって異なり、起動となりうるイベントの種類、リターン時の関数値の意味も異なるが、共通的に以下の場合には、何もせず、関数値として0を戻す。この時、\*ev の内容は変化しない。

- パーツが操作不可状態 (P\_DISABLE = 1) の場合。
- \*ev で指定したイベントのタイプがそのパーツの処理対象でない場合。
- イベントのタイプが EV\_BUTDWN または EV\_SWITCH で、イベント発生位置(ev->e.pos)がパーツの領域外であった場合。

処理の中断の場合には、アプリケーション側で適切な処理を行なった後、\*ev のイベントタイプに EV\_NULL を設定して、再び cact\_par() をコールする必要がある。

なお、指定したパーツの属するウィンドウ / パネルが入力受付状態でない場合、およびアクティブプロセスでない場合は、何もせずに関数値0を戻す。

テキストボックス / シークレットテキストボックス / 数値ボックスの場合

EV\_BUTDWN または EV\_SWITCH の場合：

- 「不能状態」の場合は「可能状態」とし、「入力状態」とする。
- プレスした位置が選択領域でない場合は、プレスした位置を選択開始位置とし、入力 / 編集処理を行なう。ダブルクリックの場合は全体が選択される。
- プレスした位置が選択領域であった場合は、選択領域枠のドラッグ処理を行なう。この時単なるドラッグ移動か、複写かの区別が行なわれる。

EV\_KEYDWN または EV\_NULL の場合：

- 「不能状態」の場合は「可能状態」とし、「入力状態」とする。
- 現在の選択領域に対して入力 / 編集処理を行なう。

その他の場合：

何もせず関数値 0 を戻す(\*ev の内容は変化しない)。

入力 / 編集処理は以下の内容となる。

- 「入力状態」とし、選択領域 / カレットの表示を行なう。同一のウィンドウ内で、以前に「入力状態」であったボックスがあれば、「非入力状態」とし、選択領域 / カレットの表示を消す。
- イベントを順次取り出しながら、[3.3.2.1 テキストボックス](#)、[3.3.2.2 シークレットテキストボックス](#)、[3.3.2.3 数値ボックス](#) で説明した動作を行なう。
- [タブ]、[入力終]または[改段落 / 改行]キーにより、「非入力状態」となり表示を更新して処理を終了する。
- ボックス外でのプレス、ドラッグによるボックス外でのリリースの場合は、「入力状態」のまま、表示を変えずに処理を終了する。
- EV\_BUTDWN, EV\_BUTUP, EV\_KEYDWN, EV\_KEYUP, EV\_AUTKEY 以外のイベントが発生した場合、または、メニュー指定が発生した場合は、そのままの状態での処理を中断する。\*ev には得られたイベントが設定され、関数値は以下のものとなる。

関数値：

他のイベントの場合 (P\_EVENT) :

0110 0000 0010 0000

メニュー指定の場合 (P\_MENU) :

0110 0000 0001 0000

なお、メニュー指定とは、以下のいずれかの場合を示す。

- (1) メニューボタンのクリック
- (2) 「命令」キーを押した状態でのプレス
- (3) 「命令」キーと別のキーの同時押し(キーマクロ)

アプリケーションは、対応する処理を行なった後、\*ev のイベントタイプに EV\_NULL を設定して、再び cact\_par() をコールして処理を再開する必要がある。

- 終了時には処理終了の契機となったイベントがそのまま \*ev に設定される。
- 処理終了時には、以下の関数値が戻る。

関数値:010C 0000 0000 0SSS

C = 0: 現在値の変化無し  
= 1: 現在値の変化有り  
S = P\_TAB 「タブ」  
= P\_NL 「改段落/改行」  
= P\_END 「入力終」  
= P\_BUT ボックス外でのプレス  
= P\_MOVE ボックス外への移動  
= P\_COPY ボックス外への複写

## シリアルボックスの場合

EV\_BUTDWN または EV\_SWITCH の場合 :

- 「不能状態」の場合は「可能状態」とし、「入力状態」とする。
- プレスした位置に最も近いフィールドを選択し、入力 / 編集処理を行なう。

EV\_KEYDWN または EV\_NULL の場合 :

- 「不能状態」の場合は「可能状態」とし、「入力状態」とする。
- 現在の選択フィールドに対して入力 / 編集処理を行なう。

その他の場合 :

何もせず関数値 0 を戻す。(\*ev の内容は変化しない)

入力 / 編集処理は以下の内容となる。

- 「入力状態」とし、選択領域の表示を行なう。同一のウィンドウ内で、以前に「入力状態」であったボックスがあれば、「非入力状態」とし、選択領域の表示を消す。
- [タブ]キーにより次のフィールドを選択する。
- イベントを順次取り出しながら[3.3.2.4 シリアルボックス](#)で説明した動作を行なう。
- 最後の[タブ]、[入力終]または[改段落 / 改行]キーにより、「非入力状態」となり表示を更新して処理を終了する。
- ボックス外でのプレスの場合は、「入力状態」のまま、表示を変えずに処理を終了する。
- EV\_BUTDWN, EV\_BUTUP, EV\_KEYDWN, EV\_KEYUP, EV\_AUTKEY 以外のイベントが発生した場合、またはメニュー指定が発生した場合は、そのままの状態での処理を中断する。\*ev には得られたイベントを設定され、関数値は以下のものとなる。

関数値：

他のイベントの場合 (P\_EVENT)：

0110 0000 0010 0000

メニュー指定の場合 (P\_MENU)：

0110 0000 0001 0000

アプリケーションは、対応する処理を行なった後、\*ev のイベントタイプに EV\_NULL を設定して、再び cact\_par() をコールして処理を再開する必要がある。

- 終了時には処理終了の契機となったイベントがそのまま \*ev に設定される。
- 処理終了時には、以下の関数値が戻る。

関数値:010C 0000 0000 0SSS

C = 0: 現在値の変化無し

= 1: 現在値の変化有り

S = P\_TAB 「タブ」

= P\_NL 「改段落/改行」

= P\_END 「入力終」

= P\_BUT ボックス外でのプレス

テキスト / ピクトグラムオルターネートスイッチの場合

EV\_BUTDWN または EV\_SWITCH の場合：

- 「不能状態」の場合は「可能状態」とする。また「点滅状態」の場合は「表示状態」とする。
- インジケータまたは図形表示を反転表示し、プレス中の位置に応じて反転表示の解除 / 復活を行なう。
- リリースされた場合は、その位置に応じて現在値 / 表示を変更し、以下の関数値を戻して処理を終了する。

関数値:010C 0000 0000 000S

C = 0: 現在値の変化無し。(矩形外でのリリース)

= 1: 現在値の変化有り。(矩形内でのリリース)

S : 現在値 (0:OFF, 1:ON)

- リリース時に発生した EV\_BUTUP はイベントキューから取り除かれる。また処理中に発生した EV\_KEYDWN, EV\_KEYUP, EV\_AUTKEY はイベントキューから取り除かれて無視されるが、その他のイベントはイベントキューに残る。

その他の場合：

何もせず関数値 0 を戻す。(\*ev の内容は変化しない)

テキスト / ピクトグラムモーメンタリスイッチの場合

EV\_BUTDWN または EV\_SWITCH の場合：

- 「不能状態」の場合は「可能状態」とする。また「点滅状態」の場合は「表示状態」とする。
- 矩形内部または図形表示を反転表示し、プレス中の位置に応じて反転表示の解除 / 復活を行なう。
- P\_PRESS = 1 (プレス中の動作を行なう) の場合は、プレス中で、かつパーツの矩形領域内にある場合 (反転表示されている状態) は、以下の関数値を戻して処理を中断す

る。この時 \*ev のイベントタイプは EV\_NULL に設定される。

関数値 : 0110 0000 0000 0000

アプリケーションは、プレス中の動作を行なった後、再び cact\_par() をコールする必要がある。

- P\_DCLICK = 0 (ダブルクリックの検出なし) の場合は、リリースされた時点ですぐ処理を終了するが、P\_DCLICK = 1 の場合は、ダブルクリックが否かをチェックした後リターンする (なお、ダブルクリックは最初のプレスから2回目のリリースまでの時間で判断される)。
- リリースされた場合は、その位置に応じて現在値 / 表示を変更し、以下の関数値を戻して処理を終了する。

関数値: 010C 0000 0000 00DS

C = 0: 状態の変化無し。(リリースが矩形領域外)  
= 1: 状態の変化有り。(リリースが矩形領域内)  
D = 1: ダブルクリック (P\_DCLICK=1の時のみ)  
S = 1: シングルクリック  
(C=0の場合は、D,Sも0となる)

- リリース時に発生した EV\_BUTUP はイベントキューから取り除かれる (ダブルクリックの場合は途中の EV\_BUTUP, EV\_BUTDWN も取り除かれる)。また処理中に発生した EV\_KEYDOWN, EV\_KEYUP, EV\_AUTKEY はイベントキューから取り除かれて無視されるが、その他のイベントはイベントキューに残る。

EV\_NULL の場合:

矩形領域内でプレスされたと見なして、EV\_BUTDWN と同様の処理を行なう。

その他の場合:

何もせず関数値 0 を戻す (\*ev の内容は変化しない)。

スイッチセレクトの場合

EV\_BUTDWN または EV\_SWITCH の場合:

- 「不能状態」の場合は「可能状態」とする。
- 対応する選択肢のインジケータを反転表示し、プレス中の位置に応じて反転表示の切り換えを行なう。
- リリースされた場合は、その位置に応じて現在値 / 表示の変更を行ない、以下の関数値を戻して処理を終了する。

関数値: 010C SSSS SSSS SSSS

C = 0: 現在値の変化無し  
= 1: 現在値の変化有り  
S : 現在値 (現在の選択肢の番号、0: 選択無し)

現在値が4095を超える場合には、返される現在値は不定となる。

- リリース時に発生した EV\_BUTUP はイベントキューから取り除かれる。また処理中に発生した EV\_KEYDOWN, EV\_KEYUP, EV\_AUTKEY はイベントキューから取り除かれて無視されるが、その他のイベントはイベントキューに残る。

その他の場合:



何もせず関数値 0 を戻す(\*ev の内容は変化しない)。

## スクロールセクタ

EV\_BUTDWN または EV\_SWITCH の場合：

- 「不能状態」の場合は「可能状態」とする。
- イベント発生位置 (ev->e.pos) がスクロール (選択肢表示) 領域の場合、対応する選択肢を反転表示し、プレス中の位置に応じて反転表示の切り換えを行なう。
- イベント発生位置 (ev->e.pos) がスクロールバー領域の場合、スクロールバー処理 (ジャンプ移動、スムーズ移動、エリア移動) を行ない、スクロール部の表示をスクロールする。
- スムーズ移動の際に、ボリュームの様な中断は起らず、表示の更新等は全てパーツマネージャ側で行なう。
- エリア移動の移動幅は、(表示されている選択肢数 - 1) で表示の更新等は全てパーツマネージャ側で行なう。
- リリースされた場合は、その位置に応じて現在値 / 表示の変更を行ない、以下の関数値を戻して処理を終了する。

関数値:010C SSSS SSSS SSSS

C = 0: 現在値の変化無し。

= 1: 現在値の変化有り。

S : 現在値 (現在の選択肢の番号、0:選択無し)

現在値が 4095 を超える場合には、返される現在値は不定となる。

- リリース時に発生した EV\_BUTUP はイベントキューから取り除かれる。また処理中に発生した EV\_KEYDWN, EV\_KEYUP, EV\_AUTKEY はイベントキューから取り除かれて無視されるが、その他のイベントはイベントキューに残る。

その他の場合：

何もせず関数値 0 を戻す(\*ev の内容は変化しない)。

【注意】キー操作によるスクロールセクタの操作は cact\_par() としては一切行なわず、スクロールセクタの現在値 (選択肢の番号) を直接設定することにより行なう。また、選択肢番号をキーボードから入力し[変換]キーを押すことで、指定した番号の選択肢を直接選択する等の操作も全てアプリケーションによって実現される。

## ボリュームの場合

EV\_BUTDWN または EV\_SWITCH の場合：

- 「不能状態」の場合は「可能状態」とする。
- 【ジャンプ移動】

イベント発生位置 (ev->e.pos) がノブ (白) 部分の場合、ノブ部分の枠の影をドラッグに追従させて移動し、リリースされた場合は、その位置に応じて現在値 (clo, chi) / 表示を更新して処理を終了する。この場合、ポインタはボリュームの方向に合わせた縦または横の「握り」になり、リリース時点で元の形に戻る。なお、選択状態でなかった場合は、一度ポインタの形状を縦または横の「移動手」とする。

- 【エリア移動】

イベント発生位置 (ev->e.pos) がパターン部分の場合、ダブルクリックか否かを判断し、ダブルクリックの場合は、そのまま処理を終了する。現在値 / 表示の更新は行な

われない。

○ 【スムーズ移動】

ダブルクリックでない場合は、プレス中で、ボリュームの矩形領域内にあり、トンボとの距離がある場合は、すぐに以下の関数値を戻して処理を中断する。この時 \*ev のイベントタイプは EV\_NULL に設定される。

関数値:0110 0000 0000 00DD

D = 0: P\_UP -- 上方向 への移動  
= 1: P\_DOWN -- 下方向 への移動  
= 2: P\_LEFT -- 左方向 への移動  
= 3: P\_RIGHT -- 右方向 への移動

アプリケーションは、対応する処理を行ないボリュームの現在値 (clo, chi) を更新した後、再び cact\_par() をコールする必要がある。

○ リリースされた場合、以下の関数値を戻して処理を終了する。

関数値:010C 0000 0000 SSDD

C = 0: 状態の変化無し  
= 1: 状態の変化有り  
D = 0: P\_UP -- 上方向 への移動  
= 1: P\_DOWN -- 下方向 への移動  
= 2: P\_LEFT -- 左方向 への移動  
= 3: P\_RIGHT -- 右方向 への移動  
S = 0: P\_SMOOTH -- スムース移動  
= 1: P\_AREA -- エリア移動  
= 2: P\_JUMP -- ジャンプ移動

○ リリース時に発生した EV\_BUTUP はイベントキューから取り除かれる (ダブルクリックの場合は途中の EV\_BUTUP, EV\_BUTDWN も取り除かれる)。また処理中に発生した EV\_KEYDOWN, EV\_KEYUP, EV\_AUTKEY はイベントキューから取り除かれて無視されるが、その他のイベントはイベントキューに残る。

EV\_NULL の場合:

パターン領域内でプレスされたと見なして、EV\_BUTDWN のスムーズ処理と同様の処理を行なう。

【エラーコード】

EX\_ADR : アドレス(ev)のアクセスは許されていない。

EX\_PID : パーツ(pid)は存在していない。

cget\_val

パーツの値の取出し

【形式】

W cget\_val(W pid, W size, W \*value)

## 【パラメータ】

W pid パーツID  
W size value のサイズ(ワード数またはハーフワード数)  
W \*value 値を格納する配列

## 【リターン値】

0 正常(関数値は現在値のワード数)  
<0 エラー(エラーコード)

## 【解説】

pid で指定したパーツの値を取り出して、value で指定した領域に格納する。size は、value の領域のデータ数(ワード数またはハーフワード数)を指定する。格納される value の内容、および関数値の意味はパーツのタイプによって異なる。

### テキストボックス/シークレットテキストボックス

- 現在の文字列を (TC\*) value で指定した領域に格納する。size はその領域のハーフワード数(文字数)を示し、領域が不足した場合は先頭から (size - 1) 文字のみが格納され、最後に TNULL が格納される。関数値として格納された文字列長が戻る。
- size 0、または value=NULL の場合は、文字列の取り出しは行なわれず、最大文字数が関数値として戻る。

### 数値ボックス

- cv の値を取り出し、value で指定した領域に格納する。size はその領域のワード数を示し、領域が不足した場合は、その size 分だけ格納される。関数値として以下に示す値(全体のワード数)が戻る。

- 1 -- 整数の場合 (32ビット)
- 2 -- 実数の場合 (64ビット浮動小数点)

- size 0、または value = NULL の場合は、値の取り出しは行なわれず、全体のワード数が関数値として戻る。

### シリアルボックス

- 各フィールドの cv 値を順番に取り出し、value で指定した領域に格納する。size はその領域のワード数を示し、領域が不足した場合は、その size 分だけ格納される。関数値として全体のワード数(フィールド数)が戻る。
- size 0、または value = NULL の場合は、値の取り出しは行なわれず、全体のワード数が関数値として戻る。

### テキスト/ピクトグラムオルタネート/モーメンタリスイッチ

### スイッチセクタ/スクロールセクタ

- 現在値を取り出し、value で指定した領域に格納する。現在値は常に 1 ワードであるため、size は通常 1 とする。関数値として現在値が戻る。

- モーメンタリスイッチの場合は、値は常に 0 であり意味がない。
- size 0、または value = NULL の場合は、値の取り出しは行なわれず、現在値が関数値として戻る。
- スクロールセレクトタの場合は、size = 2,3 とすると、value には、以下の値が格納される。

value[0] : 現在値 (size > 0の時)  
 value[1] : 表示されている選択肢の数 (size > 1の時)  
 value[2] : 一番上に表示されている選択肢の番号 (size > 2の時)

## ボリューム

- clo, chi, lo, hi の値を順番に取り出し、value で指定した領域に格納する。size はその領域のワード数を示し、領域が不足した場合は、その size 分だけ格納される。関数値としては、全体のワード数 (= 4) が戻る。
- size 0、または value = NULL の場合は、値の取り出しは行なわれず、全体のワード数 (= 4) が関数値として戻る。

## 【エラーコード】

EX\_ADR : アドレス(value)のアクセスは許されていない。  
 EX\_PID : パーツ(pid)は存在していない。

# cset\_val

パーツの値の設定

## 【形式】

W cset\_val(W pid, W size, W \*value)

## 【パラメータ】

W pid パーツID  
 W size value のサイズ  
 W \*value 設定値を格納した配列

## 【リターン値】

0 正常(関数値は設定したワード数)  
 <0 エラー(エラーコード)

## 【解説】

pid  
 で指定したパーツの現在値を  
 \*value  
 で指定した値に変更し、パーツが「表示状態」の時は、再表示を行なう。

size は、設定するデータ数 (ワード数またはハーフワード数) を指定する。設定する現在値の内容、および関数値の意味はパーツのタイプによって異なる。

## テキストボックス / シークレットテキストボックス

- value で指定した領域に格納されている文字列を設定する。size および関数値はハーフワード単位である。設定される文字列は TNULL で終了するか、または size 文字数までであり、最大文字数 (txsize) より大きい場合、後ろは切り捨てられ実際に設定された文字列長 (つまり、TNULL を除いた文字数) が関数値として戻る。
- size 0、または value = NULL の場合は、空文字列の設定となり、関数値として 0 が戻る。

## 数値ボックス

- value で指定した領域に格納されている cv の値を設定する。size および関数値は W 単位であり、整数型の場合は 1、実数型の場合は 2 になる。size が必要なワード数より少ない場合、または value = NULL の場合は、何も設定されず、関数値として 0 が戻る。また size が大きすぎる場合は必要な分のみ設定される。

## シリアルボックス

- value で指定した領域に順番に格納されている各フィールドの cv 値を設定する。size および関数値は W 単位である。size が必要なワード数より少ない場合、その分のみ設定され、実際に設定されたワード数が関数値として戻る。従って、size の値によっては一部のフィールドの値のみの設定も可能である (但し、先頭のフィールドから)。
- size 0、または value = NULL の場合は、何も設定されず、関数値として 0 が戻る。また size が大きすぎる場合は必要な分のみ設定される。

## テキスト / ピクトグラムオルタネート / モーメンタリスイッチ

### スイッチセクタ / スクロールセクタ

- \*value で指定した値を現在値として設定する。size および関数値は W 単位である。関数値としては、1 が戻る。
- size 0、または value = NULL の場合は、何も設定されず、関数値として 0 が戻る。また size が大きすぎる場合は必要な分のみ設定される。
- スクロールセクタの場合、以下の動作となる。
  - size = 1 の場合は、value[0] の値を現在値として設定し、設定した現在値が見えるように最低限のスクロールを行なう。また、P\_NOSEL で現在値が 0 の場合は、インジケータの消去のみを行なう。
  - size = 2 の場合は、value[0] の値を現在値として設定し、value[1] で指定した位置に設定した現在値が見えるようにスクロールを行なう。value[1] で指定する値は、0 が一番上で、表示できる選択肢の数 -1 が一番下となる。value[1] の値がレンジを超えている場合は、それぞれ 0 または最大値と見なされる。また、現在値が 0 の場合、value[1] は、先頭に表示する項目番号を表す。これは、キー等により現在値を変更せずにスクロールだけを行なうような場合、使用する。

## ボリューム

- value で指定した領域に順番に格納されている clo, chi, lo, hi の値を設定する。size および関数値は W 単位である。size が必要なワード数より少ない場合、その分のみ設定さ

れ、実際に設定されたワード数が関数値として戻る。従って、size の値によっては clo, chi のみの設定も可能となる。

- size 0、または value = NULL の場合は、何も設定されず、関数値として 0 が戻る。また size が大きすぎる場合は必要な分のみ設定される。

## 【エラーコード】

EX\_ADR : アドレス(value)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(設定値が範囲外)。  
EX\_PID : パーツ(pid)は存在していない。

# cget\_dat

パーツのデータ取出し

## 【形式】

ERR cget\_dat(W pid, W \*par)

## 【パラメータ】

W pid パーツID  
W \*par パーツデータ格納領域

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

pid で指定したパーツのデータを取り出して、par で指定した領域に格納する。取り出されるデータは、パーツのタイプによって異なる。

|                  |                                   |
|------------------|-----------------------------------|
| テキストボックス         | (W) txsize                        |
| シークレットテキストボックス   | (W) txsize                        |
| 数値ボックス           | (UW) fmt                          |
| シリアルボックス         | (TC*) fmt                         |
| テキストオルタネートスイッチ   | (TC*) name                        |
| テキストモーメンタリスイッチ   | (TC*) name                        |
| ピクトグラムオルタネートスイッチ | (W) onpat(par[0]), offpat(par[1]) |
| ピクトグラムモーメンタリスイッチ | (W) onpat                         |
| スイッチセクタ          | (TC*) name                        |
| スクロールセクタ         | (TC*) name                        |
| ボリューム            | (W) patnum                        |

(TC\*) fmt と (TC\*) name の場合は、ポインタの取出しであり、実際の文字列は直接は得られない。

### 【関数リターン値】

= 0 (E\_OK) : 正常終了  
< 0 : エラー (関数値はエラーコード)

## cset\_dat

パーツのデータ設定

### 【形式】

ERR cset\_dat(W pid, W par)

### 【パラメータ】

W pid パーツID  
W par パーツデータ (PA\_PARTS の場合は W\*)

### 【リターン値】

0 正常  
<0 エラー(エラーコード)

### 【解説】

pid で指定したパーツのデータを、par で指定した値に変更し、パーツが「表示状態」の場合は再表示を行なう。設定されるデータは、パーツのタイプによって異なる。

|                  |                                    |
|------------------|------------------------------------|
| テキストボックス         | (W) txsize                         |
| シークレットテキストボックス   | (W) txsize                         |
| 数値ボックス           | (UW) fmt                           |
| シリアルボックス         | (TC*) fmt                          |
| テキストオルタネートスイッチ   | (TC*) name                         |
| テキストモーメンタリスイッチ   | (TC*) name                         |
| ピクトグラムオルタネートスイッチ | (W*) onpat(par[0]), offpat(par[1]) |
| ピクトグラムモーメンタリスイッチ | (W) onpat                          |
| スイッチセクタ          | (TC*) name                         |
| スクロールセクタ         | (TC*) name                         |
| ボリューム            | (W) patnum                         |

(TC\*) fmt と (TC\*) name の場合は、ポインタの設定である。fmt, name で指定される文字列の内容を直接変更し、fmt, name 自体の値が変化していない場合でも、cset\_dat()を行なわない限り変更の反映は保証されない。

シリアルボックスの fmt を変更しても、項目の値や項目総数は現在のままとなる。このため、現在の項目の値が新しい fmt に適用できない場合や、項目総数が現在と同じでない場合は、EX\_PAR のエラーになり、fmt は変更されない。

### 【関数リターン値】

= 0 (E\_OK) : 正常終了  
< 0 : エラー (関数値はエラーコード)

### 【エラーコード】

EX\_DNUM : データ (onpat, offpat, patnum) はデータマネージャに登録されていない。

EX\_PAR : パラメータが不正である。

EX\_PID : パーツ (pid) は存在していない。

## ccut\_txt

パーツの選択領域の取出し

### 【形式】

W ccut\_txt(W pid, W size, TC \*buff, W cut)

### 【パラメータ】

W pid      パーツID  
W size     buff の文字数  
TC \*buff   文字列取り出し用領域

### 【リターン値】

0      正常 (関数値は選択領域の文字数)  
< 0    エラー (エラーコード)

### 【解説】

pid で指定したテキスト / 数値ボックスの選択領域の文字列を取り出して、buff で指定した領域に格納し、選択領域全体の文字数を関数値として戻す。

size は buff で指定した領域の文字数の指定であり、領域が不足した場合は先頭から (size - 1) 文字のみが格納され、最後に TNULL が格納される。

選択領域がヌルの場合は、buff には空文字列、即ち、\*buff に TNULL が設定され、関数値として 0 が戻る。

cut = 0 の場合は、取り出した選択領域はそのままボックスのデータとして残し、cut != 0 の場合は削除する。



buff = NULL の場合は、格納は行なわないが、選択領域全体の文字数が関数値として戻る。この場合は、cut は無視され、選択領域が削除されることはない。

なお、ボックスが「表示状態」の時は、再表示が行なわれる。

### 【エラーコード】

EX\_ADR : アドレス(buff)のアクセスは許されていない。

EX\_PAR : パラメータが不正である(size が不正)。

EX\_PID : パーツ(pid)は存在していない(テキスト / 数値ボックスではない)。

## cins\_txt

パーツの選択領域の挿入

### 【形式】

W cins\_txt(W pid, PNT pos, TC \*buff)

### 【パラメータ】

W pid      パーツID  
PNT pos    挿入位置  
TC \*buff   挿入文字列

### 【リターン値】

0      正常(関数値は全体の文字数)  
<0    エラー(エラーコード)

### 【解説】

pid で指定したテキスト / 数値ボックスの pos で指定した位置に、buff で指定した領域に格納されている文字列(最後は TNULL) を挿入し、そのボックスを「入力状態」とし、挿入した文字列を新たな選択領域とする。

pos はウィンドウ / パネルの相対座標で指定される。挿入した結果、最大の文字数を超えた場合は、後ろは切り捨てられる。関数値としては挿入後の全体の文字数が戻る。

pos として {0x8000, 0x8000} を指定することで、buff で指定した文字列によって選択領域を置換することができる。これは、メニューによるトレー操作に対応する。

なお、パーツが「表示状態」の時は、再表示が行なわれる。

### 【エラーコード】

EX\_ADR : アドレス(buff)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(挿入テキスト/挿入位置が不正)。  
EX\_PID : パーツ(pid)は存在していない(テキスト/数値ボックスではない)。

## cget\_pos

パーツの位置の取出し

### 【形式】

ERR cget\_pos(W pid, RECT \*r)

### 【パラメータ】

W pid パーツID  
RECT \*r パーツ位置矩形

### 【リターン値】

0 正常  
<0 エラー(エラーコード)

### 【解説】

pid で指定したパーツの現在の表示領域を取り出し、r で指定した領域に格納する。取り出された表示領域は、パーツの属するウィンドウ/パネルの相対座標で示される。

### 【エラーコード】

EX\_ADR : アドレス(r)のアクセスは許されていない。  
EX\_PID : パーツ(pid)は存在していない。

## cset\_pos

パーツの位置の設定

### 【形式】

ERR cset\_pos(W pid, RECT \*r)

### 【パラメータ】

W pid パーツID  
RECT \*r パーツ位置矩形

### 【リターン値】

0 正常  
<0 エラー(エラーコード)

### 【解説】

pid で指定したパーツの表示領域を r で指定した領域に変更する。これは、位置の移動と大きさの両方の変更となる。r は、そのパーツの属するウィンドウ / パネルの相対座標で指定する。

パーツが「表示状態」の場合は、新しい領域に再表示されるが、元の領域の表示に関しては何も行なわない。

### 【エラーコード】

EX\_ADR : アドレス(r)のアクセスは許されていない。  
EX\_PID : パーツ(pid)は存在していない。  
EX\_PAR : パラメータが不正である(指定矩形値が不正)。

## cget\_sts

パーツの状態の取出し

### 【形式】

W cget\_sts(W pid, W \*wid)

### 【パラメータ】

W pid パーツID  
W \*wid ウィンドウID格納領域

### 【リターン値】

0 正常(関数値はパーツのタイプ / 属性 / 状態)  
<0 エラー(エラーコード)

### 【解説】

pid で指定したパーツが属するウィンドウ / パネルIDを取り出し、wid で指定した領域に格納し、パーツのタイプ / 属性 / 状態ワード ([図79 パーツのタイプ / 属性 / 状態](#)参照) を関数値として戻す。

wid=NULL の場合は、ウィンドウ / パネルIDは戻されない。

また、表示属性のビットは常に 0 となっている(登録時のみ有効)。

### 【エラーコード】

EX\_ADR : アドレス(wid)のアクセスは許されていない。

EX\_PID : パーツ(pid)は存在していない。

## cchg\_par

パーツの状態の変更

### 【形式】

ERR cchg\_par(W pid, UW cmd)

### 【パラメータ】

W pid        パーツID  
UW cmd       cmd は以下の4種の状態指定の任意の組合せ (OR) となる。

P\_EMPHAS / P\_NORMAL -- 強調枠 / 通常枠  
P\_NOFRAME / P\_FRAME -- 枠無し / 枠有り  
P\_DISABLE / P\_ENABLE -- 操作禁止 / 操作許可  
P\_INACT / P\_ACT        -- 不能状態 / 可能状態

### 【リターン値】

0        正常  
<0       エラー(エラーコード)

### 【解説】

pid で指定したパーツの状態を、cmd で指定した内容に変更する。

パーツが「表示状態」の時は、再表示が行なわれる。

### 【エラーコード】

EX\_PAR : パラメータが不正である(cmdが不正 : P\_EMPHASとP\_NORMALの組み合わせ等)。

EX\_PID : パーツ(pid)は存在していない。

# cchg\_pwd

ウィンドウ内パーツの状態の変更

## 【形式】

ERR cchg\_pwd(W wid, UW cmd)

## 【パラメータ】

W wid ウィンドウID  
W cmd cchg\_par() の場合と同じ。

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウ / パネルに属する全てのパーツの状態を、cmd で指定した内容に変更する。但し、P\_SBAR = 1 の属性を持つ、ボリュームは対象外となる。

パーツが「表示状態」の時は、再表示が行なわれる。

## 【エラーコード】

EX\_PAR : パラメータが不正である(cmdが不正 : P\_EMPHASとP\_NORMALの組み合わせ等)。  
EX\_WID : ウィンドウ(wid)は存在していない。

# cidl\_par

パーツの定常処理

## 【形式】

ERR cidl\_par(W wid, PNT \*pos)

## 【パラメータ】

W wid ウィンドウID  
PNT \*pos ポインタ位置座標

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

wid で指定したウィンドウ / パネルに属するすべてのパーツに対しての定常処理を行なう。この場合、P\_SBAR = 1 の属性のボリュームも対象となる。指定したウィンドウが、入力受付状態でない場合は、何もしない。

\*pos は、その時点のポインタの wid で指定したウィンドウの相対座標位置であり、ポインタの形状変更で使用される。pos = NULL とすればポインタの形状変更を行なわないが、通常1つでもパーツを有するウィンドウ / パネルは pos != NULL で、この関数をコールする必要がある。

定常処理とは以下に示す内容のものである。

- 「点滅」状態のパーツに対して、点滅の周期に達していた場合に表示を更新して点滅を行なう。
- ボックスパーツにおいて、選択領域の「ちらつき枠」が表示されていた場合、ちらつき処理を行なう。または「文字カーソル」が表示されていた場合、その点滅処理を行なう。
- メニューボタンまたは、「制御」キー押下中でなく、pos が NULL でない場合、pos で示された位置に応じて、ポインタの形状の更新を行なう(「可能状態」、「許可状態」かつ「表示状態」のパーツが対象となる)。

ボリューム / スクロールセレクトのノブ部分の場合  
-- 縦、または横の「移動手」  
テキストボックス / 数値ボックスの選択領域の場合  
-- 斜めの「移動手」  
対象パーツ上で上記以外の場合  
-- 「選択指」  
対象パーツ上にない場合  
-- 変更しない

関数値として以下の意味を持っている

1:

ポインタ形状をパーツマネージャが拘束している場合 (指定位置がパーツ上で、パーツマネージャがポインタ形状を更新しているか、メニューボタンまたは、[命令]キーが押されている)

0:

アプリケーションにポインタ形状の更新を許可している場合

## 【エラーコード】

EX\_ADR : アドレス(pos)のアクセスは許されていない。  
EX\_WID : ウィンドウ(wid)は存在していない。

# cexe\_par

パーツの単イベント実行

## 【形式】

W cexe\_par(W pid, WEVENT \*ev)

## 【パラメータ】

W pid パーツID  
WEVENT \*ev 処理するイベント

## 【リターン値】

0 正常(関数値下位16bit): 01PC SSSS SSSS SSSS

P = 0: 処理の終了  
= 1: 処理の中断  
C = 0: 現在値/状態の変化無し  
= 1: 現在値/状態の変化有り  
S : 各種のコード

<0 エラー(エラーコード)

## 【解説】

pid で指定したパーツの固有の動作をおこなう。\*ev はパーツが処理するウィンドウイベントである。テキストボックスシークレットテキストボックス数値ボックスシリアルボックス (以下、ボックスパーツ) を動作させた場合にもパーツ内でイベント待ちにならないのを除いて、cact\_par と同じ機能を持つ。

ボックスパーツを実行した場合、範囲選択などのドラッグ操作中は cact\_par と同じにイベントが取り出されるが、それ以外ではイベントは取り出されず、ev に EV\_NULL を格納して P\_EVENT を返す (通常は、P\_EVENT で EV\_NULL が設定されることはない)。この場合、アプリケーションは wget\_evt によって新たなウィンドウイベントを取り出し cexe\_par に渡す必要がある (イベントを正しく渡さなかった場合の動作は保証されない)。なお、P\_EVENT が得られた場合で EV\_NULL でなかった場合は、cact\_par と同様にイベントを処理し、ev に EV\_NULL を格納して再度 cexe\_par を呼ぶ必要がある。

## 【エラーコード】

EX\_ADR : アドレス(ev)のアクセスは許されていない。  
EX\_PID : パーツ(pid)は存在していない。

[この章の目次にもどる](#)

[前頁:3.2 メニューマネージャにもどる](#)

[次頁:3.4 パネルマネージャにすすむ](#)



[この章の目次にもどる](#)

[前頁:3.3 パネルマネージャにもどる](#)

[次頁:3.5 トレーマネージャにすすむ](#)

---

## 3.4 パネルマネージャ

### 3.4.1 パネルマネージャの機能

#### 3.4.1.1 概要

パネルマネージャは、外殻の HMI 機能の 1 つとして位置付けられ、ダイアログタイプのパネルの生成 / 削除 / 操作等の機能を提供している。アプリケーションプログラムは、このパネルマネージャを通して容易にダイアログタイプのパネルを使用することができる。

また、システムメッセージパネルの表示機能もパネルマネージャが提供している。

なお、フローティングタイプのパネルに関しては、パネルマネージャでは特に機能を提供せず、アプリケーションが通常のウィンドウとして処理することになる。

なお、今後単にパネルといった場合は、ダイアログタイプのパネルを示すものとする。

本仕様書では、パネルの標準的な形状に関して説明しているが、表示形状の詳細はインプリメントに依存する。

#### 3.4.1.2 パネル

(ダイアログタイプの) パネルは、パラメータの設定や、確認等を求めるパネルであり、その設定 / 操作が終了しないと先の処理に進めない場合に使用されるもので、スクリーン上で通常ウィンドウの前面 / 前面ウィンドウの下に表示され移動 / 変形ができないものである。

パネルは表示された時点で強制的にユーザからの入力受付状態となり、そのパネルで要求している設定や操作を終了した時点でパネルは消滅し、以前に入力受付状態であったウィンドウに入力受付状態は戻る。

パネルは多重にネスタリングされて表示される場合があり、その場合は発生順にオーバーラップされて表示され、入力受付状態も最後に発生したパネルに順次切り換わっていく。パネルの操作を終了した場合は、発生 of 逆順に表示が消えて入力受付状態が戻る。

アプリケーションプログラムはパネルマネージャを使用してパネルを生成し、生成時に得られるパネル ID (>0) を使用して操作等を行なう。

生成したパネルは、基本的に登録したプロセスに依存し、そのプロセスが終了するとパネルも自動的に削除されるが、ある種のパネルは複数プロセスで使用される場合があるため、パネル ID はプロセスグローバルであり、またパネルの定義データはすべてのプロセスからアクセスできる領域にある必要がある。このような複数プロセスで使用されるパネルは、ローカルメモリ領域のデータを参照していないように作成する必要があるが、パネルマネージャは特にこのための保証は行なわない。

パネルマネージャは内部的には、ウィンドウマネージャを使用して特殊なウィンドウとしてのパネルを生成する。従ってパネル ID は、ウィンドウマネージャにより生成されたウィンドウ ID に等しいことになる。

#### 3.4.1.3 システムメッセージパネル

システムメッセージパネルはスクリーンの下部に1行分の大きさで常に表示されている特殊なパネルであり、他のウィンドウ/パネル等で決してオーバーラップされない領域である。

フルスクリーンモードでは、システムメッセージパネルは隠れて見えなくなる。

このパネルでは一切の操作はできず、単にメッセージの表示のみが行なわれる。一度表示されたメッセージは、別のメッセージによりオーバーライトされない限り消えることはない。

システムメッセージパネルはシステムの初期化の時点で自動的に生成されるため、アプリケーションが表示を行なう場合は単に表示用の関数をコールすればよいが、システムメッセージパネルの目的にあった使用法に限定すべきである。

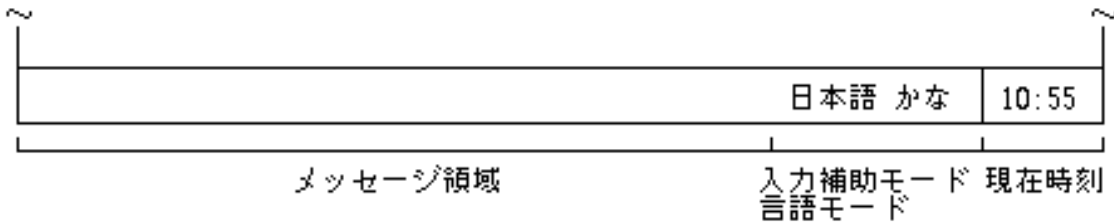


図 96 : システムメッセージパネル

#### 3.4.1.4 パネルの表示形状

パネルは、一般的にウィンドウと同様に下図のように右下に影が付いた境界線により囲まれた矩形領域として表示される。

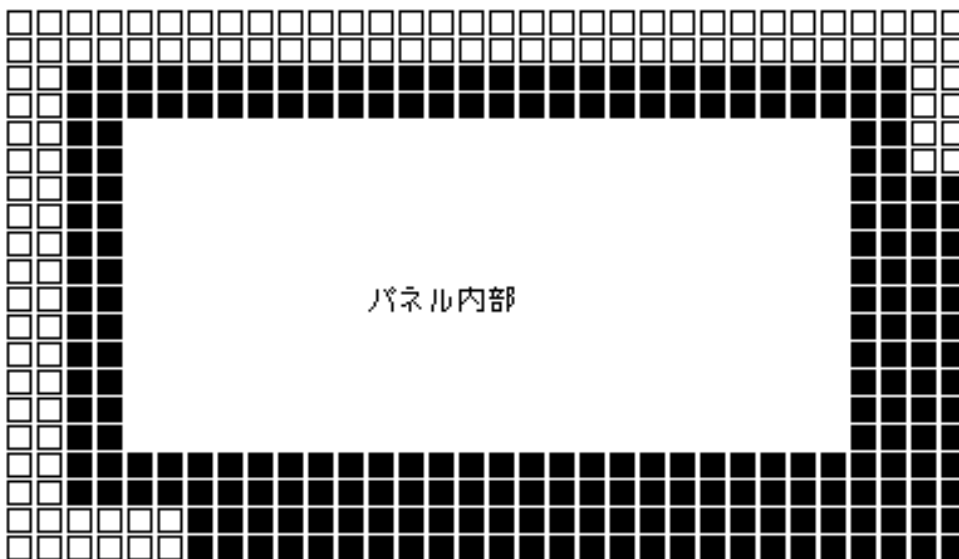


図 97 : パネルの一般的な境界線

パネルの内部には通常、強調用の内枠を表示する。内枠は、パネルの境界線の内側に狭い隙間を開けて描かれる、境界線より太い枠である。なお、境界線のことを外枠とも呼ぶ。

内枠としては任意のパターンが指定可能であるが、通常はパネルの用途に応じて「通常パネル」「警告パネル」「重大警告パネル」の3種類のパターンが使用される。

内枠の内側がパネルの作業領域であり、その表示内容はアプリケーションに依存し、各種コントロールパーツ、図形イメージデータ、文字列の任意の組み合わせとなる。また、通常はパネルの操作を終了するためのモーメンタリスイッチが存在する。



図 98 : パネルの一般的な形状

パネルの表示に関する以下の項目は、パネルの生成時に指定することができる。

- 境界線の幅
- 境界線のパターン
- 内枠の幅
- 内枠のパターン
- 境界線と内枠の間隔
- パネルの背景パターン

### 3.4.1.5 パネルのデータ構造

パネルの全体構造

パネルは以下のデータ構造により定義される。

```
typedef struct panel {
    UW oframe; /* 境界線(外枠)の属性 */
    UW iframe; /* 内枠の属性 */
    W bgpat; /* 背景パターン */
    RECT r; /* 表示領域 */
    W defsw; /* デフォルトスイッチ番号 */
    W nitem; /* 項目数 */
    PNL_ITEM *item; /* 項目配列へのポインタ */
} PANEL;
```

oframe :

パネルの境界線の属性を指定するもので、以下の内容を持つ。境界線の幅は影の部分を含まない幅である。

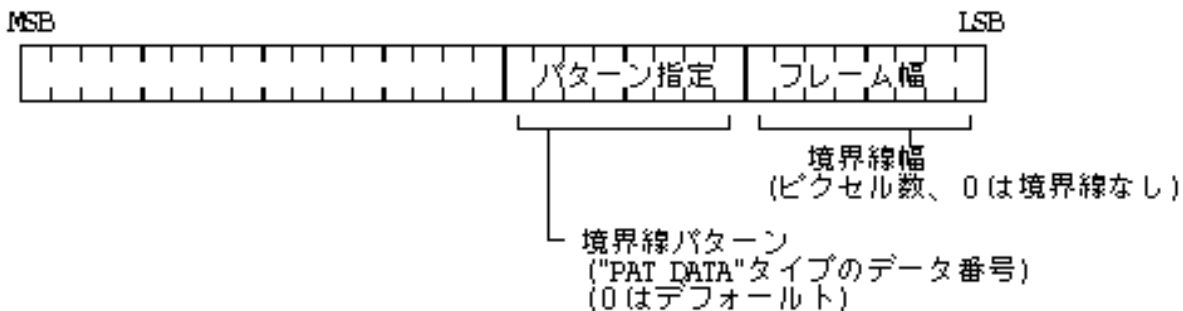


図 99 : パネルの境界線の属性

frame :

パネルの内枠の属性を指定するもので、以下の内容を持つ。

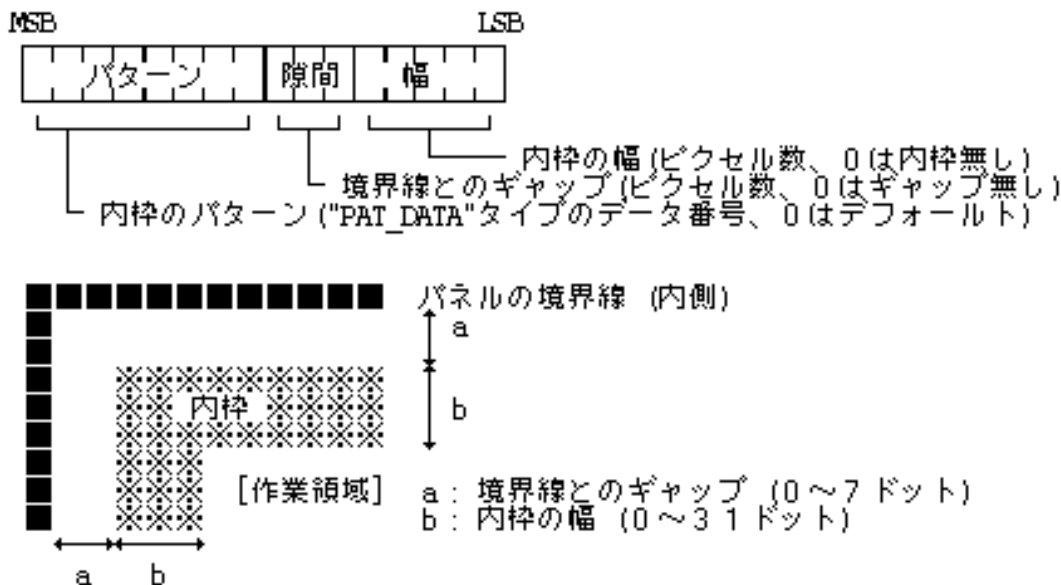


図 100 : パネルの内枠の属性

bgpat :

パネルの背景パターンを指定する、"PAT\_DAT" タイプのデータ番号。0 の場合はデフォルトを意味する。

r :

パネルの境界線を囲む全体の表示矩形領域を絶対座標で指定したものである。

defsw :

デフォルトスイッチとして使用されるモーメンタリスイッチパーツの項目番号を示す。パネルには通常「保存」「承認」等のモーメンタリスイッチが1つ存在し、このスイッチは[改段落 / 改行]キーにより ON となりパネルの処理が終了する。デフォルトのスイッチは太枠で強調表示される。デフォルトスイッチが存在しない場合は0とする。

nitem :

パネル内に表示される項目の数であり、項目配列の要素数を示す。

item :

nitem 個の要素からなる項目配列へのポインタである。  
各項目はこの配列のインデックス+1 (1 ~ nitem) である項目番号で識別される。

パネルの項目の構造

パネルの構成項目 (PNL\_ITEM) は、以下のように定義される。

```
typedef struct pnl_item {
    UW  itype;      /* 項目タイプ */
    UW  info;      /* 各種情報 */
    RECT ir;       /* 表示領域 */
    W   desc;      /* 内部ディスクリプタ */
    W   dnum;      /* データ番号 */
    H   *ptr;      /* データポインタ */
} PNL_ITEM;
```

i type:

項目のタイプを指定する以下のいずれかの値であり、この値はデータマネージャのデータタイプに1対1に対応している。

```
NULL_ITEM 0    -- 空項目
PTR_ITEM   1    -- ポインタイメージ
PICT_ITEM  2    -- ピクトグラムイメージ
PAT_ITEM   3    -- パターンイメージ
BMAP_ITEM  4    -- ビットマップイメージ
           5    -- 予約
TEXT_ITEM  6    -- 文字列
PARTS_ITEM 7    -- コントロールパーツ
ACT_ITEM   0x80 -- 動作項目(PD プレスの通知)
ATR_TEXT   0x20 -- 属性付き文字列
```

ACT\_ITEM, ATR\_TEXT の指定は上記項目タイプに OR で指定される。

ATR\_TEXT 指定は TEXT\_ITEM に対してのみ適用され、その文字列項目が属性データを含んでいることを意味する。

コントロールパーツ以外は表示のみの項目であり基本的に操作はできないが、ACT\_ITEM 指定を行なった場合は、その表示領域で PD のプレスが発生した場合、アプリケーションに通知される。コントロールパーツの場合は、ACT\_ITEM 指定は無関係であり、パーツの操作を行なった場合は常にアプリケーションに通知される。

NULL\_ITEM として指定した領域には何も表示されないが ACT\_ITEM と合わせて、指定した領域内での PD プレスを検出するために使用される。

info:

項目のタイプにより以下の意味を持つ。

- コントロールパーツのオルタネートスイッチの場合：  
"0" の場合は独立したスイッチを意味し、"0" でない場合は、グループスイッチを意味する。  
グループスイッチの場合、そのスイッチが属するグループに対応したビットが1にセットされる。
- その他の場合：  
使用しない(無視される)。

ir:

項目を表示する矩形領域の指定であり、パネルの全体領域(境界線を含む)の左上の点を (0,0) とした座標値で指定される。各項目の表示はこの領域でクリッピングされ、はみ出ることはない。

なお、表示領域が空の場合は、その項目は表示されない。

項目がコントロールパーツの場合は、パーツの定義データの中に表示領域が定義されており、その左上の点を ir の左上の点に一致するように移動したものが実際の表示領域となる。したがって、この場合は、ir の右下の点は使用されない。

desc:

以下の意味を持つ内部ディスクリプタであり、定義時には何も設定する必要はない。

- コントロールパーツ項目の場合：  
登録したパーツID

- データ番号 ( dnum ) で定義した項目の場合：  
共有メモリのアクセスキー
- その他の項目の場合：  
使用しない(0)

dnum:

項目のデータとして、データボックスに定義されているデータを使用する場合、そのデータ番号 (>0) を指定する。項目のタイプがデータタイプとして適用される。dnum = 0 の場合は ptr は無視される。

ptr:

項目のデータを直接メモリ上に定義している場合に、そのポインタを指定する。この場合は、dnum = 0 とする必要がある。

グループスイッチ

複数のオルタネートスイッチをグループ化して排他的動作を行なわせるものをグループスイッチと呼ぶ。即ち、グループ内のあるスイッチが ON 状態となった場合には、同一グループ内で既に ON 状態となっているスイッチは自動的に OFF 状態となる。グループスイッチでは、常に 0 個または 1 個のスイッチのみ ON 状態となる。

グループスイッチは、info データのグループに対応するビットを "1" にセットすることにより指定される。即ち、info データの同一ビットが "1" にセットされているスイッチ群が同一のグループに属することになる。グループは最大32個まであり、1つのスイッチを2つ以上のグループに属させることも可能である。

グループスイッチの例を以下に示す。

スイッチ1のinfo = 0x01  
スイッチ2のinfo = 0x01  
スイッチ3のinfo = 0x10  
スイッチ4のinfo = 0x10  
スイッチ5のinfo = 0x11

- スイッチ1,2,5が1つのグループで、スイッチ3,4,5がもう一つ別のグループとなる。
- スイッチ2をONにすると、スイッチ1,5は共にOFFとなる。
- スイッチ4をONにすると、スイッチ3,5は共にOFFとなる。
- スイッチ5をONにすると、スイッチ1,2,3,4はすべてOFFとなる。

属性付き文字列

ATR\_TEXT 指定の付いた文字列項目は、文字の属性を指定するために、TAD で規定されている以下の文章付箋セグメントを文字列の中を含むことができる。

- フォント指定付箋
- フォント属性指定付箋
- 文字サイズ指定付箋
- 文字カラー指定付箋

上記以外の付箋は無視される。また、属性付き文字列を ATR\_TEXT 指定せずに登録した場合、その表示は保証されない。属性付き文字列は、表示領域の上端に合わせて表示される。

3.4.1.6 パネルの描画環境

パネルは 1 つの独立した描画環境を持ち、パネルの生成時に対応した描画環境が割り当てられ、

その描画環境上でパネルの描画が行なわれる。

この描画環境の内容の詳細はインプリメントに依存したパネル描画のための標準的な設定となっているが、座標系とクリッピング領域は以下のようにになっている。

座標系：

パネルの表示領域の左上の点を(0,0)とした座標系

クリッピング領域：

フレーム長方形 -- パネルの表示領域(境界線を含む)

表示長方形 -- パネルの作業領域(内枠の内部)

ウィンドウと異なり、作業領域の左上の点は(0,0)でない点に注意が必要である。



図 101 : パネルの描画環境

アプリケーションが、パネル上に特殊な描画を行ないたい場合は、パネルに割り当てられている描画環境IDを取り出し、その描画環境に対して描画を行なうことにより可能である。ただし、アプリケーションが、描画環境のパラメータを変更した場合のパネルの表示は保証されなくなるため、変更した場合は必ず元に戻るか、描画環境の複製を作成して使用する必要がある。

#### 3.4.1.7 パネルの動作

パネルは入力受付状態を占有した形で動作し、発生したイベントは基本的にすべて現在表示されているパネルを対象とする。従って、PDプレスによる自動的なウィンドウ/パネルの入力受付状態の切り換えは行なわれず、パネル以外の場所でのPDプレスの操作は通常は無視される。

パネルの動作、即ちイベント処理は、以下に示す2つの方法により行なう。

(1)

アプリケーションが直接イベントを取り出し、その処理を行なう。

この場合、アプリケーションは取り出したイベントの発生場所をチェックし、対応する動作をパーツマネージャ等の関数を使用して直接行なう必要がある。

なお、この場合は、他のウィンドウへの切換えが発生しないようにwget\_evt()をNOMSG指定で実行する必要がある。

(2)

パネルマネージャで用意されている標準パネル動作関数(pact\_pnl)を使用する。

この場合、パネルマネージャがイベントを順次取り出し、対応する処理を行ない、その結果、イベントがアプリケーションに戻される。

通常は、(2)の方法を用いるが、イベントに対する特殊な動作を必要とする場合は、(1)の方法を使用することになる。

## 3.4.2 データ / 定数の定義

### パネルID

```
typedef W          PNID;    /* パネルID */
```

### 項目タイプ

```
#define NULL_ITEM    0    /* 無し */
#define PTR_ITEM     1    /* ポインタ */
#define PICT_ITEM    2    /* ピクトグラム */
#define PAT_ITEM     3    /* パターン */
#define BMAP_ITEM    4    /* ビットマップ */
#define FIG_ITEM     5
#define TEXT_ITEM    6    /* 文字列 (1行) */
#define PARTS_ITEM   7    /* コントロールパーツ */

#define ACT_ITEM     0x80  /* 動作項目(上記項目にORで指定) */
#define BLT_TEXT     0x40
#define ATR_TEXT     0x20  /* 属性付き文字列(上記項目にORで指定) */
```

### パネル項目

```
typedef struct pnl_item {
    UW itype;    /* 項目タイプ */
    UW info;    /* 各種情報 */
    RECT ir;    /* 表示領域 */
    W desc;     /* 内部ディスクリプタ */
    W dnum;     /* データ番号 */
    H *ptr;     /* データポインタ */
} PNL_ITEM;
```

### パネル

```
typedef struct panel {
    UW oframe;  /* 境界線(外枠)の属性 */
    UW iframe;  /* 内枠の属性 */
    W bgpat;    /* 背景パターン */
    RECT r;    /* 表示領域 */
    W defsw;    /* デフォルトスイッチ番号 */
    W nitem;    /* 項目数 */
    PNL_ITEM *item; /* 項目配列へのポインタ */
} PANEL;
```

## 3.4.3 パネルマネージャの関数

ここでは、パネルマネージャがサポートしている各関数の詳細を説明する。これらの関数群は外殻の拡張システムコールとして提供される。



各関数のエラーコードとしては、ここで示した以外にも、核や他の外殻でエラーが検出された場合は、そのエラーコードが直接戻る。

## pcre\_pnl

パネルの生成

### 【形式】

PNID pcre\_pnl(PANEL \*pnl, PNT \*p)

### 【パラメータ】

PANEL \*pnl パネル定義データ  
PNT \*p パネル表示位置

### 【リターン値】

0 正常(パネルID)  
<0 エラー(エラーコード)

### 【解説】

pnl で定義したパネルを生成し、p で指定した位置にパネルの表示を行なう。

パネルはオープンされると入力受付状態を横取りした形で占有し、パネルをオープンしたプロセスがアクティブプロセスとなる。パネルをオープンしたプロセスがアクティブプロセスでなかった場合は、元の入力受付状態のウィンドウに対してEV\_INACTが送られ、EV\_INACTが受信された後にパネルがオープンされる。パネルをオープンしたプロセスが既にアクティブプロセスであった場合は、EV\_INACTは送られない。

パネルをクローズした場合、オープンした時に入力受付状態であったウィンドウに入力受付状態が戻り、EV\_INACTを送った場合は、EV\_SWITCH(W\_SWITCHコマンド)が送られる。

p はパネルの境界線を含む全体の表示領域の左上の点を絶対座標で指定したもので、p = NULL の場合は、パネルの定義データ内に定義された位置に表示される。指定したパネルの定義データは生成後も参照されるので、保存しておかなければいけない。p->x に0x8000を指定すると水平方向にセンタリングされ、p->y に0x8000を指定すると垂直方向にセンタリングされる。

パネルの境界線, 内枠, 背景の各パターンを指定したデータ番号が不正な場合は、デフォルトパターンが適用される。また、デフォルトスイッチ番号が不正な場合は、デフォルトスイッチが無いものとしてパネルを生成する。

関数値としてパネルID(>0)が戻され、以後の操作の際に指定する。

### 【エラーコード】

EX\_ADR : アドレス(pnl, p)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である(パネルの定義データが不正)。

EX\_SAVE : イメージのセーブ領域が不足した(ネスティングしたパネルの場合)。

## popn\_pnl

パネルのデータボックスからの生成

### 【形式】

PNID popn\_pnl(W dnum, PNT \*p)

### 【パラメータ】

W dnum PANEL\_DATA タイプのデータ番号  
PNT \*p パネル表示位置

### 【リターン値】

0 正常(パネルID)  
<0 エラー(エラーコード)

### 【解説】

dnumで指定したデータ番号を持つ、"PANEL\_DATA" タイプのデータで定義したパネルを生成し、pで指定した位置にパネルの表示を行なう。

パネルはオープンされると入力受付状態を横取りした形で占有し、パネルをオープンしたプロセスがアクティブプロセスとなる。パネルをオープンしたプロセスがアクティブプロセスでなかった場合は、元の入力受付状態のウィンドウに対してEV\_INACTが送られ、EV\_INACTが受信された後にパネルがオープンされる。パネルをオープンしたプロセスが既にアクティブプロセスであった場合は、EV\_INACTは送られない。

パネルをクローズした場合、オープンした時に入力受付状態であったウィンドウに入力受付状態が戻り、EV\_INACTを送った場合は、EV\_SWITCH(W\_SWITCH コマンド)が送られる。

pはパネルの境界線を含む全体の表示領域の左上の点を絶対座標で指定したもので、p = NULLの場合は、パネルの定義データ内に定義された位置に表示される。p->xに0x8000を指定すると水平方向にセンタリングされ、p->yに0x8000を指定すると垂直方向にセンタリングされる。

パネルの境界線、内枠、背景の各パターンを指定したデータ番号が不正な場合は、デフォルトパターンが適用される。また、デフォルトスイッチ番号が不正な場合は、デフォルトスイッチが無いものとしてパネルを生成する。

関数値としてパネルID(>0)が戻され、以後の操作の際に指定する。

### 【エラーコード】

EX\_ADR : アドレス(p)のアクセスは許されていない。  
EX\_DNUM : データ(dnum)はデータマネージャに登録されていない。

EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である(パネルの定義データが不正)。  
EX\_SAVE : イメージのセーブ領域が不足した(ネスティングしたパネルの場合)。

## pdel\_pnl

パネルの削除

### 【形式】

ERR pdel\_pnl(W pnid)

### 【パラメータ】

W pnid パネルID

### 【リターン値】

0 正常  
<0 エラー(エラーコード)

### 【解説】

pnidで指定したパネルを削除し、パネルの表示を消去する。消去するパネルは最前面に表示されているパネル、即ち、最も最後に生成したパネルでなくてはならない。

パネルを削除した場合にはパネルに属しているパーツ群も自動的に削除される。また、パネルを生成したプロセスが終了した場合は、パネルは自動的に削除される。

### 【エラーコード】

EX\_PNID : パネル(pnid)は存在していない(最前面のパネルでない)。

## pact\_pnl

パネルの標準動作

### 【形式】

W pact\_pnl(W pnid, EVENT \*ev, W \*itemno)

### 【パラメータ】

W pnid パネルID

EVENT \*ev 動作結果のイベントが格納される  
W \*itemno 動作結果の項目番号が格納される

## 【リターン値】

0 正常(関数値は項目の処理状態コード)  
<0 エラー(エラーコード)

## 【解説】

pnid で指定したパネルの標準動作を行ない、動作結果としてのイベント、および項目番号をそれぞれ、ev, itemno で指定した領域に格納して、操作した項目の状態を関数値として戻す。

この関数により、以下に示す内容の処理が行なわれる。通常はデフォルトスイッチが押されてパネルの処理が完全に終了するまで、pact\_pnl() を繰り返し実行することになる。

(1)

操作許可状態のボックスタイプのパーツが存在し、かつ「入力状態」のボックスが存在しない場合は、最も小さい項目番号のボックスを「入力状態」とし、キー入力を受け付けるようにする。

(2)

(ウィンドウ)イベントを順次取り出し、そのイベントがパネル内に存在するパーツに対して適用される場合、そのパーツに対して cact\_par() を実行する。その結果パーツの状態が変更されなかった場合は (5) の処理を行ない、状態が変更された場合は (3)、(4) の処理を行なう。

(3)

ボックスタイプのパーツの cact\_par() を実行した場合：

- [タブ]、[入力終]、[改段落 / 改行]キーで終了した場合は、cact\_par() の関数値を pact\_pnl() の関数値としてリターンする。この時、\*itemno には操作したパーツの項目番号が設定され、\*ev には EV\_NULL が設定される。この場合、次に pact\_pnl() が呼ばれた場合のボックスの入力状態は、以下のようになる。

[タブ] -- 項目番号順での次のボックスが入力状態となる  
[入力終][改段落 / 改行] -- 同一のボックスが再度入力状態となる

ただし、デフォルトスイッチが定義されている場合は、「改段落 / 改行」キーにより、デフォルトスイッチも動作する。デフォルトスイッチの動作は次に pact\_pnl() が呼ばれたときに行なわれる。デフォルトスイッチの動作が行なわれると、\*itemno にデフォルトスイッチの項目番号を負にしたものが設定される。

- ボックス外の PD のプレスで終了した場合は、まず、cact\_par() の関数値を pact\_pnl() の関数値として返す。この時 \*itemno には、その時入力状態だったボックスパーツの項目番号が設定され、\*ev には EV\_NULL が設定される。そして、次に pact\_pnl が呼ばれた時に、このボックスパーツの終了の契機となったイベントに対する処理が行なわれる。
- メニュー指定で終了した場合、他のイベントによる中断の場合は、(6) と同じ条件で pact\_pnl() からリターンする。ボックスの入力状態は変わらない。

(4)

ボックス以外のパーツの cact\_par() を実行した場合：

- `cact_par()` の関数値を `pact_pnl()` の関数値としてリターンする。この時、`*itemno` には操作したパーツの項目番号が設定され、`*ev` には `EV_NULL` が設定される。
- グループスイッチ内のスイッチが ON になった場合は、今迄 ON であったスイッチは OFF に切り換わった状態となっている。
- デフォルトスイッチは [改段落/改行] キーでも動作し、この場合の項目番号は、負の値 (- 項目番号) となる。
- ボリュームおよびモーメンタリスイッチの場合は、プレス中の処理のために `cact_par()` が中断する可能性があるが、その場合も `cact_par()` の関数値がそのまま戻るため、対応する処理を行なう必要がある。

(5)

ACT\_ITEM が設定された表示項目で PD がプレスされた場合は、プレスされた時点で、`pact_pnl()` からリターンする。この時、`*itemno` にはその項目番号が設定され、`*ev` には得られたイベント (`EV_BUTDWN`) が設定される。関数値は 0 となる。

(6)

(2),(5) に該当しない、ボタン/キー関連のイベントは無視される。ボタン/キー関連以外のイベント、またはメニュー指定のイベントが得られた場合は、`*itemno` に現在「入力状態」であるボックスパーツの項目番号 (無い場合は 0) を、`*ev` には得られたイベントを設定して戻る。関数値は以下の値となる (`cact_par()` の値と同一)。

メニュー指定 : `P_MENU` (0x6010)  
 他のイベント : `P_EVENT` (0x6020)

なお、メニュー指定とは、以下のいずれかの場合を示す。

- (1) PD のメニューボタンを押した状態でのクリック
- (2) 「命令」キーを押した状態でのプレス
- (3) 「命令」キーと別のキーの同時押し(キーマクロ)
- (4) メニュー起動ボタンでのクリック(タッチパネルの場合)

なお、イベントとして `EV_REQUEST` の `W_REDISP` が返った場合は、`wid` で指定されたパネルやウィンドウの再表示をおこなう必要がある。パネルの再表示の場合、パネル構造体で定義した部分は再描画がおこなわれているので、アプリケーションで描画した部分があれば再表示することになる。なお、アクティブでないパネルや、ウィンドウに対しても再表示要求が発生することに注意が必要である。

## 【エラーコード】

EX\_ADR : アドレス(`ev, itemno`)のアクセスは許されていない。  
 EX\_PNID : パネル(`pnid`)は存在していない。

## pget\_itm

パネル項目の取出し

## 【形式】

W `pget_itm(W pnid, W itemno, PNL_ITEM *item)`

## 【パラメータ】

W pnid パネルID  
W itemno 項目番号  
PNL\_ITEM \*item パネル項目データ

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

pnid で指定したパネル内の itemno で指定した項目の定義データを取り出し、item で指定した領域に格納する。格納される内容は定義した内容と同一であるが、内部ディスクリプタ (item->desc) には、以下の値が格納され、関数値として、この内部ディスクリプタの値が戻る。

コントロールパーツの場合： -- 登録されたパーツID  
コントロールパーツ以外の場合： -- 0

item = NULL の場合は項目の定義データの取り出しは行なわれませんが、関数値として上記の内部ディスクリプタの値が戻る。

## 【エラーコード】

EX\_ADR : アドレス(item)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(itemnoが不正)。  
EX\_PNID : パネル(pnid)は存在していない。

## pset\_itm

パネル項目の設定

## 【形式】

W pset\_itm(W pnid, W itemno, PNL\_ITEM \*item)

## 【パラメータ】

W pnid パネルID  
W itemno 項目番号  
PNL\_ITEM \*item パネル項目データ

## 【リターン値】

0 正常(関数値は項目の内部ディスクリプタ)  
<0 エラー(エラーコード)

### 【解説】

pnid で指定したパネル内の itemno で指定した項目の定義データを item で指定した領域の内容に変更し、パネルの再表示を行なう。

item = NULL の場合は再設定を行わず、指定した項目の再表示のみを行なう。特に pset\_itm(pnid, -1, NULL) の場合は、すべての項目の再表示を行う。

関数値として、項目定義データ設定後の以下の内部ディスクリプタの値が戻る。

コントロールパーツの場合： -- 登録されたパーツID  
コントロールパーツ以外の場合： -- 0

## psel\_box

パネルの入力ボックスの選択

### 【形式】

W psel\_box(W pnid, W itemno)

### 【パラメータ】

W pnid パネルID  
W itemno 項目番号

### 【リターン値】

0 正常(関数値は入力状態のボックスの項目番号(該当なしの場合は0))  
<0 エラー(エラーコード)

### 【解説】

pnid で指定したパネル内で、itemno で指定した項目番号を持つボックスパーツを「入力状態」のボックスとして選択し、そのボックスパーツの項目番号を関数値として戻す。「入力状態」として選択したボックスパーツが、pact\_pnl() を実行した場合にキー入力を受け付けることになる。

itemno = 0 の場合は、ボックスパーツの入力状態の変更は行なわれず、単に現在入力状態であるボックスパーツの項目番号を関数値として戻す。該当するものが無い場合は、関数値として0が戻る。

itemno で指定した項目が操作許可状態のボックスパーツでない場合は、EX\_PAR のエラーとなる。

## 【エラーコード】

EX\_PAR : パラメータが不正である(itemno が不正,操作許可のボックスパーツでない)。  
EX\_PNID : パネル(pnid)は存在していない。

## pget\_gid

描画環境IDの取出し

## 【形式】

GID pget\_gid(W pnid)

## 【パラメータ】

W pnid パネルID

## 【リターン値】

0 正常(関数値はパネルの描画環境ID)  
<0 エラー(エラーコード)

## 【解説】

pnid で指定したパネルで使用している描画環境IDを取り出し、関数値として戻す。

アプリケーションがこの描画環境IDを用いてパネル上に直接描画する場合は、基本的に必要なパラメータを再設定し、描画後は元の設定に戻しておく必要がある。パラメータを変更したままの状態パネルの表示 / 操作等を行なった場合の動作は保証されないので注意が必要である。

## 【エラーコード】

EX\_PNID : パネル(pnid)は存在していない。

## pact\_err

標準エラー警告パネル処理

## 【形式】

W pact\_err(W kind, TC \*arg1, TC \*arg2, TC \*arg3)

## 【パラメータ】



W kind PANEL\_DATA タイプのデータ番号  
TC \*arg1 文字列1  
TC \*arg2 文字列2  
TC \*arg3 文字列3

### 【リターン値】

0 正常(関数値はスイッチの種別(デフォルトスイッチは0))  
<0 エラー(エラーコード)

### 【解説】

kind で指定した "PANEL\_DATA" タイプのデータ番号をもつ標準エラー警告パネルを表示する。  
kind < 0 の場合は、- kind をデータ番号とする。

標準エラー警告パネルは、データ番号が 1000 以上のため、kind 1000 または kind -1000  
でなくてはならない。

パネルの定義データの先頭の 3 つまでの項目が文字列項目の場合は、順番に、arg1 ~ arg3 で指定  
した文字列項目に置き換えられる。

例: 項目1 文字列項目 -- arg1 に置き換えられる。  
項目2 文字列項目 -- arg2 に置き換えられる。  
項目3 文字列項目でない -- arg3 は無視される。

例: 項目1 文字列項目 -- arg1 に置き換えられる。  
項目2 文字列項目でない -- arg2, arg3 は無視される。

デフォルトスイッチ項目は必ず存在し、デフォルトスイッチから連続した n 個のモーメンタ  
リスイッチ項目 (ピクトグラムモーメンタリも含む)のいずれかのプレスで処理を終了し、(プレス  
されたモーメンタリスイッチ項目番号) - (デフォルトスイッチ項目番号) の値 (0 ~ n-1) を関  
数値として戻す。

例: デフォルトスイッチ項目番号を 4 とし、項目 7 までが、モーメンタリスイッチの  
場合、4 ~ 7 のスイッチのプレスで処理を終了し、関数値として 0 ~ 3 を戻す。

kind で指定した番号の "PANEL\_DATA" タイプのデータが登録されていない場合、およびデフォ  
ルトスイッチ項目が規定されていない場合は、EX\_PAR のエラーとなる。

また、デフォルトスイッチ項目番号より小さい項目番号の PARTS\_ITEM 項目が存在する場合は、  
EX\_PAR のエラーとなる。

### 【エラーコード】

EX\_ADR : アドレス(arg1 ~ 3)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(-1000 < kind < 1000、パネルデータが登録され  
ていない、デフォルトスイッチが未定義、デフォルトスイッチ項目  
番号より小さい番号のパーツ項目が存在する)。  
EX\_SAVE : イメージのセーブ領域が不足した  
(kind > 0 でネスティングしたパネルの場合)。

## pdsp\_msg

システムメッセージパネルの表示

### 【形式】

W pdsp\_msg(TC \*msg)

### 【パラメータ】

TC \*msg 文字列

### 【リターン値】

0 正常(関数値は表示文字数)  
<0 エラー(エラーコード)

### 【解説】

システムメッセージパネルに現在表示されているメッセージを消去し、msg で指定したメッセージを新たに表示する。msg = NULL の場合は、単に現在の表示の消去を行なう。

メッセージは、システムのデフォルトフォント、通常体、デフォルトカラーで表示され、領域をはみでた部分はクリップされる。

関数値としてはクリップされずに実際に表示された文字数が戻る。

### 【エラーコード】

EX\_ADR : アドレス(msg)のアクセスは許されていない。

## pdsp\_tim

システムメッセージパネルの時刻表示

### 【形式】

ERR pdsp\_tim(UW stat)

### 【パラメータ】

UW stat メタキー状態

## 【リターン値】

0 正常  
< 0 エラー(エラーコード)

## 【解説】

statで指定したメタキー状態に対応した言語モード、入力補助モード、および現在時刻をシステムメッセージパネルに表示する。実際には、現在表示している言語モード、入力補助モード、現在時刻と異なっている場合のみ表示の更新を行なう。

この関数は、メタキー状態が変更された場合、および適当な周期でウィンドウマネージャにより実行されるため、通常のアプリケーションでは実行する必要はない。

## 【エラーコード】

pact\_msg

システムメッセージパネルのイベント処理

## 【形式】

ERR pact\_msg(WEVENT\* ev)

## 【パラメータ】

WEVENT \*ev ウィンドウイベント

## 【リターン値】

0 正常  
< 0 エラー(エラーコード)

## 【解説】

システムメッセージパネルに対して発生したイベント ev に応じて、システムメッセージパネルの表示の更新を行なう。NULLイベントや再表示イベントを受け付ける。関数値として、処理したイベントのタイプが返る。

このシステムコールは、ウィンドウマネージャによって自動的に実行されるため、通常のアプリケーションでは実行する必要はない。

## 【エラーコード】

EX\_ADR : アドレス(ev)のアクセスは許されていない。

---

[この章の目次にもどる](#)

[前頁:3.3 パーツマネージャにもどる](#)

[次頁:3.5 トレーマネージャにすすむ](#)

[この章の目次にもどる](#)

[前頁:3.4 パネルマネージャにもどる](#)

[次頁:3.6 データマネージャにすすむ](#)

---

## 3.5 トレーマネージャ

### 3.5.1 トレーマネージャの機能

#### 3.5.1.1 概要

トレーマネージャは外殻の1つとして位置付けられ、基本的なデータ移動 / 複写の操作である「切り取る」、「写し取る」、「貼り込む」および「取り込む」で使用されるトレーの管理を行ない、トレー内のデータを操作する機能を提供するものである。

また、ドラッグによるデータ移動 / 複写等のために一時的に使用される「一時トレー」の管理も行ない、その操作機能を提供している。

トレーマネージャは単にトレーに格納されるデータ領域の管理を行ない、その内容に関しては一切関知しない。

アプリケーションは、常にユーザからの明示的な操作によってのみトレーの操作を行なわなくてはならない。すなわち、ユーザの操作とは無関係にアプリケーション間のデータ交換等にトレーを使用してはいけない。

#### 3.5.1.2 トレー

トレーは、各ユーザ毎に1つ存在する特殊な「データ格納容器」であり、「切り取る」または「写し取る」の操作のディスティネーションとしてデータが格納され、「貼り込む」または「取り込む」の操作のソースとしてデータが取り出される容器である。

「切り取る / 取り込む」の操作ではソースデータは失われ(即ち、移動)、「写し取る / 貼り込む」の操作では、ソースデータは保存される(即ち、複写)。

トレーはスタック構造を持ち、基本的に最後に格納したデータが最初に取り出されるが、その順番を変更したり任意の位置のデータを取り出すことも可能である。データは、先頭(最後に格納されたデータ)を"1"とした連続番号であるデータ位置により識別される。

トレーに格納されたデータは、「データの削除」操作を行なうか、またはトレーの最大容量をオーバーして溢れた場合に自動的に消滅する。トレーの最大容量は、インプリメントに依存するが、原則的には最後に格納されたデータから遡って、少なくとも5個のデータは保証されるものとする(通常は10個程度)。

なお、トレーに格納されるデータの大きさは基本的に任意であるが、小さいサイズのデータを効率的に取り扱うことを前提としている。

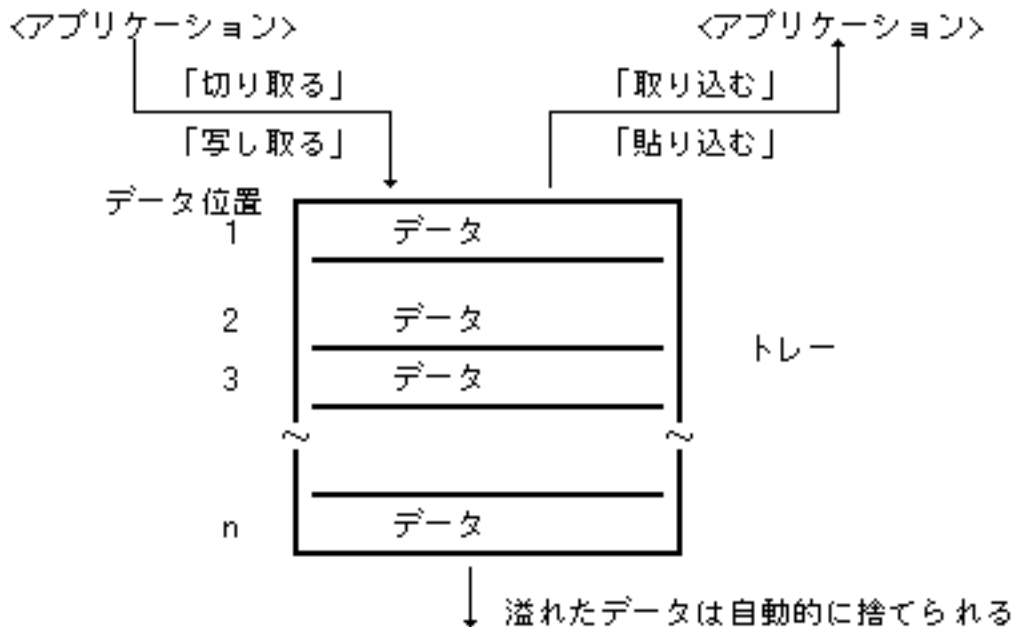


図 102: トレー

一時トレーは、ドラッグによる移動 / 複写等の一時的なデータの交換に使用される特殊な格納容器であり、1つのデータの格納 / 取り出しのみを行なう。従って、新たなデータを格納すると以前のデータは消滅する。

一時トレーは各ユーザ毎ではなくシステムに1つのみ用意されており、常にオープンされて使用できる状態となっている。システムの立上げ時には、一時トレーは空の状態になっている。

### 3.5.1.3 トレーの格納データ

アプリケーションは、トレーを利用したデータ互換を実現するために、以下に述べる形式でトレーにデータを格納する必要がある。

なお、トレーマネージャではレコードの内容、レコードタイプの正当性等に関しては一切関知しない。

トレーおよび一時トレーに格納される1つのデータ単位は、TAD データ構成の形式であり、TAD で規定されているセグメントの列となる。TAD データ構成をトレーおよび一時トレーに格納する場合は、原則的に1つの TAD セグメントを1つのトレーレコードとしたトレーレコードの列として格納する。ただし、連続した図形描画セグメント、および連続した文字コードと文章付箋セグメントは、それぞれ1つのトレーレコードとすることができる。また、1つの TAD セグメントを複数の連続したトレーレコードとして格納することもできる。

1つのトレーレコードは、以下の構造体により定義される。

```
typedef struct {
    W    id;      /* レコードタイプ( > 0 ) */
    W    len;     /* データのバイト長 */
    B    *dt;     /* データ本体へのポインタ */
} TRAYREC;
```

レコードタイプは、TAD で定義されているセグメントIDそのものであり、セグメントに対応した 0x80 ~ 0xFD の値となる(内容に関しては、「[第3章 TAD詳細仕様書](#)」を参照のこと)。

ただし、レコードタイプとしては、TAD のセグメントIDに加えて以下に示すものが追加される。

仮身レコード (TR\_VOBJ (=0x1)) :

仮身を示すレコードであり、リンクレコード、および仮身セグメントが続いた以下の形式を持つ。トレーではTADで定義されている仮身セグメント単体では使用せずに、この仮身レコードを使用する。(詳細に関しては「[3.8 実身 / 仮身マネージャ](#)」を参照のこと。)

```
typedef struct {
    VLINK    vlnk;    -- リンクレコード
    VOBJSEG  vseg;    -- 仮身セグメント
} TR_VOBJREC;
```

文章レコード (TR\_TEXT (=0x2)) :

連続した文字コードと文章付箋セグメントの列から構成されるレコード

図形レコード (TR\_FIG (=0x3)) :

連続した図形描画セグメントの列から構成されるレコード

さらに1つのTADセグメントを連続した複数のトレーレコードに分割した場合は、以下の接続フラグがレコードタイプにセットされる。

接続フラグ (TR\_CONT (=0x4000)):

分割したトレーレコードの最後のレコード以外のすべてのレコードのレコードタイプに接続フラグがセットされる。

例：1つの画像セグメントを複数のトレーレコードに分割した場合のレコードタイプは、以下のようになる。

```
先頭のレコード          : TS_IMAGE + TR_CONT (= 0x40E5)
2番目のレコード         : TS_IMAGE + TR_CONT (= 0x40E5)
      :
      :
(最後 - 1)番目のレコード : TS_IMAGE + TR_CONT (= 0x40E5)
最後のレコード          : TS_IMAGE (= 0xE5)
```

データ本体は、TAD のセグメントから、先頭のセグメントID、およびバイト長を除いたデータ本体部分となる。ただし、文章レコード(TR\_TEXT)と図形レコード(TR\_FIG)の場合は、先頭のセグメントID、およびバイト長を含んだ完全なTADセグメントの列となる。

TADデータ構成は、「文章データ」と「図形データ」のいずれかであるため、トレーレコードの先頭には、「管理情報セグメント」に続いて「文章開始セグメント」か「図形開始セグメント」のいずれかがくることになり、最後のレコードは「文章終了セグメント」か「図形終了セグメント」となる。通常は最後の「文章 / 図形終了セグメント」は省略される。また、最初の「管理情報セグメント」も省略可能である。

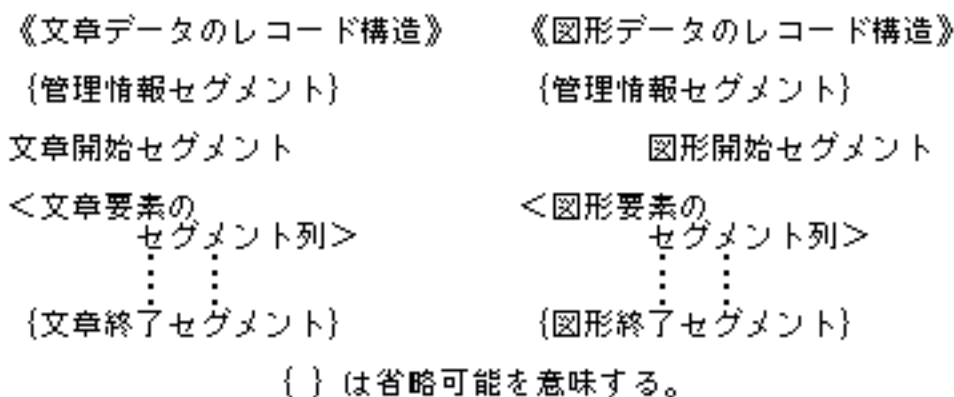


図 103: トレーレコードの構造

トレーおよび一時トレーにデータを格納する場合は、TRAYREC の配列の先頭アドレスと、トレーレコードの数(配列の要素数)を指定する。

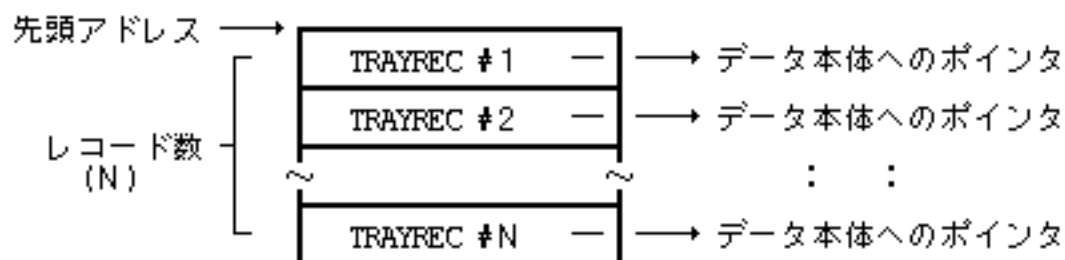


図 104: トレー/一時トレーに格納するデータ指定

トレーおよび一時トレーからデータを取り出す場合は、以下に示す 3 種類の方法を取ることができる。

(1) 全体取り出し

格納されているデータ全体の取り出しであり、取り出し用バッファの先頭に TRAYREC の配列が入り、その直後にレコードデータ本体が入る形式となる。

(2) ヘッダ取り出し

取り出し用バッファには、TRAYREC の配列のみが入り、対応するレコードデータ本体は取り出されない。この場合、TRAYREC 内のデータ本体へのポインタ (dt) は意味を持たない。

(3) 指定レコードデータ取り出し

取り出し用バッファには、指定したレコードのレコードデータ本体のみが取り出される。レコードの指定は先頭のレコードは"1"とした TRAYREC 配列のインデックス番号で行なう。この場合、指定したレコードのレコードタイプも得られる。



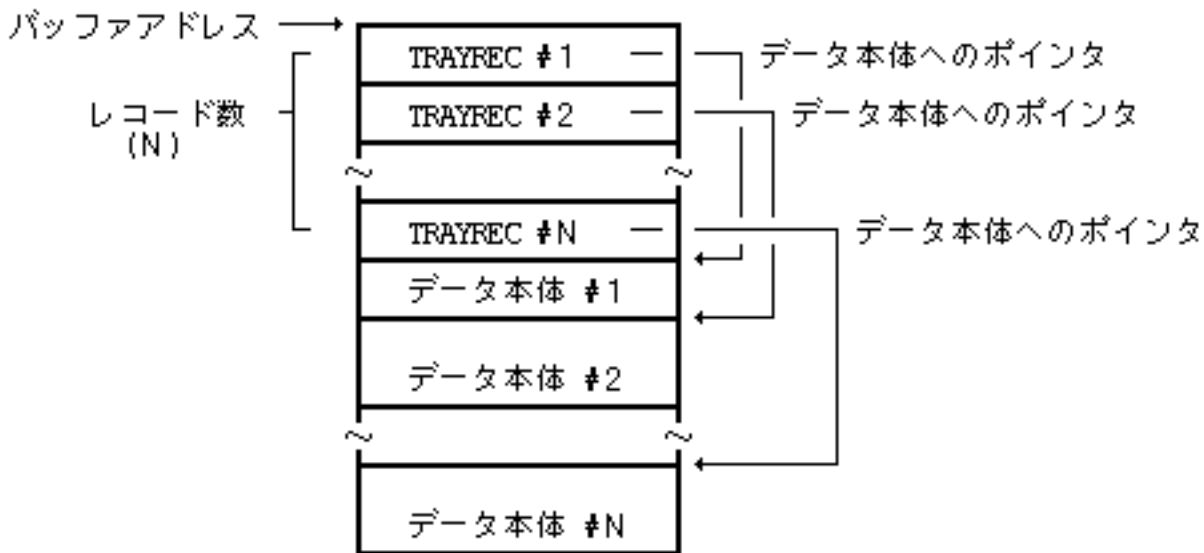


図 105: トレー / 一時トレーから取り出すデータ形式((1)全体取り出し)

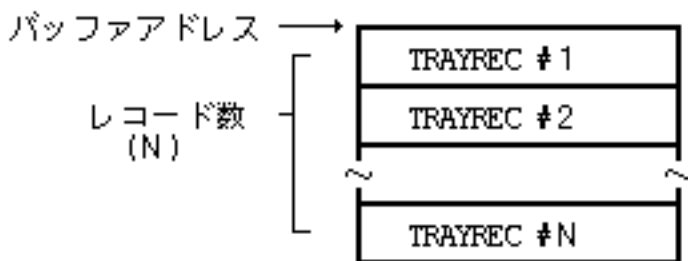


図 106: トレー / 一時トレーから取り出すデータ形式((2)ヘッダ取り出し)

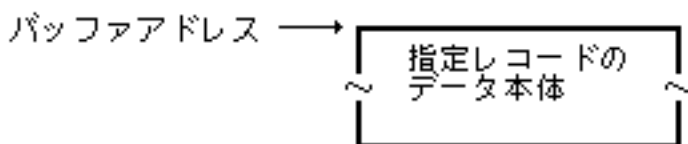


図 107: トレー / 一時トレーから取り出すデータ形式((3)指定レコードデータ取り出し)

また、トレーに格納したデータには最大 12 文字 (TCODE) の任意の名称を格納時に設定することが可能である。この名称はデータの内容を説明するためにアプリケーションプログラムが設定する。なお、一時トレーに格納するデータに対しては、名称をつけることはできない。

#### 3.5.1.4 トレーの操作

トレーは、ユーザ毎に1つ用意される。通常はユーザの初期プロセスによりオープンされ、そのユーザのプロセスはそのトレーを常に対象とする。トレーはユーザの使用が終了した時点でクローズされ、その内容は消える。

トレーはスタック構造を持ち、スタックの先頭にデータが格納され、スタックの先頭から

データが取り出されるが、取り出す場合は任意の位置のデータを取り出すことも可能である。

トレー内のデータには、「選択状態」が定義され、トレー内のどのデータが選択され処理の対象となっているかを示す。トレーは空でない限り、必ず「選択状態」のデータが1つ存在する。

トレーに対する基本操作として以下のものが用意されている。

- データをスタックの先頭に格納する。格納したデータが選択状態となる。
- 選択されているデータを複写する。
- 選択されているデータを削除する。直後のデータが選択状態となる。
- 任意のデータを選択する。
- 選択されたデータを任意の位置に移動する。

一時トレーはスタック構造を持たず、ただ1つのデータのみ格納可能である。従って一時トレーに対する操作は「格納」と「取り出し」のみとなる。

## 3.5.2 データ / 定数の定義

### トレーレコード

```
#define TR_VOBJ 0x1      /* 仮身レコード */
#define TR_TEXT 0x2     /* 文章レコード */
#define TR_FIG  0x3     /* 図形レコード */
#define TR_CONT 0x4000 /* 接続フラグ */
```

仮身レコード (TR\_VOBJ (=0x1)) :

仮身を示すレコードであり、リンクレコード、および仮身セグメントが続いた TR\_VOBJRECの形式を持つ。トレーではTADで定義されている仮身セグメント単体では使用せずに、この仮身レコードを使用する。

文章レコード (TR\_TEXT (=0x2)) :

連続した文字コードと文章付箋セグメントの列から構成されるレコード

図形レコード (TR\_FIG (=0x3)) :

連続した図形描画セグメントの列から構成されるレコード

接続フラグ (TR\_CONT (=0x4000)):

分割したトレーレコードの最後のレコード以外のすべてのレコードのレコードタイプに接続フラグがセットされる。

他のレコードタイプ (TS\_xxx)は、TADセグメントとして定義される。

### トレーレコードタイプ

```
typedef struct {
    W    id;      /* レコードタイプ ( > 0 ) */
    W    len;     /* データのバイト長 */
    B    *dt;     /* データ本体へのポインタ */
}
```

```
} TRAYREC;
```

## 仮身レコードタイプ

```
typedef struct {  
    TC  fs_name[20]; /* ファイルシステム名 */  
    UH  f_id;        /* ファイルID */  
    UH  attr;        /* 仮身タイプ / 属性 */  
    UH  rel;         /* 続柄インデックス */  
    UH  appl[3];     /* アプリケーションID */  
} VLINK;
```

```
typedef struct {  
    VLINK  vlnk;      /* リンクレコード */  
    VOBSEG vseg;     /* 仮身セグメント */  
} TR_VOBJREC;
```

データ本体は、TAD のセグメントから、先頭のセグメントID、およびバイト長を除いたデータ本体部分となる。ただし、文章レコード(TR\_TEXT)と図形レコード(TR\_FIG)の場合は、先頭のセグメントID、およびバイト長を含んだ完全なTADセグメントの列となる。

TAD データ構成は、「文章データ」と「図形データ」のいずれかであるため、トレーレコードの先頭には、「管理情報セグメント」に続いて「文章開始セグメント」か「図形開始セグメント」のいずれかがくることになり、最後のレコードは「文章終了セグメント」か「図形終了セグメント」となる。通常は最後の「文章 / 図形終了セグメント」は省略される。また、最初の「管理情報セグメント」も省略可能である。

### 《文章データのレコード構造》

{管理情報セグメント}

文章開始セグメント

<文章要素の  
セグメント列>

∴  
∴  
∴

{文章終了セグメント}

### 《図形データのレコード構造》

{管理情報セグメント}

図形開始セグメント

<図形要素の  
セグメント列>

∴  
∴  
∴

{図形終了セグメント}

{ }は省略可能を意味する。

トレーおよび一時トレーにデータを格納する場合は、TRAYREC の配列の先頭アドレスと、トレーレコードの数(配列の要素数)を指定する。

トレーおよび一時トレーからデータを取り出す場合は、以下に示す 3 種類の方法を取ることができる。

#### (1) 全体取り出し

格納されているデータ全体の取り出しであり、取り出し用バッファの先頭に TRAYREC

の配列が入り、その直後にレコードデータ本体が入る形式となる。

## (2) ヘッダ取り出し

取り出し用バッファには、TRAYREC の配列のみが入り、対応するレコードデータ本体は取り出されない。この場合、TRAYREC 内のデータ本体へのポインタ(dt)は意味を持たない。

## (3) 指定レコードデータ取り出し

取り出し用バッファには、指定したレコードのレコードデータ本体のみが取り出される。レコードの指定は先頭のレコードは"1"とした TRAYREC 配列のインデックス番号で行なう。この場合、指定したレコードのレコードタイプも得られる。

また、トレーに格納したデータには最大12文字(TC)の任意の名称を格納時に設定することが可能である。この名称はデータの内容を説明するためにアプリケーションプログラムが設定する。なお、一時トレーに格納するデータに対しては、名称をつけることはできない。

トレーは、ユーザ毎に1つ用意される。通常はユーザの初期プロセスによりオープンされ、そのユーザのプロセスはそのトレーを常に対象とする。トレーはユーザの使用が終了した時点でクローズされ、その内容は消える。

トレーはスタック構造を持ち、スタックの先頭にデータが格納され、スタックの先頭からデータが取り出されるが、取り出す場合は任意の位置のデータを取り出すことも可能である。

トレー内のデータには、「選択状態」が定義され、トレー内のどのデータが選択され処理の対象となっているかを示す。トレーは空でない限り、必ず「選択状態」のデータが1つ存在する。

トレーに対する基本操作として以下のものが用意されている。

- データをスタックの先頭に格納する。格納したデータが選択状態となる。
- 選択されているデータを複写する。
- 選択されているデータを削除する。直後のデータが選択状態となる。
- 任意のデータを選択する。
- 選択されたデータを任意の位置に移動する。

一時トレーはスタック構造を持たず、ただ1つのデータのみ格納可能である。従って一時トレーに対する操作は「格納」と「取り出し」のみとなる。

### 3.5.3 トレーマネージャの関数

topn\_tra

トレーのオープン

【形式】

ERR topn\_tra(void)

## 【パラメータ】

なし

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

自プロセスのユーザに対するトレーをオープンし、以後のトレーに関する操作を可能とする。この時、トレーは空の状態になっている。

既にトレーがオープンされている場合は何もしない。

この関数は、通常ユーザの初期プロセスによりそのユーザのセッションが開始する際にコールされ、一般のアプリケーションでは使用しない。

## 【エラーコード】

EX\_NOSPC : システムのメモリ領域が不足した。

**tcls\_tra**

トレーのクローズ

## 【形式】

ERR tcls\_tra(void)

## 【パラメータ】

なし

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

自プロセスのユーザに対するトレーをクローズしその内容を消す。

この関数は、通常ユーザの初期プロセスによりそのユーザのセッションが終了する際にコールされ、一般のアプリケーションでは使用しない。

### 【エラーコード】

EX\_TRAY : トレーはオープンされていない。

tpsh\_dat

トレーへのデータ格納

### 【形式】

W tpsh\_dat(TRAYREC \*data, W nrec, TC \*name)

### 【パラメータ】

|         |       |             |
|---------|-------|-------------|
| TRAYREC | *data | トレーに格納するデータ |
| W       | nrec  | レコード数       |
| TC      | *name | データに付ける名前   |

### 【リターン値】

0 正常(関数値は格納したレコード数)  
<0 エラー(エラーコード)

### 【解説】

トレーの先頭に data で指定した nrec 個のデータレコードを格納し、そのデータを「選択状態」とする。関数値として格納したレコード数が戻る。

name はデータに対して付ける任意の名称であり、先頭の12文字(TC)までが有効となる。名称を付けない場合は、name = NULL とする。

データの格納により、トレーに溢れが生じた場合は、最後のデータは自動的に消滅する。

### 【エラーコード】

EX\_ADR : アドレス(data,name)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した(データのサイズが大きすぎる)。  
EX\_PAR : パラメータが不正である(nrec 0、トレーレコードの内容が不正)。

EX\_TRAY : トレーはオープンされていない。

## tpop\_dat

トレーからのデータ取り出し

### 【形式】

W tpop\_dat(TRAYREC \*data, W size, W \*a\_size, W rec, TC \*name)

### 【パラメータ】

|         |        |                              |
|---------|--------|------------------------------|
| TRAYREC | *data  | トレーから取り出すデータ格納場所             |
| W       | size   | data のバイト数                   |
| W       | a_size | 全体のバイト数                      |
| W       | rec    | 取り出し方法                       |
|         | < 0 :  | 全体取り出し                       |
|         | = 0 :  | ヘッダ取り出し                      |
|         | > 0 :  | 指定レコードデータ取り出し(rec で指定したレコード) |
| TC      | *name  | データ名称の格納場所                   |

### 【リターン値】

0 正常(関数値はデータのレコード数 / レコードタイプ)  
<0 エラー(エラーコード)

### 【解説】

トレー内の選択されているデータを、rec の指定に従って取り出し、data で指定した size バイトの領域に格納する。

格納すべきデータのサイズより size が小さい場合は、size 分だけ格納され、EX\_PAR のエラーとなる。data = NULL の場合は、size の値は無視され、データは一切格納されない。いずれの場合も \*a\_size には格納すべきデータの全体のバイトサイズが戻される。このため例えば、rec < 0, data = NULL とすることにより、格納されているデータ全体のサイズを知ることができる。

なお、データを取り出した後もトレー内のデータは削除されずに残っており、選択状態のままとなる。name はデータに付けられた名称を格納する領域へのポインタであり、12 + 1文字(TC)の領域である必要がある(格納された名称の最後には TNULL が付けられる)。name = NULL の場合は、名称は格納されない。

関数値として、全体取り出し / ヘッダ取り出し (rec = 0) の時は、データの総レコード数が戻り、指定レコード取り出し (rec > 0) の時は、取り出したレコードのレコードタイプが戻る。存在しないレコード番号を指定した場合は、EX\_PAR のエラーとなり、\*a\_size には

"0" が戻される。

トレイが空の場合は、取り出し方法に無関係に、\*a\_size には "0" が戻され、関数値は "0" となる。

### 【エラーコード】

EX\_ADR : アドレス(data,a\_size,name)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である  
(size 0、領域のサイズが不足、recの値が不正)。  
EX\_TRAY : トレーはオープンされていない。

## tdel\_dat

トレーデータの削除

### 【形式】

W tdel\_dat(void)

### 【パラメータ】

なし

### 【リターン値】

0 正常(関数値は「選択状態」のデータ位置)  
<0 エラー(エラーコード)

### 【解説】

トレイ内の「選択状態」のデータを削除する。削除後は削除した直後のデータが「選択状態」となる。直後のデータが存在しない場合(即ち最後のデータの場合)は、直前のデータ(即ち新規に最後となったデータ)が「選択状態」になる。

関数値としては、削除後に「選択状態」となったデータ位置(>0)が戻るが、トレイが空の場合は何も行なわれず、関数値は "0" が戻る。

### 【エラーコード】

EX\_TRAY : トレーはオープンされていない。



# tsel\_dat

トレーデータの選択

## 【形式】

W tsel\_dat(W pos)

## 【パラメータ】

W pos データ位置

## 【リターン値】

0 正常(関数値は「選択状態」のデータ位置)  
<0 エラー(エラーコード)

## 【解説】

トレー内の、pos で指定した位置のデータを「選択状態」とする。今まで「選択状態」であったデータは「非選択状態」となり、関数値として新たな「選択状態」のデータ位置 (> 0) が戻る。

pos は、先頭のデータを "1" とした連続番号であり、pos が存在するデータ数より大きい場合は、最後のデータが「選択状態」となる。また pos = 0 の場合は、「選択状態」を変えずに現在の「選択状態」のデータ位置を関数値として戻す。

トレーが空の場合は何も行なわれず、関数値は "0" が戻る。

## 【エラーコード】

EX\_TRAY : トレーはオープンされていない。

# tmov\_dat

トレーデータの位置変更

## 【形式】

W tmov\_dat(W pos)

## 【パラメータ】

W pos データ位置

### 【リターン値】

0 正常(関数値は「選択状態」のデータ位置)  
<0 エラー(エラーコード)

### 【解説】

トレイ内の「選択状態」のデータを pos で指定したデータ位置に移動する。移動したデータは「選択状態」のままとなる。

pos 1 の場合は先頭に移動し、pos が現在のデータ数より大きい場合は、最後に移動する。

関数値として移動後のデータのデータ位置 (> 0) が戻り、トレイが空の場合は何も行なわれず、関数値は "0" が戻る。

### 【エラーコード】

EX\_TRAY : トレーはオープンされていない。

## tget\_sts

トレイの状態取り出し

### 【形式】

W tget\_sts(UW \*size, UW \*rsize)

### 【パラメータ】

UW \*size トレーの総バイト数  
UW \*rsize 使用できる残りのバイト数

### 【リターン値】

0 正常(関数値は存在するデータ数)  
<0 エラー(エラーコード)

### 【解説】

トレイ内にあるデータの総数を関数値として戻し、その総バイト数を、size に設定する。また使用できる残りのバイト数を rsize に設定する (特に制限がない場合でも適当な大きな数が戻されるものとする)。

size, rsize がそれぞれ NULL の場合は設定されない。

### 【エラーコード】

EX\_ADR : アドレス(size, rsize)のアクセスは許されていない。  
EX\_TRAY : トレーはオープンされていない。

## tset\_dat

一時トレイへのデータ格納

### 【形式】

W tset\_dat(TRAYREC \*data, W nrec)

### 【パラメータ】

TRAYREC \*data データ  
W nrec レコード数

### 【リターン値】

0 正常(関数値は格納したレコード数)  
<0 エラー(エラーコード)

### 【解説】

一時トレイに data で指定した nrec 個のレコードを格納する。以前に格納されていたデータは捨てられる。

nrec = 0 の場合は、一時トレイを空の状態とする。この場合、data はアクセスされないため、その値は何でもよい。

### 【エラーコード】

EX\_ADR : アドレス(data)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した(サイズが大きすぎる)。  
EX\_PAR : パラメータが不正である

(nrec < 0、トレーレコードの内容が不正)。

## tget\_dat

一時トレーからのデータ取り出し

### 【形式】

W tget\_dat(TRAYREC \*data, W size, W \*a\_size, W rec)

### 【パラメータ】

TRAYREC \*data データ格納場所

W size data のバイト数

W \*a\_size データの全体のバイト数

W rec 取り出し方法

< 0 : 全体取り出し

= 0 : ヘッダ取り出し

> 0 : 指定レコードデータ取り出し(rec で指定したレコード)

### 【リターン値】

0 正常(関数値はデータのレコード数/レコードタイプ)

<0 エラー(エラーコード)

### 【解説】

一時トレー内に格納されているデータを、rec の指定に従って取り出し、data で指定した size バイトの領域に格納する。

格納すべきデータのサイズより size が小さい場合は、size 分だけ格納され、EX\_PAR のエラーとなる。data = NULL の場合は、size の値は無視され、データは一切格納されない。いずれの場合も \*a\_size には格納すべきデータの全体のバイトサイズが戻される。このため例えば、rec < 0, data = NULL とすることにより、格納されているデータ全体のサイズを知ることができる。

なお、データを取り出した後も一時トレー内のデータは削除されずに残っている。

関数値として、全体取り出し/ヘッダ取り出し(rec = 0)の時は、データの総レコード数が戻り、指定レコード取り出し(rec > 0)の時は、取り出したレコードのレコードタイプが戻る。存在しないレコード番号を指定した場合は、EX\_PAR のエラーとなり、\*a\_size には "0" が戻される。一時トレーが空の場合は、取り出し方法に無関係に、\*a\_size には "0" が戻され、関数値は "0" となる。

### 【エラーコード】

EX\_ADR : アドレス(data,a\_size)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である  
(size 0、領域のサイズが不足、recの値が不正)。

---

[この章の目次にもどる](#)

[前頁:3.4 パネルマネージャにもどる](#)

[次頁:3.6 データマネージャにすすむ](#)

## 3.6 データマネージャ

### 3.6.1 データマネージャの機能

#### 3.6.1.1 概要

データマネージャは、外殻の1つとして位置付けられ、HMI マネージャ群や、アプリケーションプログラムで使用する固定的なデータ群をプログラムと切り離れた「データボックス」として定義しておき、その内に存在するデータ項目を効率的かつ統一的に取り扱う手段を提供している。データマネージャ自体はデータボックスを作成する機能は持たない。

データボックスは、以下の2種に大別される。

- システムデータボックス

システムで標準的に共通使用するデータ群を登録した、システムに唯一存在するデータボックス。

- アプリケーションデータボックス

アプリケーション(群)で固有に使用するデータ群を登録したデータボックスで、アプリケーション(群)毎に定義される。

1つのアプリケーションは同時にシステムデータボックスと、複数のアプリケーションデータボックスを使用することができる。

#### 3.6.1.2 データタイプID / データ番号

データボックスに登録されている1つのデータ項目は、データタイプIDとデータ番号で識別される。データタイプIDはデータのカテゴリを示すものであり、データ番号は1つのデータタイプ内でユニークに付けられた番号(>0)である。

データタイプIDとしては以下のものが規定されている。

|            |       |                |
|------------|-------|----------------|
| PTR_DATA   | (1)   | ポインタイメージ       |
| PICT_DATA  | (2)   | ピクトグラムイメージ     |
| PAT_DATA   | (3)   | パターンデータ        |
| BMAP_DATA  | (4)   | 図形ビットマップデータ    |
| FIG_DATA   | (5)   | 図形セグメントデータ     |
| TEXT_DATA  | (6)   | 文字列データ         |
| PARTS_DATA | (7)   | コントロールパーツ定義データ |
| MENU_DATA  | (8)   | 標準メニュー定義データ    |
| GMENU_DATA | (9)   | 汎用メニュー定義データ    |
| PANEL_DATA | (10)  | パネル定義データ       |
| USER_DATA  | (64~) | ユーザ定義データ       |

ユーザ定義データは、特定のアプリケーション(群)で使用されるデータタイプであり、64以上の任意のタイプIDをつけて使用することができる。

各データタイプID毎にデータ構造が定義されているが、一部のタイプを除いてその内容はデータマネージャとしては関知しない。

### 3.6.1.3 データマネージャの動作

データマネージャを使用するために、アプリケーションは、まず対象とするデータボックス(システムデータボックスを含む)をすべてオープンする。この結果、各アプリケーション(プロセス)毎に対象とするデータボックスが別々に管理される。ただし、システムデータボックスはすべてのアプリケーションで共通の対象となる。

アプリケーションが、データタイプIDとデータ番号を指定して、データ項目の参照要求を行なった場合、データマネージャは、そのアプリケーションがオープンしたデータボックスを順番にサーチして、要求されたデータ項目が存在するか否かをチェックする。データボックスのサーチは、オープンした逆順に行なわれ、システムデータボックスは常に一番最後になる。従って、同一データタイプID、同一データ番号を使用することにより、あるデータ項目をアプリケーション固有の別のデータ項目で再登録することができる。

要求されたデータ項目が存在する場合は、そのデータ項目を共有メモリ上で参照できるようにしてそのメモリアドレスをアプリケーションに戻す。既に共有メモリ上で参照できる状態にあった場合は単にその領域のアドレスをアプリケーションに戻すことになる。この場合は、複数のアプリケーション間で、そのデータ項目を共用していることになる。

データマネージャにより共有メモリ上にロードされたデータは、他のアプリケーションからも参照される可能性があるため、直接変更してはいけない。書き込みの保護はされていないため、アプリケーション側でこの点を注意しなくてはならない。変更したい場合は、別領域にコピーする必要がある。

### 3.6.1.4 データボックスの構造

データボックスはファイルの1つのレコードとして定義される。レコードタイプは「データボックス」タイプとなる。

通常、1つのデータボックスは、独立した1つのファイルとして定義されるが、1つのファイルの中に複数のデータボックスを入れる場合や、別のレコードと混在している場合もある。

システムデータボックスは1つのファイルとして独立しており、固定的なファイルパスを持ち、アプリケーションデータボックスはアプリケーションがそのファイルパスを指定するため、基本的にどこに存在してもよいが、通常は、アプリケーションプログラムファイル内に置くことが多い。

データボックスは大きくインデックス部とデータ部から構成され、全体として以下に示す構造を持つ。

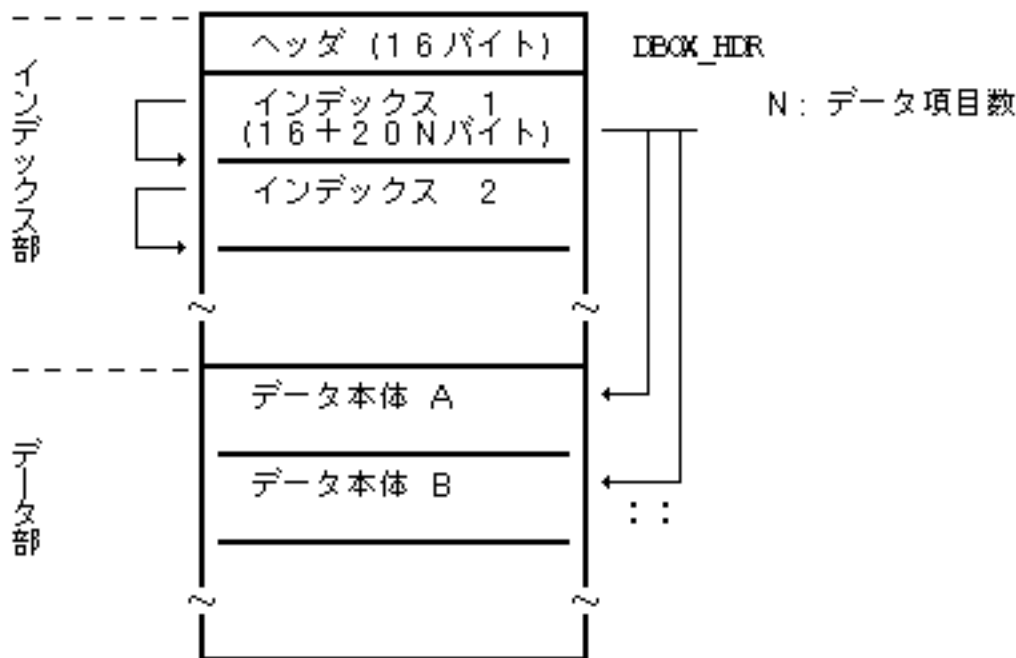


図 108 : 全体構造

インデックス部はヘッダと、そのデータボックスが含む各データタイプID毎のインデックスから構成され、基本的にオープンされた時点でデータマネージャの管理するメモリ領域上にロードされる。

インデックス部はヘッダと、そのデータボックスが含む各データタイプID毎のインデックスから構成され、基本的にオープンされた時点でデータマネージャの管理するメモリ領域上にロードされる。

ヘッダは16バイトの以下の構造を持つ。

```
typedef struct {
    H ntyp;          /* ボックス内のデータタイプIDの数 */
    H ixsize;       /* インデックス部の総バイト数 */
    H resv[6];      /* 予約 */
} DBOX_HDR;
```

各データタイプ毎のインデックスは、以下の構造を持つ。

```
typedef struct {
    W npos;         /* 次のタイプのインデックスへのオフセット */
    W typ;         /* データタイプID */
    UW attr;       /* データの属性(未使用) */
    W nd;         /* このタイプに属するデータ項目の数 */
    DNUM_INX dix[1]; /* 項目インデックス配列 (nd 個の要素) */
} DTYP_INX;
```

- オフセットは、データボックスの先頭からのバイトオフセットである。
- データの属性は以下の内容であり、そのタイプのすべてのデータ項目に適用される。

attr: xxxx .. xxxx xxxL



## L: プレロード指定

1の場合は、データボックスのオープン時に、含まれる全てのデータ項目を共有メモリブロックにロードされる。プレロードされたデータ項目はオープンしたプロセスにより参照されている状態となる。

0の場合は、オープン時にはロードせず、実際の参照時に個別にロードされる。

X:

予約 (0)

項目インデックスは20バイトの、各データ項目に対応するインデックスで、データ部に含まれるデータ本体へのオフセット、およびサイズを保持している。

```
typedef struct {
    W    id;          /* データ番号 */
    W    pos;        /* 対応するデータ本体へのオフセット */
    W    size;       /* データ本体のバイトサイズ */
    W    info[2];    /* 内部使用領域 */
} DNUM_INX;
```

1つのデータボックスに入るインデックスの数に特に制限はないが、インデックス部全体のバイト数は32Kバイト以下でなくてはならない。また、同一タイプのインデックスが複数含まれていても問題ないが、なるべく1つにまとまっている事が望ましい。

データ本体はそのデータタイプに依存した構造を持つが、データマネージャは一部のデータタイプを除いて、その構造には関知せず、単なるバイトデータ列として取り扱う。

## 3.6.2 データ / 定数の定義

### 3.6.2.1 データ / 定数の定義

データタイプID

```
#define PTR_DATA      1    /* ポインタイメージ */
#define PICT_DATA     2    /* ピクトグラムイメージ */
#define PAT_DATA      3    /* パターンデータ */
#define BMAP_DATA     4    /* 図形ビットマップデータ */
#define FIG_DATA      5    /* 図形セグメントデータ */
#define TEXT_DATA     6    /* 文字列データ */
#define PARTS_DATA    7    /* パーツ定義データ */
#define MENU_DATA     8    /* 標準メニュー定義 */
#define GMENU_DATA    9    /* 汎用メニュー定義 */
#define PANEL_DATA    10   /* パネル定義 */
#define USER_DATA     64   /* ユーザ定義データ */
```

データボックスの構造

```
typedef struct {
    H    ntyp;        /* ボックス内のデータタイプIDの数 */
    H    ixsize;     /* インデックス部の総バイト数 */
```

```

    H   resv[6];
} DBOX_HDR;

```

```

typedef struct {
    W   id;           /* データ番号 */
    W   pos;          /* 対応するデータ本体へのオフセット */
    W   size;         /* データ本体のバイトサイズ */
    W   info[2];      /* 内部使用領域 */
} DNUM_INX;

```

```

typedef struct {
    W   npos;         /* 次のタイプのインデックスへのオフセット */
    W   typ;          /* データタイプID */
    UW  attr;         /* データの属性(未使用) */
    W   nd;           /* このタイプに属するデータ項目の数 */
    DNUM_INX dix[1]; /* 項目インデックス配列 (nd 個の要素) */
} DTYP_INX;

```

### 3.6.2.2 標準データ形式

#### ポインタイメージ (PTR\_DATA)

ポインタイメージは、以下の様な構造を持つデータであり、hotpt 以降は、ディスプレイプリミティブで定義されている PTRIMAGE タイプに等しい。

|       |              |                    |
|-------|--------------|--------------------|
| W     | p_size       | -- ポインタサイズ (ピクセル数) |
| COLOR | fgcol        | -- 前景色             |
| COLOR | bgcol        | -- 背景色             |
| PNT   | hotpt        | -- ホットポイント位置       |
| B     | data[P_SIZE] | -- データパターン         |
| B     | mask[P_SIZE] | -- マスクパターン         |

P\_SIZE はポインタサイズにより異なり、以下のようになる。

|             |       |
|-------------|-------|
| 16 x 16 の場合 | : 32  |
| 24 x 24 の場合 | : 96  |
| 32 x 32 の場合 | : 128 |
| 48 x 48 の場合 | : 288 |

データ番号0~99は、システムで使用するために予約されているものとする。

|   |                         |
|---|-------------------------|
| 0 | : (PS_SELECT): 選択指      |
| 1 | : (PS_MODIFY): 修正選択指    |
| 2 | : (PS_MOVE) : 移動手       |
| 3 | : (PS_VMOVE) : 移動手(縦方向) |
| 4 | : (PS_HMOVE) : 移動手(横方向) |
| 5 | : (PS_GRIP) : 握り        |
| 6 | : (PS_VGRIP) : 握り(縦方向)  |
| 7 | : (PS_HGRIP) : 握り(横方向)  |

- 8 : (PS\_RSIZ) : 変形手
- 9 : (PS\_VRSIZ) : 変形手(縦方向)
- 10 : (PS\_HRSIZ) : 変形手(横方向)
- 11 : (PS\_PICK) : つまみ
- 12 : (PS\_VPICK) : つまみ(縦方向)
- 13 : (PS\_HPICK) : つまみ(横方向)
- 14 : (PS\_BUSY) : 湯のみ(待ち状態)
- 15 : (PS\_MENU) : プリメニュー
- 16~99 : (予約)

ピクトグラムイメージ (PICT\_DATA)

図形ビットマップイメージと同じ構造を持つ。

データ番号0~99は、システムで使用するために予約されているものとする。

パターンイメージ (PAT\_DATA)

パターンイメージは描画パターンとして使用されるイメージであり、ディスプレイプリミティブで使用される PATTERN と同様の構造を持っており、以下の2種類のいずれかの構造となる。

<タイプ0>

|   |          |   |               |   |
|---|----------|---|---------------|---|
| W | kind     | — | パターンタイプ = 0   |   |
| W | hsize    | — | パターン横サイズ      |   |
| W | vsize    | — | パターン縦サイズ      |   |
| W | fgcol    | — | 前景色           |   |
| W | bgcol    | — | 背景色           |   |
| W | off_mask | — | マスクデータへのオフセット |   |
| W | mask[*]  | — | 実際のマスクイメージ    | ← |

<タイプ1>

|      |                   |   |                   |   |
|------|-------------------|---|-------------------|---|
| W    | kind              | — | パターンタイプ = 1       |   |
| H    | hsize             | — | パターン横サイズ          |   |
| H    | vsize             | — | パターン縦サイズ          |   |
| W    | off_mask          | — | マスクデータへのオフセット     |   |
| W    | off_bmap          | — | ビットマップデータへのオフセット  |   |
| W    | mask[*]           | — | 実際のマスクイメージ        | ← |
| W    | planes            | — | プレーン数             | ← |
| H    | pixbits           | — | ピクセルビット数          |   |
| H    | rowbytes          | — | ビットマップの横バイト数(偶数)  |   |
| RECT | bounds            | — | ビットマップ境界          |   |
| W    | off_base          | — | ビットマップイメージへのオフセット |   |
| ⋮    | ⋮                 |   |                   |   |
|      | 上記が、planes 個続く    |   |                   |   |
| W    | ⋮                 | — | 実際のイメージデータ        | ← |
| ⋮    | ⋮                 |   |                   |   |
|      | 上記が、最大 planes 個続く |   |                   |   |

図 109: パターンイメージ

off\_xxxx で示されるオフセットは、全て、データの先頭からのバイトオフセットであり、データマネージャにより、メモリにロードされた時点で、オフセットから実ポインタへの変換が行なわれ、ディスプレイプリミティブで直接使用可能な PAT の形式に変換される。この変換は、dopn\_dat() でオープンしたデータだけでなく、ddef\_dat()、ddef\_ldt() で一時的に登録したデータに対しても行なわれる。

データ番号0~99は、システムで使用するために予約されているものとする。

#### 図形ビットマップイメージ (BMAP\_DATA)

図形ビットマップイメージは、ディスプレイプリミティブで使用される C\_BMP と同様の構造を持っており、以下の構造となる。

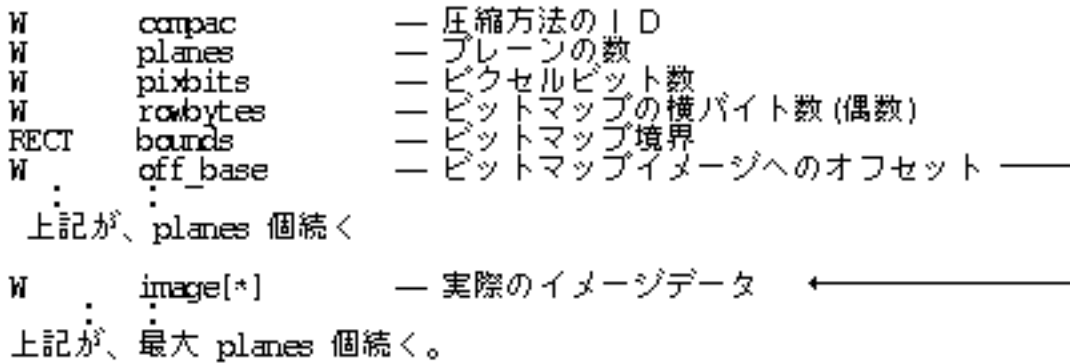


図 110: 図形ビットマップイメージ

off\_xxxx で示されるオフセットは、全て、データの先頭からのバイトオフセットであり、データマネージャにより、メモリにロードされた時点で、オフセットから実ポインタへの変換が行なわれ、ディスプレイプリミティブで直接使用可能な C\_BMP の形式に変換される。この変換は、dopn\_dat() でオープンしたデータだけでなく、ddef\_dat(), ddef\_ldt() で一時的に登録したデータに対しても行なわれる。

#### 図形セグメントイメージ (FIG\_DATA)

TAD で規定されるものと同一である。

#### 文字列データ (TEXT\_DATA)

TEXT\_DATA は単なるワード列であり、通常最後は TNULL (0) で終る。

その他

パーツ、メニュー、パネルのデータ構造はそれぞれ対応する HMI マネージャにより定義される。

### 3.6.3 データマネージャの関数

ここでは、データマネージャがサポートしている各関数の詳細を説明する。これらの関数群は、外殻の拡張システムコールとして提供される。

なお、データマネージャでは核のプロセス管理、ファイル管理、メモリ管理を使用しているため、エラーが発生した場合は、プロセス管理、ファイル管理、メモリ管理のエラーコードが直接戻る。

## dopn\_dat

データボックスのオープン

## 【形式】

W dopn\_dat(LINK \*Ink)

## 【パラメータ】

LINK \*Ink データボックスファイルのリンク

## 【リターン値】

0 正常終了(データボックスファイルID)  
<0 エラーコード

## 【解説】

Ink で指定したファイルに含まれるデータボックスをすべてオープンし、アプリケーションプロセスからのデータボックスの参照を可能とする。関数値として、オープンしたファイルのID (>0) が戻される。このIDはデータボックスのクローズ(dcls\_dat)でのみ使用される。

システムデータボックスがそのプロセスに対して、まだオープンされていない場合は、同時にシステムデータボックスも自動的にオープンされる。Ink = NULL の場合は、システムデータボックスのみのオープンとなり、関数値は0が戻る。

## 【エラーコード】

EX\_ADR : アドレス(Ink)のアクセスは許されていない。  
EX\_DATA : ファイルはデータファイルではない。  
EX\_DFMT : データ項目のデータ形式が不正である。  
EX\_SDATA : システムデータファイルは存在しない。

## dcls\_dat

データボックスのクローズ

## 【形式】

ERR dcls\_dat(W fid)

## 【パラメータ】

W fid データボックスファイルID

## 【リターン値】

0 正常終了  
<0 エラーコード

### 【解説】

fid で指定したオープン済みのファイルをクローズして、そのファイルに含まれるデータボックスの参照を終了する。これにより、データマネージャで使用していた管理用データ領域、参照データ項目の領域等が解放される。

fid = 0 の場合は、システムデータボックスを含むすべてのオープン済みのファイルをクローズする事を意味し、プロセスの終了時には、基本的に fid = 0 として dcls\_dat() を行なう必要がある。

プロセスが終了した場合は、自動的にオープンしたファイルはクローズされるが、dcls\_dat() を明示的に行なうことが望ましい。

### 【エラーコード】

EX\_FD : データファイルIDが不正である。

## dget\_dtp

データ項目の参照

### 【形式】

ERR dget\_dtp(W type, W dnum, void \*\*datap)

### 【パラメータ】

|      |         |            |
|------|---------|------------|
| W    | type    | データタイプ     |
| W    | dnum    | データ番号      |
| void | **datap | データアドレス格納用 |

### 【リターン値】

0 正常終了  
<0 エラーコード

### 【解説】

type で指定したデータタイプIDを持つ、dnum で指定したデータ番号のデータ項目を共有メモリ上で参照できるようにして、そのアドレスを \*datap に戻す。

データ項目は、最後にオープンしたデータボックスから、オープンされた逆順にサーチされ、システムデータボックスが最後にサーチされる。サーチの結果、最初に見付かったデータ項目が対象となる。

### 【エラーコード】

EX\_ADR : アドレス(datap)のアクセスは許されていない。  
EX\_DFMT : データ項目のデータ形式が不正である。  
EX\_DNUM : データ(type, dnum)はデータマネージャに登録されていない。  
EX\_PAR : パラメータが不正である。

## dget\_siz

データ項目のサイズ取出し

### 【形式】

W dget\_siz(B \*addr)

### 【パラメータ】

B \*addr データボックスデータのアドレス

### 【リターン値】

0 正常終了(関数値はデータ項目のバイトサイズ)  
<0 エラーコード

### 【解説】

addr で指定した共有メモリ上のデータ項目のバイトサイズを関数値として戻す。

### 【エラーコード】

EX\_AKEY : データ項目のアドレスが不正である。

## dget\_num

データ番号 / タイプIDの取出し

### 【形式】

W dget\_num(B \*addr, W \*type)

### 【パラメータ】

B \*addr データボックスデータのアドレス  
W \*type データタイプIDの格納場所

### 【リターン値】

0 正常終了(関数値はデータ番号)  
<0 エラーコード

### 【解説】

addr で指定した共有メモリ上のデータ項目のデータ番号を関数値として戻し、データタイプIDを type で指定した領域に格納する。type = NULL の時は、データタイプIDは格納されない。

### 【エラーコード】

EX\_ADR : アドレス(type)のアクセスは許されていない。  
EX\_AKEY : データ項目のアドレスが不正である。

## ddef\_dat

システムデータ項目の再定義

### 【形式】

ERR ddef\_dat(W type, W dnum, void \*ptr, W size)

### 【パラメータ】

|      |      |          |
|------|------|----------|
| W    | type | データタイプ   |
| W    | dnum | データ番号    |
| void | *ptr | データアドレス  |
| W    | size | データのバイト数 |

### 【リターン値】

0 正常終了  
<0 エラーコード



## 【解説】

ptr で指定した領域にある size バイトのデータを、 type で指定したデータタイプID、 dnum で指定したデータ番号を持つすでに存在するシステムデータ項目と置き換える。

この関数により再定義したデータ項目は、一時的な登録であって、ファイルに格納されないため、解放した場合には、再度定義し直さない限り参照できなくなる。

## 【エラーコード】

EX\_ADR : アドレス(ptr)のアクセスは許されていない。

EX\_DFMT : データ項目のデータ形式が不正である。

EX\_DNUM : データ(type, dnum)はデータマネージャに登録されていない。

EX\_PAR : パラメータが不正である。

# ddef\_ldt

ローカルデータ項目の登録

## 【形式】

ERR ddef\_ldt(W type, W dnum, void \*ptr, W size)

## 【パラメータ】

|      |      |          |
|------|------|----------|
| W    | type | データタイプ   |
| W    | dnum | データ番号    |
| void | *ptr | データアドレス  |
| W    | size | データのバイト数 |

## 【リターン値】

0 正常終了  
<0 エラーコード

## 【解説】

ptr で指定した領域にある size バイトのデータを、 type で指定したデータタイプID、 dnum で指定したデータ番号を持つローカルデータ項目として登録する。

登録したデータ項目は、そのプロセスからのみ参照可能であり、最後にオープンしたデータボックスとして取り扱われるため、常に最初にサーチされる。

この関数により登録したデータ項目は、一時的な登録であって、ファイルに格納されないため、解放した場合には、再度登録し直さない限り参照できなくなる。

## 【エラーコード】

EX\_ADR : アドレス(ptr)のアクセスは許されていない。  
EX\_DFMT : データ項目のデータ形式が不正である。  
EX\_PAR : パラメータが不正である。

---

[この章の目次にもどる](#)

[前頁:3.5 トレーマネージャにもどる](#)

[次頁:3.7 テキスト入力プリミティブにすすむ](#)

[この章の目次にもどる](#)

[前頁:3.6 データマネージャにもどる](#)

[次頁:3.8 実身 / 仮身マネージャにすすむ](#)

---

## 3.7 テキスト入力プリミティブ

### 3.7.1 テキスト入力プリミティブの機能

#### 3.7.1.1 概要

テキスト入力プリミティブ (TIP) は、 外殻の HMI 機能の 1 つとして位置付けられ、 かな漢字変換およびローマ字かな変換を通した、 テキスト入力の機能を提供しているものである。

アプリケーションプログラムは、 このテキスト入力プリミティブを使用することにより、 かな漢字変換方式に依存しない共通的なインタフェースにより、 テキストの入力を行なうことが可能となる。

テキスト入力プリミティブで提供している機能は、 アプリケーション側でのきめの細かい処理を可能としているため、 かなりプリミティブなレベルであり、 また文字の表示機能は、 一切含まれていない。

テキスト入力プリミティブの機能としては、 かな漢字変換、 ローマ字かな変換の他に、 学習辞書の保存状態の設定 / 変更機能が提供される。 また、 辞書アクセスや単漢字変換の機能も含まれているが、 アプリケーションはこれらを特に意識する必要はない。 なお、 辞書にない単語の登録機能や、 辞書データ交換のための辞書変換機能などの辞書メンテナンス機能は、 1 つの独立したシステムアプリケーションとして別に提供されることになる。

なお、 テキスト入力プリミティブでは、 関連する機能として、 カレット表示を行なう関数も提供している。

#### 3.7.1.2 テキスト入力プリミティブ

テキスト入力プリミティブは、 一種のフィルターとして以下に示すようなモデルとして表現される。 即ち、 入力として「キー入力イベント」を受け付け、 出力として「確定文字列」を戻すことになる。 さらに補助的な出力として「未確定文字列」、「文節情報」等を戻すことになる。 なお、 テキスト入力プリミティブで保持している文字列全体を「変換中文字列」と呼ぶ。 この文字列には、 最新の操作で確定した「確定文字列」と「未確定文字列」が含まれる。

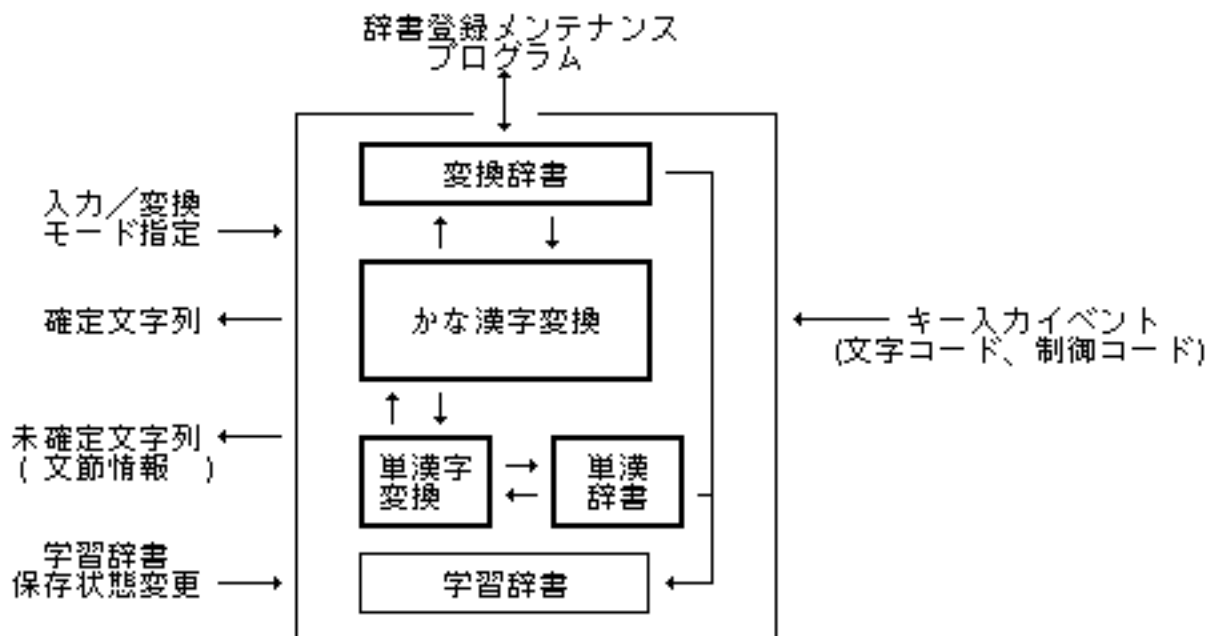


図 111: テキスト入力プリミティブのモデル

入力モードとしては、以下の3種類のいずれかを指定することができる。通常、入力モードはユーザ毎に固定的なものが使用されるが、動的に変更することも可能である。

- かな入力モード
- ローマ字入力モード1 (n方式)
- ローマ字入力モード2 (nn方式)  
(n方式、nn方式はローマ字入力における「ん」の入力方式である)

かな漢字変換モードは、以下の2種類のいずれかであり、自動変換モードでは、入力した「よみ」が[変換/逆変換]なしに自動的に変換されるモードであり、指定変換モードは、[変換/逆変換]によってのみ変換されるモードである。自動変換の場合は、変換のタイミングにより各種の方法が考えられるが、[変換/逆変換]なしに変換されるものは、基本的にすべて自動変換とみなすものとする。たとえば、句読点の入力時や、入力文字種の切り替わり時点で変換されるものも自動変換と見なす。なお、両方の変換モードをサポートしているか否かはインプリメントに依存する。

- 自動変換モード
- 指定(マニュアル)変換モード

変換モードは、以下の2種類のいずれかであり、直接モードでは入力した「よみ」が変換なしに直接出力されるモードであり、変換モードはかな漢字変換モードで指定したかな漢字変換方式で変換されるモードである。

- 直接モード
- 変換モード

テキスト入力プリミティブでは、以下に示すような機能をサポートすることが可能であり、これらの操作は、機能キーに対応するキー入力イベントによって行なわれる。ただし、これらの機能を実際にサポートしているか否かは、かな漢字変換のアルゴリズムに依存する。

- 対象文節の候補の変更

- 未確定文字列全体の候補の変更
- 未確定文字列後端への文字の追加 / 削除
- 未確定文字列内へのよみの挿入
- 未確定文字列内でのよみの削除
- 文節区切り位置の変更

### 3.7.1.3 テキスト入力ポート

テキスト入力プリミティブの動作環境をテキスト入力ポートと呼ぶ。アプリケーションはテキスト入力ポートをオープンし、オープン時に得られるIDを使用して以後の入力処理を行なうことになる。テキスト入力ポートはオープンしたプロセスにのみ有効であり、そのプロセスが終了した場合は自動的に削除される。

テキスト入力ポートのオープン時には、以下に示すテキスト入力レコードを指定する必要があり、指定したテキスト入力レコードにテキスト入力ポートの現在の状態が戻されることになる。

```
typedef struct {
    W  n_out;      /* 確定文節数 */
    W  n_cl;      /* 全文節数 */
    W  n_roman;   /* 部分ローマ字数 */
    W  update;    /* 表示更新開始文字位置 */
    UW caret;    /* 現在のcaret位置 */
    UW clause;    /* 現在対象とする文節番号および状態 */
    W* cl_cnv;    /* 変換中文字列の文節位置配列 */
    TC* cnv;      /* 変換中文字列 */
    W* cl_in;     /* 入力(よみ)文字列の文節位置配列 */
    TC* in;       /* 入力(よみ)文字列 */
} TIPREC;
```

n\_out :

確定した文節数を示し、cnv で示される変換中文字列の先頭からの文節数を示す。n\_out で示される数の文節は、続くテキスト入力ポートへの入力処理で削除されるので、n\_out = 0 の時は、アプリケーションは、指定された文節数のデータを必ず取りだしておく必要がある。

なお、テキスト入力プリミティブ内部で保持可能な文節数または文字バッファの制限を超えた場合にも、n\_out として先頭のいくつかの文節が確定されることがある。

n\_cl :

変換中文字列全体の文節数を示す。

n\_roman :

変換中文字列中でのローマ字入力の英字数 (0 ~ インプリメント依存) を示す。

|        |          |            |
|--------|----------|------------|
| 例: cnv | .....き   | n_roman =0 |
|        | .....きS  | n_roman =1 |
|        | .....きSH | n_roman =2 |
|        | .....汽車  | n_roman =0 |

update :

変換中文字列で変更があった先頭の文字位置を、変換中文字列の先頭文字を "0" とした文字位置で示す。これは、変換中文字列の表示の更新に使用され、<0 の場合は変換中文字列は変化しなかったことを示す。

caret :

変換中文字列内でのカレットの現在位置を示す以下の値であり、カレット表示のために使用される。

XXXX XXXX XXXX XXXX xPPP PPPP PPPP PPPP

P :

カレットの現在位置

変換中文字列内で n\_out で示される確定文節の直後の文節の先頭文字を "0" とした場合の、カレットの直後に位置する文字の位置。すなわち、未確定文字列内での文字位置。

X :

予約

clause :

変換中文字列内での、現在の対象文節、またはよみ列修正文節を示す以下の値であり、対象文節 / よみ列修正文節の表示のために使用される。対象文節は、変換操作の対象となっている文節であり、よみ列修正文節は、修正操作の対象となっている文節である。

XXXX XXXX XXXX XXXX MPPP PPPP PPPP PPPP

M :

= 0 対象文節

= 1 よみ列修正文節

P :

対象 / よみ列修正文節の文節位置

変換中文字列内で n\_out で示される確定文節の直後の文節を "0" とした場合の文節の位置。すなわち、未確定文字列内での文節位置。

X :

予約

cl\_cnv :

変換中文字列の文節区切り位置を示す (n\_cl+1) 個の要素からなるワード配列へのポインタであり、各文節の先頭の文字位置を表わす。文字位置は "0" から始まるため先頭の要素は常に "0" となり、最後の要素は変換中文字列の最後の TNULL(0) の文字位置であり、変換中文字列全体の文字数となる。

なお、未確定文字列中の文節区切り位置は、対象文節 / よみ列修正文節の特定のために使用されるため、厳密な意味での文節区切り位置とは異なる場合がある。

cnv :

変換中文字列へのポインタである。一般に確定文字列と未確定文字列からなっているが、n\_out = 0 ときは、未確定文字列のみとなり、n\_out = n\_cl のときは、確定文字列のみとなる。変換中文字列の最後には TNULL(0) が入っている。

in\_cnv :

入力(よみ)文字列の文節区切り位置を示す (n\_cl + 1) 個の要素からなるワード配列へのポインタであり、各文節の先頭の文字位置を表わす。文字位置は "0" から始まるため先頭の要素は常に "0" となり、最後の要素は入力(よみ)文字列の最後の TNULL(0) の文字位置であり、入力(よみ)文字列全体の文字数となる。

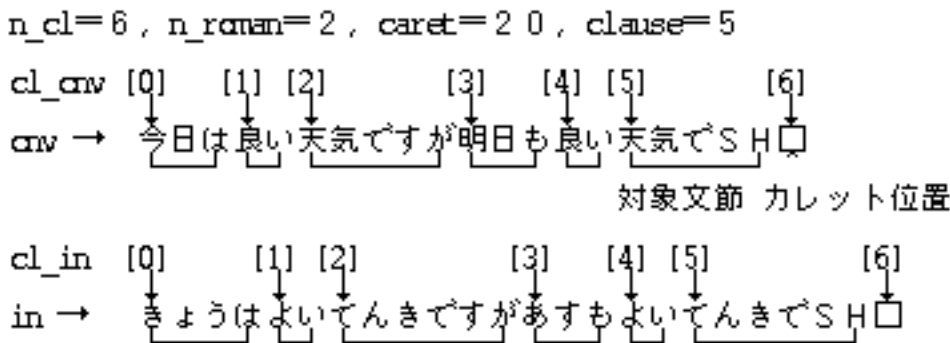
in :

変換中文字列に対応する入力(よみ)文字列へのポインタである。入力(よみ)文字列の最後には TNULL(0) が入っている。

- cl\_cnv, cnv, cl\_in, in は、いずれもテキスト入力ポート内部に確保されたバッファへのポインタであり、TIPREC には、このポインタ自体が設定される。

一般に自動変換の場合は、テキスト入力ポートへのキー入力のたびに、変換中文字列の状態が変化し、逐次確定された文節が掃き出されることになるが、指定変換の場合は、[変換/逆変換]を入力するまでは、変換中文字列と入力文字列は同一であり、文節数は常に "1" となる。

以下にテキスト入力レコードの内容の例を示す。



上記の状態で、"0" を入力した場合： ( □ は TNULL を意味する )

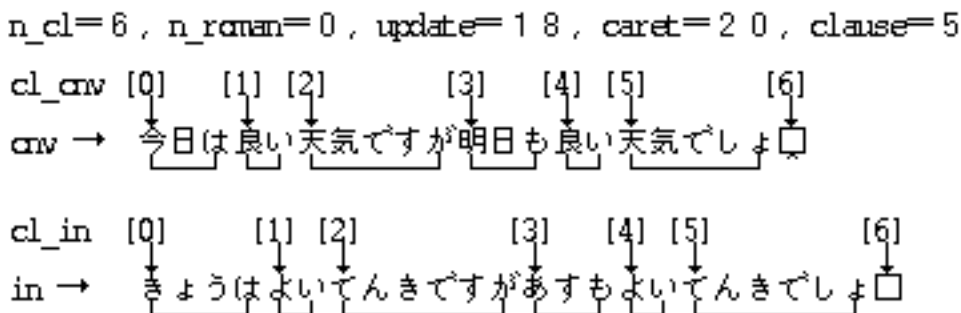


図 112: テキスト入力レコードの内容

### 3.7.1.4 カレット表示

カレットは文字の挿入位置を示す山形のシンボルであり、通常は適当な周期でブリンクしている。カレットは、何も選択されていないことを示す、選択ギャップ(ヌル選択)の縦棒と結びついた場合は、文字カーソルの意味を持つことになる。

テキスト入力プリミティブでは、カレット(文字カーソル)の表示をサポートするための関数が用意されており、以下の構造体によりカレットが定義される。カレット表示の関数では、表示するカレットの状態を保持していないため、この CARET 構造体の中に、カレットの状態が保持されることになる。

```

typedef struct {
    W sts;          /* 0:消去状態 1:表示状態 */
    W gid;          /* 描画環境ID */
    PNT pos;        /* 表示位置 */
    W height;       /* 表示高さ */
    UW kind;        /* カレット種類 0:横 1:縦 */
    COLOR color;    /* カレットの表示カラー */
} CARET;

```

sts :

カレットの表示状態であり、"0"で消去状態、"1"で表示状態を示す。  
この値はカレットの表示関数 ( idsp\_car() ) により自動的に更新されるが、カレットの表示状態に応じて直接変更してもよい。

gidn :

カレットを描画する描画環境のIDである。

pos :

カレットの山型の頂点の位置(相対座標)の指定である。

height :

選択ギャップの表示の高さであり、通常はその位置の文字の高さに合わせるようになる。height = 0 の場合は、選択ギャップは表示されず、カレットのみの表示となる。

kind :

カレットの種類を指定する以下の値である。

```

XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXD

```

D:

= 0 : 横書き

= 1 : 縦書き

X:

予約

color :

カレットの表示カラーの指定であるが、インプリメントによってはサポートされない場合もある。

通常カレットの標準形状を、以下に示すが詳細はインプリメントに依存する。



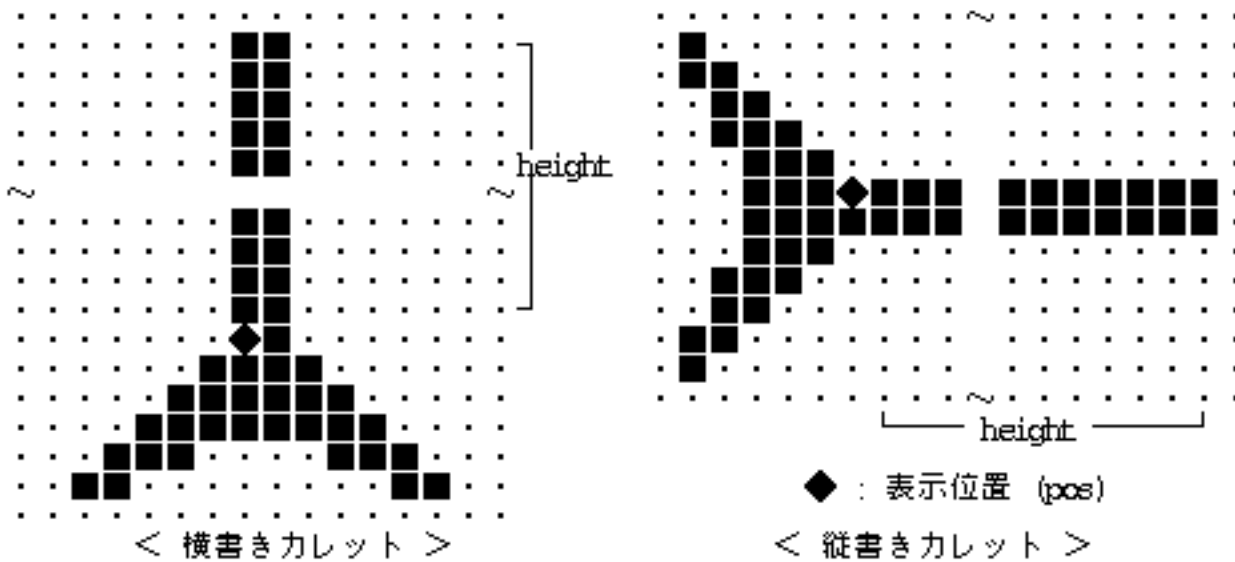


図 113: 通常カレットの標準形状

カレットの表示は、基本的に入力受付状態のウィンドウに対してのみ行なわれる。

### 3.7.2 データタイプ/定数の定義

#### 定数定義

```
#define TIP_KANA      0x0000 /* かな入力モード */
#define TIP_ROMAN1   0x0001 /* ローマ字入力モード(n) */
#define TIP_ROMAN2   0x0002 /* ローマ字入力モード(nn) */

#define TIP_DIRMD    0x0100 /* 直接モード */
#define TIP_CNVMD    0x0000 /* 変換モード */

#define TIP_AUTO     0 /* 自動変換モード */
#define TIP_MANUAL   1 /* 指定変換モード */

#define TIP_OUT      0x1 /* 確定文節が発生した */
#define TIP_CNV      0x2 /* 未確定文節が変更された */
#define TIP_CAR      0x4 /* カレット位置が変化した */
#define TIP_CL       0x8 /* 対象文節、または区切り位置が変化した */
```

#### ichg\_lrn()

```
#define TIP_DIC_COMMON 0x00000001 /* 基本辞書 */
#define TIP_DIC_SINGLE 0x00000002 /* 単漢字辞書 */
#define TIP_DIC_USR    0x00000004 /* ユーザ辞書 */
```

#### TIP

```

typedef struct {
    W    n_out;      /* 確定文節数 */
    W    n_cl;      /* 全文節数 */
    W    n_roman;   /* 部分ローマ字数 */
    W    update;    /* 表示更新開始文字位置 */
    UW   caret;     /* 現在のcaret位置 */
    UW   clause;    /* 現在対象とする文節番号および状態 */
    W*   cl_cnv;    /* 変換中文字列の文節位置配列 */
    TC*  cnv;       /* 変換中文字列 */
    W*   cl_in;     /* 入力文字列の文節位置配列 */
    TC*  in;        /* 入力文字列 */
} TIPREC;

```

```
#define TIP_YOMIMOD 0x80000000 /* 文節の M フラグ */
```

## caret

```

typedef struct {
    W    sts;       /* 0:消去状態 1:表示状態 */
    W    gid;       /* 描画環境ID */
    PNT  pos;       /* 表示位置 */
    W    height;    /* 表示高さ */
    UW   kind;      /* caret種類 0:横 1:縦 */
    COLOR color;   /* caretの表示カラー */
} CARET;

```

### 3.7.3 テキスト入力プリミティブの関数

ここでは、テキスト入力プリミティブが外殻の拡張システムコールとしてサポートしている各関数の詳細を説明する。

関数はすべて WERR 型の関数値をとり、何らかのエラーがあった場合は「負」のエラーコードが戻る。正常終了時には「0」または「正」の値が戻る。

各関数のエラーコードとしては、ここで示した以外にも、核や他の外殻でエラーが検出された場合は、以下のエラーコードが直接戻る場合がある。

E\_xxx :  
核のエラー (E\_NOMEM, E\_NOSPC, E\_PAR 等)

EG\_xxx :  
ディスプレイプリミティブのエラー (EG\_ADR 等)

また、各関数のパラメータの説明では、以下に示す記述方法を使用している。

```

( x    y    z ) -- x, y, z のいずれか1つを意味する。
|          -- OR で指定可能なことを意味する。
[ ]       -- 省略可能なことを意味する。

```

# ichg\_mod

テキスト入力モードの変更

## 【形式】

WERR ichg\_mod(W mode)

## 【パラメータ】

W mode ::= ( TIP\_KANA TIP\_ROMAN1 TIP\_ROMAN2)

|            |   |                   |
|------------|---|-------------------|
| TIP_KANA   | 0 | かな入力モード           |
| TIP_ROMAN1 | 1 | ローマ字入力モード1 (n方式)  |
| TIP_ROMAN2 | 2 | ローマ字入力モード2 (nn方式) |

## 【リターン値】

0 正常(関数値は変更前の入力モード)  
<0 エラー(エラーコード)

## 【解説】

テキスト入力モードを mode で指定した内容に変更し、変更前のテキスト入力モードを関数値として戻す。mode<0 の場合は入力モードを変更せずに現在の入力モードを関数値として戻す。

テキスト入力モードはグローバルに有効であり、入力モードを変更した場合は、全てのプロセスでオープンしているテキスト入力ポートに対して、変更した時点から適用される。

## 【エラーコード】

EX\_PAR : パラメータが不正である(mode が不正)

# ichg\_lrn

学習辞書の保存状態変更

## 【形式】

WERR ichg\_lrn(UW kind, W stat)

## 【パラメータ】

UW kind 辞書の種類  
W stat 0で保存しないことを指定  
>0で保存することを指定

## 【リターン値】

0 正常(関数値は変更前の学習辞書の保存状態)  
<0 エラー(エラーコード)

## 【解説】

kind で指定した種類の学習辞書の保存状態を、stat で指定した内容に変更し、変更前の学習辞書の保存状態を関数値として戻す。stat<0 の場合は、保存状態を変更せずに現在の保存状態を関数値として戻す。

kind は、ビット対応で辞書の種類を指定する(対応ビット=1で指定)。

```
0XXX XXXX XXXX XXXX
|-----+-----|  ||
|               |  |+- 基本辞書
インプリメント依存辞書 +----- 単漢字変換辞書
```

なお、学習辞書の保存状態の変更による実際の効果はインプリメントに依存する。

## 【エラーコード】

EX\_NOSPT : 現在のインプリメントではその機能をサポートしていない  
EX\_PAR : パラメータが不正である(kind または stat が不正)

## iopn\_tip

テキスト入力ポートのオープン

## 【形式】

WERR iopn\_tip(TIPREC \*tip, W mode)

## 【パラメータ】

TIPREC \*tip テキスト入力ポート用レコード  
W mode ::= ( TIP\_DIRMD TIP\_CNVM ) | ( TIP\_AUTO TIP\_MANUAL )

|            |       |                |
|------------|-------|----------------|
| TIP_DIRMD  | 0x100 | 直接モード(変換なし)    |
| TIP_CNVM   | 0     | 変換モード          |
| TIP_AUTO   | 0     | 自動変換モード        |
| TIP_MANUAL | 1     | 指定(マニュアル)変換モード |

TIP\_DIRMD を指定した時、TIP\_AUTO / TIP\_MANUAL の指定は、意味を持たず無視される。

### 【リターン値】

0 正常(関数値はテキスト入力ポートID(tipid))  
<0 エラー(エラーコード)

### 【解説】

テキスト入力ポートを新規にオープンし、そのID (tipid 0) を関数値として戻す。

tip はオープンしたテキスト入力ポートに割り当てられる、TIPREC の領域へのポインタであり、以後の操作でのテキスト入力ポートの状態がセットされることになる。オープン時には、tip で指定した TIPREC の内容はすべて "0" に初期化される。

mode は変換モードの指定であるが、インプリメントによってサポートされていない場合はエラーとなる。

### 【エラーコード】

EX\_ADR : アドレス(tip)のアクセスは許されていない  
EX\_LIMIT : システムの制限を超えた  
EX\_NOSPC : システムのメモリ領域が不足した  
EX\_NOSPT : 現在のインプリメントではその機能をサポートしていない  
(サポートしない変換モード)  
EX\_PAR : パラメータが不正である(mode が不正)

## icls\_tip

テキスト入力ポートのクローズ

### 【形式】

ERR icls\_tip(W tipid)

### 【パラメータ】

W tipid テキスト入力ポートID

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

tipid で指定したテキスト入力ポートをクローズする。

# iput\_key

テキスト入力ポートへのキー入力

## 【形式】

WERR iput\_key(W tipid, EVENT \*evt)

## 【パラメータ】

W tipid テキスト入力ポートID  
EVENT \*evt キーイベント

## 【リターン値】

0 正常(関数値はテキスト入力ポートの変化状態)  
<0 エラー(エラーコード)

## 【解説】

tipid で指定したテキスト入力ポートに \*evt で指定したキー入力イベント (EV\_KEYDWN または EV\_AUTKEY) を入力し、その結果のテキスト入力ポートの状態をオープン時に割り当てた TIPREC に戻す。

関数値として、テキスト入力ポートの状態変化が以下のビット対応で戻される。何の変化も発生しなかった場合は、関数値は"0"となる。アプリケーションはこの関数値に応じた適切な処理を行なう必要がある。

|         |     |                         |
|---------|-----|-------------------------|
| TIP_OUT | 0x1 | 確定文節が発生した               |
| TIP_CNV | 0x2 | 未確定文字列が変更された            |
| TIP_CAR | 0x4 | カレット位置が変化した             |
| TIP_CL  | 0x8 | 対象文節が移動した、または区切り位置が変化した |

\*evt で指定したイベントの内容が以下のいずれかの場合はエラーとなり、TIPREC の内容は一切変化しない。

- イベントのタイプが EV\_KEYDWN、EV\_AUTKEY 以外の時

- 文字コード自体が不正の時
- ローマ字入力として定義されていない文字コードの組み合わせの時(インプリメント依存)
- サポートしていない機能の機能キーコードの時

## 【エラーコード】

|          |  |
|----------|--|
| EX_CKEY  | : 変換中文字列が存在しない状態で機能キーコードが入力された               |
| EX_KEY   | : 不正キーコードである                                 |
| EX_LIMIT | : システムの制限を超えた(変換中文字列が長すぎる)                   |
| EX_PAR   | : パラメータが不正である(EV_KEYDOWN、または EV_AUTKEY ではない) |
| EX_TID   | : テキスト入力ポート(tipid)は存在していない                   |

## input\_str

テキスト入力ポートへの文字列入力

## 【形式】

ERR input\_str(W tipid, TC \*str)

## 【パラメータ】

W tipid テキスト入力ポートID  
TC \*str 文字列

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

tipid で指定したテキスト入力ポートに str で指定した文字列を入力して変換を行ない、その結果のテキスト入力ポートの状態を、オープン時に割り当てた TIPREC に戻す。

str で指定した文字列は、ローマ字コードを含まない通常の文字コード、[スペース(=0x20)]および[リクワイアードスペース(=0xA0)]のみからなる TNULL(0) で終了する文字列であり、途中にローマ字コードや、[改行]等の特殊文字キーコード、[変換]等の機能キーコードが含まれている場合は、EX\_KEY のエラーとなる。

また、直接モード(TIP\_DIRMD)でオープンされたテキスト入力ポートを指定した場合は、EX\_NOSPT のエラーとなる。

エラーが発生した場合は、TIPREC の内容は保証されない。

この関数は、主に入力済みの文字列に対する再変換の最初の変換時に使用され、テキスト入力ポートに変換中文字列が存在していない状態で使用される。テキスト入力ポートに、変換中文字列が存在していた状態で使用した場合の結果は、保証されない。

### 【エラーコード】

EX\_KEY : 不正キーコードである  
EX\_LIMIT : システムの制限を超えた(変換中文字列が長すぎる)  
EX\_NOSPT : 現在のインプリメントではその機能をサポートしていない  
(直接モードでオープンされている)  
EX\_TID : テキスト入力ポート(tipid)は存在していない

## ichg\_blk

カレットのブリンク周期の取出し / 設定

### 【形式】

WERR ichg\_blk(W intvl)

### 【パラメータ】

W intvl ブリンク周期

### 【リターン値】

0 正常(関数値は設定前のブリンク周期)  
<0 エラー(エラーコード)

### 【解説】

カレットのブリンク周期を intvl で指定した値に設定し、設定以前の値を関数値として戻す。intvl = 0 の場合は、ブリンクしない事を意味し、intvl < 0 の場合は、設定せずに現在の設定値を戻す。

ブリンク周期の単位はミリ秒であるが、実際にはインプリメントに依存した単位で、大きな値の方向に丸められることになる。

カレットのブリンク周期はグローバルに適用される。

### 【エラーコード】

エラーは発生しない。



# idsp\_car

カレットの表示

## 【形式】

ERR idsp\_car(CARET \*car, W mode)

## 【パラメータ】

CARET \*car カレットデータ

W mode 表示 / 消去 / ブリンク指定

## 【リターン値】

0 正常  
<0 エラー(エラーコード)

## 【解説】

car で指定したカレットを mode の指定に従って、消去 / 表示 / ブリンクする。

mode = 0:

消去

car->sts = 0 の場合、car->pos, car->height で示されるカレットを消去し、car->sts を 0 に設定する。  
car->sts = 0 の場合は何も行なわない。

> 0:

表示

car->sts = 0 の場合、car->pos, car->height で示されるカレットを表示し、car->sts を 1 に設定する。  
car->sts = 0 の場合は何も行なわない。

< 0:

ブリンク

ブリンクの間隔に達していた場合、car->sts の値に応じて、car->pos, car->height で示されるカレットを表示または消去し、car->sts を更新する。  
ブリンク間隔に達していない場合は何も行なわない。

ブリンクの場合は、ブリンクの周期以内に周期的に idsp\_car (&car, -1) を実行する必要がある。

カレットの表示を移動する場合は、まず消去を行なった後、カレットの位置、高さを変更して、表示を行なうことになる。また、表示状態で、カレット構造体のパラメータを変更した場合の動作は保証されない。

## 【エラーコード】

EX\_ADR : アドレス(car)のアクセスは許されていない

---

[この章の目次にもどる](#)

[前頁:3.6 データマネージャにもどる](#)

[次頁:3.8 実身 / 仮身マネージャにすすむ](#)

[この章の目次にもどる](#)

[前頁:3.7 テキスト入力プリミティブにもどる](#)

[次頁:3.9 フォントマネージャにすすむ](#)

## 3.8 実身 / 仮身マネージャ

### 3.8.1 実身 / 仮身マネージャの機能

#### 3.8.1.1 概要

実身 / 仮身マネージャは、 外殻の 1 つとして「実身 / 仮身モデル」を実現する上での各種の高レベルの機能を提供しており、 アプリケーションプログラムは実身 / 仮身マネージャで提供される拡張システムコールを使用して容易に実身 / 仮身を表示 / 操作することが可能となる。

また、実身 / 仮身マネージャは、 アプリケーションプログラムの登録 / 削除機能を提供しており、 基本的に全てのアプリケーションプログラムは実身 / 仮身マネージャを通して起動されることになる。

実身 / 仮身マネージャの機能としては、 以下のものがあげられる。

1. 仮身の各種表示 / 操作機能
2. 付箋の各種表示 / 操作機能
3. アプリケーションプログラムの管理 ( 登録 / 削除 / 実行 ) 機能
4. メニュー設定機能
5. ファイルシステムの接続管理機能
6. その他のアプリケーション機能

以下、この章ではHMI外部仕様としての実身 / 仮身マネージャを用いたアプリケーションによる、実身 / 仮身の取扱いを記述しているが、 具体的な表示項目や表示のデザインなどの詳細についてはインプリメントに依存する。

#### 3.8.1.2 仮身 / 実身 / 虚身

「実身」とは意味のあるデータの集まりそれ自体のことであり、 実際のファイルに一対一に対応するものである。 実身をユーザが操作するための手掛かりが「仮身」であり、 実身は常に仮身を通して参照される。 仮身は実身を参照するための一種のタグであり、 一つの実身をもつ以上の仮身が参照することができる。

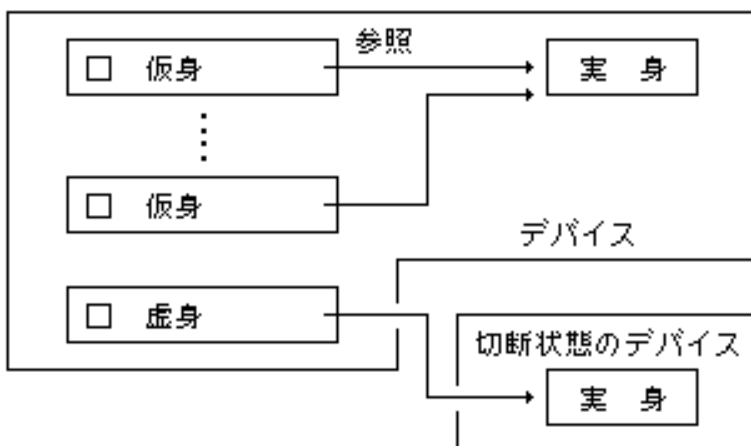


図 114 : 仮身 / 実身 / 虚身

実身自体は目で見える形で直接的に表示されることはなく、その実身を参照する仮身として表示される。仮身は1つのデータ単位として、文字や図形等と同様に実身内の任意の位置に埋め込まれて表示される。

仮身には実身の管理情報のみを表示している「閉じた仮身」と、管理情報と共に実身の内容も表示している「開いた仮身」がある。

仮身を「ウィンドウに開く」ことにより、その仮身が指す実身に適用されるアプリケーションが実行されて、実身の内容が表示される。どのアプリケーションを実行するかを指定せずに仮身からウィンドウを開いた場合には、その仮身に保持されているデフォルトアプリケーションが起動される。

仮身が格納されていた実身が存在するデバイスを、仮身の「所属元デバイス」と呼び、仮身が格納される実身が存在するデバイス、即ち、仮身が表示されているウィンドウの元となった実身が存在するデバイスを仮身の「所属デバイス」と呼ぶ。仮身の所属デバイスは、仮身を、異なるデバイスに存在する実身を元にして開かれたウィンドウ間で移動することにより動的に変化する。

仮身の所属元デバイスと異なるデバイスもしくはその中の実身を参照する仮身、即ちリンクファイルを参照している仮身を「間接参照仮身」と呼ぶ。間接参照仮身と実身は疎な結合であるため、その参照は保証されない。

また、仮身の所属デバイスと異なるデバイスもしくはその中の実身を参照する仮身を「一時間接参照仮身」と呼ぶ。一時間接参照仮身は格納することによりリンクファイルを参照する間接参照仮身となる。

デバイスには、そのデバイスが物理的または論理的に接続されておらず直接アクセスできない「切断状態」と接続されておりアクセス可能な「接続状態」の2つの状態があり、初期状態は切断状態となる。

「切断状態」のデバイスもしくはその中の実身を参照する間接参照仮身を「虚身」と呼ぶ。虚身を通常の仮身のようにウィンドウに開こうとした場合は、「デバイス接続パネル」が現われ、その虚身の参照する実身が最後に「接続状態」だった時点での所在等の管理情報を表示して、接続状態にするか否かの判断を求める。接続状態にすることにより虚身は通常の仮身と同様に操作できる。

一般の虚身は「閉じた虚身」であり開くことはできないが、すでに開いている仮身が切断状態にされた場合は「開いた虚身」となる。

### 3.8.1.3 仮身の表示

#### 仮身の表示

「閉じた仮身」は、横長の矩形枠で囲まれた図形として表示され、参照する実身に関する情報が内部に表示される。

「開いた仮身」は、閉じた仮身の境界線が下に拡がり、実身の内容自体を表示する表示エリアが追加されたものであり、表示エリアは実身の内容が対応するアプリケーションにより表示される。

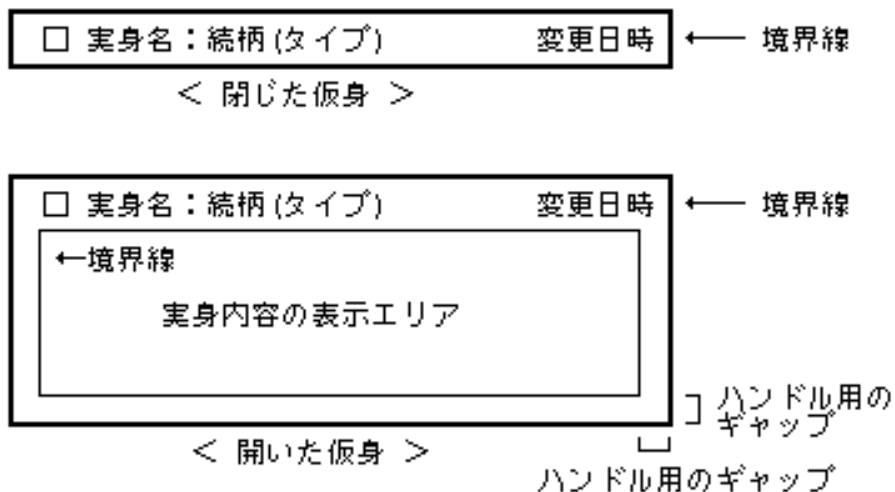


図 115: 仮身の表示

「選択状態」の場合、仮身の周囲に点線の枠をチラツキ表示させる。

仮身から開かれたウィンドウが存在している状態を「処理中状態」と呼び、元の仮身の表示全体に灰色パターンが重ねられる。「間接参照仮身」および「一時間接参照仮身」の場合、仮身の枠の左上角が境界線色で塗りつぶされる。この間接参照を示す表示は、仮身を格納した場合間接参照仮身となることを示すものである。よって「間接参照仮身」をその仮身が参照する実身のデバイスのウィンドウへ戻した場合は、間接参照表示が解除される。

仮身の表示の大きさ、デザイン等の詳細は実身 / 仮身マネージャのインプリメントに依存する。

#### タイトルバー

閉じた仮身、および開いた仮身の最上行には、ピクトグラム、実身名、続柄、タイプ、変更日時の項目が表示される。各項目は指定した文字サイズに対応した大きさの文字で表示され、文字サイズに応じて全体の表示領域の大きさが決定される。

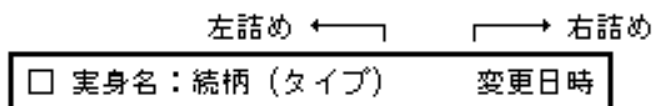


図 116: タイトルバー

実身名、続柄、タイプはそれぞれ仮身に付随する文字列であり、実身名と続柄の間には、":"が入り、タイプは "(" と ")" で括られる。

変更日時は、実身の内容を変更した最新の日時が表示される。通常、変更日時は指定した文字サイズの縦横半分の大きさで表示される。

仮身の長さが短い場合は、右よりのものから見えなくなる。

通常、これらの表示項目はすべて表示するが、仮身の属性として外枠も含めて表示しない項目を設定することが可能である。

虚身の場合は、ピクトグラムを含めたタイトルバーの表示は灰色化表示となる。

虚身でない場合、参照する実身が実際には存在しない仮身は、ピクトグラムの部分に斜線が引かれる。

| □ 組み込みコマンド:一覧 |

| □ 組み込みコマンド:一覧表 (OAファイ) |

| □ 組み込みコマンド:一覧表 (OAファイラリスト) 85 |

| □ 組み込みコマンド:一覧表 (OAファイラリスト) 85/12/25 16:42 |

図 117: タイトルバーの表示例

## 表示エリア

開いた仮身の場合のみ存在し、仮身の内側の境界線で囲まれた領域であり、参照する実身の内容が対応するアプリケーションにより表示される。

表示エリア内に表示されている仮身(内部仮身)の場合は、仮身としての各種の操作ができないことを区別するために、仮身全体を囲う境界線を細い点線で描く。

内容を表示するアプリケーションが存在しない場合、またはアプリケーションが表示を行なえなかった場合は、表示エリアに左上から右下への対角線が引かれ、表示エリアの表示ができなかったことを示す。

同様に、開いた虚身の場合も内容の表示ができないため、表示エリアに左上から右下への灰色の対角線が引かれる。この場合、タイトルバーの表示は灰色化表示となっている。

すべての項目を表示しない属性の仮身の場合は、開いた仮身のタイトルバー部分は表示されなくなる。さらに外枠も表示しない属性の場合には、表示エリアのみが表示され、境界線を含めた外側は表示されない。

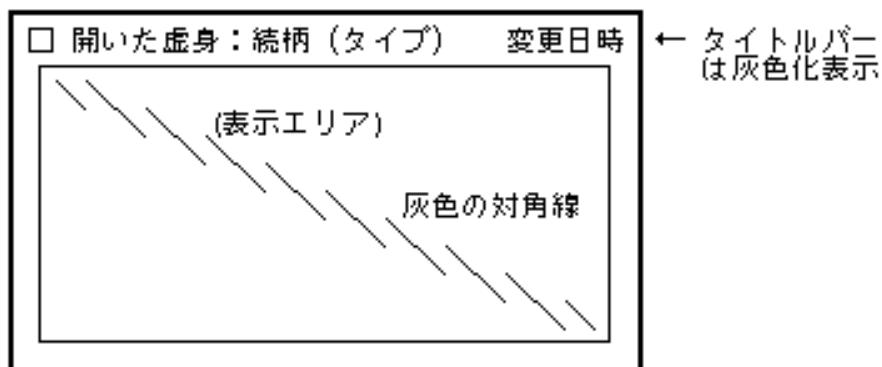


図 118: 開いた虚身の表示

## ハンドル

ハンドルは仮身の4隅に存在する領域であり、ドラッグして仮身を変形するために使用される。ユーザが仮身の長さを変える場合は、水平にドラッグして変形を行ない、「開いた仮身」にする、または「閉じた仮身」に戻す場合は、水平方向以外にドラッグして変形を行なう。

ハンドルには特に境界線はないが、アプリケーションは仮身が選択状態のときは、ポインタがその上に来ると「変形手」に変化させ、ドラッグにより変形できることを示す。

ハンドルによる変形は、ドラッグしたハンドルと対角位置の点により決定される矩形領域となるが、仮身の最小長さ、および最小高さより小さくなるような変形はできない。仮身の最小高さは

閉じた仮身の高さであり、最小長さは実身 / 仮身マネージャのインプリメントに依存する。また、開いた仮身の最小高さもインプリメントに依存し、これよりも小さい高さの開いた仮身とすることはできない。

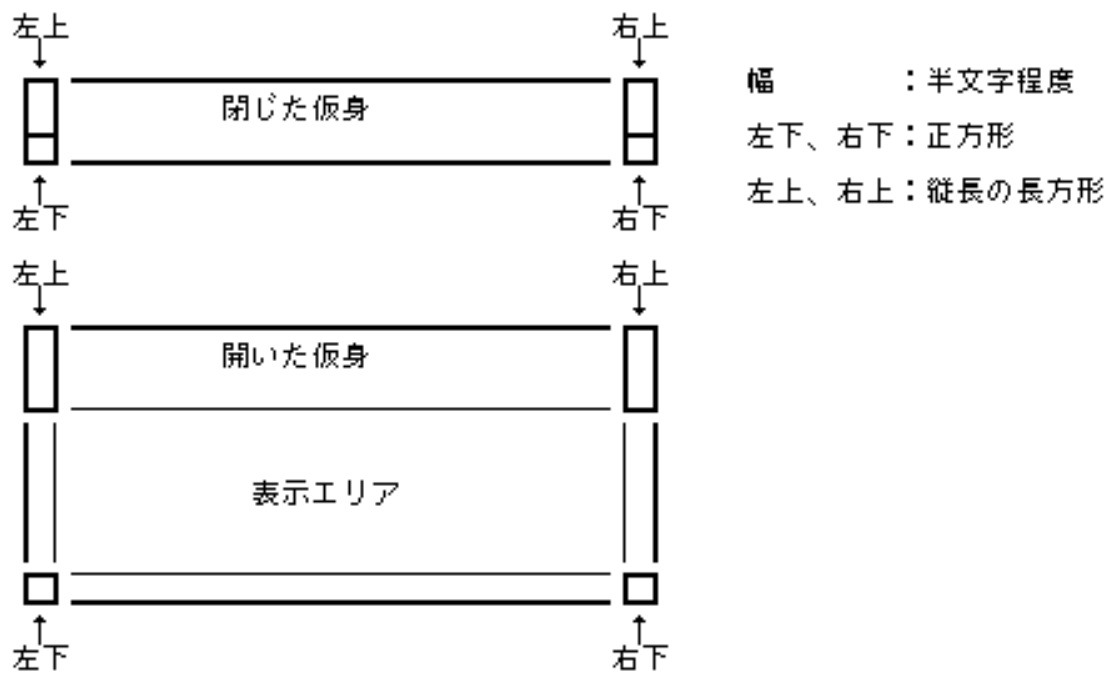


図 119 : 仮身内でのハンドルの位置

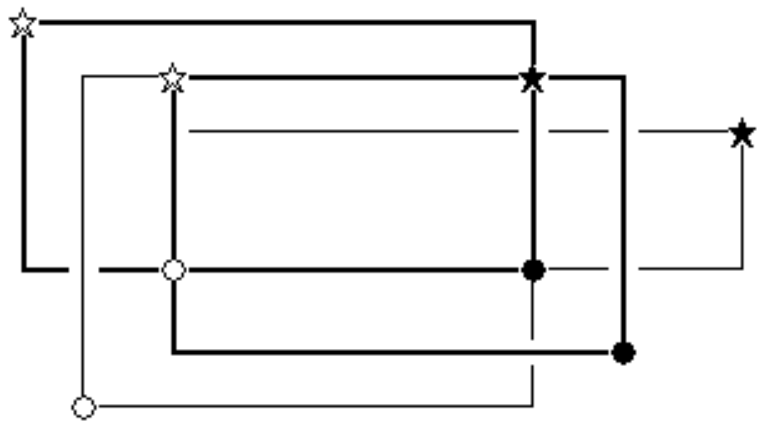


図 120 : ハンドルによる変形

境界線  
 仮身は、境界線により囲まれているが、表示エリア内に表示されている仮身 ( 内部仮身 ) は、仮身としての操作ができないことを示すため、境界線を細い点線とする。

境界線の四隅はハンドルの一部としてドラッグして変形するための手掛かりとなる。

仮身の境界線の内部は、ハンドルの部分を除いてドラッグして移動するための手掛かりとなり、アプリケーションは仮身が選択状態のときはポインタがその上にくると「移動手」とする。

### 3.8.1.4 仮身の操作

#### 仮身の操作

仮身に対する操作としては、以下のものがあげられる。これらの操作は、ポインティングデバイスにより画面上で直接的に指示される場合と、 `oget_men()` または `oget_vmn()` によって得られる

「仮身 (/ 実身 / ディスク) 操作」メニューにより指示される場合がある。

#### メニュー項目

|                       |               |
|-----------------------|---------------|
| (1) 仮身の選択             | ----          |
| (2) 仮身の切断             | 切り離し          |
| (3) 仮身の移動 / 複写        | ----          |
| (4) 仮身の変形             | ----          |
| (5) 仮身を開く             | 開く            |
| (6) 仮身を閉じる            | 閉じる           |
| (7) 実身名の変更            | 実身名変更         |
| (8) 続柄の変更             | 続柄変更          |
| (9) 新版作成              | 新版作成(または実身複製) |
| (10) 実身管理情報の表示        | 管理情報          |
| (11) 仮身をウィンドウに開く      | (実行メニュー)      |
| (12) デバイスの再フォーマット     | フォーマット        |
| (13) 仮身の属性の変更         | 属性変更          |
| (14) アプリケーションの登録 / 削除 | アプリ登録 / 削除    |
| (15) トレーからのデータ貼り込み    | トレーから書込み      |
| (16) 仮身の接続            | 接続            |
| (17) ディスク状態の表示        | 状態表示          |
| (18) ディスク整理の実行        | ----          |
| (19) ユーティリティの実行       | (ユーティリティ名)    |

- 「メニュー項目」の実際の表記は実身 / 仮身マネージャのインプリメントに依存する。

なお、開いた仮身の表示エリアに表示されている仮身 (内部仮身) に対しては、上記の仮身操作は一切できない。

以下は、仮身に関するアプリケーションの標準的な操作方法の説明であり、実身 / 仮身マネージャでは、これらの操作を行なうための基本的な機能を提供している。実際の操作に伴う処理の細部はアプリケーションのインプリメントによって異なる場合がある。

#### 仮身の選択

- 仮身の領域内をポインティングデバイスでプレスするか、または選択フレームで囲むことにより仮身を「選択状態」とし、周囲に `adsp_slt()` または `adsp_sel()` を用いて、チラツキ枠を表示する。「選択状態」は、その仮身を表示しているアプリケーションのコマンド (メニュー) の対象であることを示すものであり、複数の仮身を同時に「選択状態」としてもよい。
- 選択状態の仮身では、ポインタの形状を以下のように変化させる。
  - ピクトグラム部分 -- 「選択指」
  - ハンドルの部分 -- 「変形手」
  - その他の部分 -- 「移動手」
- 選択状態でない仮身を、プレスしてドラッグした場合は、プレスの時点で選択状態とし、「移動手」または「変形手」を一瞬表示した後、「握り」または「つまみ」に変化させ、ドラッグを行なう。

#### 仮身の切断



- 切断したい仮身を選択し、「仮身操作」メニューの「切り離し」を指定された場合、`oexe_vmn()` を実行すると、選択状態のすべての仮身が参照する実身が存在するデバイス (ファイルシステム) が切断状態となる。
- 「切断状態」となった仮身は、虚身となり灰色化表示される。同時に、切断したデバイスに含まれる実身を参照する他のすべての仮身も虚身となる。
- 開いた仮身は開いた虚身となり、表示エリアはクリアされて左上から右下への灰色の対角線が表示される。
- 虚身となった仮身から開かれていたウィンドウは、虚身ウィンドウとなり、ウィンドウのタイトルは自動的に灰色化表示となる。

### 3.8.2 仮身の移動 / 複写

- 仮身のハンドル部以外 (表示エリアを含む) をポイントし (この時点で仮身が選択状態ならば、ポインタ形状を「移動手」にする。ただし、ピクトグラム上では「選択指」にする)、ドラッグすることによって移動を行なう。リリース時にメニューボタンが押下されていれば複写を行なう。
- ドラッグ中はポインタを「握り」にする。ただし、ピクトグラム上ではプレスの時点で「移動手」を一瞬表示した後「握り」にする。仮身の「影」を `adsp_slit()` または `adsp_sel()` を用いて、ドラッグにあわせて移動させる。「影」は仮身を囲む大きさの点線の矩形枠である。
- ボタンをリリースした時点で、ポインタを「移動手」、または「選択指」に戻し、仮身の位置を決定し、`omov_vob()` / `odup_vob()` を用いて、移動 / 複写を行なう。移動 / 複写移動後の仮身は選択状態を維持する。
- 「処理中状態」の仮身を複写した場合、複写された仮身は「処理中状態」でなく通常の仮身となる。

#### 仮身の変形

- 仮身のハンドル部をポイントし (この時点で仮身が選択状態ならば、ポインタ形状を「変形手」にする)、ドラッグすることによって変形を行なう。ドラッグ中はポインタを「つまみ」にする。仮身の「影」を `adsp_slit()` または `adsp_sel()` を用いて表示し、ドラッグにあわせて変形させる。
- ボタンをリリースした時点で、ポインタを「変形手」に戻し、仮身の形を決定し、`orsz_vob()` を用いて変形を行なう。変形後の仮身は選択状態を維持する。
- 水平方向の変形は仮身の長さを変えるだけであるが、水平以外の方向の変形は、仮身を開いたり閉じたりすることになる。
- 仮身の変形が親ウィンドウの枠を越えた場合の処理はアプリケーションの仕様の範囲である。
- 虚身は変形できないが、閉じることは可能とする。
- 仮身は、実身 / 仮身マネージャのインプリメント依存の最小サイズ以下にはすることはできない。また、開いた仮身を、その最小サイズ以下にした場合は、`orsz_vob()` によって閉じた仮身となる。

#### 仮身を開く

仮身は、変形により開くことができるが、変形以外にもメニュー指定により開くことができる。

- 開きたい「閉じた仮身」を選択状態とし、「仮身操作」メニューの「開く」を指定された場合、`oexe_vmn()` を実行すると、選択状態のすべての閉じた仮身が開かれ「開いた仮身」となる。
- 表示エリア内部にデフォルトアプリケーションにより最後に閉じたときの内容が表示される。開いた仮身の選択状態は維持する。
- デフォルトアプリケーションを起動できなかった場合は、表示エリアには左上から右下への対角線が表示される。
- 虚身は `oexe_vmn()` によって開かれることはない。

#### 仮身を閉じる

仮身は、変形により閉じることができるが、変形以外にもメニュー指定により閉じることができる。

- 閉じたい「開いた仮身」を選択状態とし、「仮身操作」メニューの「閉じる」を指定された場合、`oexe_vmn()` を実行すると、選択状態のすべての開いた仮身が閉じられ「閉じた仮身」となる。
- タイトルバーを残し表示エリアが消え、仮身が閉じられる。閉じた仮身の選択状態は維持する。

#### 実身名の変更

- 仮身の選択後、「仮身操作」メニューの「実身名変更」を指定された場合、`oexe_vmn()` を実行する。
- 「実身名変更パネル」が表示され、パネル内のテキストボックスにより実身名を変更することができる。複数の仮身が選択されている場合は、それぞれの仮身に対して「実身名変更パネル」が順に表示される。
- 同一の実身を指す他の仮身が表示されている場合、その実身名の表示も同時に変更される。
- 仮身の選択状態は維持する。

#### 続柄の変更

- 仮身の選択後、「仮身操作」メニューの「続柄変更」を指定された場合、`oexe_vmn()` を実行する。
- 「続柄変更パネル」が表示され、新しい続柄を選択することができる。適当な続柄がない場合、既存の続柄の変更または新しい続柄を追加することができる。複数の仮身が選択されている場合は、それぞれの仮身に対して「続柄変更パネル」が順に表示される。
- 既存の続柄を変更したときは、同一の続柄を持つ他の仮身が表示されている場合、その続柄の表示も同時に変更される。
- 仮身の選択状態は維持する。

#### 新版作成

仮身の複写では同一の実身を参照する仮身が複写されるだけであり、実身は複写されないため、「新版作成」により実身を複写する。

- 複写したい実身を参照する仮身を選択状態とし、「仮身操作」メニューの「新版作成」を指定された場合、`oexe_vmn()` を実行する。あるいは、仮身をクリックドラッグにより移動し

た場合、 `ocre_obj()` を実行する。

- 選択した仮身がフロッピーディスクのデバイス仮身の場合は、「デバイス複製パネル」が表示され、デバイスの複製か新版作成かを選択できる。デバイスの複製を選択した場合は、「デバイス接続パネル」が表示され、接続したフロッピーディスクに選択したフロッピーディスクの複製が生成される。接続したフロッピーディスクの元の内容は削除される。通常、この機能はフロッピーディスクドライブが2台以上装着されているときにサポートされる。
- 新版作成の場合、「新版作成パネル」が表示され、パネル内のテキストボックスにより複写する新版の実身名を設定することができる。テキストボックスにはデフォルトとして元の実身名が設定されている。
- パネル内のスイッチにより新版作成を開始すると、仮身が参照する実身を元として参照している実身全てが仮身の所属デバイス上に複写されて、完全に新しい実身群が生成される。
- 複写する実身群に含まれる間接参照仮身の参照先のデバイスが接続されている場合は、「間接参照パネル」が表示され、間接参照を解消して参照先の実身を複写するか、間接参照仮身のままとするかを指定できる。
- 複写した新版または複製したフロッピーディスクを参照する仮身が、選択した仮身の近くに新たに生成されるので、アプリケーションはその仮身の表示を行なわなければならない。元の実身の選択状態は維持する。
- 複数の仮身が選択されている場合は、それぞれの仮身に対して「新版作成パネル」が順に表示されて新版作成を行なうことができる。

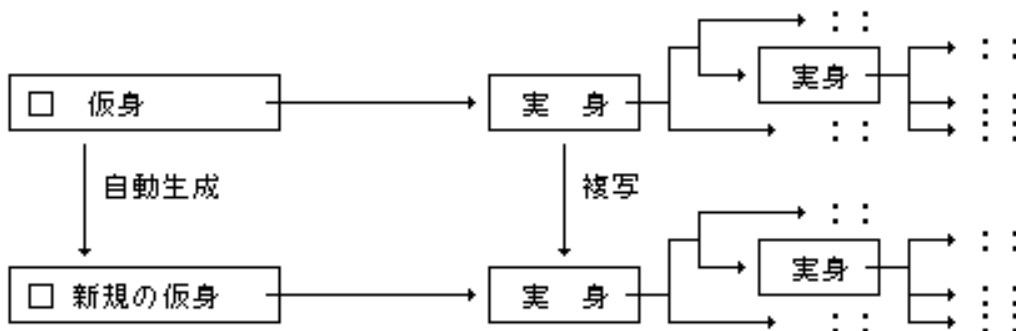


図 121 : 新版作成

#### 実身管理情報の表示

- 管理情報を表示したい実身を参照する仮身を選択し、「仮身操作」メニューの「管理情報」が指定された場合、`oexe_vmn()` を実行する。
- 「管理情報ウィンドウ」が開かれ、その内部に実身管理情報が表示される。このウィンドウにより一部の情報の変更もできる。
- 複数の仮身が選択されている場合は、それぞれの仮身に対して「管理情報ウィンドウ」が同時に表示される。
- 仮身の選択状態は維持する。

「管理情報ウィンドウ」には、以下に示すような情報が表示され、一部の変更が可能となる(詳細はインプリメントに依存する)。

#### 変更の可否

実身情報 :

|         |   |
|---------|---|
| 実身名     | - |
| 続柄      | - |
| 所属デバイス名 | - |
| デバイス所在  | - |
| 実身サイズ   | - |
| 実身合計サイズ | - |
| 作成日時    | - |
| 変更日時    | - |
| 参照する仮身数 | - |
| 含む仮身数   | - |

使用者管理：

|                 |   |
|-----------------|---|
| 所有者名            | - |
| 所属グループ名         | - |
| 所有者アクセスモード      | 可 |
| 所属(グループ)アクセスレベル | 可 |
| 一般アクセスレベル       | 可 |
| 合言葉             | 可 |
| 書込不可属性          | 可 |
| 削除不可属性          | 可 |

付箋指定：

|                 |   |
|-----------------|---|
| データタイプ          | - |
| 適用機能付箋一覧        | 可 |
| デフォルトアプリ        | 可 |
| デフォルトアプリ固定 / 最新 | 可 |

仮身をウィンドウに開く

- ウィンドウに開きたい仮身を選択状態とし、「実行」メニューの中の適用するアプリケーションを指定された場合、oexe\_apg() を用いて、指定されたアプリケーションの起動を行なう。
- ウィンドウに開きたい仮身のピクトグラム (または、仮身の任意の箇所) がダブルクリックされた場合は、oexe\_apg() を用いて、その仮身のデフォルトアプリケーションの起動を行なう。
- ウィンドウが開かれ、開いたウィンドウは入力受付状態となる。ウィンドウの大きさ / 位置は、起動されたアプリケーションに依存する。
- 起動すべきアプリケーションの起動ができなかった場合は、その旨のエラーパネルが表示される。ただし、インプリメントによっては、起動すべきアプリケーションが見つからなかった場合は、アプリケーションの機能付箋名をパネルに表示して、アプリケーションの入ったデバイスの接続を要求する場合もあり、その場合はデバイスの接続が行なわれた後にアプリケーションが起動される。
- 起動すべきアプリケーションが複数存在する場合は、最新のアプリケーションを起動するか、または、「アプリケーション選択パネル」により起動するアプリケーションを選択する (どちらかはインプリメントに依存する)。
- 「処理中状態」の仮身のピクトグラム (または、仮身の任意の箇所) がダブルクリックされた場合は、oexe\_apg() を実行すると、処理中のウィンドウが入力受付状態になり、新たに

アプリケーションが起動されることはない。なお、「処理中状態」の仮身の「実行」メニューは不能状態となる。

- 虚身をダブルクリックによりウィンドウに開こうとした場合も、`oexe_apg()` を実行するとデバイスの接続を求める「デバイス接続パネル」が表示され、デバイスの接続が行なわれた後にデフォルトアプリケーションが起動される。
- 複数の仮身を選択している場合の動作はアプリケーションのインプリメントに依存する。

#### デバイスの再フォーマット

- 再フォーマットを行ないたい仮身を選択状態とし、「仮身操作」メニューの「フォーマット」を指定された場合、`oexe_vmn()` を実行する。この項目は接続状態のデバイス仮身に対してのみ `oget_men()` または `oget_vmn()` で取り出される。
- 「フォーマットパネル」が表示され、指定により「フォーマット」を開始する。
- フォーマット中の動作はシステムで用意されるフォーマッタに依存する。
- フォーマット後は、元のデバイス仮身は新たに再フォーマットしたデバイス仮身となり、元のデバイスを参照していた他の仮身は虚身となる。
- 一般に、複数の仮身を選択している場合は、この操作は適用されない。

#### 仮身の属性の変更

- 属性を変更したい仮身を選択し、「仮身操作」メニューの「属性変更」を指定された場合、`oexe_vmn()` を実行する。
- 「属性変更パネル」が表示され、以下のような属性を変更できる(詳細はインプリメントに依存する)。

##### 表示項目

ピクトグラム、実身名、続柄、データタイプ、更新日時、仮身枠、それぞれの表示の有無

##### 表示色

枠、タイトル文字、タイトル背景、表示エリア背景、それぞれの表示色

##### 文字サイズ

文字の大きさ

##### その他

展開印刷、自動起動などの指定

- 一般に、複数の仮身を選択している場合は、すべて同一属性に変更される。

#### アプリケーションの登録 / 削除

- アプリケーションの登録または、削除を行ないたい仮身を選択し、「仮身操作」メニューの「アプリ登録」または「アプリ削除」を指定された場合、`oexe_vmn()` を実行する。この項目は実行可能な実身を参照する仮身に対してのみ `oget_men()` または、`oget_vmn()` で取り出される。
- 選択した仮身が参照する実行可能な実身を、その仮身の所属デバイス上のアプリケーションとして登録、または削除する。
- 一般に、複数の仮身を選択している場合は、この操作は適用されない。

## トレーからのデータ貼り込み

- トレーからのデータ貼り込みを行ないたい仮身を選択し、「仮身操作」メニューの「トレーから書込」を指定された場合、`oexe_vmn()` を実行する。
- 選択した仮身が参照する実身のデフォルトアプリケーションを起動して、データ貼り込みが要求される。トレー内のデータがどのように貼り込まれるかはデフォルトアプリケーションに依存する。
- 貼り込みができなかった場合は、その旨のエラーパネルが表示される。
- トレーの内容は変化しない。
- 複数の仮身を選択している場合の動作はアプリケーションのインプリメントに依存する。

## 仮身の接続

- 接続したい仮身(虚身)を選択状態とする。複数の仮身を同時に選択状態としても良い。
- 「仮身操作」メニューの「接続」を指定することにより、デバイスの接続を求める「デバイス接続パネル」が表示され、仮身の所属デバイスの接続を行なうことができる。取り外しできないデバイスの場合は、「デバイス接続パネル」は表示されずに、所属デバイスが接続される。
- 「仮身操作」メニューの「接続」を指定することにより、仮身の所属デバイスの接続を行なうことができる。所属デバイスが取り外し可能な場合は、デバイスの接続を求める「デバイス接続パネル」が表示される。
- 仮身の接続は、デバイスを直接的に接続したり、実行等の他の操作に伴って行なうことができる。

## ディスク状態の表示

- ディスク状態を表示したいディスク上の仮身を選択状態とする。
- 「仮身操作」メニューの「ディスク状態」を指定することにより、「ディスク状態パネル」が表示される。
- 「ディスク状態パネル」には以下のような項目が表示される(詳細はインプリメントに依存する)。

ファイルシステム(ディスク)名  
全体サイズ (K バイト)  
使用サイズ (K バイト)  
空きサイズ (K バイト)  
使用比率 (%)  
使用実身数  
くず実身数  
[ディスク整理]の実行スイッチ

- くず実身は、参照する仮身が1つも存在しない状態の実身を意味する。

## ディスク整理の実行

- 「ディスク状態パネル」を表示して[ディスク整理]を指定することにより、「ディスク整理」ウィンドウが開かれ、その内部にくず実身が表示される。このウィンドウによりくず実身の削除、回収が可能となる。

## ユーティリティの実行

- 対象とする仮身を選択状態とする。
- 「仮身操作」メニューに表示される、ユーティリティ名を指定することにより、選択した仮身に対して、そのユーティリティが実行される。
- 具体的にどのようなユーティリティが提供されるかはインプリメントに依存する。

### 3.8.2.1 付箋

#### 付箋の種類

付箋は大きく以下に示す種類がある。

#### 機能付箋：

アプリケーションプログラム起動用(アプリケーションプログラムファイル側で保持する)

#### 実行機能付箋：

アプリケーションプログラム起動用(起動対象実身側で保持する)

#### 小物付箋：

小物アプリケーションプログラム起動用

#### 設定付箋：

システムの設定変更用

#### 指定付箋：

アプリケーションデータ指定用

指定付箋以外は、いずれもアプリケーションプログラムを直接参照し、付箋の処理のために基本的に対応するアプリケーションプログラムの起動を必要とする。

実身/仮身マネージャでは、指定付箋以外の付箋に関する表示/操作/実行処理のための各種の関数を提供している。

#### 付箋の表示

付箋は仮身とほぼ同様に表示されるが、続柄、および変更日時は定義されない。また、ピクトグラムは、通常付箋を示す共通シンボルであるが、付箋独自のピクトグラムとする場合もある。

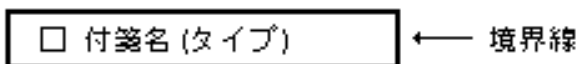


図 122 : 付箋の表示

#### 付箋の操作

付箋に対する操作としては、以下のものがあげられる。これらの操作は、ポインティングデバイスにより画面上で直接的に指示される場合と、`oget_men()` または `oget_vmn()` によって得られる「付箋操作」メニューにより指示される場合がある。

#### メニュー項目

- |                |       |
|----------------|-------|
| (1) 付箋の選択      | ----  |
| (2) 付箋の移動 / 複写 | ----  |
| (3) 付箋の変形      | ----  |
| (4) 付箋名の変更     | 付箋名変更 |

(5) 付箋の属性の変更 属性変更

(6) 付箋の実行 起動

- 「メニュー項目」の実際の表記は実身 / 仮身マネージャのインプリメントに依存する。

以下は、アプリケーションの付箋に関する標準的な操作方法の説明であり、実身 / 仮身マネージャでは、これらの操作を行なうための基本的な機能を提供している。実際の操作に伴う処理の細部はアプリケーションのインプリメントによって異なる場合がある。

付箋の選択：

仮身の場合と同様。

付箋の移動 / 複写：

仮身の場合と同様。

付箋名の変更：

仮身の場合の「実身名の変更」と同様。

付箋の変形：

仮身の場合と同様であるが、縦に変形することはできない。

付箋の属性の変更：

仮身の場合の「属性の変更」と同様であるが、以下のような項目となる。

表示項目

付箋名、データタイプ、それぞれの表示の有無

表示色

枠、タイトル文字、タイトル背景、それぞれの表示色

文字サイズ

文字の大きさ

その他

自動起動などの指定

付箋の実行：

仮身の場合の「ウィンドウに開く」と同様であり、`oexe_apg()` によって付箋の参照するアプリケーションが起動される。通常はアプリケーションによりセッティングパネルが表示されることが多い。

### 3.8.3 実身 / 仮身マネージャの詳細

#### 3.8.3.1 仮身

仮身の分類

仮身は、内部的にはファイル管理で定義されている「リンク」により表現され、その取り扱い上、以下のように分類される。

- 通常仮身：

ウィンドウ内に表示され、標準の属性 / 操作が適用される仮身。

- 内部仮身：

開いた仮身の表示エリア内に表示される、単なる静的な表示としての仮身。内部仮身は、選択できず直接的に操作の対象となり得ず、通常、仮身の境界線は細い点線で描かれる。開いた内部仮身内にネスティングしている仮身も全て内部仮身となる。

- 隠蔽仮身：



通常の状態では表示されない仮身であり、システムで内部的に使用される実身に対して使用する。ただし、実身/仮身マネージャでは隠蔽仮身も通常の仮身と同様に取り扱う。

今後、単に仮身という場合は通常仮身を示すものとする。

実身/仮身マネージャでは仮身を、指定したウィンドウ/パネル上に存在する一種のパーツとして登録し、登録した仮身に対して仮身ID ( vid > 0 ) を割り当てて管理する。アプリケーションは登録時に得られる仮身IDを使用して、仮身の表示/操作を行なうことになる。ただし、内部仮身のサポートのため、仮身を登録せずに直接表示する機能も提供している。

### 仮身のデータ構造

仮身は、内部的に以下に示す仮身リンク構造体により表現される。このデータ構造体は、ファイル管理で使用している LINK 構造体と同一であるが、フィールドの定義が異なっている。

```
typedef struct {
    TC fs_name[20]; /* ファイルシステム名 */
    UH f_id;        /* ファイルID */
    UH attr;        /* 仮身タイプ/属性 */
    UH rel;         /* 続柄インデックス */
    UH appl[3];     /* アプリケーション ID */
} VLINK;
```

fs\_name と f\_id は仮身の対象とするファイルを示すもので、核のファイル管理レベルで使用され、それ以外のデータはファイル管理レベルでは使用しないデータであり、仮身固有のデータとして以下のように定義される。

attr :

仮身属性 / 状態を示す以下の内容のものである。

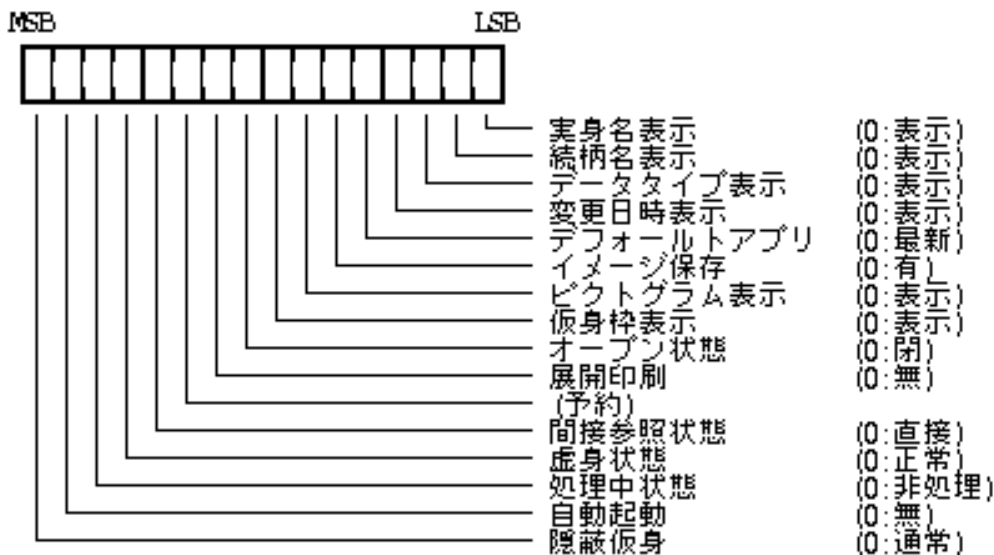


図 123 : 仮身の属性 / 状態

- 隠蔽仮身が "1" の場合は、隠蔽仮身を示し、通常の状態では表示されないことを意味する。この処理はアプリケーションレベルで行なわれ、実身/仮身マネージャでは、通常の仮身と同様に取り扱う。
- 自動起動が、"1" の場合は、仮身を登録すると同時に自動的にデフォルトアプリケーションを実行することを意味する。

- 虚身状態が "1" の場合は、参照するデバイスが切断状態であることを示し、虚身としての表示が行なわれる。虚身状態は動的な状態として定義される。
- 処理中状態が "1" の場合は、仮身がウィンドウに開かれている状態であることを示し、処理中状態としての表示が行なわれる。処理中状態は動的な状態として定義される。
- 間接参照状態が "1" の場合は、その仮身が他のデバイス上のファイルを参照していることを示し、"0" の場合は、同一デバイス上のファイルを直接参照していることを示す。
  - 虚身状態、処理中状態、間接参照状態は、実身 / 仮身マネージャ内部で設定される状態であり、仮身の登録時の値は意味を持たない。
- 展開印刷が、"1" の場合は、印刷するときに仮身を展開してその内容を印刷することを意味する。
- オープン状態が "0" の場合は、閉じた仮身、"1" の場合は開いた仮身であることを示す。
- イメージ保存が "0" の場合は、開いた仮身の表示イメージをできるだけ保存し、再表示を高速に行なうことを意味し、"1" の場合は、開いた仮身の表示イメージは保存しないことを意味する。
- デフォルトアプリが、"0" の場合、最新実行のアプリケーションをデフォルトアプリケーションとすることを示し、"1" の場合は、デフォルトアプリケーションは固定であることを示す。
- ピクトグラム表示、仮身枠表示、実身名表示 ~ 変更日時表示は、それぞれ対応する項目を仮身として表示するか否かを示すもので、"0" で表示することを示す。

appl :

仮身にデフォルトとして適用されるアプリケーションプログラムのアプリケーション ID を示す 3 つのハーフワード (16 ビット) の配列である。

アプリケーション ID は全てのアプリケーションプログラムに対して付けられたユニークな ID であり、最初の 2 つが適用するデータタイプを示し、3 つ目は同一データタイプに適用されるアプリケーションを区別するための識別番号となる。

データタイプ (最初の 2 つ) が 0 のときは、参照する実身(ファイル)内に存在する先頭の実行機能付箋がデフォルトアプリケーションであると見なされる。

ただし、3 つ目の 0x4000 に対応するビットが "1" の場合は、そのアプリケーションはデフォルトアプリケーションとなることはできない。

rel :

仮身の続柄インデックスであり、仮身の所属デバイスに含まれている続柄ファイル内のインデックスを示す。この続柄インデックスにより、続柄名の表示、および仮身内での続柄名の表示カラーが定義される。

仮身セグメント

仮身自体は VLINK 構造体により表わされるが、仮身を実際に実行したり表示するために VLINK 構造体と対になる「仮身セグメント」が必要となる。

仮身セグメントは以下に示すデータ構造を持つ。

```
typedef struct {
```

```
RECT    view;      /* 表示領域 */
H       height;   /* 開いた場合の高さ */
CHSIZE  chsz;     /* 文字サイズ */
COLOR   frcol;    /* 枠の色 */
COLOR   chcol;    /* 文字色 */
COLOR   tbcoll;   /* タイトル背景色 */
COLOR   bgcol;    /* 開いた場合の背景色 */
UH      dlen;     /* 固有データのバイト長 */
                          /* dlen バイトの固有データが続く */
```

```
} VOBJSEG;
```

view:

仮身の実際の表示領域をウィンドウ / パネルの相対座標で示す。

chsz:

仮身の文字サイズを示す以下の値。

UUSS SSSS SSSS SSSS

U:

単位指定

0:

ピクセル単位

1:

1/20 級(1/80 mm)単位

2:

1/20 ポイント(1/1440 inch)単位

3:

予約

S:

Uで指定された単位での文字サイズ指定。0の場合は、実身 / 仮身マネージャで規定するデフォルト文字サイズを意味する。

height:

仮身を開いた場合のデフォルトの仮身の高さをウィンドウ / パネルの座標系で示す。 < 0 の場合は、その仮身は開けないことを示す。また 0 の場合は実身 / 仮身マネージャで規定するデフォルトの大きさで開くことを示す。

frcol:

仮身枠の色を示す。

chcol:

仮身のタイトルバーの文字色を示す。

tbcoll:

仮身のタイトルバーの背景色を示す。

bgcol:

開いた仮身の表示エリアの背景色を示す。

dlen:

デフォルトアプリケーションで使用する固有データのバイト数を示す。このバイト数のデータが直後に続くことになる。固有データは、デフォルトアプリケーションの機能付箋に含まれているものと同じのデータ構造の内容を持ち、デフォルトアプリケーションを起動して仮身をウィンドウに開いた場合にアプリケーションに渡されるデータである。

仮身セグメントは、固有データを含んだ1つのデータ構造であり、VLINK構造体と対になり仮身を定義するために使用される。

### 続柄

続柄は仮身に対して任意に付けることのできる一種の属性としての名前(最大16文字)である。

続柄はファイルシステム(デバイス)単位で定義され、仮身の参照する実身(間接参照仮身の場合はリンクファイル)の存在するファイルシステムのルートファイル直下に存在する「続柄管理ファイル」に以下のデータが保持され、そのインデックス(0~)が仮身のデータとして格納される。通常、インデックス0は続柄名なし(全て0)となっている。

```
typedef struct {
    TC      name[16]; /* 続柄名 */
    COLOR  col;      /* 続柄名の色 */
} VOBJREL;
```

「続柄管理ファイル」の内容は、仮身操作の「続柄変更パネル」により登録/削除することが可能である。

続柄はファイルシステム単位で定義されているため、仮身を移動/複製した後、異なるファイルシステム内へ保存する場合、同一の続柄名とするために、続柄インデックスの変換、および続柄名の新規登録を行なう必要があり、このための関数 `ocnv_vob()` が用意されている。この場合、「続柄管理ファイル」への新規登録のみが行なわれ、削除は一切行なわれない。つまり、登録した続柄名の削除は仮身操作の「続柄変更パネル」によつてのみ行なうことが可能となる。

### 実身のタイプ

実身の基本データタイプは、ファイルのアプリケーションタイプ (`atype`) として定義される。ファイルのアプリケーションタイプはファイルの生成時に指定するファイル管理レベルでは使用されないデータであり、以下の内容を持つ。



図 124 : 実身のアプリケーションタイプ

ピクトグラムデータ番号は、データボックスに定義されている "PICT\_DATA" タイプのデータ番号であり、対応する仮身やウィンドウのピクトグラムとして使用される。

上位バイトは、そのファイルの属性を示し、実身/仮身マネージャで使用される。特に実身のタイプが「実行可能ファイル」タイプでないものはアプリケーションプログラムとしての登録は不可となる。

### 3.8.3.2 付箋

#### 付箋の取り扱い

実身 / 仮身マネージャでは付箋についても、仮身と同様に登録を行ない、登録した付箋に対しての各種の操作を提供する。すなわち、付箋は特殊な仮身として取り扱われ、登録時に得られる ID も仮身 ID として取り扱われる。

実身 / 仮身マネージャとして提供される関数の多くは、仮身と付箋の両方に対して適用可能であり、このためアプリケーションは仮身と付箋の区別をせずに済むようになっている。

#### 付箋のデータ構造

付箋は以下に示す共通的なデータ構造のヘッダを持ち、その直後にアプリケーションの固有データが続いて入ることになる。

```
typedef struct {
    RECT    view;          /* 表示領域 */
    CHSIZE  chsz;         /* 文字サイズ */
    UH      frcol[2];     /* 枠の色 */
    UH      chcol[2];     /* 文字色 */
    UH      tbcoll[2];    /* タイトル背景色 */
    UH      pict;        /* ピクトグラム/タイプ */
    UH      appl[3];      /* アプリケーションID */
    TC      name[16];     /* 付箋名 */
    TC      dtype[16];    /* データタイプ名 */
    UH      dlen;         /* 固有データのバイト長 */
                    /* dlen バイトの固有データが続く */
} FUSENSEG;
```

view :

付箋の実際の表示領域をウィンドウ / パネルの相対座標で示す。

chsz :

付箋の文字サイズを示す。仮身の文字サイズと同じデータ形式となる。

frcol :

付箋枠の色を示す。

chcol :

付箋のタイトルバーの文字色を示す。

tbcoll :

付箋のタイトルバーの背景色を示す。

pict :

付箋表示の場合に使用されるピクトグラムのデータ番号が下位バイトに入る。0 は標準の付箋用ピクトグラムを意味する。上位バイトは付箋の属性を示す。

HAXD NTxx PPPP PPPP

P ピクトグラムのデータ番号

H = 1 隠蔽付箋 ( 隠蔽仮身と同様に実身 / 仮身マネージャでは無視する )

A = 1 自動起動  
D = 1 虚身 ( 灰色化 ) 表示  
N = 1 付箋名表示なし  
T = 1 データタイプ名表示なし  
x 予約

name :

付箋名を示す。

dtype :

データタイプ名を示す。

appl :

付箋が参照するアプリケーションプログラムのアプリケーション ID を示す。

dlen :

続く固有データのバイト数を示す。

付箋は、実身内の1つの独立したレコードとして格納され、レコードタイプは、「実行機能付箋タイプ」または「機能付箋タイプ」となる。サブタイプは特に使用しない。「実行機能付箋タイプ」の場合は、その実身に適用可能なアプリケーションプログラムを意味し、実行メニューとしてその付箋名が表示されることになる。

さらに実行機能付箋のテンプレートとなる付箋がアプリケーションプログラムファイル内の1つの独立したレコードとして格納され、レコードタイプは、「機能付箋タイプ」となる。サブタイプは特に使用しない。

仮身のデフォルトアプリケーションに対応する付箋の固有データは仮身セグメントの中に含まれることになる。

### 3.8.3.3 アプリケーションプログラム管理

#### 実行環境ファイル

ユーザの処理の実行環境となっているファイルを「実行環境ファイル」と呼ぶ。

実行環境ファイルはシステム全体で唯一のものとして定義され、システム立ち上げ時に最初にウィンドウを開くプロセスは、必ず実行環境ファイルを設定しなければならない。また、以降のプロセスは `ochg_env()` によって実行時にネスティングして変更することができる。変更した実行環境ファイルは、スタック状に保持されるため、元の実行環境ファイルに戻る事が可能である。

実行環境ファイルの存在するデバイス ( ファイルシステム ) は、アプリケーションプログラムの検索の対象となる点で重要であり、また、フロッピーディスク等のデバイスを挿入した場合、自動的にその時点の実行環境ファイル上に対応するデバイス仮身が生成されることになる。

実行環境ファイルは、そのファイルを開いているウィンドウと対となって定義され、そのウィンドウを実行環境ウィンドウと呼ぶ。

#### アプリケーションプログラムの登録

アプリケーションプログラムは登録することにより、初めて実行可能となる。登録は、デバイス ( ファイルシステム ) 単位で行なわれ、通常、そのファイルシステム内に存在する全てのアプリケーションプログラムは登録されることになる。

アプリケーションプログラムの登録は、`oreg_apg()` により、そのアプリケーションへのリンクが存在するファイルシステム内のルートファイルの直下に存在する「アプリケーション登録ファイル」に、アプリケーションプログラムのファイルへのリンクを格納することにより行なわれ、同時にアプリケーションのデータタイプ名も格納される。なお、この登録ファイルは通常は仮身として表示されない。

登録するアプリケーションプログラムの所在には、特に制限がなく、別デバイスに存在していてもよい。

アプリケーションプログラムは、3つのハーフワードのユニークな「アプリケーション ID」を持ち、この ID により識別される。アプリケーション ID の最初の 2 つは適用するデータタイプを示し、最後の 1 つは同一データタイプに適用されるアプリケーションを区別するための識別番号となる。

同一のアプリケーション ID を持つ異なるファイルのアプリケーションプログラムを複数個登録することも可能である。

#### アプリケーションプログラムの検索

アプリケーションプログラムは対応する付箋を元に実行される。付箋にはアプリケーション ID が入っており、これを元に登録済みのアプリケーションプログラムが検索されて実行される。

アプリケーションプログラムの検索は、その時点の実行環境ファイルが属するデバイス（ファイルシステム）に登録されているアプリケーションプログラムに対して行なわれ、さらに検索対象デバイス（ファイルシステム）が `oset_sea()` によって設定してあった場合には、設定してある検索対象デバイスに対しても検索される。

なお、`oprc_dev()`、`oatt_vob()` によって接続されたデバイスが、自動的に検索対象デバイスとして設定されるかどうかはインプリメントに依存する。

デフォルトでは、検索対象デバイスは設定されていないため、実行環境のデバイスのみが検索対象となる。

対応するアプリケーションプログラムが複数個登録してあった場合は、最新の更新日時のアプリケーションを自動的に選択するか、「アプリケーション選択パネル」を表示してアプリケーションを選択させるかは、インプリメントに依存する。このような状況はアプリケーションのバージョンアップ時に古いバージョンと新しいバージョンの両方が登録された場合に発生することになる。

なお、異なるアプリケーションであることは、ファイルの総バイト数、総レコード数、ファイル名のいずれかが異なっていることで判断され、これらがすべて同一である場合は、同一のアプリケーションとみなされ、最初に検索されたアプリケーションのみが有効となる。

#### アプリケーションプログラムのファイル構成

アプリケーションプログラムファイルのアプリケーションタイプは実行可能ファイルでなくてはならない。また、以下に示すレコード構成でなくてはならない。実行プログラムレコードは先頭レコードでなくてはならないが、他のレコードの順番は任意である。

< 実行プログラムレコード > 1 個

標準オブジェクト形式の内容

< 機能付箋レコード > 1 ~ N 個

登録時にアプリケーション ID、およびデータタイプ名を取り出すために使用される。この機能付箋は、アプリケーションプログラムに対する付箋のテンプレートとしても使用されるため、固有データ部分はデフォルトとしての標準値となる。

< その他の任意のレコード > 0 ~ M 個

アプリケーション自体で解釈 / 使用する。

### 3.8.3.4 アプリケーションプログラムの起動

#### 起動方法

登録されたアプリケーションプログラムに対しては、以下の 4 種類の起動が行なわれる。

#### (1) 仮身のオープン起動：

指定された仮身に対してウィンドウを開いてアプリケーション本来の機能を実行する。実行メニュー、小物メニューまたはピクトグラム (または、仮身の任意の箇所) のダブルクリックにより起動される。

#### (2) 開いた仮身の表示起動：

指定された開いた仮身の表示エリア内の表示を行ない、表示完了後に終了する。開いた仮身の表示要求により起動される。

#### (3) データ貼り込み起動：

指定された仮身の参照する実身にトレーに格納されているデータを貼り込み、貼り込み完了後に終了する。仮身への貼り込み操作で起動される。

#### (4) 付箋のオープン起動：

指定された付箋に対して対応するアプリケーションを起動する。付箋のピクトグラム (または、付箋の任意の箇所) のダブルクリック、付箋操作メニューにより起動される。

#### (5) 開いた仮身のTADデータ作成起動：

指定された開いた仮身の表示エリア内の表示に相当する TAD データを作成し、データ作成完了後に終了する。

起動時には、各起動タイプに応じた起動メッセージが、起動したプロセスに対して送られるため、アプリケーションは要求された処理を起動メッセージ内のパラメータに従って行なう必要がある。

#### 仮身のオープン起動

仮身のオープン起動では、起動したプロセスに対して、以下の起動メッセージが渡されるので、起動されたプロセスは起動メッセージに従って、対象実身 (ファイル) に対してアプリケーションとしての処理を行なうことになる。

```
typedef struct {
    W      type;          /* = EXECREQ (= 0x10) */
    W      size;         /* メッセージ本体サイズ */
    LINK   self;        /* 起動ファイル */
    LINK   lnk;         /* 対象ファイル */
    W      vid;         /* 対象仮身 ID */
    W      pwid;        /* 親ウィンドウ ID */
    W      info;        /* レコード番号/付箋データポインタ */
    RECT   r;          /* 生成元 */
    COLOR  bgcol;      /* 元の仮身の背景色 */
    W      mode;        /* 起動モード */
} M_EXECREQ;
```

size はメッセージ本体のバイトサイズであり、メッセージ全体のバイトサイズ - sizeof(W) \* 2



の値となる。

self は、起動されたアプリケーションプログラム自体のリンクとなる。LINK 構造体の atr3 ~ atr5 には、起動の対象となったアプリケーション ID がセットされている。

Ink は、起動の対象となった仮身の参照する実身へのリンクとなる。小物アプリケーションの場合は、「小物入れ」ファイルへのリンクとなる。

vid は、仮身のオープン起動の場合、対象仮身 (生成元の仮身) の仮身 ID であり、小物アプリケーションの起動の場合、実身 / 仮身マネージャで内部的に管理している ID (対応する小物メニューの項目番号と 0x4000 との OR をとったもの) が格納される。初期起動のときは、vid = 0 となる。

pwid は、起動の対象となった仮身が登録されているウィンドウ ID であり、小物アプリケーションの場合は 0 となる。

info は、実身 / 仮身マネージャで管理している付箋の固有データへのポインタ、または対応する実行機能付箋のレコード番号となり、前者の場合は、info で指定されたポインタから固有データを取り出し、後者の場合は、起動対象となった実行機能付箋内の固有データ部分を oget\_fsn() を使用して取り出すことになる。小物アプリケーションの場合、「小物入れ」ファイル内の対応する機能付箋のレコード番号となり、対応する機能付箋の固有データ部分を oget\_fsn() を使用して取り出すことになる。

info は、付箋全体へのポインタではなく、固有データ部分のみの以下に示す構造体へのポインタである。

```
info ----->UH dlen      -- 固有データのバイトサイズ
                UB dat[dlen] -- 固有データ
```

通常、アプリケーションは、起動時に適用された実行機能付箋の固有データを必要に応じて終了前に oput\_fsn() を使用して自分で更新し、さらに更新した固有データを実身 / 仮身マネージャに終了通知として渡す必要がある。ただし、起動時に指定された固有データを使用するか否か、および終了時に更新するか否かの最終判断はアプリケーションに依存する。

r は、起動の対象となった仮身の表示領域を示し、pwid で示されるウィンドウの相対座標で表される。

bgcol は、起動の対象となった仮身の表示エリアの背景色を示し、通常、ウィンドウの背景色に使用される。

mode は、起動のモードを示す以下の値である。

```
xxxx xxxx xxxx xxxx lxxx xxxx xxxx xxTR
```

R = 0 : info は実身 / 仮身マネージャで管理している付箋の固有データへのポインタとなる (デフォルトアプリケーションの起動時)

= 1 : info は対象ファイル内の実行機能付箋レコードのレコード番号 (実行メニューによるデフォルトアプリケーションでないアプリケーションの起動時)

T = 0 : 仮身オープン起動

= 1 : 小物アプリケーション起動

l = 0 : 通常起動

= 1 : 初期起動 (初期ユーザプロセスの起動)

x : 予約

この起動は、実身 / 仮身マネージャの oexe\_apg() 関数による起動であり、起動されたプロセスは、ウィンドウを開いた後、必ず、osta\_prc() により処理の開始を通知し、実身を更新した後、

ウィンドウを閉じる前に、必ず、`oend_prc()` により処理の終了を通知しなくてはならない。`oend_prc()` を実行した場合は、開いた仮身の表示要求が戻る場合があるため、その場合は、得られた描画環境 ID に対して開いた仮身の表示起動と同様の処理を行なう必要がある。

ウィンドウやパネルを開かないアプリケーションの場合は、`osta_prc()` を実行する必要はないが、`oend_prc()` は必ず実行しなくてはならない。また、何らかのエラーが発生し、`osta_prc()` を実行しなかった場合でも、`oend_prc()` は必ず実行しなくてはならない。

#### 開いた仮身の表示起動

開いた仮身の表示起動では、起動したプロセスに対して、以下の起動メッセージが渡されるので、起動されたプロセスは起動メッセージに従って、開いた仮身の表示エリアの表示を行なうことになる。

```
typedef struct {
    W      type;          /* = DISPREQ (= 0x11) */
    W      size;         /* メッセージ本体サイズ */
    LINK   self;        /* 起動ファイル */
    LINK   lnk;         /* 対象ファイル */
    W      vid;         /* 対象仮身 ID */
    W      pwid;        /* 親ウィンドウ ID */
    W      info;        /* 付箋データポインタ */
    COLOR  bgcol;       /* 元の仮身の背景色 */
    W      gid;         /* 描画環境 ID */
} M_DISPREQ;
```

`info` は、実身 / 仮身マネージャで管理している付箋の固有データへのポインタであり、仮身のオープン起動の場合と同じであるが、処理終了後に固有データを更新する必要はない。

アプリケーションは開いた仮身の表示エリアの表示を、`gid` で指定された描画環境に対して行なう。表示エリアはフレーム長方形として設定されている。通常、この描画環境はメモリ描画環境となっており、`gid` に対する内部ウィンドウをオープンして表示を行うことになる。

この起動は、実身 / 仮身マネージャに対する仮身の表示要求によって間接的に発生する起動であり、起動されたプロセスは、表示を行なった後、必ず、`oend_req()` により、処理の正常終了、または異常終了を通知しなくてはならない。このときは描画環境をクローズする必要はない。通常はその後、プロセスを終了 (`ext_prc()`) することになる。

#### データ貼り込み起動

データ貼り込み起動では、起動したプロセスに対して、以下の起動メッセージが渡されるので、起動されたプロセスは起動メッセージに従って、トレーに格納されたデータの対象実身 (ファイル) への貼り込み処理を行なうことになる。

```
typedef struct {
    W      type;          /* = PASTEREQ (= 0x12) */
    W      size;         /* メッセージ本体サイズ */
    LINK   self;        /* 起動ファイル */
    LINK   lnk;         /* 対象ファイル */
    W      vid;         /* 対象仮身 ID */
    W      pwid;        /* 親ウィンドウ ID */
    W      info;        /* 付箋データポインタ */
} M_PASTEREQ;
```

この起動は、実身 / 仮身マネージャの `oput_dat()` 関数による仮身へのデータ貼り込み要求により発生する起動であり、起動されたプロセスは、貼り込み処理を行なった後、必ず、`oend_req()` により、処理の正常終了、または異常終了を通知しなくてはならない。通常はその後、プロセスを終了 (`ext_prc()`) することになる。

#### 付箋のオープン起動

付箋のオープン起動では、起動したプロセスに対して、以下の起動メッセージが渡されるので、起動されたプロセスは起動メッセージに従ってアプリケーションとしての処理を行なうことになる。

```
typedef struct {
    W      type;    /* = FUSENREQ (= 0x13) */
    W      size;    /* メッセージ本体サイズ */
    LINK   self;    /* 起動ファイル */
    W      vid;     /* 対象付箋 ID */
    W      pwid;    /* 親ウィンドウ ID */
    W      info;    /* 付箋データポインタ */
    RECT   r;       /* 生成元 */
} M_FUSENREQ;
```

各データの内容は、仮身のオープン起動と同様となる。特に `info` は、実身 / 仮身マネージャで管理している付箋の固有データへのポインタであり、付箋全体のポインタではない。

この起動は、実身 / 仮身マネージャの `oexe_apg()` 関数による起動であり、起動されたプロセスは、付箋の固有データを更新した場合は、必ず、`oend_prc()` により付箋データの更新を通知しなくてはならない。

一般に、付箋のオープン起動により起動されたアプリケーションは、必要に応じて起動元のアプリケーションに対して `oreq_prc()` により要求を行なう。この要求は、仮身要求イベントとして付箋のオープン起動を行なったアプリケーションに渡されるので、起動元のアプリケーションは、`orsp_prc()` によりその応答を戻す必要がある。

以下に、付箋のオープン起動で起動されたアプリケーションの典型的な動作例 (例えば、印刷アプリケーションプログラム) を示す。

1. 起動時に得られた付箋セグメントの内容に従って、セッティングパネルを開き、ユーザからの各種のパラメータの設定を受け付ける。
2. パラメータの設定が行なわれた場合は、`oend_prc()` により、付箋固有データの更新を行なう。
3. 「実行」を要求された場合は、`oreq_prc()` により、起動元のプロセスに対して、現在の編集結果を一時ファイルとすることを要求する。この応答により得られた一時ファイルを対象として「実行」としての機能を実施する。通常は、応答が得られた時点でパネルをクローズするが、プロセスはまだ終了しない。
4. 「実行」が要求されなかった場合は、パネルをクローズしプロセスを終了する。

#### 開いた仮身の TAD データ作成起動

指定された開いた仮身の表示エリア内の表示に相当する TAD データを作成し、データ作成完了後に終了する。

開いた仮身の TAD データ作成起動では、起動したプロセスに対して、以下の起動メッセージが渡されるので、起動されたプロセスは起動メッセージにしたがって、開いた仮身の表示エリアの

表示に相当する TAD データを作成し、保存先ファイルに格納する。

<開いた仮身の TAD データ作成起動メッセージ>

```
typedef struct {
    W      type;      /* = TADREQ (= 0x14) */
    W      size;      /* メッセージ本体サイズ */
    LINK   self;      /* 起動ファイル */
    LINK   lnk;       /* 対象ファイル */
    W      vid;       /* 対象仮身 ID */
    W      pwid;      /* 親ウィンドウ ID */
    W      info;      /* 付箋データポインタ */
    COLOR  bgcol;     /* 元の仮身の背景色 */
    LINK   save;      /* TADの保存先ファイル */
    RECT   r;        /* 表示領域 */
} M_TADREQ;
```

この起動は、実身 / 仮身マネージャに対する仮身の TAD データ作成要求によって間接的に発生する起動であり、起動されたプロセスは、データの作成を行った後、必ず、oend\_req() により、処理の正常終了、または異常終了を通知しなくてはならない。このときには保存先ファイルをクローズしておかなくてはならない。通常はその後、プロセスを終了 (ext\_prc()) することになる。

r は開いた仮身の表示領域を表わしている ( lefttop の座標は 0, 0 )。

### 3.8.3.5 メニュー管理

実身 / 仮身マネージャでは、以下に示す共通メニューに対するメニュー項目リストの生成、更新の機能を提供している。

実行メニュー：

指定した仮身が対象とする実身に対して適用可能な付箋名のメニュー。実身に含まれるすべての実行機能付箋レコードの付箋名を順番に並べたメニューであり、実身ファイルの更新が通知された場合は、対応する実行メニューも更新される。

小物メニュー：

現在実行可能な小物アプリケーションの付箋名のメニュー。

「小物入れ」ファイルと呼ばれる特定のファイル群に含まれるすべての機能付箋レコードの付箋名を順番に並べたメニューであり、これらのファイルの更新が通知された場合は、小物メニューも更新される。

「小物入れ」ファイルには oset\_tmf() によって任意のファイルを指定することができる。

インプリメントによっては、指定したファイルと別デバイスのルートファイル直下の同一名称のファイルも自動的に「小物入れ」ファイルとみなす場合もあり、その場合は、小物メニューの内容はデバイスの接続 / 切断に伴って自動的に変化することになる。

仮身操作メニュー：

指定した仮身に対する操作をまとめたメニュー。基本的に固定的メニューであるが、仮身のタイプにより一部異なる。仮身操作メニューは、仮身操作、実身操作、ディスク操作の 3 つのメニューに分割して取り出すこともできる。

付箋操作メニュー：

指定した付箋に対する操作をまとめたメニュー。固定的メニューである。

さらに、上記の各種メニュー項目が選択された場合の、選択番号をパラメータとした処理機能も提供されているため、アプリケーションはメニューの内容に無関係に統一的な処理が可能となる。

## 仮身要求イベント

仮身の操作に伴って、操作の対象となった仮身以外の仮身の表示の更新が必要な場合がある。この場合は、勝手に他のウィンドウの仮身の表示を更新することはできないため、「仮身要求イベント」として、仮身の再表示を要求するイベントが仮身が登録されたウィンドウの管理プロセスに対して送信される。

仮身要求イベントはウィンドウイベントの EV\_REQUEST の 1 つであり、以下の内容となる。このイベントを受信したアプリケーションは速やかにその処理を行なわなくてはならない。なお、これらのイベントに対しての応答は一部を除いて必要ない。

```
wevent.g.type = EV_REQUEST
.g.data[0] 0
.g.data[1] 0
.g.data[2] 要求タイプ
.g.data[3] 対象仮身 ID (vid)
.g.cmd     仮身要求イベント(= W_VOBJREQ)
.g.wid     対象のウィンドウ ID
.g.src     ソース PID
```

要求タイプ = 0 : 名称 / 続柄等が変更された  
1 : 処理中状態になった  
2 : 処理中状態が解除された  
3 : 虚身状態になった  
4 : 虚身状態が解除された  
5 : 虚身ウィンドウになった  
6 : 虚身ウィンドウが解除された  
7 ~ 14: 予約  
15 : 仮身 / 付箋の変更通知  
16 : 仮身の挿入要求 (ファイルシステム接続)  
17 : 仮身の挿入要求 (新規実身への保存)  
128 : 編集内容の一時ファイルへの格納要求

- 0 ~ 4 の場合は、対象の仮身の再表示を速やかに行なう必要がある。
- 5, 6 の場合は、そのウィンドウに含まれる仮身に対する再表示要求は発生しないため、そのウィンドウに含まれるすべての仮身を再表示する必要がある。この場合、ウィンドウのタイトルは自動的に虚身状態表示、または虚身状態表示解除となっているため、アプリケーションはウィンドウタイトルの表示を更新する必要はない。なお、この場合、対象仮身 ID は、ウィンドウの生成元の仮身 ID となる。
- 15 は、対象の仮身 / 付箋の固有データ、状態、デフォルトアプリケーションが oend\_prc() の結果、または実身管理情報パネルで変更された場合に送信される。従って、アプリケーションはこのイベントを受信した場合は、登録した仮身 / 付箋の内容が変更されているため、保存時には、仮身 / 付箋の状態を取り出して保存する必要がある。

- 16 は、ファイルシステムの接続に伴って生成された新規デバイス仮身の挿入を要求するもので、実行環境ウィンドウに対してのみ送信される。生成されたデバイス仮身は、既に自ウィンドウに登録されており、wevt.g.data[3] にその仮身IDが入っているので、その仮身の情報を取り出して自プロセスの管理下に置く必要がある。登録された仮身はウィンドウの右上に位置しているが、必要に応じて位置を移動してもよい。
- 17 は、新規実身への保存に伴って生成された新規仮身の挿入を要求するもので、仮身を含むすべてのウィンドウに対して送信される可能性がある。生成された仮身は、既に自ウィンドウに登録されており、wevt.g.data[3] にその仮身IDが入っているので、その仮身の情報を取り出して自プロセスの管理下に置く必要がある。登録された仮身は元の仮身の上または下に位置しているが、必要に応じて位置を移動してもよい。
- 128 以上の要求は、oreq\_prc() により送信された仮身要求イベントであり、これを受けた場合は、必ず orsp\_prc() により、応答を戻さなくてはならない。
- 128 は、編集内容の一時ファイルへの格納要求であり、速やかに一時ファイルを生成し、現在の編集内容を格納しなくてはならない。生成した一時ファイルのリンクを orsp\_prc() により、応答として戻す必要がある。

### 3.8.4 データ / 定数の定義

仮身ID

```
typedef W    VID;          /* 仮身 ID */
```

アプリケーション起動メッセージ

```
#define EXECREQ      0x10
#define DISPREQ      0x11
#define PASTEREQ     0x12
#define FUSENREQ     0x13
#define TADREQ       0x14
```

仮身のオープン起動メッセージ

```
typedef struct {
    W    type;          /* = EXECREQ (= 0x10) */
    W    size;          /* メッセージ本体サイズ */
    LINK self;         /* 起動ファイル */
    LINK lnk;          /* 対象ファイル */
    W    vid;          /* 対象仮身 ID */
    W    pwid;         /* 親ウィンドウ ID */
    W    info;         /* レコード番号 / 付箋データポインタ */
    RECT r;           /* 生成元 */
    COLOR bgcol;      /* 元の仮身の背景色 */
    W    mode;         /* 起動モード */
} M_EXECREQ;
```

開いた仮身の表示起動メッセージ

```

typedef struct {
    W        type;          /* = DISPREQ (= 0x11) */
    W        size;         /* メッセージ本体サイズ */
    LINK     self;         /* 起動ファイル */
    LINK     lnk;          /* 対象ファイル */
    W        vid;          /* 対象仮身 ID */
    W        pwid;         /* 親ウィンドウ ID */
    W        info;         /* 付箋データポインタ */
    COLOR    bgcol;        /* 元の仮身の背景色 */
    W        gid;          /* 描画環境 ID */
} M_DISPREQ;

```

### データ貼り込み起動メッセージ

```

typedef struct {
    W        type;          /* = PASTEREQ (= 0x12) */
    W        size;         /* メッセージ本体サイズ */
    LINK     self;         /* 起動ファイル */
    LINK     lnk;          /* 対象ファイル */
    W        vid;          /* 対象仮身 ID */
    W        pwid;         /* 親ウィンドウ ID */
    W        info;         /* 付箋データポインタ */
} M_PASTEREQ;

```

### 付箋のオープン起動メッセージ

```

typedef struct {
    W        type;          /* = FUSENREQ (= 0x13) */
    W        size;         /* メッセージ本体サイズ */
    LINK     self;         /* 起動ファイル */
    W        vid;          /* 対象付箋 ID */
    W        pwid;         /* 親ウィンドウ ID */
    W        info;         /* 付箋データポインタ */
    RECT     r;            /* 生成元 */
} M_FUSENREQ;

```

### 開いた仮身のTADデータ作成起動メッセージ

```

typedef struct {
    W        type;          /* = TADREQ (= 0x14) */
    W        size;         /* メッセージ本体サイズ */
    LINK     self;         /* 起動ファイル */
    LINK     lnk;          /* 対象ファイル */
    W        vid;          /* 対象仮身 ID */
    W        pwid;         /* 親ウィンドウ ID */
    W        info;         /* 付箋データポインタ */
    COLOR    bgcol;        /* 元の仮身の背景色 */
    LINK     save;         /* TADの保存先ファイル */
    RECT     r;            /* 表示領域 */
} M_TADREQ;

```

## 実身 / 仮身マネ - ジャの定義

```
#define FSNM_LEN    40 /* ファイルシステム名のバイト長 */
#define RELNM_LEN   32 /* 続柄名のバイト長 */
#define DTYPE_LEN   32 /* データタイプ名のバイト長 */
```

```
typedef union {
    VOBJSEG    *vobj; /* 仮身セグメント */
    FUSENSEG    *fsn; /* 付箋セグメント */
} VFPTR;
```

### 仮身

```
typedef struct {
    TC    fs_name[20]; /* ファイルシステム名 */
    UH    f_id;        /* ファイル ID */
    UH    attr;        /* 仮身タイプ / 属性 */
    UH    rel;         /* 続柄インデックス */
    UH    appl[3];     /* アプリケーション ID */
} VLINK;
```

### 続柄データ

```
typedef struct {
    TC    name[16]; /* 続柄名 */
    COLOR col;     /* 続柄名の色 */
} VOBJREL;
```

### 仮身の表示モード

```
#define V_NODISP    0 /* 表示しない */
#define V_ERASE     0 /* 消去する */
#define V_DISP     1 /* 表示エリアクリア */
#define V_DISPALL  2 /* 表示エリア表示 */
#define V_DISPAREA 3 /* 表示エリアのみ表示 */
#define V_ERAORG   4 /* 元を消去する */
#define V_NOFRAME  8 /* 境界線なし */
```

### 仮身の位置コード

```
#define V_WORK      0 /* 表示エリア内 */
#define V_FRAM     1 /* 仮身枠 */
#define V_PICT     2 /* ピクトグラム */
#define V_NAME     3 /* 実身名 */
#define V_LTHD    4 /* ハンドル (左上) */
#define V_RTHD    5 /* ハンドル (右上) */
#define V_LBHD    6 /* ハンドル (左下) */
#define V_RBHD    7 /* ハンドル (右下) */
```



```
#define V_RELN 8 /* 続柄 */
```

## 仮身の変形指定

```
#define V_CHECK 0 /* 大きさのチェック */
#define V_SIZE 0x10 /* 変形する */
#define V_OPEN 0x20 /* 仮身を開く */
#define V_CLOSE 0x30 /* 仮身を閉じる */
#define V_ADJUST 0x40 /* 全体表示の長さ */
#define V_ADJUST1 0x50 /* 名前表示の長さ */
#define V_ADJUST2 0x60 /* 続柄まで表示の長さ */
#define V_ADJUST3 0x70 /* 日付以外表示の長さ */
```

## 仮身の属性 / 状態

```
#define V_NONAME 0x0001 /* 実身名の表示なし */
#define V_NORELN 0x0002 /* 続柄名の表示なし */
#define V_NOTYPE 0x0004 /* データタイプの表示なし */
#define V_NOTIME 0x0008 /* 変更日時を表示なし */
#define V_FIXDEF 0x0010 /* 固定デフォルトアプリ */
#define V_NOIMG 0x0020 /* イメージ保存なし */
#define V_NOPICT 0x0040 /* ピクトグラム表示なし */
#define V_NOFDISP 0x0080 /* 仮身枠表示なし */
#define V_OPENED 0x0100 /* オープン仮身 */
#define V_NOEXPND 0x0200 /* 印刷時展開なし */
#define V_LINKFL 0x0800 /* リンクファイル */
#define V_DETACH 0x1000 /* 虚身状態 */
#define V_INPROC 0x2000 /* 処理中状態 */
#define V_AUTEXE 0x4000 /* 自動起動 */
#define V_HIDDEN 0x8000 /* 隠蔽仮身 */

#define V_GETSTS 0x8000 /* 現在の状態の取り出し */
#define V_PURGE 0xffff /* 保存表示イメージを廃棄 */
#define V_CHKREF 0x8888 /* 参照状態のチェック */
#define V_CHKDUP 0x9999 /* 重複状態のチェック */

#define F_NONAME 0x0800 /* 付箋名の表示なし */
#define F_NOTYPE 0x0400 /* データタイプの表示なし */
#define F_HIDDEN 0x8000 /* 隠蔽付箋 */
```

## 仮身メニューの操作タイプ

```
#define VM_NONE 0 /* 何も変更されなかった */
#define VM_OPEN 1 /* 仮身が開かれた */
#define VM_CLOSE 2 /* 仮身が閉じられた */
#define VM_NAME 3 /* 実身名が変更された */
#define VM_RELN 4 /* 続柄が変更された */
#define VM_NEW 5 /* 新版が作成された */
#define VM_DETACH 6 /* 切断状態となった */
#define VM_DISP 7 /* 仮身の表示属性が変更された */
```

```

#define VM_REFMT      8 /* 再フォーマットされた */
#define VM_PASTE      9 /* 仮身へ埋め込まれた */
#define VM_EXREQ     10 /* 付箋起動が要求された */
#define VM_ATTACH    11 /* 接続状態となった */
#define VM_APLREG    12 /* アプリ登録された */
#define VM_APLDEL    13 /* アプリ登録が削除された */
#define VM_INFO      14 /* 管理情報が表示された */
#define VM_DISK      15 /* ディスク状態が表示された */
#define VM_GABAGE    16 /* ディスク整理が起動された */

```

### 選択枠領域

```

/* 選択枠領域 */
typedef struct {
    UW      sts; /* 状態 */
    union {
        POLY  p; /* 多角形選択領域 */
        RECT  r; /* 長方形選択領域 */
    } rgn;
} SEL_RGN;

#define SEL_POLYRGN 0x4000 /* 多角形選択領域 */

```

### 選択枠領域リスト

```

typedef struct sel_list {
    struct sel_list *next; /* 次の選択枠へのポインタ(最後はNULL) */
    SEL_RGN          rgn; /* 選択枠領域 */
} SEL_LIST;

```

## 3.8.5 実身 / 仮身マネージャの関数

ここでは、実身 / 仮身マネージャがサポートしている各関数の詳細を説明する。これらの関数群は外殻の拡張システムコールとして提供される。

関数値は、何らかのエラーがあった場合は「負」のエラーコードが戻る。正常終了時には「0」または「正」の値が戻る。

各関数のエラーコードとしては、ここで示した以外にも、核や他の外殻でエラーが検出された場合は、そのエラーコードが直接戻る場合がある。

また、各関数のパラメータの説明では、以下に示す記述方法を使用している。

```

( x    y    z ) -- x, y, z のいずれか1つを意味する。
|
|             -- OR で指定可能なことを意味する。
[ ]          -- 省略可能なことを意味する。

```

実身 / 仮身マネージャの関数は他の外殻の関数と比較して、よりアプリケーションレベルに近い  
ため、関数によってはパネルを利用したユーザとのインタラクションが含まれている点に注意が  
必要である。

ここで示している多くの関数は、特に記述がない限り、仮身だけでなく付箋に対しても適用され

る。したがって、単に「仮身」と記述してある場合は「付箋」も含まれるものとする。

## oreg\_vob

仮身の登録

### 【形式】

VID oreg\_vob(VLINK \*vlnk, VP vseg, W wid, UW disp)

### 【パラメータ】

VLINK \*vlnk 仮身(リンク)  
= NULL : 付箋  
VP vseg 仮身セグメント/付箋セグメント  
W wid p 0 : ウィンドウ ID  
< 0 : - (描画環境 ID)  
UW disp 仮身の表示方法  
::= (V\_NODISP V\_DISP V\_DISPALL V\_DISPAREA)  
| [V\_NOFRAME] | [V\_AUTEXE]

V\_NODISP

表示は行なわない

V\_DISP

表示する (開いた仮身の表示エリアはウィンドウの背景色で塗り潰す)

V\_DISPALL

表示する(開いた仮身の表示エリアは対応するデフォルトアプリケーションを起動して表示する)

V\_DISPAREA

開いた仮身の表示エリアのみを表示し、表示エリアの境界線を含めた外側は一切表示しない。閉じた仮身の場合は何も表示しない。これは清書表示として開いた仮身の中味のみを表示する場合に使用される。この場合、V\_NOFRAME 指定は意味を持たない。

V\_NOFRAME

仮身の境界線を点線とする。V\_NODISP 指定の場合は意味を持たない。これは内部仮身の表示に使用される。

V\_AUTEXE

vlnk の attr に V\_AUTEXE が設定されているとき、登録後にデフォルトアプリケーションを起動する。

### 【リターン値】

0 正常 (仮身 ID (vid > 0))  
<0 エラー (エラーコード)

### 【解説】

vlnk で指定した仮身 (リンク) に対する vseg で指定した仮身セグメントを、wid で指定したウィンドウ / パネル上に登録し、関数値として仮身 ID (vid > 0) を戻す。

wid < 0 の場合は、- wid を描画環境 ID とみなして登録を行ない、その描画環境に仮身の表示が行なわれる。この場合、登録した仮身に対するウィンドウ ID を使用した操作は制限される。- wid の描画環境 ID が存在しない場合も登録は可能であり、仮身を登録して表示以外の処理をしたい場合に使用される。この場合、表示を行なった時点でエラー (EG\_GID) となる。

vlnk で指定した仮身の参照するファイルシステムが接続されておらず、かつ同一のファイルを参照する他の仮身 (虚身) が登録されていない場合は、「デバイス接続パネル」が表示され、デバイスの接続を求める。接続が行なわれなかった場合はエラー (ER\_NOFS) リターンする。

vseg で指定した仮身セグメント内の view により仮身の領域が指定される。view で示される領域の大きさが不正である場合は、最も近い正しい領域に更新される。また view が空の領域の場合は、仮身はデフォルトの大きさになる。

登録した後は、vseg で指定したメモリ領域は参照されないため、データを変更してもよい。

vlnk = NULL の場合は、付箋の登録となり、vseg は FUSESEG へのポインタとなる。この場合、V\_DISP ~ V\_DISPAREA は、すべて同一の意味となる。

### 【エラーコード】

EX\_ADR : アドレス(vlnk,vseg)のアクセスは許されていない。  
EX\_LIMIT : システムの制限を越えた(仮身の登録最大数を越えた)。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である(disp,vseg の内容が不正)。  
EX\_WID : ウィンドウ(wid)は存在していない。

## b\_odup\_vob

仮身の複製

### 【形式】

VID b\_odup\_vob(W org)

### 【パラメータ】

W org 仮身 ID

### 【リターン値】

0 正常 (仮身 ID (vid > 0))  
<0 エラー (エラーコード)

### 【解説】

org で指定した仮身 ID を持つ仮身と同一内容をもつ仮身を新規に登録 (複製) し、その仮身 ID (

vid > 0) を関数値として戻す。

この関数は、おもに仮身の複写に使用される。

### 【エラーコード】

EX\_LIMIT : システムの制限を越えた(仮身の登録最大数を越えた)。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_VID : 仮身 ID (vid) は存在していない。

## odel\_vob

仮身の登録削除

### 【形式】

W odel\_vob(W vid, W clr)

### 【パラメータ】

W vid 0 : 仮身 ID  
< 0 : - (ウィンドウ ID)  
W clr = 0 : 消去を行わない  
0 : 仮身の表示領域をウィンドウ背景色で塗り潰す

### 【リターン値】

0 正常 (削除した仮身の数)  
< 0 エラー (エラーコード)

### 【解説】

vid で指定した仮身の登録を削除する。

vid < 0 の場合は、- vid で指定した値をウィンドウ ID とみなして、そのウィンドウ / パネル上に登録されているすべての仮身を削除する (ただし、wid < 0 で登録した仮身に対しては適用できない)。この場合、ウィンドウ / パネルが実際に存在するかはチェックされない。

clr = 0 の場合は、仮身の表示はそのままであり特に消去は行わず、clr 0 の場合は、仮身の表示領域がウィンドウの背景色で塗り潰される (wid < 0 で登録した仮身の場合は、常に白で塗りつぶされる)。

関数値として削除した仮身の数が戻る。従って、vid > 0 の場合は常に "1" となる。vid < 0 の場合で、指定したウィンドウに仮身が 1 つも登録されていない場合は何もせず、関数値 "0" が戻る。

ウィンドウ / パネルが削除された場合、そのウィンドウ / パネル上に登録されている仮身は自動的に削除される (ocls\_wnd() を参照のこと)。ただし、wid < 0 で登録した仮身は自動的に削除されることはないため、必ず odel\_vob() により、1 つずつ削除しなくてはならない。

なお、処理中状態の仮身に対しては削除された後も、`ocre_obj()`、`oend_prc()`、`oatt_vob()`、`oopn_obj()`、`ocnv_vob()`、`oget_fsn()`、`oput_fsn()` の各関数は実行できる。

### 【エラーコード】

EX\_VID : 仮身 ID (vid) は存在していない。

## ocls\_wnd

ウィンドウのクローズ処理

### 【形式】

ERR ocls\_wnd(W wid)

### 【パラメータ】

W wid ウィンドウ ID

### 【リターン値】

=0 正常

### 【解説】

wid で指定したウィンドウ / パネルがクローズされたことを通知し、ウィンドウ / パネルに属する仮身の登録の削除、仮身の処理中状態の解除、仮身のデータの更新等を行なう。wid で指定したウィンドウ / パネルが実際に存在するかはチェックされない。

指定したウィンドウ / パネルに仮身が 1 つも登録されていない場合は何もしない。

この関数はウィンドウマネージャにより、ウィンドウ / パネルがクローズされた場合に実行されるため、一般のアプリケーションでは通常は使用しない。

### 【エラーコード】

なし

## odsp\_vob

仮身の表示

### 【形式】

W odsp\_vob(W vid, RECT \*r, UW disp)

### 【パラメータ】

W vid 0 : 仮身 ID  
 = 0x8000 : 仮身リスト指定  
 < 0 : - (ウィンドウ ID)  
 RECT \*r 表示領域 (仮身リスト指定のときは、仮身リストへのポインタ)  
 UW disp 仮身の表示 / 消去方法  
 ::= (V\_ERASE V\_DISP V\_DISPALL V\_DISPAREA) | [V\_NOFRAME]

V\_ERAS

消去する (ウィンドウの背景色で塗り潰す)。

V\_DISP

表示する(開いた仮身の表示エリアはウィンドウの背景色で塗り潰す)。

V\_DISPALL

表示する (開いた仮身の表示エリアは対応するデフォルトアプリケーションを起動するか、保存されている表示イメージにより表示する)。

V\_DISPAREA

開いた仮身の表示エリアのみを表示し、表示エリアの境界線を含めた外側は一切表示しない。閉じた仮身の場合は何も表示しない。これは清書表示として開いた仮身の中味のみを表示する場合に使用される。この場合、V\_NOFRAME 指定は意味を持たない。

V\_NOFRAME

仮身の境界線を点線とする。V\_ERASE 指定の場合は意味を持たない。これは内部仮身の表示に使用される。

## 【リターン値】

0 正常 (処理した仮身の数)  
 <0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の表示 / 消去を行なう。r は表示すべき領域の相対座標での指定であり、仮身の領域と r の共通部分のみが表示される。r = NULL の場合は、仮身の領域全体が表示される。

仮身の表示は、その時点の仮身の状態 (虚身状態、処理中状態等) に応じて行なわれる。

vid < 0 の場合は、- vid で指定した値をウィンドウ ID とみなして、そのウィンドウ / パネル上に登録されているすべての仮身の表示 / 消去を行なう。

vid = 0x8000 のときは、仮身リスト指定となり、r を以下の仮身リスト構造へのポインタとみなして、複数の仮身の表示を高速に行なう。この場合、指定する仮身はすべて同一のウィンドウ / パネル、または描画環境に属していなくてはならない (そうでない場合の表示は保証されない)。また、途中の仮身 ID が不正のときは、それ以降の仮身の表示は保証されない。

```
Wffff vid -- 仮身 ID
RECT r -- 表示領域
: :
W 0 -- 終了を示す
```

関数値として処理した仮身の数が戻る。従って、vid > 0 の場合は常に "1" となる。vid < 0 の場

合は、指定したウィンドウ / パネルに仮身が1つも登録されていない場合は何もせず、関数値 "0" が戻る。この場合、ウィンドウ / パネルが実際に存在するかはチェックされない。また、vid = 0x8000 の場合は、関数値は常に "0" となる。

付箋の場合は、V\_DISP ~ V\_DISPAREA は、すべて同一の意味となる。

### 【エラーコード】

EX\_ADR : アドレス(r)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(disp が不正)。  
EX\_VID : 仮身 ID (vid)は存在していない。  
EX\_WID : ウィンドウ(wid)は存在していない(vid < 0 のときのみ)。

## odsp\_vor

仮身の倍率表示

### 【形式】

W odsp\_vor(W vid, RECT \*r, UW disp, PNT scal)

### 【パラメータ】

W vid 0 : 仮身 ID  
= 0x8000 : 仮身リスト指定  
< 0 : - (ウィンドウ ID)  
RECT \*r 表示領域 (仮身リスト指定のときは、仮身リストへのポインタ)  
UW disp 仮身の表示方法 (odsp\_vob と同じ)  
::= (V\_ERASE V\_DISP V\_DISPALL V\_DISPAREA) | [V\_NOFRAME]  
PNT scal 表示倍率

### 【リターン値】

0 正常 (処理した仮身の数)  
< 0 エラー (エラーコード)

### 【解説】

仮身の拡大 / 縮小表示を行なう。scal で指定された倍率により拡大 / 縮小を行なう以外は odsp\_vob() と同一である。

scal.x は横方向の倍率を示し、scal.y は縦方向の倍率を示す。倍率は 256 倍の値で表現され、0x100 で 1 倍を示す。scal.x = 0 または scal.y = 0 の場合は、縦横 1 倍とみなす。

倍率は、仮身の表示座標、文字サイズ、ピクトグラムの大きさ、開いた仮身の表示領域のイメージに影響するが、最大文字サイズの制限はインプリメントに依存する。拡大した結果、座標がオーバーフローした場合の表示は保証されない。

### 【エラーコード】



EX\_ADR : アドレス(r)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(disp が不正)。  
EX\_VID : 仮身 ID (vid)は存在していない。  
EX\_WID : ウィンドウ(wid)は存在していない(vid < 0 のときのみ)。

## odra\_vob

仮身の一時的表示

### 【形式】

ERR odra\_vob(VLINK \*vInk, VP vseg, W wid, UW disp)

### 【パラメータ】

VLINK \*vInk 仮身(リンク)  
= NULL : 付箋  
VP vseg 仮身セグメント/付箋セグメント  
W wid 0 : ウィンドウ ID  
< 0 : - (描画環境 ID)  
UW disp 仮身の表示方法 (odsp\_vob と同じ)  
::= (V\_ERASE V\_DISP V\_DISPALL V\_DISPAREA) | [V\_NOFRAME]

### 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

### 【解説】

vInk で指定した仮身 (リンク) に対する vseg で指定した仮身セグメントを、wid で指定したウィンドウパネル上に表示する。

wid < 0 の場合は、- wid を描画環境 ID とみなして表示を行なう。描画環境 ID が存在しない場合はエラー (EG\_GID) となる。

vInk で指定した仮身の参照するファイルシステムが接続されておらず、かつ同一のファイルを参照する他の仮身 (虚身) が登録されていない場合は、「デバイス接続パネル」が表示され、デバイスの接続を求める。接続が行なわれなかった場合はエラー (ER\_NOFS) リターンする。

vseg で指定した仮身セグメント内の view により仮身の領域が指定される。view で示される領域の大きさが不正である場合は、最も近い正しい領域に更新される。特に view が空の領域の場合は、仮身はデフォルトの大きさとなる。

仮身の表示は、その時点の仮身の状態(虚身状態等)に応じて行なわれる。

disp により、仮身の境界線の有無、および表示 / 消去方法を指定する。

vInk = NULL の場合は、付箋の一時的表示となり、vseg は FUSENSEG へのポインタとなる。この場合、V\_DISP ~ V\_DISPAREA は、すべて同一の意味となる。

## 【エラーコード】

EX\_ADR : アドレス(vlnk,vseg)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である(disp,vseg の内容が不正)。  
EX\_WID : ウィンドウ(wid)は存在していない。

# odra\_vor

仮身の一時的倍率表示

## 【形式】

ERR odra\_vor(VLINK \*vlnk, VP vseg, W wid, UW disp, PNT scal)

## 【パラメータ】

VLINK \*vlnk 仮身(リンク)  
= NULL : 付箋  
VP vseg 仮身セグメント/付箋セグメント  
W wid 0 : ウィンドウ ID  
< 0 : - (描画環境 ID)  
UW disp 仮身の表示方法 (odsp\_vob と同じ)  
::= (V\_ERASE V\_DISP V\_DISPALL V\_DISPAREA) | [V\_NOFRAME]  
PNT scal 表示倍率

## 【リターン値】

= 0 正常 < 0 エラー (エラーコード)

## 【解説】

仮身の一時的拡大 / 縮小表示を行なう。scal で指定された倍率により拡大 / 縮小を行なう以外は odra\_vob() と同一である。

scal.x は横方向の倍率を示し、scal.y は縦方向の倍率を示す。倍率は 256 倍の値で表現され、0x100 で 1 倍を示す。scal.x = 0 または scal.y = 0 の場合は、縦横 1 倍とみなす。

倍率は、仮身の表示座標、文字サイズ、ピクトグラムの大きさ、開いた仮身の表示領域のイメージに影響するが、最大文字サイズの制限はインプリメントに依存する。拡大した結果、座標がオーバーフローした場合の表示は保証されない。

## 【エラーコード】

EX\_ADR : アドレス(vlnk,vseg)のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である(disp,vseg の内容が不正)。  
EX\_WID : ウィンドウ(wid)は存在していない。

# ofnd\_vob

## 仮身の位置検索

### 【形式】

W ofnd\_vob(W wid, PNT pos, W \*vid)

### 【パラメータ】

W wid            0 : ウィンドウ ID  
                 < 0 : - (仮身 ID)  
PNT pos        位置  
W \*vid        見つかった仮身 ID の格納場所

### 【リターン値】

0    正常 (位置コード)  
<0   エラー (エラーコード)

### 【解説】

wid で指定したウィンドウ / パネルに登録してある全ての仮身に対して、pos で指定した位置の点が含まれているか否かをサーチし、含まれていた場合はその仮身 ID を、含まれていない場合は "0" を vid で指定した領域に格納する。サーチの順番は特に定義されない。

wid < 0 の場合は、- wid を仮身 ID とする仮身に対して、pos で指定した位置の点が含まれているか否かをチェックする。この場合、ウィンドウ / パネルが実際に存在するかはチェックされない。

pos はウィンドウ / パネル内の相対座標で指定する。

関数値として以下の位置コードが戻る。

V\_WORK (0) -- 表示エリア内、または含まれていない  
V\_FRAM (1) -- 仮身枠  
V\_PICT (2) -- ピクトグラム  
V\_NAME (3) -- 実身名  
V\_LTHD (4) -- ハンドル (左上)  
V\_RTHD (5) -- ハンドル (右上)  
V\_LBHD (6) -- ハンドル (左下)  
V\_RBHD (7) -- ハンドル (右下)  
V\_RELN (8) -- 続柄

V\_FRAMは、上記以外の仮身の矩形領域内の部分を示す (境界線も含む)。付箋の場合は V\_WORK (表示エリア内)、V\_RELN は適用されない。

指定したウィンドウに仮身が1つも登録されていない場合は、vid には "0" が格納され、関数値 "0" が戻る。

なお、wid < 0 で登録した仮身に対しては、この関数は適用できない。

## 【エラーコード】

EX\_ADR : アドレス(vid)のアクセスは許されていない。  
EX\_VID : 仮身 ID (vid)は存在していない(wid < 0 のとき)。

# omov\_vob

仮身の移動

## 【形式】

ERR omov\_vob(W vid, W wid, RECT \*newr, UW disp)

## 【パラメータ】

W vid 0 : 仮身 ID  
< 0 : - (ウィンドウ ID)  
W wid 0 : ウィンドウ ID  
= 0xFFFF8000 : 仮身の暫定削除  
< 0 : - (描画環境 ID)  
RECT \*newr 移動位置  
UW disp 仮身の表示方法  
::= (V\_NODISP V\_DISP V\_DISPALL V\_DISPAREA) | [V\_ERAORG]

V\_NODISP

表示は行なわない。

V\_DISP ~ V\_DISPAREA

odsp\_vob() での指定と同じ。

V\_ERAORG

移動した仮身の元の位置の表示を消去して移動する。この指定が無い場合は、仮身の元の位置の表示に関しては何も行なわない。

## 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の現在位置を、wid で指定したウィンドウ / パネル内の newr で指定した位置に移動する。wid = 0 の場合は、ウィンドウの変更がないことを意味する。

wid = 0xFFFF8000 (- 0x8000) は、特別に指定した仮身を暫定的に削除することを意味する。暫定的に削除した仮身は、再度、omov\_vob() により、正しいウィンドウに移動することにより復旧することができ、odel\_vob() により完全に削除することができる。この場合、newr は無視され、位

置の移動は行なわれない。また、disp パラメータも無視され、常に V\_NODISP となる。この機能は、アプリケーションで仮身を削除する場合や、トレーに切り取る場合に「復旧」機能を実現するために使用される。トレーに仮身を切り取る場合、切り取った仮身は必ず、暫定削除、もしくは「復旧」機能を実現しない場合は削除しなくてはならない。暫定的に削除した仮身は、omov\_vob()、odel\_vob() 以外の関数を実行した場合、EX\_VID のエラーとなる。

暫定削除された仮身に対して、omov\_vob()、odel\_vob() 以外の関数を実行すると、EX\_VID のエラーとなる。ただし、処理中状態の仮身は暫定削除された後も、ocre\_obj()、oend\_prc()、oatt\_vob()、oopn\_obj()、ocnv\_vob()、oget\_fsn()、oput\_fsn() の各関数は実行できる。

wid < 0 の場合は、- wid を描画環境 ID とみなして移動を行ない、描画環境に仮身の表示が行なわれる。V\_NODISP 指定以外で - wid の描画環境 ID が存在しない場合はエラー (EG\_GID) となる。

vid < 0 の場合は、- vid をウィンドウ ID とするウィンドウ内のすべての仮身を wid で指定したウィンドウへ移動する。この場合、newr は無視され、位置の移動は行なわれない。また、disp パラメータも無視され、常に V\_NODISP となる (ただし、wid < 0 で登録した仮身に対しては適用できない)。

newr->p.lefttop により仮身の表示矩形の左上の点をウィンドウ / パネルの相対座標で指定する。仮身の大きさは変更されないため、pos->p.rightbot は無視され、実行終了時に移動後の仮身の表示矩形領域が、newr で指定した領域に戻される。newr = NULL の場合は、位置の移動を行わずにウィンドウの移動のみを行なう。

処理中状態の仮身を別のウィンドウに移動した場合は、その仮身から開かれているウィンドウの親ウィンドウ、生成元が変更される。wid で指定した移動先のウィンドウ / パネルが存在していない場合はエラー (EX\_WID) となる。ただし、wid < 0 で V\_NODISP 指定の場合は - wid の描画環境の存在はチェックされない。

付箋の場合は、V\_DISP ~ V\_DISPAREA は、すべて同一の意味となる。

## 【エラーコード】

EX\_ADR : アドレス(newr)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(disp が不正)。  
EX\_VID : 仮身 ID(vid)は存在していない。  
EX\_WID : ウィンドウ(wid)は存在していない(wid > 0の場合)。

## orsz\_vob

仮身の変形

## 【形式】

ERR orsz\_vob(W vid, RECT \*newr, UW mode)

## 【パラメータ】

W vid 仮身 ID  
RECT newr 変形位置  
UW mode ::= (V\_SIZE V\_ADJUST(1~3) V\_OPEN V\_CLOSE V\_CHECK)  
| (V\_NODISP V\_DISP V\_DISPALL V\_DISPAREA) | [V\_ERAORG]

## V\_SIZE

newr で指定した矩形領域に変形する。newr は、ウィンドウ / パネルの相対座標で指定され、元の仮身の矩形領域の 4 隅の点のうち少なくとも 1 点は一致していなくてはならない。仮身のタイトル部の高さは、文字サイズに依存するため、newr で指定した大きさと、実際に変形された大きさは少し異なる場合がある。指定した大きさによって、仮身のオープン / クローズも行なわれることになる。

## V\_ADJUST, V\_ADJUST1, V\_ADJUST2, V\_ADJUST3

仮身の長さをそれぞれ、全体、名前まで、名前 + 続柄まで、名前 + 続柄 + データタイプまで表示する長さとなるように、仮身の右側の座標のみを変更する。

## V\_OPEN

仮身をデフォルトの大きさで開く。既に開いている場合は何も行なわない。

## V\_CLOSE

仮身を閉じる。既に閉じている場合は何も行なわない。

## V\_CHECK

変形せずに、現在の仮身の矩形領域を戻す。

## V\_NODISP ~ V\_ERAORG

変形後の表示指定(omov\_vob() と同じ)。実際の変形が行なわれなかった場合 (V\_CHECK 指定を含む)にも、この表示指定は有効であり、指定に従った再表示を行なう。

### 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

### 【解説】

vid で指定した仮身を mode で指定した方法により変形し、変形の結果の新しい仮身の矩形領域を、newr で指定した領域に戻す。

処理中状態の仮身の大きさが変更された場合は、その仮身から開かれているウィンドウの生成元が変更される。

付箋の場合は、V\_DISP ~ V\_DISPAREA は、すべて同一の意味となる。また、V\_OPEN, V\_CLOSE は EX\_PAR のエラーとなる。

### 【エラーコード】

EX\_ADR : アドレス(newr)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(mode が不正、変形の点が一致していない)。  
EX\_VID : 仮身 ID (vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(仮身は開けない)。

## ochg\_chs

仮身の文字サイズの変更

## 【形式】

W ochg\_chs(W vid, W chsz, RECT \*newr, UW mode)

## 【パラメータ】

W vid 仮身 ID  
W chsz 文字サイズ (-1 は変更なし, 0 はデフォルト)  
RECT \*newr 仮身矩形領域が格納される  
UW mode ::= [V\_ADJUST] |  
(V\_NODISP V\_DISP V\_DISPALL V\_DISPAREA) | [V\_ERAORG]

V\_ADJUST

仮身の長さを、文字サイズの変更に比例して変更する。この指定がない場合は、仮身の長さ  
は変更されない。

V\_NODISP ~ V\_ERAORG

変形後の表示指定(omov\_vob()と同じ)。仮身の大きさが実際に変更されなかった場合に  
も、この示指定は有効であり、指定に従った再表示を行なう。

## 【リターン値】

0 正常 (変更前の文字サイズ)  
<0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の文字サイズを chsz で指定した大きさに変更し、変更前の文字サイズを関数  
値として戻す。chsz = -1 の場合は変更せずに現在の文字サイズを戻す。chsz = 0 はデフォルト  
の文字サイズを意味する。

仮身の高さは、文字サイズに応じて変更される。開いた仮身の場合は、表示エリアの高さは変更  
されないが、タイトル部の高さが変更されるため、全体の高さが変更されることになる。

newr が NULL でない場合は結果の仮身の矩形領域を、newr で指定した領域に戻す。

処理中状態の仮身の大きさが変更された場合は、その仮身から開かれているウィンドウの生成元  
が変更される。

付箋の場合は、V\_DISP ~ V\_DISPAREA は、すべて同一の意味となる。

## 【エラーコード】

EX\_ADR : アドレス(newr)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(mode, chsz が不正)。  
EX\_VID : 仮身 ID(vid)は存在していない。

# ochg\_col

仮身表示色の変更

## 【形式】

ERR ochg\_col(W vid, COLOR frcol, COLOR chcol, COLOR tbcoll, COLOR bgcol, UW disp)

## 【パラメータ】

|       |        |  |
|-------|--------|--|
| W     | vid    | 仮身 ID  |
| COLOR | frcol  | 枠の色 (-1 は変更なし)   |
| COLOR | chcol  | 文字色 (-1 は変更なし)   |
| COLOR | tbcoll | タイトル背景色 (-1 は変更なし)   |
| COLOR | bgcol  | 開いた場合の表示エリアの背景色 (-1 は変更なし)   |
| UW    | disp   | 仮身の表示方法(omov_vob と同じ)<br>::= (V_NODISP V_DISP V_DISPALL V_DISPAREA)   [V_ERAORG] |

## 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の表示色を、frcol ( 枠の色 ), chcol ( 文字色 ), tbcoll ( タイトル背景色 ), bgcol ( 開いた場合の表示エリアの背景色 ) で指定した色に変更する。それぞれの値が - 1 のときは、対応する色は変更しないことを意味する。

disp により表示の指定を行なう。内容は、omov\_vob() での指定と同じである。

付箋の場合は、V\_DISP ~ V\_DISPAREA は、すべて同一の意味となる。また、bgcol は適用されず無視される。

## 【エラーコード】

EX\_PAR : パラメータが不正である(disp が不正)。  
EX\_VID : 仮身 ID (vid)は存在していない。

ochg\_nam

実身名の変更

## 【形式】

W ochg\_nam(W vid, TC \*name)

## 【パラメータ】

W vid 仮身 ID



TC \*name 実身名文字列 (NULL はユーザ設定)

### 【リターン値】

=0 正常 (変更なし)  
=1 正常 (変更した)  
<0 エラー (エラーコード)

### 【解説】

vid で指定した仮身が参照している実身の名前を、 name で指定した名前に変更する。 name = NULL の場合は、「実身名変更パネル」が表示されて、ユーザによる実身名の変更が行なわれる。

関数値として実身名が変更された場合に "1" が戻り、変更されなかった場合は "0" が戻る。従って "1" が戻った場合は、仮身の再表示を行なう必要がある。

実身名を変更した実身と同一の実身を参照している他の仮身が存在した場合には、その仮身に対して仮身要求イベントが送信される (vid で指定した仮身に対しては送信されない)。

実身名を変更した仮身が処理中状態の場合、開かれているウィンドウのタイトルも同時に変更される。

name = NULL の場合、指定した仮身が切断状態のときは自動的に「デバイス接続パネル」が表示され、ファイルシステムの接続を求める。また、ファイル管理のエラーにより実身名の変更ができなかった場合は、その旨のエラーパネルが表示された後にエラーリターンする。

name NULL の場合、指定した仮身が切断状態のときは単にエラーリターンする。また、ファイル管理のエラーにより実身名の変更ができなかった場合は単にエラーリターンし、パネルは一切表示されない。

指定した仮身がデバイス仮身の場合は、対応するファイルシステムの名称の変更となり、そのファイルシステム内に処理中の仮身が存在する場合は、EX\_VOBJ のエラーとなる。なお、デバイス仮身の名称の変更は、name = NULL の場合のみ可能である。

付箋の場合も同様の処理を行なう。

### 【エラーコード】

EX\_ADR : アドレス(name)のアクセスは許されていない。  
EX\_VID : 仮身 ID (vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(name NULLで切断状態、デバイス仮身)。

## ochg\_rel

続柄の変更

### 【形式】

W ochg\_rel(W vid, W index)

### 【パラメータ】

W vid 仮身 ID  
W index 続柄のインデックス(< 0 はユーザ設定)

### 【リターン値】

= 0 正常 (変更なし)  
= 1 正常 (変更した)  
< 0 エラー (エラーコード)

### 【解説】

vid で指定した仮身の続柄のインデックスを、index で指定した値に変更する。指定したインデックスが存在しない場合はエラーとなる。

index < 0 の場合は、「続柄変更パネル」が表示され、ユーザによる続柄の変更および登録が行なわれる。

「続柄変更パネル」により既に存在する続柄が変更された場合、変更された続柄インデックスを持つ他の仮身に対して仮身要求イベントが送信される (vid で指定した仮身に対しては送信されない)。

関数値として続柄が変更された場合に "1" が戻り、変更されなかった場合は "0" が戻る。従って "1" が戻った場合は、仮身の再表示を行なう必要がある。

index < 0 の場合、指定した仮身が切断状態のときは自動的に「デバイス接続パネル」が表示され、ファイルシステムの接続を求める。ファイル管理のエラーが発生した場合は、その旨のエラーパネルが表示された後にエラーリターンする。

index 0 の場合、指定した仮身が切断状態のときは単にエラーリターンする。また、ファイル管理のエラーが発生した場合は単にエラーリターンし、パネルは一切表示されない。

付箋の場合は、EX\_VOBJ のエラーとなる。

### 【エラーコード】

EX\_PAR : パラメータが不正である(index が不正)。  
EX\_VID : 仮身ID (vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(index 0 で虚身状態、付箋)。

## onew\_obj

新版作成

### 【形式】

VID onew\_obj(W vid, VLINK \*vInk)

### 【パラメータ】

W vid 仮身 ID  
VLINK \*vlnk 仮身(リンク)

## 【リターン値】

0 正常 (新版または複製した仮身 ID)  
<0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の新版の作成またはデバイスの複製を行ない、作成した新版または複製したデバイスを参照する仮身 ID を関数値として戻す。

vid で指定した仮身がフロッピーディスクのデバイス仮身の場合は、「デバイス複製パネル」が表示され、デバイスの複製か新版作成かを選択する。ただし、通常、この機能はフロッピーディスクドライブが2台以上装着されているときにサポートされる。フロッピーディスクのデバイス仮身でない場合は、常に新版作成となる。

新版作成の場合は、「新版作成パネル」が表示され、新版の実身名を指定することができる。新版作成が終了すると、新版を参照する仮身が指定した仮身の下または上に自動的に生成され、関数値としてその仮身 ID が戻るため、その仮身を表示する必要がある。新版は、vid で指定した仮身の所属デバイス上に生成される。所属デバイスが未定義 (ウィンドウの元となった実身が実身 / 仮身マネージャに登録されていない) の場合は、この関数を実行したプロセスの現在の作業ファイルと同一ファイルシステム上に生成される。

デバイスの複製の場合は、「デバイス接続パネル」により複製先のフロッピーディスクの接続が要求され、接続した複製先のフロッピーディスクに指定したデバイス仮身の参照するフロッピーディスクの内容がすべて複製される。この場合、接続したフロッピーディスクの元の内容は削除される。複製が終了すると、接続した複製先のフロッピーディスクを参照する仮身が指定した仮身の下または上に自動的に生成され、関数値としてその仮身 ID が戻るため、その仮身を表示する必要がある。

デバイスの複製、および新版作成の処理中は「新版作成中パネル」が表示される。

vlnk NULL の場合、vlnk で指定した領域には、作成した新版または複製したデバイスを参照するリンクが格納される。

指定した仮身が切断状態のときは自動的に「デバイス接続パネル」が表示され、ファイルシステムの接続を求める。

ファイル管理のエラーが発生した場合は、その旨のエラーパネルが表示された後にエラーリターンする。

付箋の場合は、EX\_VOBJ のエラーとなる。

## 【エラーコード】

EX\_ADR : アドレス(vlnk)のアクセスは許されていない。  
EX\_LIMIT : システムの制限を越えた(ファイルの数が多すぎる)。  
EX\_VID : 仮身 ID(vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(処理中状態、付箋)。

# ocre\_obj

## 新規実身の生成

### 【形式】

W ocre\_obj(W vid, TC \*name, W \*nvid, LINK \*lnk, W copy)

### 【パラメータ】

|      |       |                 |
|------|-------|-----------------|
| W    | vid   | 仮身 ID           |
| TC   | *name | 実身名文字列          |
| W    | *nvid | 新規実身の仮身 ID 格納場所 |
| LINK | *lnk  | リンク格納場所         |
| W    | copy  |                 |

0:

vid で指定した仮身の参照する実身と同一の内容を持った実身を生成

=0:

単に空の実身を生成

### 【リターン値】

0 正常 (生成した実身のファイルディスクリプタ)

<0 エラー (エラーコード)

### 【解説】

name で指定した名前の実身を新規に生成し、そのファイルディスクリプタを関数値として戻す。

name が空文字列の場合は、「保存先指定パネル」が表示され、生成する実身のデバイスを接続済みデバイスから選択、および実身名の設定を行なう。name で指定された領域には指定された実身名が戻されるため、20 + 1 文字以上の領域が確保されていなくてはならない。パネルにより「取り消し」が指定された場合は、EX\_PAR のエラーとなる。

nvid NULL の場合、nvid で指定した領域には、新規に生成した実身を参照する仮身 ID が格納され、lnk NULL の場合は、lnk で指定した領域にリンクが格納される。

name が空文字列でない場合、実身は、vid で指定した仮身の所属デバイス上に生成される。所属デバイスが未定義 (ウィンドウの元となった実身が実身 / 仮身マネージャに登録されていない) の場合は、この関数を実行したプロセスの現在の作業ファイルと同一ファイルシステム上に生成される。

name が空文字列の場合、実身は、選択したデバイス上に生成される。

vid で指定した仮身の属するウィンドウに対して、生成した実身に対する「新規仮身の挿入要求」の仮身要求イベントが送信される。

指定した仮身が虚身状態の場合は、EX\_VOBJ のエラーとなる。

ファイル管理のエラーが発生した場合は、その旨のエラーパネルが表示された後にエラーリターン

する。

この関数は、新規実身への保存の際に使用される。

付箋の場合は、EX\_VOBJ のエラーとなる。

### 【エラーコード】

- EX\_ADR : アドレス(name,nvid,lnk)のアクセスは許されていない。
- EX\_PAR : パラメータが不正である(取り消し)。
- EX\_VID : 仮身 ID(vid)は存在していない。
- EX\_VOBJ : 仮身の状態 / 属性が不適當(虚身状態、付箋)。

## odsp\_inf

管理情報の表示

### 【形式】

ERR odsp\_inf(W vid)

### 【パラメータ】

W vid 仮身 ID

### 【リターン値】

- = 0 正常
- < 0 エラー (エラーコード)

### 【解説】

「管理情報ウィンドウ」を表示するシステムアプリケーションを起動し、vidで指定した仮身の参照する実身の「管理情報ウィンドウ」を表示する。

すでに、vidに対する「管理情報ウィンドウ」が表示されているときは、ウィンドウの切り替えを行う。

管理情報ウィンドウを表示するシステムアプリケーションが存在しない場合は、EX\_NOEXS のエラーとなる。

指定した仮身が虚身状態の場合は、EX\_VOBJ のエラーとなる。

付箋の場合は、EX\_VOBJ のエラーとなる。

### 【エラーコード】

- EX\_NOEXS : 管理情報パネルを表示するシステムアプリケーションが登録されていない。
- EX\_VID : 仮身 ID(vid)は存在していない。
- EX\_VOBJ : 仮身の状態 / 属性が不適當(虚身状態、付箋)。

# odsk\_inf

ディスク情報の表示

## 【形式】

ERR odsk\_inf(W vid, W used, W gabage)

## 【パラメータ】

W vid 仮身 ID  
W used 使用実身数  
W gabage くず実身数

## 【リターン値】

=0 正常  
<0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の属するファイルシステムの「ディスク状態パネル」を表示する。

パネルには、「ディスク状態」メニューを指定した場合と同様に、以下のような項目が表示されるが、[ディスク整理]の実行スイッチは表示されない。

ファイルシステム(ディスク)名  
全体サイズ (K バイト)  
使用サイズ (K バイト)  
空きサイズ (K バイト)  
使用比率 (%)  
使用実身数  
くず実身数

used は使用実身数、gabage はくず実身数を示し、used = 0 のときは、使用実身数、くず実身数の両方を実際にチェックした結果を表示するが、0 でない場合はチェックを行わず、used、gabage の値を、そのまま使用実身数、くず実身数としてパネルに表示する。

## 【エラーコード】

EX\_VID : 仮身ID(vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(虚身状態、付箋)。

# oget\_vob

仮身の取り出し

## 【形式】

W oget\_vob(W vid, VLINK \*vlnk, VP vseg, UW size, UW \*rsize)

## 【パラメータ】

W vid 仮身 ID  
VLINK \*vlnk 仮身(リンク)  
VP vseg 仮身セグメント  
UW size vseg で指定した領域のバイト数  
UW \*rsize 仮身セグメント全体のバイト数が戻される

## 【リターン値】

=0 正常 (vid は 仮身)  
=1 正常 (vid は 付箋)  
<0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の現在の状態を、vlnk で指定した領域に取り出し、対応する仮身セグメントを vseg で指定した領域に格納する。size は vseg で指定した領域のバイトサイズを示し、このサイズが仮身セグメントのサイズより小さい場合は、そのサイズ分のみ格納される。

rsize が NULL でない場合は、仮身セグメントの全体のバイト数が戻される。

vlnk が NULL の場合は、仮身データは格納されず、vseg が NULL の場合は、仮身セグメントは格納されない。

仮身を実体内に格納する場合は、この関数により最終的な仮身データおよび、仮身セグメントを取り出す必要がある。

また、起動されたアプリケーションが元の仮身の表示環境を取り出すためにも使用される。

付箋の場合は、vlnk には何も格納されず、vseg には FUSENSEG の内容が格納される。

関数値として、仮身のときは、"0"、付箋のときは、"1" が戻る。

特殊機能として、vid < 0 のときは - vid で指定した仮身が属するウィンドウの元となった仮身(親)が属するファイルシステムを vlnk に取り出す。不明のときは、EX\_NOEXS のエラーとなる。この場合、vseg, size, rsize は無視され、vlnk は NULL であってはならない。

## 【エラーコード】

EX\_ADR : アドレス(vlnk,vseg,rsize)のアクセスは許されていない。  
EX\_VID : 仮身ID(vid)は存在していない。  
EX\_NOEXS : 親のファイルシステムは不明 (vid < 0 のとき)

# ochg\_sts

仮身の状態の変更

## 【形式】

W ochg\_sts(W vid, UW mode)

## 【パラメータ】

W vid 0 : 仮身 ID  
< 0 : - (ウィンドウ ID)  
UW mode 状態変更モード  
::= V\_GETSTS V\_PURGE V\_CHKREF V\_CHKDUP  
(V\_NONAME | V\_NORELN | V\_NOTYPE | V\_NOTIME |  
V\_FIXDEF | V\_NOIMG | V\_NOPICT | V\_NOFDISP |  
V\_NOEXPND | V\_AUTEXE)

V\_GETSTS 0x8000

変更せずに現在の状態を取り出す

V\_PURGE 0xFFFF

保存されている開いた仮身の表示イメージを廃棄し、以後の表示時には必ず対応するアプリケーションを起動するようにする。

V\_CHKREF 0x8888

vid で指定した仮身の参照する実身 (リンクファイルの場合は参照先の実身) が他の仮身からも参照されているか否かをチェックする。

V\_CHKDUP 0x9999

vid で指定した仮身の参照するファイル (リンクファイルの場合はリンクファイルそのもの) が他の仮身からも参照されているか否かをチェックする。

または、以下の組み合わせで指定した属性に変更する。

V\_NONAME 0x0001 -- 実身名の表示なし  
V\_NORELN 0x0002 -- 続柄名の表示なし  
V\_NOTYPE 0x0004 -- データタイプの表示なし  
V\_NOTIME 0x0008 -- 変更日時の表示なし  
V\_FIXDEF 0x0010 -- 固定デフォルトアプリ  
V\_NOIMG 0x0020 -- イメージ保存なし  
V\_NOPICT 0x0040 -- ピクトグラム表示なし  
V\_NOFDISP 0x0080 -- 仮身枠表示なし  
V\_NOEXPND 0x0200 -- 印刷時展開なし  
V\_AUTEXE 0x4000 -- 自動起動

## 【リターン値】

0 正常 (変更前の属性 / 状態)  
< 0 エラー (エラーコード)

## 【解説】



vid で指定した仮身の状態を mode で指定した内容に変更し、変更前の状態を関数値として戻す。vid < 0 の場合は、- vid で指定した値をウィンドウ ID とみなして、そのウィンドウ / パネル上に登録されているすべての仮身の状態を変更する (ただし、wid < 0 で登録した仮身に対しては適用できない)。

関数値としては V\_CHKREF, V\_CHKDUP 指定の場合を除いて、vid 0 のときは仮身の属性 / 状態ワード全体が戻され、vid < 0 のときは "0" が戻る。変更に伴う再表示は一切行なわれない。V\_GETSTS 指定の場合は vid < 0 指定は意味を持たない。

V\_CHKREF, V\_CHKDUP 指定の場合、vid 0 のときは、他からも参照されているとき "1"、参照されていないとき "0" が戻る。vid < 0 のときは何にもせずに "0" を戻す。

付箋の場合は、V\_GETSTS, V\_NONAME, V\_NOTYPE, V\_AUTEXE のみ適用され、他は、EX\_PAR のエラーとなる。関数値として戻される値は、FUSENSEG の pict ワードの内容となる。

仮身、付箋とも、関数値として戻される属性のうち隠蔽属性ビットは実際の属性と無関係に常に "0" となる。

### 【エラーコード】

EX\_PAR : パラメータが不正である (mode が不正)。  
EX\_VID : 仮身ID(vid)は存在していない。

osta\_prc

仮身の実行処理の開始

### 【形式】

ERR osta\_prc(W vid, W wid)

### 【パラメータ】

W vid 仮身 ID  
W wid ウィンドウ ID

### 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

### 【解説】

実身 / 仮身マネージャから仮身のオープン起動で起動されたアプリケーション (小物アプリケーションを含む) は、起動されてウィンドウを開いた後、必ずこの関数を実行して処理の開始を知らせなくてはならない。

vid は、実身 / 仮身マネージャから渡される起動メッセージ (M\_EXECEQ 構造体) の中の対象仮身 ID (vid) を指定する。

wid は、アプリケーションがオープンしたウィンドウのウィンドウ ID を指定する。ウィンドウを開かず、単にパネルのみを使用するアプリケーション (小物アプリケーションを含む) では、オープンしたパネルのパネル ID を指定する。

小物アプリケーションでないアプリケーションの起動の場合、vid で指定した仮身を処理中状態とし、wid で指定したウィンドウをその対象ウィンドウとする。この時点で、仮身の表示は処理中状態の表示に変化し、以後は、その仮身の移動 / 変形に伴って、wid で指定したウィンドウの親ウィンドウ / 生成元が自動的に変更される。

小物アプリケーションの起動の場合、表示上の変化は発生せず、単に実身 / 仮身マネージャで内部的に管理している vid で指定した ID に対して wid で指定したウィンドウをその対象ウィンドウとする内部処理が行なわれることになる。

小物アプリケーションの場合は、wid として負の値を指定することにより、同一小物の多重起動を許すようにすることができる。そうでない場合は、多重起動はすでに起動されている同一の小物ウィンドウへの切り替えとなる。

付箋の場合は、EX\_VOBJ のエラーとなる。

### 【エラーコード】

EX\_PAR : パラメータが不正である (小物以外で wid 0)。  
EX\_VID : 仮身 ID (vid) は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當 (付箋)。

## oend\_prc

仮身の実行処理の終了

### 【形式】

GID oend\_prc(W vid, VP dat, W update)

### 【パラメータ】

W vid 仮身 ID  
VP dat 処理結果の状態を保存した付箋の固有データへのポインタ  
W update 対象ファイルの更新状態を通知  
= 0 : 実身は更新されていない。  
> 0 : 実身は更新されているが、実行メニューの更新は不要。  
< 0 : 実身は更新されており、実行メニューの更新も必要。

### 【リターン値】

0 正常 (開いた仮身の表示用描画環境 ID または 0)  
< 0 エラー (エラーコード)

### 【解説】

実身 / 仮身マネージャから仮身のオープン起動で起動されたアプリケーション (小物アプリケー

ションを含む)は、処理を終了した後、ウィンドウ(またはパネル)を閉じる前に、必ずこの関数を実行して終了を知らせなくてはならない。

アプリケーションは、osta\_prc() を実行しなかった場合でも、oend\_prc() は必ず実行しなくてはならない。

処理対象のファイルを更新する必要がある場合には、この関数を実行する前にファイルの更新を行ない、クローズしておく必要がある。

vid は、実身/仮身マネージャから渡される起動メッセージ(M\_EXECREQ 構造体)の中の対象仮身ID(vid)を指定する。

小物アプリケーションでないアプリケーションの起動の場合、vidで指定したウィンドウを対象とした仮身の処理中状態を解除し、通常状態とする。この時点で、仮身の表示は通常状態の表示に変化する。

小物アプリケーションの起動の場合、表示上の変化は発生せず、単に実身/仮身マネージャで内部的に管理しているvidで指定したIDに対する内部処理が行なわれることになる。

dat は処理結果の状態を保存した付箋の固有データへのポインタであり、最初のハーフワードが続くデータのバイト数を示す。この固有データは指定した仮身に付随するデータとして保存されることになる。NULLの場合は保存すべき固有データはないことを意味する。

関数値として、元の仮身が開いている場合は、その表示エリアの表示のための描画環境ID(gid > 0)が戻されるので、その描画環境ID(gid)に対して開いた仮身の内容の描画を行なう必要がある。描画領域は描画環境のフレーム長方形となる。開いた仮身の表示が不要の場合は、関数値"0"が戻る。

付箋の場合は、update 0 のとき、dat で指定した付箋の固有データの更新を意味する。

### 【エラーコード】

EX\_ADR : アドレス(dat)のアクセスは許されていない。  
EX\_VID : 仮身ID(vid)は存在していない。

## oend\_req

仮身の要求処理の終了

### 【形式】

ERR oend\_req(W vid, W stat)

### 【パラメータ】

W vid 仮身ID  
W stat 処理状態  
= 0: 正常終了  
0: 異常終了

### 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

### 【解説】

実身 / 仮身マネージャから仮身の表示起動または貼り込み起動で起動されたアプリケーションは、処理を終了した後、必ずこの関数を実行して終了を知らせなくてはならない。

vid は、実身 / 仮身マネージャから渡される起動メッセージの中の対象仮身ID (vid) を指定する。付箋の場合は、EX\_VOBJ のエラーとなる。

### 【エラーコード】

EX\_VID : 仮身ID(vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(付箋)。

## oreq\_prc

仮身要求イベントの送信

### 【形式】

W oreq\_prc(W wid, W req, B \*resp)

### 【パラメータ】

W wid ウィンドウ ID  
W req 仮身要求イベント  
B \*resp 応答の格納場所

### 【リターン値】

0 正常 (応答状態)  
< 0 エラー (エラーコード)

### 【解説】

wid で指定したウィンドウの管理プロセスに対して、req で指定した仮身要求イベントを送信し、その応答を resp で指定した領域に格納する。resp は要求に対する応答を格納できるだけの十分な大きさがなくてはならない。

送信される仮身要求イベントの要求タイプは、req で指定した値に + 128 された値となる。

関数値として、送信先から応答として戻された応答状態が戻る。正常応答を 0 とし、エラー応答を < 0 とする。

### 【エラーコード】

EX\_ADR : アドレス(resp)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(req < 0)。  
EX\_WID : ウィンドウ(wid)は存在していない。  
E(X)\_xxx : 送信先から戻されたエラー応答

## orsp\_prc

仮身要求イベントの応答

### 【形式】

ERR orsp\_prc(EVENT \*evt, B \*resp, UW size, W status)

### 【パラメータ】

|       |        |                |
|-------|--------|----------------|
| EVENT | *evt   | 仮身要求イベント       |
| B     | *resp  | 応答情報           |
| UW    | size   | 応答情報を取り出すサイズ   |
| W     | status | 応答状態として相手に戻す状態 |

### 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

### 【解説】

evt で指定した仮身要求イベントに対して、 resp で指定した領域に格納されている size バイトの情報を応答として戻す。

status は、応答状態として相手に戻す状態であり、正常応答を 0 とし、エラー応答を < 0 とする。

### 【エラーコード】

EX\_ADR : アドレス(evt, resp)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(evt の内容が不正)。

## oput\_dat

仮身への貼り込み

### 【形式】

ERR oput\_dat(W vid)

## 【パラメータ】

W vid 仮身 ID

## 【リターン値】

=0 正常  
<0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の参照する実身に、トレに格納されているデータを貼り込む。

データの貼り込みは、仮身のデフォルトアプリケーションを起動して行ない、貼り込みが正常に行なわれた場合は、関数値 "0" が戻り、データの貼り込みが行なわれなかった場合は、デフォルトアプリケーションから戻されたエラーコードがそのまま関数値として戻る。

指定した仮身が虚身状態の場合は、EX\_VOBJ のエラーとなる。

付箋の場合は、EX\_VOBJ のエラーとなる。

## 【エラーコード】

EX\_VID : 仮身 ID(vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(付箋)。  
E(X)\_xxx : 貼り込み先のアプリケーションからのエラーコード

# oupd\_fil

ファイル更新の通知

## 【形式】

ERR oupd\_fil(W vid, LINK \*lnk, W update)

## 【パラメータ】

W vid 仮身ID  
LINK \*lnk リンク  
W update 実行メニューの更新の必要性を指定  
0 : ファイルは更新されているが、実行メニューの更新は不要。  
< 0 : ファイルは更新されており、実行メニューの更新も必要。

## 【リターン値】

=0 正常  
<0 エラー (エラーコード)

## 【解説】

vid または lnk で指定したファイルを更新したことを通知する。vid = 0 の場合は、vid で指定した仮身の参照するファイルの更新を通知し、vid = 0 の場合は、lnk で指定したファイルの更新を意味する。vid = 0 の場合は、lnk の値は参照されないののでどんな値でも構わない。

仮身からウィンドウを開いて、対象のファイルを更新した場合は、この関数で更新を通知する必要はなく、対象ファイル以外を更新した場合に、この関数を実行する必要がある。更新されたファイルが「小物入れ」ファイルの場合、小物メニューは自動的に更新される。

付箋の場合は、EX\_VOBJ のエラーとなる。

## 【エラーコード】

EX\_ADR : アドレス(lnk)のアクセスは許されていない。  
EX\_VID : 仮身 ID(vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(付箋)。

# oset\_tmf

小物メニューファイルの設定

## 【形式】

ERR oset\_tmf(LINK \*lnk)

## 【パラメータ】

LINK \*lnk 「小物入れ」へのリンク

## 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

## 【解説】

lnk で指定したファイルを「小物入れ」ファイルとして設定し、小物メニューを更新する。lnk = NULL の場合は、小物メニューを再生成することを意味する。

インプリメントによっては、指定したファイルと別デバイスのルートファイル直下の同一名称のファイルも自動的に「小物入れ」ファイルとみなす場合もある。

システム立ち上げ時には、小物メニューは不能状態となっているため、ユーザ環境の初期化時に必ず、この関数を実行して小物メニューを有効とする必要がある。

lnk で指定したファイルは存在しない場合、アクセスできない場合、または機能付箋が1つも存在しない場合には、小物メニューは不能状態となる。

## 【エラーコード】

EX\_ADR : アドレス(Ink)のアクセスは許されていない。

## oget\_men

仮身のメニュー項目の取り出し

### 【形式】

ERR oget\_men(W vid, TC \*\*emenu, TC \*\*tmenu, TC \*\*vmenu, TC \*\*fmenu)

### 【パラメータ】

|    |         |                         |
|----|---------|-------------------------|
| W  | vid     | 仮身 ID                   |
| TC | **emenu | 実行メニューの項目リストポインタの格納場所   |
| TC | **tmenu | 小物メニューの項目リストポインタの格納場所   |
| TC | **vmenu | 仮身操作メニューの項目リストポインタの格納場所 |
| TC | **fmenu | 付箋操作メニューの項目リストポインタの格納場所 |

### 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

### 【解説】

実行メニュー、小物メニュー、仮身操作メニュー、付箋操作メニューの項目リストを共有メモリ領域に生成し、項目リストへのポインタを、それぞれ emenu, tmenu, vmenu, fmenu で指定した領域に格納する。emenu, tmenu, vmenu, fmenu のいずれかが NULL の場合は対応するポインタが格納されない。

vid の指定により以下に示すメニュー項目が取り出される。

vid > 0

vid で指定した単独の仮身または付箋に適用される実行 / 仮身操作 / 付箋操作メニューが取り出される。

仮身操作 / 付箋操作メニューのいずれかは常に不能状態となる。仮身操作メニューには指定した仮身の状態が反映される。

< 0

不能状態の実行 / 仮身操作 / 付箋操作メニューが取り出される。

= 0

不能状態の実行メニュー、共通項目のみの仮身操作メニュー、付箋操作メニューが取り出される。

= 0x4000

不能状態の実行 / 付箋操作メニュー、共通項目のみの仮身操作メニューが取り出される。

= 0x4001



不能状態の実行 / 仮身操作メニュー、付箋操作メニューが取り出される。

- 小物メニューは、vid の指定に無関係に常に同じものが取り出される。
- 付箋操作メニュー項目は共通項目のみである。

取り出されるメニュー項目は動的に変化する可能性があるため、基本的にアプリケーションはメニュー表示の直前に、選択状態に対応した vid を指定してこの関数を実行する必要がある。

## 【エラーコード】

EX\_ADR : アドレス (emenu, tmenu, vmenu, fmenu) のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_VID : 仮身 ID (vid) は存在していない。

## oget\_vmn

仮身のメニュー項目の分割取り出し

## 【形式】

```
ERR oget_vmn(W vid, TC **emenu, TC **tmenu, TC **vmenu, TC **omenu,  
             TC **dmenu, TC **fmenu)
```

## 【パラメータ】

|    |         |                           |
|----|---------|---------------------------|
| W  | vid     | 仮身 ID                     |
| TC | **emenu | 実行メニューの項目リストポインタの格納場所     |
| TC | **tmenu | 小物メニューの項目リストポインタの格納場所     |
| TC | **vmenu | 仮身操作メニューの項目リストポインタの格納場所   |
| TC | **omenu | 実身操作メニューの項目リストポインタの格納場所   |
| TC | **dmenu | ディスク操作メニューの項目リストポインタの格納場所 |
| TC | **fmenu | 付箋操作メニューの項目リストポインタの格納場所   |

## 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

## 【解説】

実行メニュー、小物メニュー、仮身操作メニュー、実身操作メニュー、ディスク操作メニュー、付箋操作メニューの項目リストを共有メモリ領域に生成し、項目リストへのポインタを、それぞれ emen, tmen, vmen, omen, dmen, fmen で指定した領域に格納する。指定した領域ポインタが NULL の場合は対応するポインタが格納されない。

oget\_men() と基本的に同様の機能であるが、oget\_men() で取り出す仮身操作メニューの内容が、仮身操作、実身操作、ディスク操作の 3 つのメニューに分割されて取り出される。

## 【エラーコード】

EX\_ADR : アドレス(emen, tmen, vmen, omen, dmen, fmen)  
のアクセスは許されていない。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_VID : 仮身 ID(vid)は存在していない。

## oexe\_vmn

仮身 / 付箋操作メニューの実行

### 【形式】

W oexe\_vmn(W vid, W item, VP buf)

### 【パラメータ】

W vid 仮身 ID  
W item 項目番号  
VP buf 操作に対応した情報が戻される

### 【リターン値】

0 正常 (操作タイプ)  
<0 エラー (エラーコード)

### 【解説】

vid で指定した仮身 / 付箋に対する仮身 / 付箋操作メニューで、item で指定した項目番号に対応する処理を実行する。item の値により以下の意味を持つ。

item > 0x100

oget\_men(), oget\_vmn() で取り出した付箋操作メニュー

item の下位バイトは付箋操作メニューの子項目番号

item 0

oget\_men() で取り出した仮身操作メニュー

item は仮身操作メニューの子項目番号

item < 0

oget\_vmn() で取り出した仮身 / 実身 / ディスク操作メニュー

N :

N は仮身操作メニューの子項目番号

(0x100 + N):

N は実身操作メニューの子項目番号

(0x200 + N):

N はディスク操作メニューの子項目番号

vid 0 の場合は、複数の仮身に対する操作を意味し、buf で指定した領域には、操作の対象となる仮身 ID の配列が入る。配列の最後は "0" で終わっていないといけない。関数値として操作の種類を示す以下の値が戻される。この値は、仮身 / 付箋操作メニューの項目番号とは無関係である。また、buf で指定した領域には、操作に対応した情報が戻される。

| 関数値              | buf の内容                        |
|------------------|--------------------------------|
| VM_NONE ( 0 )    | : 何も変更されなかった -                 |
| VM_OPEN ( 1 )    | : 仮身が開かれた  新しい仮身領域 ( RECT )    |
| VM_CLOSE ( 2 )   | : 仮身が閉じられた  新しい仮身領域 ( RECT )   |
| VM_NAME ( 3 )    | : 実身名が変更された -                  |
| VM_RELN ( 4 )    | : 続柄が変更された  新しい続柄インデックス        |
| VM_NEW ( 5 )     | : 新版が作成された  新版を参照する仮身 ID       |
| VM_DETACH ( 6 )  | : 切断状態となった -                   |
| VM_DISP ( 7 )    | : 表示属性が変更された  新しい仮身領域 ( RECT ) |
| VM_REFMT ( 8 )   | : 再フォーマットされた -                 |
| VM_PASTE ( 9 )   | : 仮身へ埋込まれた  アプリケーションからのリターン値   |
| VM_EXREQ ( 10 )  | : 付箋起動が要求された -                 |
| VM_ATTACH ( 11 ) | : 接続状態となった -                   |
| VM_APLREG ( 12 ) | : アプリ登録された -                   |
| VM_APLDEL ( 13 ) | : アプリ登録が削除された -                |
| VM_INFO ( 14 )   | : 管理情報が表示された -                 |
| VM_DISK ( 15 )   | : ディスク状態が表示された -               |
| VM_GABAGE ( 16 ) | : ディスク整理が起動された -               |

vid 0 の場合は、buf には、実際に操作が行なわれて状態が変化した仮身 ID とその結果の値がペアで戻され、最後は仮身 ID = 0 で終わる。エラーが発生したり、何も変更されなかった仮身に対しては、その仮身 ID は戻されない。

```

例：実行時：          W      仮身ID-1
                      :      :
                      W      仮身ID-n
                      W      0

実行終了時：
  VM_OPEN のとき：   W      仮身ID-1
                    RECT   1 の新しい仮身領域
                    :      :
                    W      仮身ID-n
                    RECT   n の新しい仮身領域
                    W      0

  VM_NAME のとき：   W      仮身ID-1
                    :      :
                    W      仮身ID-n
                    W      0

  VM_NEW のとき：    W      仮身ID-1
                    W      1 の新版を参照する仮身ID
                    :      :
                    W      仮身ID-n
                    W      n の新版を参照する仮身ID
  
```

buf は、vid > 0 の場合は、sizeof ( RECT ) の大きさを持つ必要があり、vid = 0 の場合は、(仮身の数) × (sizeof(RECT) + sizeof(W)) + sizeof(W) の大きさを持つ必要がある。

関数値が VM\_NONE、VM\_NEW、VM\_EXREQ、VM\_ATTACH ~ VM\_GABAGE 以外の場合は、指定した仮身の再表示を行なう必要がある。VM\_NEWの場合は、新版を参照する仮身が新規に生成されたため、その仮身を自ウィンドウ内に追加して表示する必要がある。

関数値が VM\_ATTACH ~ VM\_GABAGE の場合は、通常は何もする必要がない。

VM\_EXREQ は付箋操作メニューの「起動」が指定されたことを示す。この関数内では付箋の実行は行なわないため、oexe\_apg() により付箋の実行を行なう必要がある。

指定した仮身に対しては仮身要求イベントは送信されないが、指定した以外の仮身の表示の変更が必要な場合は、その仮身に対して仮身要求イベントが送信される。

vid = 0 の場合、適用されないメニュー項目を指定された場合は、EX\_VOBJ のエラーとなる。

### 【エラーコード】

EX\_ADR : アドレス(buf)のアクセスは許されていない。  
 EX\_PAR : パラメータが不正である(item が不正)。  
 EX\_VID : 仮身 ID(vid)は存在していない(vid > 0 のとき)。  
 EX\_VOBJ : 仮身の状態 / 属性が不適當(付箋)。  
 EX\_xxx : 対応する関数から戻されたエラー(vid > 0 のとき)

## ochg\_env

実行環境ファイルの変更

### 【形式】

W ochg\_env(LINK \*lnk, W wid)

### 【パラメータ】

LINK \*lnk 実行環境ファイルへのリンク  
 W wid ウィンドウ ID

### 【リターン値】

0 正常 (実行環境ファイルのスタックレベル(0: 無))  
 <0 エラー (エラーコード)

### 【解説】

lnk で指定したファイルを現在の実行環境ファイルとして設定し、wid で指定したウィンドウを実行環境ウィンドウとする。lnk で指定したファイルの存在、及び wid で指定したウィンドウの存在はチェックされない。

実行環境ファイル/ウィンドウを設定した場合、以前の実行環境ファイル/ウィンドウはスタック内に保存される。

指定したファイルが以前の実行環境ファイルとして定義されている場合は、スタック内に保存されている以前の定義は自動的に削除される。

Ink = NULL の場合で、wid = 0 または指定した wid が現在の実行環境ウィンドウである場合はスタック内に保存されている直前の実行環境ファイルに戻す。

Ink = NULL の場合で、指定した wid が現在の実行環境ウィンドウでない場合は、指定した wid がスタック内に保存されていればその定義を削除する。この場合、実行環境ファイルは変更されない。

実行環境ウィンドウとして定義されているウィンドウが削除された場合は、自動的にスタック内の定義が削除され、場合によっては直前の実行環境ファイルに切り換わる。

システム立ち上げ時には、実行環境ファイルは未定義となっているため、ユーザ環境の初期化時に必ず、この関数を実行して実行環境ファイルを設定する必要がある。

関数値として実行環境ファイルのスタックレベルが戻る。"0" の場合は実行環境ファイルが未定義であることを意味する。

### 【エラーコード】

EX\_ADR : アドレス(Ink)のアクセスは許されていない。

EX\_LIMIT : システムの制限を越えた(実行環境ファイルの数が多すぎる)。

## oget\_env

実行環境ファイルの取り出し

### 【形式】

WID oget\_env(LINK \*Ink)

### 【パラメータ】

LINK \*Ink 実行環境ファイルへのリンク

### 【リターン値】

0 正常 (実行環境ウィンドウ ID)

<0 エラー (エラーコード)

### 【解説】

現在の実行環境ファイルを取り出し、Ink で指定した領域に格納し、実行環境のウィンドウ ID を関数値として戻す。Ink が NULL のときは、実行環境ファイルは取り出さない。

実行環境ファイルが未定義である場合は、EX\_NOEXS のエラーとなる。

## 【エラーコード】

EX\_ADR : アドレス(lnk)のアクセスは許されていない。  
EX\_NOEXS : 実行環境ファイルは未定義。

# oreg\_apg

アプリケーションプログラムの登録

## 【形式】

ERR oreg\_apg(LINK \*lnk)

## 【パラメータ】

LINK \*lnk アプリケーションプログラムへのリンク

## 【リターン値】

=0 正常  
<0 エラー (エラーコード)

## 【解説】

lnk で指定したファイルを、そのファイルが属するファイルシステム内のルートファイル直下に存在するアプリケーション登録ファイルにアプリケーションプログラムとして登録し、同時にアプリケーションのデータタイプ名も格納する。登録すべきファイルシステムは接続されていないといけない。

指定したファイルがアプリケーションプログラムの形式でない場合、およびアプリケーション ID のデータタイプ ID が (0, 0) の場合は、EX\_PAR のエラーとなる。また、既に同一ファイルが登録されていた場合は、EX\_EXS のエラーとなる。

同一のアプリケーション ID を持つ異なるアプリケーションプログラムが複数登録されていた場合は、実行時にどれを実行するかを選択することになる。

## 【エラーコード】

EX\_ADR : アドレス(lnk)のアクセスは許されていない。  
EX\_EXS : 既に同一ファイルが登録されていた。  
EX\_PAR : パラメータが不正である  
(アプリケーションプログラム形式ではない、データタイプ ID が 0)。

# odel\_apg

アプリケーションプログラムの登録削除

## 【形式】

ERR odel\_apg(LINK \*Ink)

**【パラメータ】**

LINK \*Ink アプリケーションプログラムへのリンク

**【リターン値】**

= 0 正常  
< 0 エラー (エラーコード)

**【解説】**

Ink で指定したファイルを、そのファイルが属するファイルシステム内のアプリケーション登録ファイルからアプリケーションプログラムとしての登録を削除し、同時にアプリケーションのデータタイプ名も削除する。登録削除すべきファイルシステムは接続されていないといけない。

登録を削除するだけで、アプリケーションプログラム自体が削除されることはない。

指定したファイルがアプリケーションプログラムとして登録されていない場合はエラーとなる。

**【エラーコード】**

EX\_ADR : アドレス(Ink)のアクセスは許されていない。  
EX\_NOEXS : アプリケーションは登録されていない。

oset\_sea

検索対象ファイルシステムの設定

**【形式】**

W oset\_sea(W cnt, LINK \*Ink)

**【パラメータ】**

W cnt リンクの個数  
LINK \*Ink リンク配列

**【リターン値】**

0 正常 (検索対象ファイルシステム数)  
< 0 エラー (エラーコード)

## 【解説】

Ink で指定したファイルの属するファイルシステムをアプリケーションプログラムの検索対象として設定する。Ink で指定したファイルシステムの存在はチェックされない。

Ink は、cnt で指定した要素数のリンク配列へのポインタであり、要素の順番に検索対象とする。

アプリケーションプログラムの検索は、まず現在の実行環境ファイルの存在するファイルシステムに対して行なわれ、その次に設定したファイルシステムを順番に検索することになる。cnt = 0 または、Ink = NULL の場合は、現在の実行環境ファイルの存在するファイルシステムのみを検索対象とすることを意味する。

関数値として、設定した結果としての検索対象ファイルシステムの数を戻す。

## 【エラーコード】

- EX\_ADR : アドレス(Ink)のアクセスは許されていない。
- EX\_LIMIT : システムの制限を越えた(検索対象ファイルシステムの数が多すぎる)。
- EX\_PAR : パラメータが不正である(cnt < 0)。

## oget\_sea

検索対象ファイルシステムの取り出し

## 【形式】

W oget\_sea(W cnt, LINK \*Ink)

## 【パラメータ】

|      |      |          |
|------|------|----------|
| W    | cnt  | Ink の要素数 |
| LINK | *Ink | リンク配列    |

## 【リターン値】

- 0 正常 (検索対象ファイルシステム数)
- <0 エラー (エラーコード)

## 【解説】

現在の検索対象ファイルシステムを取り出し、そのファイルシステムのルートファイルへのリンクを、Ink で指定した領域に格納する。関数値として現在、検索対象ファイルシステムとして設定されているファイルシステム数を戻す。

cnt は Ink で指定した配列領域の要素数であり、検索対象ファイルシステムの数が cnt で指定した値より大きい場合は、Ink で指定した領域には cnt 個のリンクのみ戻されるが、関数値は全体の数が戻る。

現在の実行環境ファイルの存在するファイルシステムのみが検索対象の場合は、関数値として "0" が戻り、Ink には何も設定されない。



## 【エラーコード】

EX\_ADR : アドレス(lnk)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(cnt < 0)。

# ofnd\_apg

アプリケーションプログラムの検索

## 【形式】

W ofnd\_apg(VLINK \*vlnk, LINK \*lnk, TC \*dtype, W cnt)

## 【パラメータ】

|       |        |                |
|-------|--------|----------------|
| VLINK | *vlnk  | 仮身(リンク)        |
| LINK  | *lnk   | 検索したリンクを格納する配列 |
| TC    | *dtype | データタイプを格納する配列  |
| W     | cnt    | 検索回数           |

## 【リターン値】

0 正常 (格納したアプリケーションの数)  
<0 エラー (エラーコード)

## 【解説】

vlnk で指定した条件に適合するアプリケーションプログラムを cnt で指定した回数だけ検索し、見つかったアプリケーションのリンクを lnk で指定した領域に、そのデータタイプ名を dtype で指定した領域に格納する。lnk, dtype がそれぞれ NULL の場合は格納されない。

lnk は、cnt 個の LINK 構造体配列へのポインタであり、dtype は cnt × 32 バイトの領域へのポインタである。データタイプ名が 32 バイトに満たない場合は後ろに "0" が詰められる。

検索の条件は以下に示すものとなる。

vlnk->fs\_name[0] 0 のとき :

fs\_name で指定したファイルシステム内のみを検索する。接続されていない場合は、ファイル管理の E\_NOFS エラーが戻る。

vlnk->fs\_name[0] = 0 のとき :

実行環境ファイルの存在するファイルシステム、次に、設定されている検索対象ファイルシステム全体を順番に検索する(接続されていないものは除かれる)。これは、oexe\_apg() で検索される条件に等しい。

vlnk->appl[0][1] 0 のとき :

vlnk->appl[0][1][2] で指定したアプリケーション ID を持つアプリケーションを検索する。

vlnk->appl[0][1] = 0 のとき :

すべてのアプリケーションを検索する。

関数値として、見つかったアプリケーションの数を戻す。"0" の場合は、アプリケーションは存在しないことを意味する。検索は、最大 cnt 個のアプリケーションが見つかった時点で終了するため、関数値 cnt となる。

### 【エラーコード】

EX\_ADR : アドレス(vInk, Ink, dtype)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(cnt 0)。

## oexe\_apg

アプリケーションプログラムの実行

### 【形式】

W oexe\_apg(W vid, W item)

### 【パラメータ】

W vid 仮身 ID  
W item アプリケーションの番号

### 【リターン値】

0 正常 (起動したプロセス ID (または 0))  
<0 エラー (エラーコード)

### 【解説】

vid で指定した仮身に対して item の下位バイトで指定したアプリケーションを起動し、関数値として起動したアプリケーションのプロセス ID を戻す。item の上位バイトは無視される。

アプリケーションの起動は、vid の値により以下の意味となる。

vid > 0:

item は、vid で指定した仮身の実行メニューの項目番号であり、item の下位バイト = 0 の場合はデフォルトアプリケーションの起動を意味する。

vid 0:

item は、小物メニューの項目番号であり、小物アプリケーションの起動となる。

仮身が既に処理中状態の場合、もしくは小物アプリケーションが既に実行中の場合は、item の値に無関係に対応するウィンドウに入力受付状態が切り換えられ、関数値として "0" が戻る。ただし、実行中の小物アプリケーションが多重起動を許している場合は、新規に小物アプリケーションが実行される(osta\_prc() を参照のこと)。

vid で指定した仮身が虚身状態の場合、「デバイス接続パネル」によりデバイスの接続を求める。接続された後、デフォルト起動のときはアプリケーションが起動されるが、デフォルト起動

でないときは接続のみ行なわれて関数値として "0" が戻る。

起動すべきアプリケーションが複数存在する場合は、最新の更新日時のアプリケーションを起動するか、「アプリケーション選択パネル」を表示して、起動するアプリケーションを選択するかはインプリメントに依存する。「アプリケーション選択パネル」を表示して、選択が取り消された場合は、EX\_NOEXS のエラーとなる。

起動に際して何らかのエラーが発生した場合は、その旨を示すエラーパネルが表示された後にエラーリターンする。ただし、インプリメントによっては、起動すべきアプリケーションが見つからなかった場合に、アプリケーションの機能付箋名をパネルに表示して、アプリケーションの入ったデバイスの接続を要求する場合もある。この場合、デバイスの接続が行なわれた後にアプリケーションが起動され、デバイスの接続が取り消された場合は、EX\_NOEXS のエラーとなる。

付箋の場合、item の下位バイトが "0" でない場合は、EX\_PAR のエラーとなる。

## 【エラーコード】

EX\_NOEXS : アプリケーションは登録されていない、起動が取り消された。  
EX\_PAR : パラメータが不正である(item が不正)。  
EX\_VID : 仮身 ID(vid)は存在していない。

## odet\_fls

ファイルシステムの切断

## 【形式】

ERR odet\_fls(TC \*dev, W eject)

## 【パラメータ】

TC \*dev デバイス名  
W eject イジェクト指定

## 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

## 【解説】

dev で指定したディスクデバイス上のファイルシステムをシステムから切り離す。eject 0 の場合は切り離れた後、ファイルシステムの媒体をイジェクトして物理的に取り外す。イジェクトが不可能なデバイスの場合はこの指定は無視される。

この関数は、OS 核の det\_fls() と同一の機能であるが、実身 / 仮身マネージャでデバイスの接続管理を正しく行なうために、基本的にアプリケーションは、det\_fls() の代わりに odet\_fls() を使用しなくてはならない。

## 【エラーコード】

EX\_ADR : アドレス(dev)のアクセスは許されていない。

## ofdt\_fls

ファイルシステムの切断(強制)

### 【形式】

ERR ofdt\_fls(TC \*dev, W eject)

### 【パラメータ】

TC \*dev デバイス名  
W eject イジェクト指定

### 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

### 【解説】

odet\_fls() と完全に同一の機能であるが、実身/仮身マネージャでの他のシステムコールを実行中であっても、待たされずに強制的に処理を行う。

### 【エラーコード】

EX\_ADR : アドレス(dev)のアクセスは許されていない。  
ER\_BUSY : ビジー状態のため切断できない。

## oatt\_fls

ファイルシステムの接続

### 【形式】

ERR oatt\_fls(TC \*dev, TC \*name, LINK \*lnk, W ronly)

### 【パラメータ】

TC \*dev デバイス名  
TC \*name 接続名  
LINK \*lnk リンク  
W ronly 読みだし専用指定

## 【リターン値】

=0 正常  
<0 エラー (エラーコード)

## 【解説】

dev で指定した名前のディスクデバイス上に存在するファイルシステムを、name で指定した接続名により、システムに接続する。

name は、接続したファイルシステムを一意的に区別するために使用される任意の名前であり、先頭から 8 文字までが適用される。既に接続済みの接続名と同じであってはいけない。この接続名は、接続したファイルシステムのルートファイルを絶対パス名により指定するために使用される。

Ink は、接続したファイルシステムのルートファイルへのリンクが格納される領域であり、Ink = NULL の場合は格納されない。

r\_only 0 の場合は、書き込み不可のファイルシステムとして接続される。この場合、そのファイルシステム内のいかなる書き込みも禁止される。

この関数は、OS 核の att\_fls() と同一の機能であるが、実身 / 仮身マネージャでデバイスの接続管理を正しく行なうために、基本的にアプリケーションは、att\_fls() の代わりに oatt\_fls() を使用しなくてはならない。

## 【エラーコード】

EX\_ADR : アドレス(dev, name, Ink)のアクセスは許されていない。

# ofat\_fls

ファイルシステムの接続(強制)

## 【形式】

ERR ofat\_fls(TC \*dev, TC \*name, LINK \*Ink, W ronly)

## 【パラメータ】

|      |       |          |
|------|-------|----------|
| TC   | *dev  | デバイス名    |
| TC   | *name | 接続名      |
| LINK | *Ink  | リンク      |
| W    | ronly | 読みだし専用指定 |

## 【リターン値】

=0 正常  
<0 エラー (エラーコード)

## 【解説】

oatt\_fls() と完全に同一の機能であるが、実身/仮身マネージャでの他のシステムコールを実行中であっても、待たされずに強制的に処理を行う。

## 【エラーコード】

EX\_ADR : アドレス(dev, name, lnk)のアクセスは許されていない。  
ER\_BUSY : ビジー状態のため接続できない。

# odet\_vob

仮身の切断

## 【形式】

W odet\_vob(W vid, TC \*dev)

## 【パラメータ】

W vid 仮身ID  
TC \*dev デバイス名

## 【リターン値】

= 0 正常 (切断済み)  
= 1 正常 (切断した)  
< 0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の示す実身の存在するファイルシステム、または dev で指定した名前の論理デバイス上のファイルシステムを切断する。

vid > 0 の場合は、vid の指定が有効となり dev は無視されるが、dev NULL で実際に切断された場合は、切断された論理デバイス名が dev に戻される。vid = 0 の場合には dev が有効となり、dev が空文字列の場合は、特別にシステムデバイスの切断を意味し、dev には論理デバイス名が戻される。dev で指定した領域は 8 + 1 文字の領域が確保されていなくてはならない。

イジェクト可能なメディアの場合は、同時にイジェクトされる。

既に切断されている場合は何もせずに関数値 "0" を戻す。

切断した場合は関数値 "1" を戻し、切断したファイルシステム内の登録された全ての仮身を虚身状態とし、これらの仮身が登録されたウィンドウの管理プロセスに対して仮身要求イベント(虚身移行)を送信する (vid で指定した仮身に対しても送信される)。

処理中状態の仮身に対しては、その仮身から開かれたウィンドウの管理プロセスに対して仮身要求イベント(虚身ウィンドウ移行)を送信する。この場合、そのウィンドウのタイトル表示は自動的に虚身ウィンドウ表示に変更される。虚身ウィンドウ内に含まれる仮身に対しては、仮身要求イ

ベント(虚身移行)は送信されないため、虚身ウィンドウ移行の仮身要求イベントを受信した場合は、基本的にそのウィンドウ内の全ての仮身の再表示を行なう必要がある。

切断したファイルシステムは自動的に検索対象ファイルシステムから削除される。また、必要に応じて小物メニューが自動的に更新される。

切断できない場合は、その旨のエラーパネルが表示された後にエラーリターンする。

付箋の場合は、EX\_VOBJのエラーとなり、エラーパネルは表示されない。

### 【エラーコード】

EX\_ADR : アドレス(dev)のアクセスは許されていない。  
EX\_VID : 仮身 ID (vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(付箋)。

## oatt\_vob

仮身の接続

### 【形式】

W oatt\_vob(W vid, W ronly)

### 【パラメータ】

W vid 仮身 ID  
W ronly 読みだし専用指定

### 【リターン値】

= 0 正常 (取り消し)  
= 1 正常 (接続済み)  
= 2 正常 (接続した)  
< 0 エラー (エラーコード)

### 【解説】

vidで指定した仮身の示す実身の存在するファイルシステムを接続する。r\_only 0の場合は、書込不可のファイルシステムとして接続する。接続名はシステムディスクの場合は"SYS"であり、他の場合は論理デバイス名と同じ名前となる。

接続すべきファイルシステムがイジェクト可能なメディアの場合は、ディスクの挿入を求めるパネルを表示する。挿入が取り消された場合は何もせず関数値 "0" を戻す。

既に接続済みの場合は何もせずに関数値 "1" を戻す。この場合、r\_onlyのパラメータは無視される。

接続した場合は関数値 "2" を戻し、接続したファイルシステム内の登録されたすべての仮身の虚身状態を解除し、虚身が登録されたウィンドウの管理プロセスに対して仮身要求イベント(虚身

解除)を送信する(vidで指定した仮身に対しても送信される)。

処理中状態の虚身に対しては、その虚身から開かれたウィンドウの管理プロセスに対して仮身要求イベント(虚身ウィンドウ解除)を送信する。この場合、そのウィンドウのタイトル表示は自動的に通常ウィンドウ表示に変更される。虚身ウィンドウ内に含まれる仮身に対しては、仮身要求イベント(虚身解除)は送信されないため、虚身ウィンドウ解除の仮身要求イベントを受信した場合は、基本的にそのウィンドウ内の全ての仮身の再表示を行なう必要がある。

接続したファイルシステムは自動的に検索対象ファイルシステムの最後に追加される。また、必要に応じて小物メニューが自動的に更新される。

接続できない場合は、その旨のエラーパネルが表示された後にエラーリターンする。

付箋の場合は、EX\_VOBJのエラーとなり、エラーパネルは表示されない。

### 【エラーコード】

EX\_VID : 仮身ID(vid)は存在していない。  
EX\_VOBJ : 仮身の状態/属性が不適當(付箋)。

opr\_c\_dev

デバイス挿入イベント処理

### 【形式】

ERR opr\_c\_dev(EVENT \*evt, TC \*dev, W ronly)

### 【パラメータ】

|       |       |          |
|-------|-------|----------|
| EVENT | *evt  | デバイスイベント |
| TC    | *dev  | デバイス名    |
| W     | ronly | 読みだし専用指定 |

### 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

### 【解説】

evtで指定したデバイスイベント(ディスク挿入イベント)に従って、挿入されたデバイス上のファイルシステムを接続する。r\_only 0の場合は、書込不可のファイルシステムとして接続する。接続名はシステムディスクの場合は"SYS"であり、他の場合は論理デバイス名と同じ名前となる。

evtがデバイスイベントでない場合は、EX\_PARのエラーとなるが、ディスク挿入イベント以外のデバイスイベントの場合は、何もせずに正常リターンする。

evt = NULLの場合は、devで指定した論理デバイスの接続となる。



デバイスが未フォーマットまたは標準形式でない場合の対応は、デバイスの種別やインプリントによって異なってくる。フロッピーディスクの場合は、通常、その旨のパネルを表示し、フォーマットや対応アプリケーションの実行を選択し、フォーマットした場合は、フォーマット後のファイルシステムを接続し、取り消した場合は、接続せずにエラーリターンする。

既に同一のファイルシステム名のデバイスが接続されている場合は、その旨のパネルを表示し、ファイルシステム名の変更を求める。変更した場合は接続が行なわれ、変更を取り消した場合は、接続せずにエラーリターンする。

他の原因で接続できない場合は、その旨のエラーパネルを表示し、確認された後に、エラーリターンする。

接続した場合は、接続したファイルシステム内の登録されたすべての仮身の虚身状態を解除し、虚身が登録されたウィンドウの管理プロセスに対して仮身要求イベント(虚身解除)を送信する。

処理中状態の虚身に対しては、その虚身から開かれたウィンドウの管理プロセスに対して仮身要求イベント(虚身ウィンドウ解除)を送信する。この場合、そのウィンドウのタイトル表示は自動的に通常ウィンドウ表示に変更される。虚身ウィンドウ内に含まれる仮身に対しては、仮身要求イベント(虚身解除)は送信されないため、虚身ウィンドウ解除の仮身要求イベントを受信した場合は、基本的にそのウィンドウ内の全ての仮身の再表示を行なう必要がある。

接続したファイルシステムのデバイス仮身が実行環境ウィンドウに存在しない場合、実行環境ウィンドウの管理プロセスに対して、仮身要求イベント(デバイス仮身の挿入要求)を送信する。

正常に接続された場合、`evt` `NULL` で `dev` `NULL` のときは、デバイス挿入イベントが発生した論理デバイス名が格納される。従って、`dev` は、8 + 1文字分の領域がなくてはならない。

接続したファイルシステムは自動的に検索対象ファイルシステムの最後に追加される。また、必要に応じて小物メニューが自動的に更新される。

この関数は、デバイスイベント(ディスク挿入イベント)が発生した場合に必ず実行しなくてはならない。

## 【エラーコード】

EX\_ADR : アドレス(`evt`,`dev`)のアクセスは許されていない。

EX\_PAR : パラメータが不正である(`evt` の内容が不正)。

## ofmt\_vob

仮身の再フォーマット処理

## 【形式】

W ofmt\_vob(W vid)

## 【パラメータ】

W vid 仮身ID

## 【リターン値】

- =0 正常 (再フォーマットされた)
- =1 正常 (取り消された)
- <0 エラー (エラーコード)

**【解説】**

vid で指定したデバイス仮身が示すデバイスを再フォーマットする。新たにフォーマットされたデバイスは接続され、以後、vid が指す仮身はこのデバイスを参照するデバイス仮身に変換される。

指定した仮身が切断状態、またはデバイス仮身でない場合は、EX\_VOBJ のエラーとなる。

デバイス内に処理中状態の実身が存在する場合は、その旨のエラーパネルを表示した後、EX\_VOBJ でエラーリターンする。

デバイスが切断できない場合は、その旨のエラーパネルを表示した後、エラーリターンする。

再フォーマットされた場合、元のデバイスを参照する他のデバイス仮身に対して、名称変更の仮身要求イベントが送信され、元のデバイス内の実身を参照する他の仮身に対して、虚身状態移行の仮身要求イベントが送信される (vid で指定した仮身に対しては送られない)。

再フォーマットされた場合は関数値 "0" が戻り、再フォーマットが取り消された場合は関数値 "1" が戻る。

付箋の場合は、X\_VOBJ のエラーとなる。

**【エラーコード】**

- EX\_VID : 仮身 ID(vid)は存在していない。
- EX\_VOBJ : 仮身の状態 / 属性が不適當  
(処理中状態、切断状態、デバイス仮身でない、付箋)。

## ocnv\_vob

仮身の変換

**【形式】**

W ocnv\_vob(W org, W vid, LINK \*Ink)

**【パラメータ】**

- W org 仮身 ID
- W vid 仮身 ID
- LINK \*Ink 実身のリンク

**【リターン値】**

- =0 正常 (リンクファイルは生成しなかった)

=1 正常 (リンクファイルを生成した)

<0 エラー (エラーコード)

### 【解説】

org で指定した仮身の参照する実身内に、vid で指定した仮身を保存するために、仮身の変換を行なう。Ink NULL の場合は、Ink で指定した領域に変換後の仮身が参照する実身のリンクが格納される。

vid で指定した仮身の参照する実身が、org で指定した仮身の参照する実身と、同一のファイルシステムに存在する場合は、何の変換も行なわれず、関数値として "0" を戻す。

そうでない場合は、vid で指定した仮身の参照する実身を参照するためのリンクファイルが、org で指定した仮身の参照する実身が存在するファイルシステム上に生成され、vid で指定した仮身は、生成したリンクファイルを参照する仮身に変換される。同時に元の続柄名を保つために仮身の続柄インデックスの変更、および続柄ファイルへの自動登録が行なわれる。この場合は、関数値として "1" が戻る。

org で指定した仮身が虚身状態の場合は、EX\_VOBJ のエラーとなる。

この関数は、アプリケーションが仮身を実身に保存する場合に使用される。

付箋の場合は、EX\_VOBJ のエラーとなる。

### 【エラーコード】

EX\_ADR : アドレス(Ink)のアクセスは許されていない。  
EX\_VID : 仮身ID(vid,org)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(切断状態、付箋)。

## oopn\_obj

実身のオープン

### 【形式】

W oopn\_obj(W vid, LINK \*Ink, UW omode, TC \*pwd)

### 【パラメータ】

|      |       |          |
|------|-------|----------|
| W    | vid   | 仮身 ID    |
| LINK | *Ink  | 実身のリンク   |
| UW   | omode | オープンモード  |
| TC   | *pwd  | パスワード文字列 |

### 【リターン値】

0 正常 (オープンした実身のファイルディスクリプタ)  
<0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の参照する実身を、 o\_mode で指定したモード、 pwd で指定した合言葉でオープンし、 そのファイルディスクリプタを関数値として戻す。

Ink NULL の場合は、 Ink で指定した領域に仮身の参照する実身のリンクが格納される。

指定した仮身が虚身状態の場合は「デバイス接続パネル」が表示されて、 ファイルシステムの接続処理が行なわれる。 接続されなかった場合はエラーリターン ( EX\_VOBJ )する。

ファイルがオープンできなかった場合は、 その旨のエラーパネルの表示が行なわれ、 ファイル管理から戻されたエラーコードを関数値としてエラーリターンする。 ただし、 合言葉が正しくないエラーのときは、 合言葉の入力処理を行ない、 正しい合言葉が入力された場合は正常リターンする。 合言葉の入力が取り消された場合は、 エラーリターン ( EX\_PWD )する。

通常、 アプリケーションは実身をオープンする場合はファイル管理の opn\_fil() の代わりにこの関数を使用する。

付箋の場合は、 EX\_VOBJ のエラーとなる。

## 【エラーコード】

EX\_ADR : アドレス(Ink,pwd)のアクセスは許されていない。  
EX\_VID : 仮身ID(vid)は存在していない。  
EX\_VOBJ : 仮身の状態/属性が不適當(切断状態、付箋)。

## odel\_obj

実身の削除

## 【形式】

W odel\_obj(LINK \*Ink, W mode)

## 【パラメータ】

LINK \*Ink 実身のリンク  
W mode 削除モード  
0 : 削除できなかった場合にエラーパネルを表示する。  
= 0 : エラーパネルを表示しない。

## 【リターン値】

0 正常 (実際に削除した実身数) < 0 エラー (エラーコード)

## 【解説】

Ink で指定したファイルが削除可能である場合に削除し、 さらに、 そのファイルに含まれるリンクの示すファイル (ネストしたファイル) が削除可能になった場合はネストして削除する。

削除可能であるのは、 ファイルの参照カウント = 0 で、 かつ、 そのファイルを参照している仮身

が登録されていない場合となる。この条件はネストしたファイルに対しても適用される。

## 【エラーコード】

EX\_ADR : アドレス(Ink)のアクセスは許されていない。

oget\_fsn

付箋固有データの取り出し

## 【形式】

ERR oget\_fsn(W vid, W fd, VP dat, UW size)

## 【パラメータ】

|    |      |                |
|----|------|----------------|
| W  | vid  | 仮身 ID          |
| W  | fd   | 実身のファイルディスクリプタ |
| VP | dat  | 固有データ格納領域      |
| UW | size | dat のバイト数      |

## 【リターン値】

=0 正常  
<0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の示す実身内に存在する起動対象となった実行機能付箋レコード内の固有データを取り出して dat で指定した領域に格納する。size は dat で指定した領域のバイトサイズを示し、このサイズが固有データのサイズより小さい場合は、そのサイズ分のみが格納される。

格納される固有データの先頭には、固有データのバイトサイズ (dlen) が格納され、その後には固有データ自体が格納される。

vid は、実身 / 仮身マネージャから渡される起動メッセージ (M\_EXECREQ 構造体) の中の対象仮身ID (vid) を指定する。

この関数の実行前に、アプリケーションは自分で対象の実身をリードまたは更新オープンしておき、オープン時に得られたファイルディスクリプタを fd として渡す必要がある。

この関数は、仮身のオープン起動で起動された場合にアプリケーションが実行機能付箋レコードの固有データを対象実身から取り出すために使用する。ただし、デフォルト起動の場合にこの関数を使用すると、実身 / 仮身マネージャで保持されている固有データではなく、対象実身内に存在する実行機能付箋レコードの固有データを取り出すため、デフォルト起動の場合は、通常、この関数を使用せずに起動時に渡された固有データへのポインタを使用して固有データを取り出す必要がある。指定した仮身が処理中状態でない場合、および切断状態の場合は、EX\_VOBJ のエラーとなる。

付箋の場合は、EX\_VOBJ のエラーとなる。

## 【エラーコード】

EX\_ADR : アドレス(dat)のアクセスは許されていない。  
EX\_VID : 仮身 ID(vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(切断状態、処理中状態でない、付箋)。  
EX\_PAR : パラメータが不正である(size < 0)。

## oput\_fsn

付箋固有データの格納

## 【形式】

ERR oput\_fsn(W vid, W fd, VP dat)

## 【パラメータ】

W vid 仮身 ID  
W fd 実身のファイルディスクリプタ  
VP dat 固有データ

## 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

## 【解説】

vid で指定した仮身の示す実身内に存在する起動対象となった実行機能付箋レコード内の固有データを、dat で指定した領域の内容で更新する。dat で指定した領域の先頭に、固有データのバイトサイズ(dlen)があり、その後には固有データ自体があるものとみなす。

vid は、実身 / 仮身マネージャから渡される起動メッセージ (M\_EXECREQ 構造体) の中の対象仮身ID (vid) を指定する。

この関数の実行前に、アプリケーションは自分で対象の実身を更新オープンしておき、オープン時に得られたファイルディスクリプタを fd として渡す必要がある。

この関数は、アプリケーションが対象実身内の実行機能付箋レコードの固有データを更新するために使用する。デフォルト起動の場合でも、通常は対象実身内の実行機能付箋レコードの固有データを更新しておく必要があるため、この関数を使用して固有データの更新を行なう。さらに、実身 / 仮身マネージャ内で保持されている固有データも oend\_prc() 関数を使用して更新する必要がある。

なお、この関数では対象ファイルの更新日時は変更されない。

指定した仮身が処理中状態でない場合、および切断状態の場合は、EX\_VOBJ のエラーとなる。従って、この関数は oend\_prc() を実行する以前に実行しなくてはならない。

付箋の場合は、EX\_VOBJ のエラーとなる。

## 【エラーコード】

EX\_ADR : アドレス(dat)のアクセスは許されていない。  
EX\_VID : 仮身ID (vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(切断状態、処理中状態でない、付箋)。

## ocre\_chd

子実身の生成

## 【形式】

W ocre\_chd(W vid, TC \*name, W \*nvid)

## 【パラメータ】

W vid 仮身ID  
TC \*name 実身のデフォルト名称  
W \*nvid 生成した実身の仮身ID

## 【リターン値】

0 正常 (生成した実身のファイルディスクリプタ)  
<0 エラー (エラーコード)

## 【解説】

vid で指定した仮身に対する子実身を生成し、そのファイルディスクリプタを関数値として戻す。

name は生成する実身のデフォルト名称であり、「保存先指定パネル」が表示され、最終的に指定された名称が戻る。

nvid には新規に生成された実身を参照する仮身 ID が格納される。生成された仮身は、vid が開いているウィンドウに対して登録されるが、その位置は保証されないため、必ず正しい位置に移動する必要がある。

新規に生成される実身は、vid で指定した実身と同一タイプであり、実行機能付箋のみが複写される。

仮身要求イベントは発生しない。

## 【エラーコード】

EX\_ADR : アドレス(name, nvid)のアクセスは許されていない。  
EX\_VID : 仮身ID(vid)は存在していない。  
EX\_VOBJ : 仮身の状態 / 属性が不適當(切断状態、付箋)。  
EX\_PAR : 取り消された。

# otad\_vob

仮身の表示用TADデータ作成処理

## 【形式】

W otad\_vob(W vid, RECT \*r, UW disp, LINK \*lnk)

## 【パラメータ】

W vid 0 : 仮身 ID  
= 0x8000 : 仮身リスト指定  
< 0 : - (ウィンドウ ID)

RECT \*r 表示領域  
(仮身リスト指定のときは、仮身リストへのポインタ)

UW disp 仮身の表示 / 消去方法 (odsp\_vob と同じ)  
::= (V\_ERASE V\_DISP V\_DISPALL V\_DISPAREA)  
| [V\_NOFRAME]

LINK \*lnk 作成した TAD データファイルへのリンク

## 【リターン値】

0 正常 (処理した仮身の数)  
< 0 エラー (エラーコード)

## 【解説】

仮身を表示する (odsp\_vob) 代わりに、表示した状態を表現する TAD データを作成する。

vid で指定した仮身の表示 / 消去を行なった表示のための TAD データを一時ファイルに作成し、そのリンクを lnk に格納する。r は表示すべき領域の相対座標での指定であり、仮身の領域と r の共通部分のみのデータが作成される。r = NULL の場合は、仮身の領域全体のデータが作成される。

仮身のデータは、その時点の仮身の状態 (虚身状態、処理中状態等) に応じた表示を表現するように作成される。

開いた仮身の表示エリアのデータは、仮身のデフォルトアプリケーションに対して、TAD データ作成起動 (M\_TADREQ) を実行して作成する。ただし、TAD データ作成起動がエラーとなった場合は、開いた仮身の表示起動 (M\_DISPREQ) を実行して作成したイメージを TAD データとして作成する。

vid < 0 の場合は、- vid で指定した値をウィンドウ ID とみなして、そのウィンドウ / パネル上に登録されているすべての仮身のデータを作成する。

vid = 0x8000 のときは、仮身リスト指定となり、r を以下の仮身リスト構造へのポインタとみなして、複数の仮身のデータを高速に作成する。途中の仮身 ID が不正のときは、それ以降の仮身の表示は保証されない。

W vid -- 仮身 ID



```
RECT    r    -- 表示領域
:      :
W      0    -- 終了を示す
```

vid < 0 の場合は、関数値として処理した仮身の数が戻る。指定したウィンドウ / パネルに仮身が 1 つも登録されていない場合は何もせず、関数値 "0" が戻る。

vid 0 または、vid = 0x8000 の場合は、関数値は常に "0" となる。

付箋の場合は、V\_DISP ~ V\_DISPAREA は、すべて同一の意味となる。

ファイル管理のエラーが発生した場合には、そのエラーコードがそのままエラーリターンする。

### 【エラーコード】

```
EX_ADR    : アドレス(r)のアクセスは許されていない。
EX_PAR    : パラメータが不正である(displ が不正)。
EX_VID    : 仮身 ID (vid)は存在していない。
EX_WID    : ウィンドウ(wid)は存在していない(vid < 0 のときのみ)。
```

## 3.8.6 アプリケーション支援関数

ここでは、実身 / 仮身の処理には直接的には関連しないが、アプリケーション支援のために、実身 / 仮身マネージャとしてサポートしている各関数の詳細を説明する。これらの関数群は外殻の拡張システムコールとして提供される。

関数値は、何らかのエラーがあった場合は「負」のエラーコードが戻る。正常終了時には「0」または「正」の値が戻る。

### aset\_sel

選択枠のチラツキ間隔設定

### 【形式】

```
W    aset_sel(W period)
```

### 【パラメータ】

```
W    period    チラツキ間隔(ミリ秒単位)
```

### 【リターン値】

0 正常 (以前のチラツキ間隔)

### 【解説】

選択枠のチラツキ間隔を period で指定した間隔に設定し、以前のチラツキ間隔を関数値として返す。

```
period    0
```

の場合は、設定を行わず、単に現在のチラツキ間隔を関数値として戻す。

period の値はミリ秒単位であるが、実際の分解能はインプリメントに依存する。

### 【エラーコード】

なし

adsp\_sel

選択枠の表示

### 【形式】

ERR adsp\_sel(GID gid, SEL\_RGN \*selp, W mode)

### 【パラメータ】

GID gid 描画環境 ID  
SEL\_RGN \*selp 選択枠領域  
W mode 表示モード  
= 0 : 消去  
> 0 : 表示  
< 0 : チラツキ表示

### 【リターン値】

= 0 正常 < 0 エラー (エラーコード)

### 【解説】

selp で指定した選択枠を、gid で指定した描画環境上に表示 / 消去 / チラツキ表示する。

selp->sts は、選択枠の状態を保持する以下の内容のデータである。

xxxx xxxx xxxx xxxx IPBD WWWW TTTT TTTT

I = 0 可能状態

1 不可状態 (表示されない)

P = 0 矩形選択枠

1 多角形選択枠 (角丸めは無視される)

B = 0 チラツキ状態でない

1 チラツキ状態

D = 0 消去状態

1 表示状態

W 枠幅 (0 はデフォルト)

T チラツキ間隔カウンタ

x 予約

B, D, T の値は、adsp\_sel() の中で、状態を保持するために使用しているため、変更してはなら

ない。

枠幅を表示状態に変更してはいけない(表示は保証されない)。

チラツキ間隔カウンタの初期値は問わない(通常は0とする)。

### 【エラーコード】

EX\_ADR : アドレス(sel<sub>p</sub>)のアクセスは許されていない。

EG\_xxx : 描画(ディスプレイプリミティブ)のエラー。

## adsp\_sl<sub>t</sub>

選択枠リストの表示

### 【形式】

ERR adsp\_sl<sub>t</sub>(GID gid, SEL\_LIST \*sel<sub>p</sub>, W mode, W dh, W dv)

### 【パラメータ】

|          |                   |   |
|----------|-------------------|---|
| GID      | gid               | 描画環境 ID                                       |
| SEL_LIST | *sel <sub>p</sub> | 選択枠領域リスト                                      |
| W        | mode              | 表示モード<br>= 0 : 消去<br>> 0 : 表示<br>< 0 : チラツキ表示 |
| W        | dh                | 水平移動量   |
| W        | dv                | 垂直移動量   |

### 【リターン値】

= 0 正常  
< 0 エラー (エラーコード)

### 【解説】

sel<sub>p</sub> で指定した選択枠リストに含まれる選択枠群を、gid で指定した描画環境上に表示 / 消去 / チラツキ表示する。

dh, dv は、それぞれ選択枠の水平、垂直座標の移動量を示し、選択枠群は、dh, dv だけ座標を移動して表示される。

### 【エラーコード】

EX\_ADR : アドレス(sel<sub>p</sub>)のアクセスは許されていない。

EG\_xxx : 描画(ディスプレイプリミティブ)のエラー。

# apnl\_men

パネルメニュー処理

## 【形式】

W apnl\_men(W pnid, W pid, EVENT \*epv)

## 【パラメータ】

|       |      |                |
|-------|------|----------------|
| W     | pnid | パネル ID         |
| W     | pid  | テキストボックスパーツ ID |
| EVENT | *epv | メニュー起動イベント     |

## 【リターン値】

=0 正常  
<0 エラー (エラーコード)

## 【解説】

pnid のパネル ID で指定したパネル上の、pid のパーツ ID で指定したテキストボックスに対してトレーからの編集メニューの処理を行なう。

\*evt はパネルの動作実行で得られたメニューイベントでなくてはならない。

\*evt が EV\_KEYDOWN のときは、キーメニューとしての処理を行ない、そうでないときは、以下の内容のメニューを画面に表示して、選択された項目に応じた処理を行なう。

|    |         |   |
|----|---------|---|
| 編集 | トレーへ複写  | C |
|    | トレーから複写 | Z |
|    | トレーへ移動  | V |
|    | トレーから移動 | X |
|    | 削除      |   |

メニュー処理の結果として、トレーや対象のテキストボックスの内容が更新される。

## 【エラーコード】

EX\_ADR : アドレス(epv)のアクセスは許されていない。

# achg\_bgc

ウィンドウ背景色変更処理

## 【形式】

W achg\_bgc(W \*mask, COLOR \*color, COLOR bgc)

### 【パラメータ】

W \*mask マスク(0, FILL0 (1) ~ FILL100 (7)) 0 は仮身背景色と同じ  
COLOR \*color 背景色  
COLOR bgc 仮身背景色

### 【リターン値】

= 0 正常 (取り消し、または変更なし)  
= 1 正常 (変更された)  
< 0 エラー (エラーコード)

### 【解説】

mask, color, bgc で指定された現在のウィンドウ背景色に対して、「ウィンドウ背景色変更パネル」を表示して、ウィンドウ背景色の変更を行い、変更した結果を、mask, color に格納する。

bgc < 0 のときは、「仮身背景色と同じ」という選択はないようになる。

### 【エラーコード】

EX\_ADR : アドレス(mask, color)のアクセスは許されていない。

---

[この章の目次にもどる](#)

[前頁:3.7 テキスト入力プリミティブにもどる](#)

[次頁:3.9 フォントマネージャにすすむ](#)

[この章の目次にもどる](#)

[前頁:3.8 実身 / 仮身マネージャにもどる](#)

[次頁:3.10 TCP/IPマネージャにすすむ](#)

## 3.9 フォントマネージャ

### 3.9.1 フォントマネージャの機能

#### 3.9.1.1 概要

フォントマネージャは、外殻の一部として位置付けられ、文字描画に使用されるフォントデータの登録 / 削除 / 情報の問い合わせ / データの読み出し等の機能を持ち、マルチプロセスの環境下での柔軟なマルチフォントの機能を提供している。

フォントマネージャは、スクリーン表示およびイメージ転送方式によるプリントに使用されるフォント、即ちシステムに内蔵されるフォントのみを対象とし、プリンタ等の外部機器に組み込まれているフォントに関しては対象としていない。

通常のアプリケーションは、実際の文字描画を行う場合は、ディスプレイプリミティブとして提供されている文字描画関数を使用すればよく、フォントマネージャの関数を使用する必要はない。フォントの一覧取り出しを行う場合のみフォントマネージャの関数を使用する。フォントの登録、削除等はフォントマネージャの関数を使用するが、通常のアプリケーションは行う必要はなく、主としてシステムアプリケーションが行うことになる。

ディスプレイプリミティブは、フォントマネージャを使用してフォントに関する情報および実際のフォントのイメージを取り出した後、必要に応じたイメージの後処理 ( 変形 / 拡大 / 縮小 / 回転等 ) を行ない、文字の描画を行なうことになる。

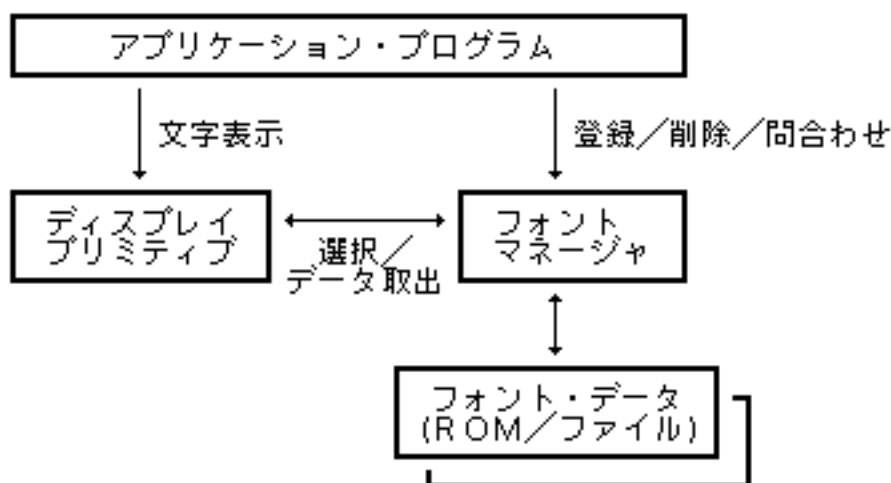


図 125 : フォントマネージャとディスプレイプリミティブ

フォントデータは、ROM またはファイルの形式で供給されることになり、そのデータ形式としていくつかの標準形式が規定されるが、フォントデータの形式は基本的にフォントマネージャ以外には見えない。

なお、フォントを作成するためのツールとしてのフォントエディタは1つの独立したアプリケーションとして位置付けられ、フォントマネージャの機能には含まれない。

### 3.9.1.2 フォント

#### フォント

フォント(書体)とは、目に見える形の文字表現(文字体)の集合である。

フォントは、その文字表現の基本的なデザインの違いによりフォントファミリーに体系付けられる。

#### フォントファミリー

フォントファミリーは、12文字(12文字に満たない場合は後ろに0が詰められる)のフォントファミリー名により区別される。また、フォントファミリーには、文字体の特徴を表わすフォントクラスが対応づけられる。

一つのフォントファミリーは、一つもしくは複数のフォントデータ群から構成される。異なる文字セットのフォントデータも一つのフォントファミリーに属することができる。

最初に登録したフォントファミリーがシステムのデフォルトフォントファミリーとなる。デフォルトフォントファミリーは文字セットごとに決定される。デフォルトフォントファミリーには、ある文字セットの標準規格において規定されている全ての文字が、一般的なデザインにより定義されていると期待してよい。

#### フォントクラス

フォントクラスは、文字体の特長を表わす以下に示すデータである。

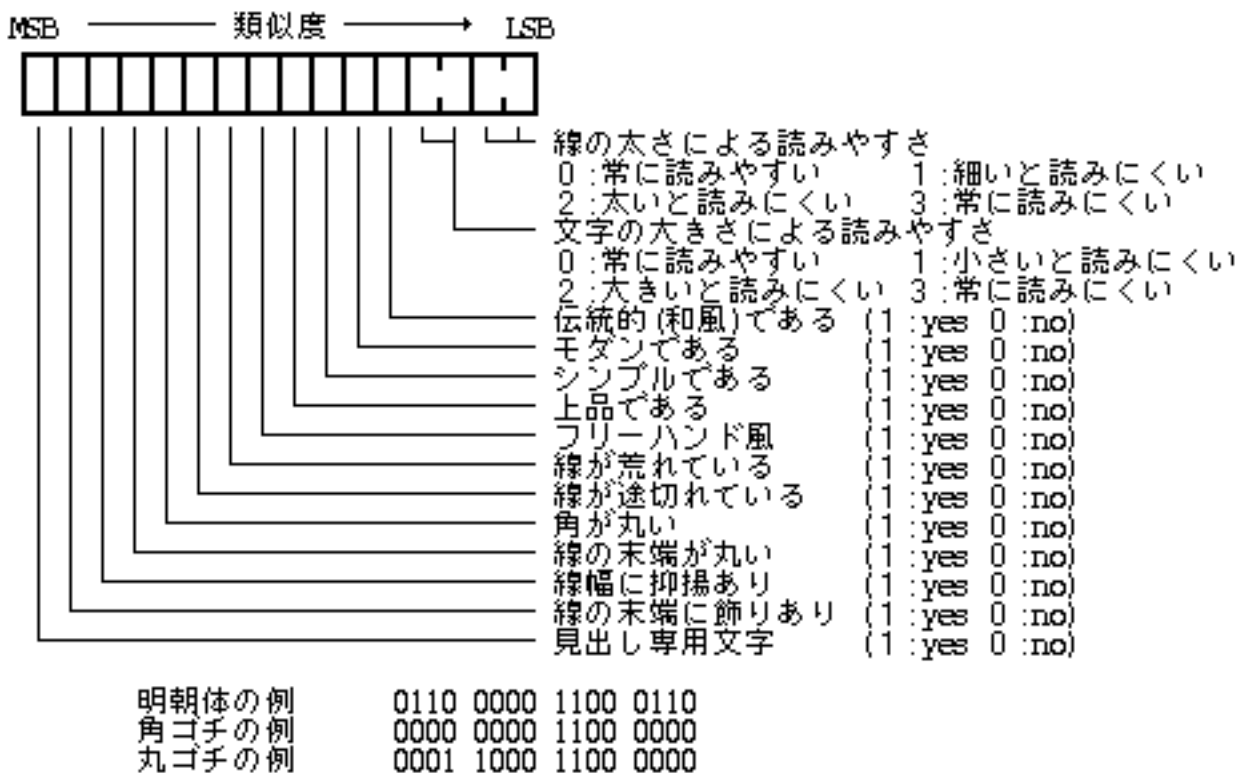


図 126: フォントクラス

フォントクラスの各ビットに対応した基準は、上位のビットが最も類似度が高く、下位にビットが最も類似度が低いため、フォントファミリーの類似度は、フォントクラスを上位

のビットから比較して、最初に異なったビットの位置により判断することができる。

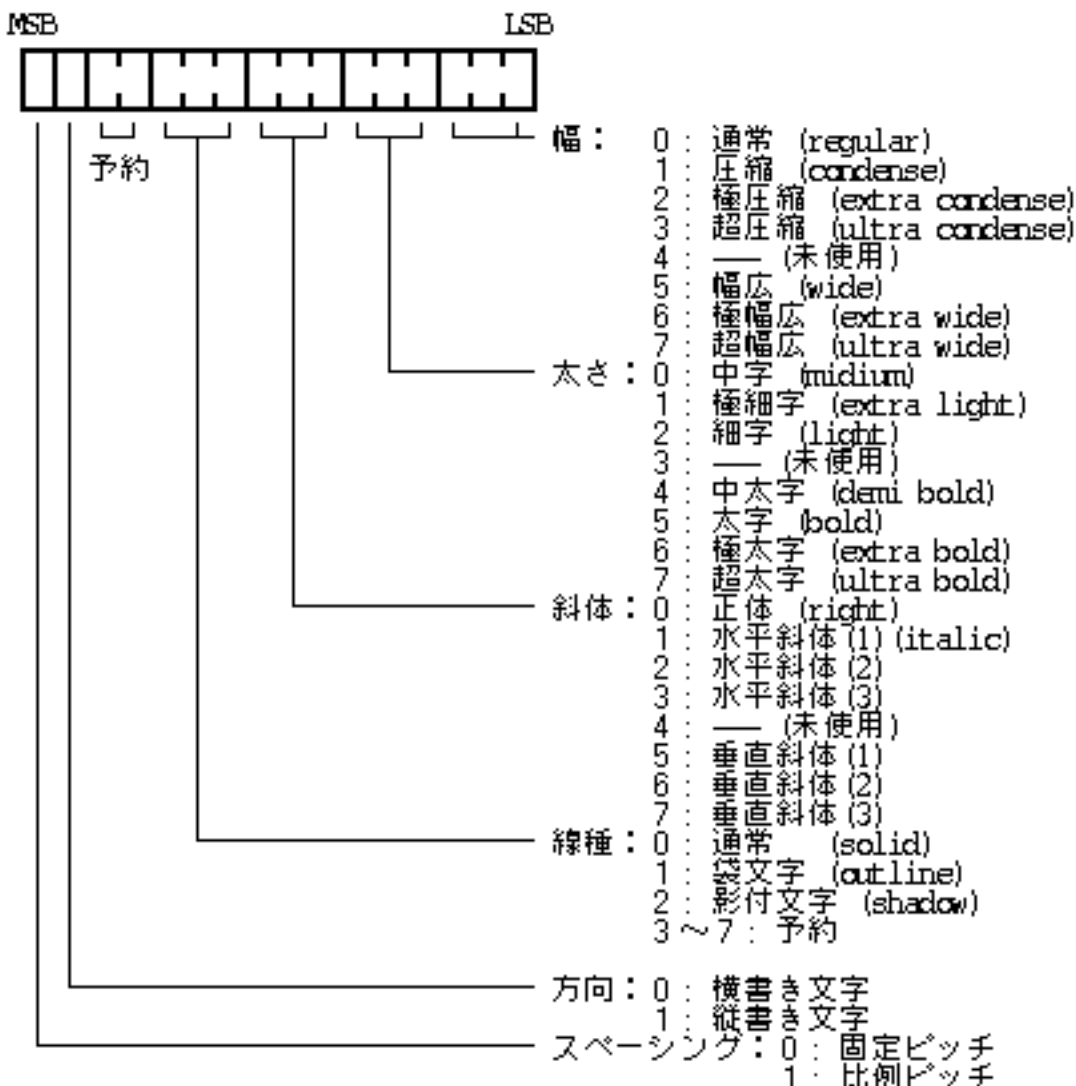
フォントクラスは、フォントファミリーをその特徴により指定する場合、または、特定のフォントファミリーが存在しない時に最も良く似ている代替フォントファミリーを選択するために使用される。即ち、文字セットIDが同一でかつ、フォントクラスの上位ビット側から最も多くのビットが一致しているフォントクラスに対応するフォントファミリーが代替フォントファミリーとなる。

同一のフォントファミリー名を持つフォントデータは、同一のフォントクラスでなくともいけないが、異なるフォントファミリー名を持つフォントデータが、同一のフォントクラスであってもよい。この場合、その2つのフォントファミリーの文字体はかなり似ているが微妙に違うということになる。

FTC\_DEFAULT ( = 0x80000000 )は、常にデフォルトフォントファミリーに対応する特別なフォントクラスである。

### フォント属性

同一フォントファミリーの中で、主として加工による文字体の相違はフォント属性により区別される。フォントのバリエーションは基本的には無限とも言えるが、大部分のバリエーションは以下に定義されるフォント属性により区別可能となる。



※ 水平斜体/垂直斜体(1)~(3)では、(1)より(3)の方が傾きが大きいことを意味する。また、超 (ultra) は 極 (extra) よりも程度が大きいことを意味する。



図 127 : フォント属性

フォントデータにはフォントファミリーを示すフォント名が付けられているが、さらにフォント属性等の意味を付加した 20 文字の詳細フォント名を定義される。この詳細フォント名はフォント情報の表示にのみ使用される。

### 3.9.1.3 フォントデータ

#### フォントデータ

フォントデータはフォントを定義している実際のデータの集まりの単位であり、対象とする文字セットで定義されている文字コード範囲内の 1 つの連続範囲に対して定義され、その範囲を定義域と呼ぶ。定義域は第 1 バイトと第 2 バイトの値を座標値とする (0, 0) ~ (255, 255) の矩形領域中の 1 つの矩形領域となる。

フォントデータは文字の大きさ毎に定義される。文字の大きさは、ポイント数、級数等の実スケールではなく、最終的に得られる文字の高さのビットマップイメージ上でのドット数で表わされ、これをフォントサイズと呼ぶ。なお、実スケールとドット数との変換は上位レベルで行なわれるものとする。

従って、1 つのフォントファミリーは以下に示す複数のフォントデータ群から構成されることになる。

- 文字セットの異なるフォントデータ
- フォント属性の異なるフォントデータ
- フォントサイズの異なるフォントデータ
- 定義域の異なるフォントデータ

フォントファミリーの例として以下のものがあげられる。

#### ファミリー : XX明朝体

- 24ドット第一水準フォントデータ
- 24ドット第二水準フォントデータ
- 24ドットボールドかなフォントデータ
- 32ドット第一水準フォントデータ
- 32ドット第二水準フォントデータ
- 第一水準フォントデータ(スケーラブル)
- 第二水準フォントデータ(スケーラブル)

フォントマネージャは、フォントファミリーを 1 つの単位として取り扱い、要求された文字セット、フォント属性、フォントサイズ、および文字コードに適用できるフォントデータをファミリー内から選択して使用する。

フォント属性、フォントサイズが適合するフォントが存在しない場合は、所望のフォントを各種の変形処理、拡大 / 縮小 / 回転処理により機械的に発生させることが最も容易なフォントデータを選択することになる。

フォントファミリーで定義されていない文字コードに対して、他のフォントファミリーを適用するように指定することが可能である。この場合、そのフォントファミリーを部分フォントファミリーと呼び、未定義文字コードを適用するフォントをベースフォントファミリーと呼ぶ。なお、ベースフォントファミリーの適用は 1 レベルのみに限られ、多重のベースフォントファミリーの適用は行なわれない。

部分フォントファミリーは、ある特定の文字コード領域のフォントのみを変更するために使用する。例えば、ひらがな、カタカナだけを定義する「丸文字」のフォントファミリーに対して「丸ゴシック」のベースフォントが定義されている場合、「丸文字」フォントファミリーを指定したときには、ひらがな、カタカナの文字コードに対しては「丸文字」が適用され、それら以外の文字コードに対しては「丸ゴシック」が適用されることになる。

## フォントデータの構造

フォントデータは、メモリ上のデータまたはファイルとして存在し、基本的に以下の4つのデータブロックにより構成される。

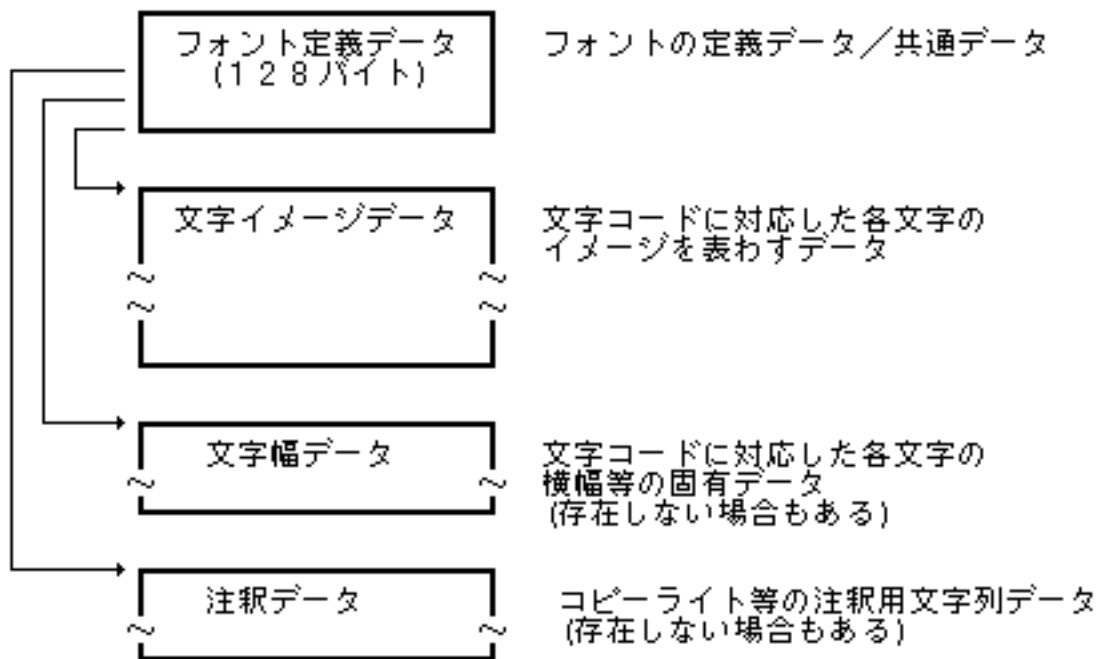


図 128 : フォントデータの全体構造

フォント定義データは、フォントデータに関する各種の定義データ、共通データを含むもので以下に示す内容を持つ。

```
typedef struct {
    SCRIPT  script;          /* 文字セット(スクリプト)指定 */
    FCLASS  fclass;         /* フォントクラス */
    FATTR   attr;           /* フォント属性 */
    UB      size;           /* フォントサイズ (文字枠の最大高さ) */
    UB      width;          /* 文字枠の最大幅 */
    UB      base;           /* ベース位置 */
    UB      leading;        /* レディング */
    TC      name[12];        /* フォントファミリー名 */
    FCLASS  baseclass;      /* ベースフォントクラス */
    TC      basename[12];    /* ベースフォントファミリー名 */
    TC      fullname[20];    /* 詳細フォント名 */
    TC      topcode;         /* 最初の文字コード */
    TC      lastcode;        /* 最後の文字コード */
    UB      sheight;         /* 基準文字高さ */
};
```

```

UB      swidth;          /* 基準文字幅 */
H      rsv[2];          /* 予約 */
UB      imgform;        /* 文字イメージの形式 */
UB      widform;        /* 文字幅等データの形式 */
W      datasize;       /* データ全体のバイトサイズ */
OFFSET offimage;       /* 文字イメージへのオフセット */
OFFSET offwidth;       /* 文字幅等データへのオフセット */
OFFSET offnote;        /* 注釈データへのオフセット */
} FDEF;

```

script :

文字セットを示すIDである。TRON 多国語コードの基本言語面の場合は 0x0021 である。

fclass :

フォントクラスを示す。

1つのフォントファミリー内では、fclass はすべて一致してはいけない。

name :

フォント (ファミリー) 名を示す。12文字に満たない場合は後ろに0が詰められる。

attr :

フォント属性を示す。

size :

フォントサイズ (文字枠の最大高さ) のドット数 (0 ~ 255) を示す。ベクトル形式フォント等の任意のフォントサイズが適用可能な場合は0となる。

width :

文字枠の最大幅のドット数 (0 ~ 255) を示す。任意フォントサイズの場合は0となる。

base :

文字描画のベースライン (基準位置) からの文字枠の高さのドット数 (0 ~ 255) を示す。任意フォントサイズの場合は0となる。

leading :

シングルスペースの行間のドット数 (0 ~ 255) を示す。任意フォントサイズの場合は0となる。

baseclass :

ベースフォントのフォントクラスを示す。部分フォントでない場合は意味を持たない。

basename :

ベースフォント (ファミリー) 名を示す。12文字に満たない場合は後ろに0が詰められる。部分フォントでない場合は、すべて0が詰められる。先頭の文字が0か否かにより、部分フォントか否かの区別が行なわれる。

1つのフォントファミリー内では、baseclass, basename はすべて一致してはいけない。

いけない。

fullname :

詳細フォント名を示す。20文字に満たない場合は後ろに0が詰められる。  
これは、参照用としてのみ使用される。

topcode :

lastcode :

フォントデータで定義している文字コードの最小値、および最大値を示す。topcodeの(第1バイト, 第2バイト)で示される点と、lastcodeの(第1バイト, 第2バイト)で示される点で囲まれる矩形領域が定義域となる。

swidth :

sheight :

サイズの基準となる文字枠の幅 / 高さのドット数 (1 ~ 255) を示す。文字サイズ指定時に参照される。

ベクトル形式フォント等の任意のフォントサイズが適用可能な場合は、フォントサイズに関らず sheight : swidth が文字枠の 縦 : 横 の比とみなされる。  
どちらかの値が0の場合は、size, width の値と等しいとみなされる。

imgform :

文字イメージデータの形式を示す。

0 ~ 63 : 標準形式

0 : ドット形式 (固定イメージサイズ)

1 : ドット形式 (可変イメージサイズ)

2 : ベクトル形式(アウトライン)

3 : TrueType形式(アウトライン)

4 ~ : 予約

64 ~ : 特殊形式

widform :

文字幅等データの形式を示す。

0 ~ 63 : 標準形式

0 : 文字幅データ

1 : 文字幅 / インデックスデータ

2 : イメージ幅付き文字幅データ

3 ~ : 予約

64 ~ : 特殊形式

datasize :

フォント定義データを除いた、フォントデータの全体のバイトサイズ、即ち、文字イメージデータ、文字幅データ、注釈データのバイトサイズの合計を示す。

offimage :

offwidth :

offnote :

それぞれフォント拡張定義データ、文字イメージデータ、文字幅データ、注釈データの先頭アドレスへのフォント定義データの先頭からのバイトオフセットを示す。0の場合はそのデータが存在しない事を示す。文字イメージデータは必ず存在する。offimage が 0 ~ 127 の場合は、特別に文字イメージデータのレコードサブタイプを表わす。0の場合は、レコードサブタイプに関係なく最初の文字イメージデータレコードが対象となる。1 ~ 127 の場合は、該当するレコードサブタイプを持つ文字イメージデータレコードが対象となる。

文字イメージデータ、および文字幅等データの形式は、imgform および widform により規定され、いくつかの標準データ形式が規定される。

注釈データは、最後が TNULL(0) で終了している1つの文字列である。

### 3.9.1.4 フォントのアクセス

#### フォントデータの登録

フォントを使用するためには、そのフォントデータをあらかじめ登録しておく必要がある。登録したフォントデータはフォントID( 0 )により識別される。フォントIDは登録した順に0からの連続番号が付けられる。

文字セットごとに、最初に登録したフォントデータの属するフォントファミリーが、デフォルトフォントファミリーとして登録される。

すでに登録されているフォントとフォントファミリー、フォント属性が同じで、定義域が重なるフォントを登録した場合、後から登録したフォントデータの文字が優先される。既に登録されているフォントデータと同一のフォントデータは登録できない。

システムのスタートアップ直後は、フォントデータが1つも登録されていない状態であり、システムの初期化処理、およびユーザの切替処理として必要なフォントデータの登録が行なわれる。登録はシステムで標準的に使用するフォントデータを先に行ない、特にデフォルトフォントファミリーとして使用するフォントデータは一番最初に登録しなくてはならない。なお、フォントの登録は初期化以外の任意の時点でも可能である。

フォントデータには、すべてのユーザに共通な共通フォントと、ユーザ毎に定義される個別フォントの2種類がある。共通フォントは、すべてのユーザの初期化処理で登録され、個別フォントはユーザに応じて登録される点が異なるだけであり、一度登録されたフォントは、実行時のプロセスのユーザ情報に無関係にすべてのプロセスから使用可能となる。

#### フォントデータの所在

フォントデータは、その所在により以下に示す2つのタイプに分類される。

固定フォント：

ROM等に固定的に存在しているフォント。

動的フォント：

ファイルとして存在しているフォント。

固定フォントの登録は、そのメモリアドレスを、動的フォントの登録は、そのファイルを、それぞれフォントマネージャに通知することにより行なわれる。即ち、以下に示すFLOC構造体により、フォントの所在を指定して登録する。

```
typedef union {
    LINK    *lnk;    -- フォントファイル指定(動的フォント)
    FDEF    *addr;   -- メモリ上のフォントデータ指定(固定フォント)
} FLOC;
```

固定フォントは、システム空間上のメモリ領域に存在しなければいけない。

動的フォントは、必要に応じてメモリ領域に読み込まれる。フォントデータがフォントマネージャに登録されている状態で、ファイルを操作してはいけない。

## フォントセット

フォントの文字イメージへのアクセスは、フォントセットを介して行う。フォントセットは、ある特定のフォント属性 / 大きさに変形され、ある条件に基づいて選択されたフォントファミリーの集合である。フォントファミリーの実体は、あるフォントファミリーに属するフォントデータの集合であるので、フォントセットの実体は複数のフォントデータの集合である。

フォントセットは動的に生成され、フォントディスクリプタという正の整数値が割り当てられる。実際のフォントへのアクセスはフォントディスクリプタを使用する。

フォントセットはそれを生成したプロセスに属しておらず、生成したプロセスの終了時にも解放されずに残る。アプリケーションがフォントセットを生成した場合、責任をもって削除する必要がある。

フォントセットを生成した時点では、以下のデフォルト値が設定される。

```
FSSPEC :
    name    : 空文字列
    fclass  : FTC_DEFAULT ( = 0x80000000 )
    attr    : 0
    size    : (16, 16)
    angle   : 0
```

## フォント情報の設定

フォントの文字イメージにアクセスするには、最初に目的とするフォントの情報をフォントセットに設定する。設定には、以下に示す FSSPEC 構造体を使用する。

```
typedef struct FontSetSpecifier {
    TC    name[12]; /* フォントファミリー名 */
    UW    fclass;   /* フォントクラス */
    UW    attr;     /* フォント属性 */
    SIZE  size;     /* 文字サイズ */
} FSSPEC;
```

name :

目的とするフォントのフォントファミリー名を指定する。フォントクラスのみによる指定を行う場合は、空文字列とする。

fclass :

name で指定したフォントファミリーが存在しない場合に、代替フォントファミリーを検索するためのフォントクラスを指定する。

ただし、fclass が FTC\_DEFAULT ( = 0x80000000 ) の場合は、デフォルトフォントファミリーが代替フォントファミリーとして選択される。

attr :

目的とするフォントのフォント属性を指定する。

size :

目的とするフォントのサイズを指定する。

size.c.v -- 目的とするフォントの基準文字高さのドット数

size.c.h -- 目的とするフォントの基準文字幅のドット数

name で指定したフォントファミリーが、実際に存在するかどうかは環境に依存する。

fclass は代替フォントファミリーの検索にのみ使用される。

目的のフォントとしてデフォルトフォントファミリーを指定したい場合は、name として空文字列を、fclass として FTC\_DEFAULT ( 0x80000000 ) を指定すればよい。

目的のフォントファミリーが存在しない場合に、任意のフォントファミリーではなくデフォルトフォントファミリーを代替フォントファミリーとして指定したい場合は、name として目的のフォントファミリー名を、フォントクラスとして FTC\_DEFAULT ( 0x80000000 ) を指定すればよい。

実際の文字イメージへのアクセスは、フォントセット、文字セット、文字コードを指定して行う。実際に文字イメージが生成されるまでのロジックを以下に述べる。

1. システムに登録されたフォントファミリーの中から、フォントセットで指定されたフォントファミリーを選択する。
2. 指定されたフォントファミリーが環境に存在しない場合、もしくはフォントセットで指定されたフォントファミリー名が空文字列の場合は、フォントセットで指定されたフォントクラスに基づいて最も近似したフォントファミリーを検索し、代替フォントファミリーとして選択する。
3. 選択されたフォントファミリーの中から、指定された文字コードが定義されたフォントデータを選択する。
4. フォントデータが複数登録されている場合、フォントセットで指定されたフォント属性 / フォントサイズに適合するフォントデータを検索して選択する。
5. 適合するものがない場合は、各種の変形処理、拡大 / 縮小 / 回転処理により機械的に発生させることが最も容易なフォントデータを選択する。
6. 選択されたフォントデータから、文字イメージを生成する。
7. 選択されたフォントファミリーの中に、指定された文字コードが定義されたフォントデータが存在しない場合、ベースフォントファミリーが定義されていれば、第2のフォントファミリーとしてベースフォントファミリーを選択する。
8. ベースフォントファミリー名で指定されたフォントファミリーそのものが存在しない場合でも、ベースフォントクラスによる代替フォントファミリーの検索を行う。

9. ベースフォントファミリーとして選択されたフォントファミリーに対し、3～6の処理を行う。
10. ベースフォントファミリーが定義されていない場合、もしくはベースフォントファミリーにも指定された文字コードが定義されたフォントデータが存在しない場合、第3のフォントファミリーとしてデフォルトフォントファミリーを選択する。
11. デフォルトフォントファミリーとして選択されたフォントファミリーに対し、3～6の処理を行う。
12. デフォルトフォントファミリーにも指定された文字コードが定義されたフォントデータが存在しない場合、未定義文字イメージが生成される。未定義文字イメージの詳細はフォントマネージャの実装に依存する。

## 文字の回転角度の設定

フォントセットには、文字イメージの回転角度を指定することができる。

文字描画位置を中心として、反時計方向に任意の角度回転させた文字イメージを取得することができる。回転角度は、0～の度単位の非負の数値で指定し、360の剰余が有効となる。

## 文字イメージ形式

フォントマネージャから得られる文字イメージは、フォントデータの文字イメージデータ構造とは無関係に、常にビットマップデータ形式として、以下に示す形で得られる。

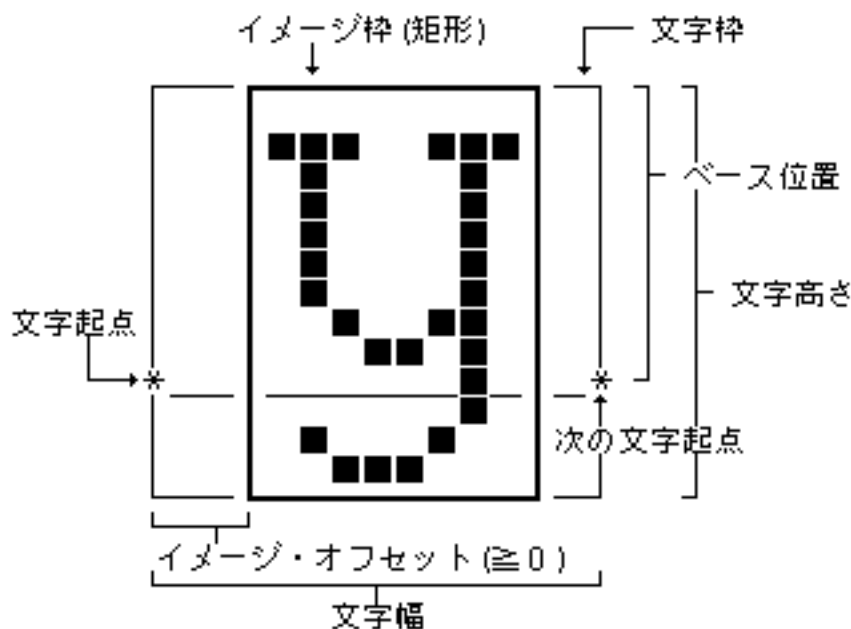


図 129: フォントの文字イメージ - 1

文字枠は、ある1つの文字を描画する場合に文字が占める仮想的な矩形領域であり、その幅が文字幅、その高さが文字高さとなる。フォントデータ内に含まれるすべての文字の文字枠の最大高さ (size)、最大幅 (width) がフォントデータの定義データに設定される。

また、文字枠は、フォントの属性により以下のようになる。横書きの場合は、文字描画の起点に文字幅を加えた位置が文字間ギャップ0の場合の次の文字描画の起点となり、縦書



きの場合は、文字描画の起点に文字高さを加えた位置が文字間ギャップ0の場合の次の文字描画の起点となる。

|             | 横書き                     |                 | 縦書き                     |                  |
|-------------|-------------------------|-----------------|-------------------------|------------------|
|             | 固定ピッチ                   | 比例ピッチ           | 固定ピッチ                   | 比例ピッチ            |
| 文字高さ<br>文字幅 | 固定 (size)<br>固定 (width) | 固定 (size)<br>可変 | 固定 (size)<br>固定 (width) | 可変<br>固定 (width) |

図 130 : 文字高さと文字幅

ベース位置は、横書きの場合に文字を揃える位置であり、文字枠の上端からのドット数で表わされる。これは、フォントデータに固定の値である。この値は縦書きでは使用しない。

イメージ枠は、実際に得られるビットマップ形式の文字イメージを囲む矩形枠である。文字枠と同じとは限らず、文字枠をはみだす場合もある。文字枠をはみだす場合直前 / 直後の文字に重なることになる。これは英文字の飾りヒゲ (kerning) や斜体などに使用される ( [図 131 : フォントの文字イメージ2](#) 参照)。

文字枠より小さい場合は、文字枠の残りの部分は背景部分となる。

イメージオフセットはイメージ枠が文字枠のどの部分に位置するかを示すデータであり、文字枠左上からの相対座標で示される。イメージオフセットは回転角度の影響を受ける。回転した文字イメージを取り出す場合、文字枠に対して回転操作を行った後の文字枠左上からの相対座標となる。

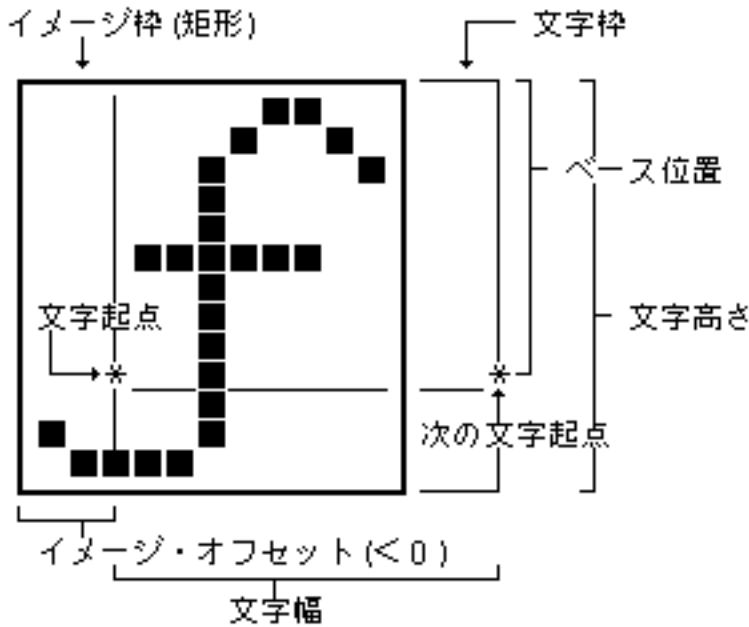


図 131 : フォントの文字イメージ2

フォントマネージャは、指定されたフォントセット、属性、サイズ、文字セット、文字コードに対する文字イメージの情報を提供するだけであり、ディスプレイプリミティブが、これらの情報をもとに実際の文字描画領域、文字描画位置を計算して文字描画を行なうことになる。

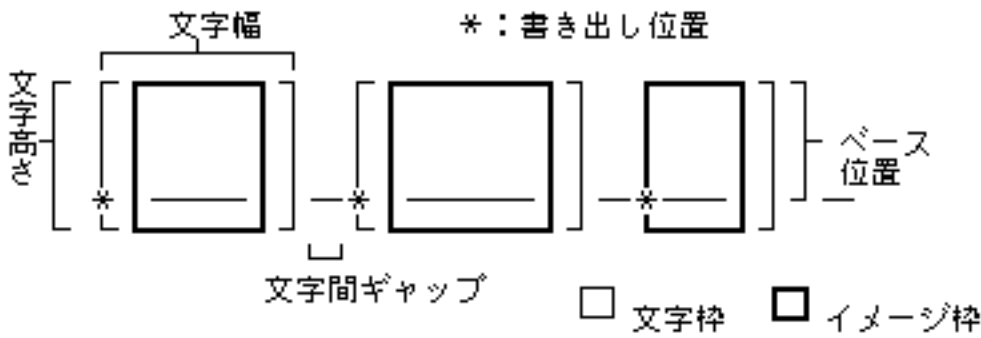


図 132 : 文字の描画位置(横書き)

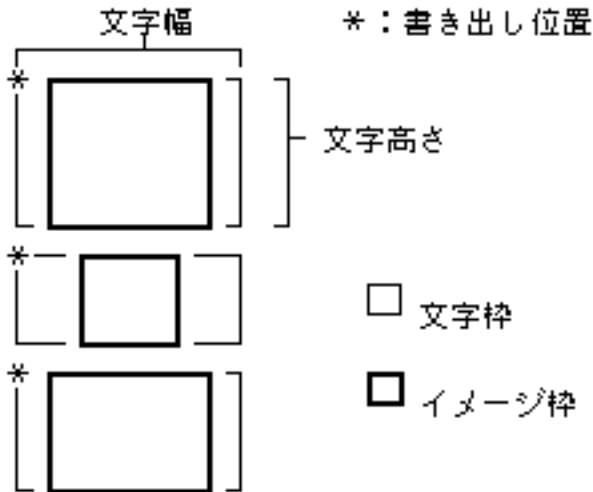


図 133 : 文字の描画位置(縦書き)

### 文字イメージの取り出し

指定したフォントセットに対応するフォントディスクリプタを使用して、文字セット、文字コードを指定することにより対応する文字イメージを取り出すことができる。

要求したフォント属性、フォントサイズ、回転角度と完全に一致するイメージが取り出せた場合は文字イメージに対する後処理は不用であるが、取り出せなかった場合は、取り出した文字イメージに対して拡大/縮小/変形/回転の後処理を行なう必要がある。

ビットマップ形式のフォントデータの場合、フォントマネージャはイメージに対する処理を行わない。アウトライン形式フォントの場合は、フォントマネージャは指定されたサイズの文字イメージとして展開するので拡大/縮小の後処理は不要である。フォントマネージャが変形/回転を行うかどうかは実装に依存する。

取り出す文字イメージは以下に示す 2 種のデータ構造により表現される。

```
typedef struct {
    RECT    frame;        /* イメージ枠 */
    H       width;       /* 文字幅 */
    H       height;      /* 文字高さ */
    PNT     imgofs;      /* イメージオフセット */
    UH     family;      /* フォントファミリーの種別 */
}
```

```

    UH    fid;        /* フォントID */
} FCDATA;

typedef struct {      /* イメージ取り出し用データ定義 */
    UW    attr;      /* フォント属性 */
    UH    height;    /* 文字枠の最大高さ */
    UH    width;     /* 文字枠の最大幅 */
    UH    base;      /* ベース位置 */
    UH    leading;   /* レディング */
    H     fkind;     /* 線種の変形係数 */
    H     fslope;    /* 傾斜の変形係数 */
    H     fweight;   /* 太さの変形係数 */
    H     fwidth;    /* 幅の変形係数 */
    H     rowbytes;  /* ビットマップの幅 */
    UH    resv;      /* 予約 */
    SIZE  asize;     /* フォントの文字サイズ */
    UH    aangle;    /* フォントの回転角度 */
    UH    pixbits;   /* ビットマップのピクセルビット数 */
    UB    *image;    /* ビットマップ開始アドレス */
    FCDATA ch[1];    /* 実際は指定した文字数個の配列 */
} FDATA;

```

rowbytes :

文字イメージが格納されているビットマップの幅のバイト数を示し、必ず偶数の値となる。

なお、文字幅 / 高さのみの取り出し時には、この値は保証されない。

pixbits :

文字イメージが格納されているビットマップのピクセルビット数を示す。通常は 0x0101 である。

グレースケールによる文字イメージの取り出しを行った場合のピクセルビット数は実装に依存するが、通常は 0x0808 (白黒 256 階調) とする。

なお、文字幅 / 高さのみの取り出し時には、この値は保証されない。

image :

文字イメージが格納されているビットマップの先頭アドレスを示す。このビットマップは、プレーン = 1 であり、左上の座標は常に ( 0, 0 ) となる。

ドット形式フォントの場合、イメージ枠の外側には他の文字の文字イメージが存在するので、無視しなければいけない。

なお、文字幅 / 高さのみの取り出し時には、この値は保証されない。

attr :

得られた文字イメージのフォント属性を示す。フォントセットで指定したフォント属性と一致するとは限らない。

height :

width :

base :

leading :

得られた文字イメージの属するフォントデータの最大文字高さ ( フォントサイズ )、最大文字幅、ベース位置、レディングである。これらの値は適用したフォントデータの生のデータであり、asize による拡大 / 縮小が必要となる。可変フォントサイズのフォントデータの場合、フォントデータ全体を取り出した文字イメージのフォントサイズに展開した場合の値となる。

asize :

得られた文字イメージの実際のサイズを示す。文字描画を行なう場合には、フォントセット設定時に指定した size と asize の比率に応じた拡大 / 縮小を行なう必要がある。即ち、以下の倍率で拡大 / 縮小を行なう必要がある。

size.c.v / asize.c.v -- 縦の倍率  
size.c.h / asize.c.h -- 横の倍率

aangle :

得られた文字イメージの実際の回転角度を示す。文字描画を行なう場合には、フォントセット設定時に指定した angle と aangle の差に応じた回転を行う必要がある。

fkind :

fslope :

fweight :

fwid :

それぞれ得られた文字イメージに対して後処理として必要な文字の変形処理を示す。これらの値は、フォントセット設定時に指定したフォント属性と実際に得られたフォント属性 ( attr ) との違いにより、フォントサイズに対応した適当な値が設定される。後処理は、asize による拡大 / 縮小処理を行なう以前に行なわれる。いずれも 0 の場合は、対応する後処理が不要なことを示す。

fkind :

>0 の場合は、要求する文字種が袋文字または影付きの場合で、得られた文字イメージが通常文字の場合で、下位 8 ビットが右下のアウトラインのドット数を示し、上位 8 ビット ( 実際は 7 ビット ) が左上のアウトラインのドット数を示す。袋文字の場合は、右下と左上のアウトラインのドット数は等しくなるが、影付きの場合は、右下のドット数が大きくなる。

< 0 の場合は、上記以外で要求する文字種と得られた文字イメージの文字種が異なる場合であり、後処理は不可能であることを示す。

fslope :

> 0 の場合は水平傾斜を示し、上辺を右方向へずらすドット数を示す。< 0 の場合は垂直傾斜を示し、左辺を下方向へずらすドット数を示す。

fweight :

> 0 の場合は ( 右方向へ ) 太くするドット数を示し、< 0 の場合は ( 右側を ) 細くするドット数を示す。ただし、細くする後処理は無理な場合が多く、処理方法はインプリメントに依存する。

fwid :

> 0 の場合は幅を広げるドット数を示し、< 0 の場合は、幅を狭めるドット数を示す。

ch :

指定した文字コードに対する情報であり、文字列を指定した場合は、文字数分の配列が得られることになる。この場合、文字列の途中で適用すべきフォントデータが切り換わった結果、asize, fweight 等の、inf 以外の共通情報が変化するような場合は、そこまでの文字の情報のみが得られる。

frame :

image, rowbytes, pixbits で示されるビットマップ上の文字イメージ矩形を示す。通常は、1文字単位に独立したイメージ矩形となるが、ベクトル形式のフォント等の場合は、文字列中の連続した複数文字のイメージを1つのイメージ矩形の中にまとめて取り出す場合もある。この場合は、先頭の文字に対するイメージ矩形が全体の矩形領域を示し、続く文字のイメージ矩形は lefttop = rightbot となり、lefttop の点はその文字のイメージ矩形の左上の位置を示すことになる。  
なお、文字幅 / 高さのみの取り出し時には、この値は保証されない。

width :

height :

それぞれ、文字の文字幅、文字高さを示す。横書きの場合の文字高さは、最大文字高さに固定である。縦書きの場合は、文字幅 / 文字高さ共に可変となる。文字イメージの回転の影響は受けない。

imgofs :

イメージオフセットを示す。文字枠の左上を原点とした場合の、イメージ枠 (frame) の相対位置を示す。  
なお、文字幅 / 高さのみの取り出し時には、この値は保証されない。

family :

得られた文字イメージのフォントファミリーの種類を示す。

- 0 : 対象フォントファミリー
- 1 : 対象フォントファミリー(代替)
- 2 : ベースフォントファミリー
- 3 : ベースフォントファミリー(代替)
- 4 : デフォルトフォントファミリー
- 1 : 未定義文字

fid: 得られた文字イメージのフォントIDを示す。

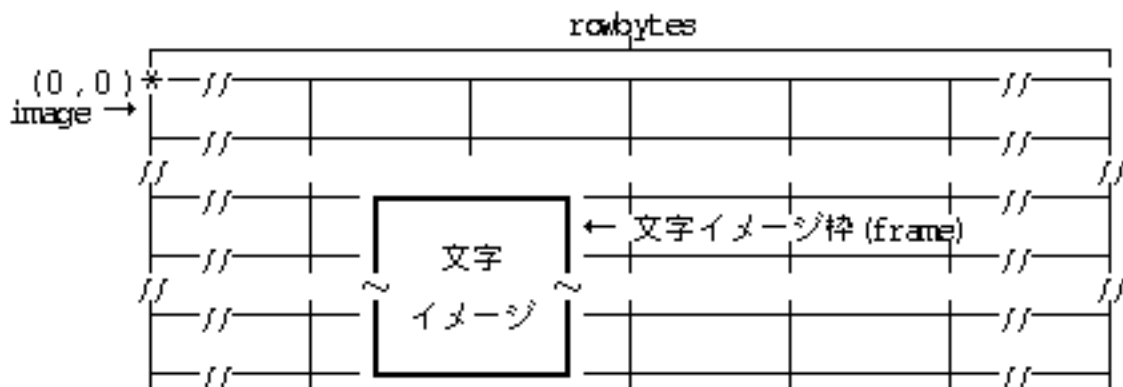


図 134 : イメージビットマップ上の文字イメージ



先頭文字の `inf[n].frame` は全体を囲む矩形を示し、  
 2番目以降の文字の `inf[n+i].frame` は \* で指定した点を示す。  
 (`inf[n+i].frame` の `lefttop = rightbot` となる)

図 135 : 連続文字イメージ

## 3.9.2 データ / 定数の定義

### フォント指定

```
typedef struct ExtendFontSpecifier {
    TC    name[12]; /* フォントファミリー名 */
    UW    fclass; /* フォントクラス */
    UW    attr; /* フォント属性 */
    SIZE  size; /* 文字サイズ */
} FSSPEC;

/* フォント属性：幅 */
#define FT_REGULAR 000000 /* 通常: regular */
#define FT_COND 000001 /* 圧縮: condense */
#define FT_XCOND 000002 /* 極圧縮: extra condense */
#define FT_UCOND 000003 /* 超圧縮: ultra condense */
#define FT_WIDE 000005 /* 幅広: wide */
#define FT_XWIDE 000006 /* 極幅広: extra wide */
#define FT_UWIDE 000007 /* 超幅広: ultra wide */

/* フォント属性：太さ */
#define FT_MIDI 000000 /* 中字: midium */
#define FT_XLIGHT 000010 /* 極細字: extra light */
#define FT_LIGHT 000020 /* 細字: light */
#define FT_DBOLD 000040 /* 中太字: demi bold */
#define FT_BOLD 000050 /* 太字: bold */
#define FT_XBOLD 000060 /* 極太字: extra bold */
#define FT_UBOLD 000070 /* 超太字: ultra bold */

/* フォント属性：斜体 */
#define FT_RIGHT 000000 /* 正体: right */
#define FT_HSLOPE1 000100 /* 水平斜体 : italic */
#define FT_HSLOPE2 000200 /* 水平斜体 */
#define FT_HSLOPE3 000300 /* 水平斜体 */
#define FT_VSLOPE1 000500 /* 垂直斜体 */
```

```

#define FT_VSLOPE2 000600 /* 垂直斜体 */
#define FT_VSLOPE3 000700 /* 垂直斜体 */

/* フォント属性：線種 */
#define FT_SOLID 000000 /* 通常: solid */
#define FT_OUTLINE 001000 /* 袋文字: out line */
#define FT_SHADOW 002000 /* 影付文字: shadow */
#define FT_WSHADOW 003000 /* 白影付文字: white shadow */

/* フォント属性：方向 */
/* FontDirectionAttribute */
#define FT_HDRAW 0x0000 /* 横書き文字 */
#define FT_VDRAW 0x4000 /* 縦書き文字 */

/* フォント属性：ピッチ */
#define FT_FIXED 0x0000 /* 固定ピッチ */
#define FT_PROP 0x8000 /* 比例ピッチ */

/* フォント属性：階調 */
#define FT_BW 0x0000 /* 白黒二値 */
#define FT_GRAYSCALE 0x10000 /* グレースケール */

/* フォント定義データ */
typedef struct {
    SCRIPT script; /* 文字セット(スクリプト)指定 */
    FCLASS fclass; /* フォントクラス */
    FATTR attr; /* フォント属性 */
    UB size; /* フォントサイズ (文字枠の最大高さ) */
    UB width; /* 文字枠の最大幅 */
    UB base; /* ベース位置 */
    UB leading; /* レディング */
    TC name[12]; /* フォント(ファミリー)名 */
    FCLASS baseclass; /* ベースフォントクラス */
    TC basename[12]; /* ベースフォント(ファミリー)名 */
    TC fullname[20]; /* 詳細フォント名 */
    TC topcode; /* 最初の文字コード */
    TC lastcode; /* 最後の文字コード */
    H rsv[3]; /* 予約 */
    UB imgform; /* 文字イメージの形式 */
    UB widform; /* 文字幅等データの形式 */
    W datasize; /* データ全体のバイトサイズ */
    OFFSET offimage; /* 文字イメージへのオフセット */
    OFFSET offwidth; /* 文字幅等データへのオフセット */
    OFFSET offnote; /* 注釈データへのオフセット */
} FDEF;

/* 文字イメージ形式 */
#define FT_FIXIMG 0 /* 固定イメージ幅形式 */
#define FT_VALIMG 1 /* 可変イメージ幅形式 */

```

```

#define FT_VECTOR 2 /* ベクトル形式(アウトライン) */
#define FT_TRUETYPE 3 /* TrueType形式 */

/* 文字幅等データ形式 */
#define FT_SIMPLEWDATA 0 /* 文字幅データ */
#define FT_INDEXWDATA 1 /* 文字幅/インデックスデータ */
#define FT_IMGWDATA 2 /* イメージ幅付き文字幅データ */

/* フォント所在指定 */
#define FT_MEM 0 /* 固定フォント */
#define FT_FILE 1 /* 動的フォント */
#define FT_SYSTEM 2 /* システムフォント */

/* フォント所在情報 */
typedef union {
    LINK *Ink; /* フォント実身ポインタ(動的フォント) */
    FDEF *addr; /* メモリ上のフォントデータポインタ (固定フォント) */
} FLOC;

/* フォント常駐指定 */
#define FT_RES 0x0100 /* 常駐タイプ指定 */

/* フォント一覧指定 */
#define FT_FONT 0 /* 同一フォントファミリーの一覧 */
#define FT_SCALL 1 /* 同一文字セットの一覧 */
#define FT_SCFAMILY 2 /* 同一文字セットの異なるフォントファミリーの一覧 */
#define FT_LOC 3 /* 同一ファイルの一覧 */
#define FT_ALL -1 /* すべての一覧 */
#define FT_FAMILY -2 /* 異なるフォントファミリーの一覧 */

/* フォントクラス */
#define FTC_MINCHO 0x000060c6
#define FTC_DEFAULT 0x80000000

/* インデックスデータ形式 */
#define FT_IDXNONE 0 /* インデックスデータなし */
#define FT_IDXINDIRECT 1 /* 間接インデックス形式 */

/* フォント一覧 */
typedef struct {
    FID fid; /* フォントID */
    SCRIPT script; /* 文字セット(スクリプト)指定 */
    FCLASS fclass; /* フォントクラス */
    FATTR attr; /* フォント属性 */
    UB size; /* フォントサイズ */
    UB width; /* 文字枠の最大幅 */
    UB base; /* ベース位置 */
    UB leading; /* レディング */
}

```



```
TC      name[12]; /* フォント(ファミリー)名 */
H       swidth;  /* 基準文字幅 */
H       sheight; /* 基準文字高さ */
} FLIST;
```

```
/* 文字イメージデータ */
```

```
typedef struct {
RECT    frame; /* イメージ枠 */
H       width; /* 文字幅 */
H       height; /* 文字高さ */
PNT     imgofs; /* イメージオフセット */
UH      family; /* フォントファミリーの種別 */
UH      fid; /* フォントID */
} FCDATA;
```

```
/* フォントファミリー種別 */
```

```
#define FT_TARGET      0 /* 対象フォントファミリー */
#define FT_ALTTARGET  1 /* 対象フォントファミリー(代替) */
#define FT_BASE        2 /* ベースフォントファミリー */
#define FT_ALTBASE     3 /* ベースフォントファミリー(代替) */
#define FT_DEFAULT     4 /* デフォルトフォントファミリー */
#define FT_UNDEF       -1 /* 未定義文字 */
```

```
/* イメージ取り出し用データ定義 */
```

```
typedef struct {
UW      attr; /* フォント属性 */
UH      height; /* 最大文字高さ */
UH      width; /* 最大文字幅 */
UH      base; /* ベース位置 */
UH      leading; /* レディング */
H       fkind; /* 線種の変形係数 */
H       fslope; /* 傾斜の変形係数 */
H       fweight; /* 太さの変形係数 */
H       fwidth; /* 幅の変形係数 */
H       rowbytes; /* ビットマップの幅 */
UH      resv; /* 予約 */
SIZE    asize; /* フォントの文字サイズ */
UH      aangle; /* フォントの回転角度 */
UH      pixbits; /* ビットマップのピクセルビット数 */
UB      *image; /* ビットマップ開始アドレス */
FCDATA  ch[1]; /* 実際は指定した文字数個の配列 */
} FDATA;
```

```
/* fget_img() 時のオプション */
```

```
#define FT_IMAGE      0x00000001 /* イメージの取り出しを指定 */
#define FT_SYS        0x00000002 /* (システムで使用) */
```

```
/* 標準フォントデータ構造の定義 */
```

```
/* 固定イメージサイズ形式ヘッダ */
```

```
typedef struct {  
    UB    height;    /* 文字高さ */  
    UB    width;     /* 文字幅 */  
    H     rowbytes;  /* ビットマップ横バイト数 */  
    H     margin;    /* ビットマップ左端マージン */  
    UH    offset;    /* イメージデータオフセット */  
} FIDATA;
```

```
/* 可変イメージサイズ形式ヘッダ */
```

```
typedef struct {  
    UB    height;    /* 文字高さ */  
    UB    width;     /* 文字幅 */  
    H     rowbytes;  /* ビットマップ横バイト数 */  
    UH    offset;    /* イメージデータオフセット */  
    UH    pos[1];    /* 文字位置テーブル */  
} VIDATA;
```

```
/* イメージヘッダ */
```

```
typedef union {  
    FIDATA    fidata;    /* 固定イメージサイズ形式 */  
    VIDATA    vidata;    /* 可変イメージサイズ形式 */  
} IMGH;
```

```
/* 文字幅 / 高さデータ */
```

```
typedef struct {  
    B     offset;    /* 文字オフセット */  
    UB    hw;        /* 文字幅 / 高さ */  
} HW;
```

```
/* 文字幅データ */
```

```
typedef struct {  
    UH    s;         /* 開始インデックス */  
    UH    e;         /* 終了インデックス */  
    H     hwdata[1]; /* 文字幅 / 高さデータ */  
} WDATA;
```

```
/* インデックス文字幅データ */
```

```
typedef struct {  
    UB    widform;    /* 文字幅等データの形式 */  
    UB    idxform;    /* インデックスデータ形式 */  
    UB    rsv[2];     /* 予約 */  
    OFFSET offwidth; /* 文字幅データへのオフセット */  
    OFFSET offidxtable; /* インデックスデータへのオフセット */  
} IDXWDATA;
```

```
/* イメージ幅付き文字幅データ */
```

```
typedef struct {  
    UH    s;         /* 開始インデックス */
```

```

UH      e;          /* 終了インデックス */
struct _imgwd {
    B    offset;    /* 文字オフセット */
    UB   hw;        /* 文字幅(高さ) */
    UB   imghw;     /* イメージ幅(高さ) */
} i[1];
} IMGWDATA;

```

### 3.9.3 フォントマネージャの関数

## fdef\_fnt

フォントデータの登録

#### 【形式】

```
WERR    fdef_fnt(FLOC loc, W spec)
```

#### 【パラメータ】

```

FLOC    loc   フォントの所在
W       spec  ::= ( FT_MEM      FT_FILE ) | ( FT_RES ) | ( FT_SYSTEM )

```

```

FT_MEM   : loc はフォントデータへのポインタとなる
FT_FILE  : loc はフォントファイルのリンクを示す。
FT_RES   : 常駐フォントとして登録する。
FT_SYSTEM : システムフォントとして登録する。

```

#### 【リターン値】

```

0       正常(関数値はフォントID)
<0     エラー(エラーコード)

```

#### 【解説】

loc で指定したフォントデータを登録し、そのフォントIDを関数値として戻す。

ファイル指定の場合は、指定したファイルに含まれるフォントレコードタイプのレコードの内容をフォントデータとして登録することになる。複数のフォントレコードが含まれる場合は、すべてのフォントデータを登録し、最後に登録したフォントIDを関数値として戻す。

FT\_RES 指定を行なった場合は、フォントデータを常駐フォントとして登録する。常駐フォントとして登録されたフォントデータは、メモリ上にロードされ高速にアクセスされることが期待できる。実際にフォントデータがメモリ上にロードされ常駐するかどうかはマ

ネージャの実装に依存する。ロードするメモリ領域が不足した場合は、EX\_NOSPC のエラーとなり、登録されない。

FT\_SYSTEM 指定を行なった場合は、フォントデータをシステムフォントとして登録する。システムフォントとして登録されたフォントデータは、決して登録を削除することができない。

指定したフォントが既に登録済みの場合は、特に何もせず、そのフォントIDを関数値として戻す。また、フォントデータの形式が不正、またはサポートしていない形式の場合は、EX\_FTFMT のエラーとなる。

## 【エラーコード】

EX\_ADR : アドレス(loc.addr)のアクセスは許されていない。  
EX\_FONT : ファイルはフォントファイルではない。  
EX\_FTFMT : フォントデータの形式が不正 / サポートしていない。  
EX\_FTID : フォントID/フォントは存在しない。  
EX\_LIMIT : システムの制限を超えた(登録可能な数をオーバー)。  
EX\_NOSPC : システムのメモリ領域が不足した。  
EX\_PAR : パラメータが不正である(specが不正)。

## fdel\_fnt

フォントデータの削除

## 【形式】

ERR fdel\_fnt(FID fid)

## 【パラメータ】

FID fid フォントID

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

fid で指定したフォントデータの登録を削除して、以後は使用できない状態とする。

なお、指定したフォントデータが現在選択されている場合にも登録は削除され、以後使用できなくなってしまうため、注意が必要である。

fid < 0 の場合は、システムフォントを除いて登録してあるフォントデータをすべて削除す

る。

## 【エラーコード】

EX\_FTID : フォントID/フォントは存在しない。

# fdel\_loc

フォントデータの所在による削除

## 【形式】

WERR fdel\_loc(FLOC loc, W spec)

## 【パラメータ】

FLOC loc フォントの所在  
W spec ::= ( FT\_MEM FT\_FILE )

FT\_MEM : loc はフォントデータへのポインタとなる  
FT\_FILE : loc はフォントファイルのリンクを示す。

## 【リターン値】

0 正常(関数値は削除したフォントデータの個数)  
<0 エラー(エラーコード)

## 【解説】

loc で指定したフォントデータを削除して、以後は使用できない状態とする。

ファイル指定の場合は、指定したファイルに含まれるフォントレコードタイプのレコードから登録されたフォントデータをすべて削除する。フォントデータが登録されていない場合は何もしない。

なお、指定したフォントデータが現在選択されている場合にも登録は削除され、以後使用できなくなってしまうため、注意が必要である。

削除したフォントデータの数を関数値として戻す。一つも削除しなかった場合は、0 を返す。

## 【エラーコード】

EX\_ADR : アドレス(loc.addr)のアクセスは許されていない。  
EX\_PAR : パラメータが不正である(specが不正)。

# fget\_def

フォント定義データの取り出し

## 【形式】

WERR fget\_def(FID fid, FDEF \*def)

## 【パラメータ】

FID fid フォントID  
FDEF \*def フォント定義データ

## 【リターン値】

0 正常(関数値は FT\_FILE | FT\_MEM)  
<0 エラー(エラーコード)

## 【解説】

fid で指定したフォントデータの定義データを取り出し、def で指定した領域に格納する。def の領域はあらかじめ確保されていなくてはならない。

def 内に得られた FDEF データの offimage, offwidth, offnote の値は意味を持たないため、参照してはならない。

## 【エラーコード】

EX\_ADR : アドレス(buff)のアクセスは許されていない。  
EX\_FTID : フォントID / フォントは存在しない。

# fget\_not

フォント注釈データの取り出し

## 【形式】

WERR fget\_not(FID fid, B \*buff, UW len)

## 【パラメータ】

FID fid      フォントID  
B    \*buff    注釈データ格納領域  
UW  len      buff のバイト数

### 【リターン値】

0      正常(関数値は注釈データの文字数)  
<0    エラー(エラーコード)

### 【解説】

fid で指定したフォントデータの注釈データ(文字列)を取り出し、buff で指定した領域に格納する。

len は buff の領域の文字数を示し、注釈データの文字数が len 以上の場合は len 文字数のみ格納され、最後の TNULL は格納されない。

関数値として注釈データの文字数が戻る。関数値 0 は、注釈データが存在しないことを意味する。

buff = NULL、または len = 0 の場合は、注釈文字列は格納されずに、その文字数が関数値として戻る。

### 【エラーコード】

EX\_ADR      : アドレス(buff)のアクセスは許されていない。  
EX\_FTID     : フォントID/フォントは存在しない。  
EX\_PAR      : パラメータが不正である(len<0)。

## flst\_fon

フォントの一覧取り出し

### 【形式】

WERR      flst\_fon(FID fid, W mode, FLIST \*buff, UW len)

### 【パラメータ】

FID fid      フォントID  
W    mode      フォントデータ一覧条件

FT\_FONT (0) :

fid で指定したフォントデータと同一フォントファミリーのフォントデータの一覧を取り出す(指定したフォントを含む)。

FT\_ALL (-1) :

すべてのフォントデータの一覧を取り出す。fidの指定は無視される。

FT\_FAMILY(-2) :

異なるフォントファミリーの一覧を取り出す (各フォントファミリーの中から1つのフォントデータのみを取り出す)。fidの指定は無視される。

FT\_SCALL (1) :

fidで指定したフォントデータと同一文字セットのフォントデータの一覧を取り出す。

FT\_SCFAMILY(2) :

fidで指定したフォントデータと同一文字セットで、異なるフォントファミリーの一覧を取り出す (各フォントファミリーの中から1つのフォントデータのみを取り出す)。

FT\_LOC (3) :

fidで指定したフォントデータと同一ファイルのフォントデータ一覧を取り出す。

FLIST \*buff フォントデータ一覧格納領域  
UW len buffの要素数

### 【リターン値】

0 正常(関数値は対象とするフォントデータ個数)  
<0 エラー(エラーコード)

### 【解説】

現在登録してあるフォントデータの内、fid, modeで指定した条件に適合するフォントデータの一覧を取り出し、buffで示される領域に格納する。関数値として、対象とするフォントデータの個数が戻る。

対象とするフォントデータの個数が、lenより大きい場合は、lenで示される個数の一覧のみ取り出される。

buff = NULL、またはlen = 0の場合は一覧は格納されずに、対象とするフォントデータの個数が関数値として戻る。

### 【エラーコード】

EX\_ADR : アドレス(buff)のアクセスは許されていない。  
EX\_FTID : フォントID/フォントは存在しない。  
EX\_PAR : パラメータが不正である(len<0, modeが不正)。

## fopn\_fon

フォントセットの生成



## 【形式】

WERR fopn\_fon()

## 【パラメータ】

なし

## 【リターン値】

0 正常(関数値はフォントディスクリプタ)  
<0 エラー(エラーコード)

## 【解説】

フォントセットを生成する。関数値としてそのフォントディスクリプタが戻る。

## 【エラーコード】

EX\_NOMEM : メモリ領域が不足した。

# fcls\_fon

フォントセットの削除

## 【形式】

ERR fcls\_fon(W fdesc)

## 【パラメータ】

W fdesc フォントディスクリプタ

## 【リターン値】

=0 正常  
<0 エラー(エラーコード)

## 【解説】

fdesc で指定されたフォントセットを削除する。

## 【エラーコード】

EX\_FTD : フォントディスクリプタは存在しない。

# fset\_fon

フォントセットの設定

## 【形式】

ERR fset\_fon(W fdesc, FSSPEC \*spec)

## 【パラメータ】

|        |       |             |
|--------|-------|-------------|
| W      | fdesc | フォントディスクリプタ |
| FSSPEC | *spec | フォント指定      |

## 【リターン値】

= 0 正常  
< 0 エラー(エラーコード)

## 【解説】

fdesc で指定されたフォントセットに \*spec で指定したフォント情報(フォントファミリー、属性、サイズ)を設定する。

## 【エラーコード】

EX\_ADR : アドレス(fspec)のアクセスは許されていない。  
EX\_FTD : フォントディスクリプタは存在しない。  
EX\_PAR : パラメータが不正である(文字サイズが不正)。

# fget\_fon

フォントセットの取り出し

## 【形式】

ERR fget\_fon(W fdesc, FSSPEC \*spec)

## 【パラメータ】

W            fdesc    フォントディスクリプタ  
FSSPEC    \*spec    フォント指定を格納する領域

## 【リターン値】

= 0    正常  
< 0    エラー(エラーコード)

## 【解説】

fdesc で指定されたフォントセットのフォント情報を取り出し、\*spec で指定した領域に戻す。

## 【エラーコード】

EX\_ADR        : アドレス(fspec)のアクセスは許されていない。  
EX\_FTD        : フォントディスクリプタは存在しない。

# fset\_ang

回転角度の設定

## 【形式】

ERR fset\_ang(W fdesc, W ang);

## 【パラメータ】

W    fdesc    フォントディスクリプタ  
W    ang      回転角度

## 【リターン値】

= 0    正常  
< 0    エラー(エラーコード)

## 【解説】

fdesc で指定されたフォントセットに回転角度を設定する。

## 【エラーコード】

EX\_FTD : フォントディスクリプタは存在しない。  
EX\_PAR : パラメータが不正である(angが不正)。

## fget\_ang

回転角度の取り出し

### 【形式】

```
WERR fget_ang(W fdesc);
```

### 【パラメータ】

W fdesc フォントディスクリプタ

### 【リターン値】

0 正常(回転角度)  
<0 エラー(エラーコード)

### 【解説】

fdesc で指定されたフォントセットの回転角度 (0 ~ 359 の度単位の非負の数値) を取り出す。

### 【エラーコード】

EX\_FTD : フォントディスクリプタは存在しない。

## fget\_fam

文字フォント情報の取り出し

### 【形式】

```
WERR fget_fam(W fdesc, W script, FNTINFO *inf);
```

### 【リターン値】

0 正常(フォントID)

## <0 エラー(エラーコード)

### 【解説】

フォントディスクリプタから、script で指定した文字セットのフォントファミリーのフォント情報を inf で指定した領域に取り出す。

infの内容は、FSSPEC で指定した文字サイズにスケーリングされた値である。

通常、フォントファミリーは複数のフォントデータから構成されるが、いずれのフォントデータの情報が取り出されるかはインプリメントに依存する。

フォントデータの選択によっては、fget\_fam() で取り出したフォント情報と fget\_img() で取り出したフォント情報とは異なる場合がある。

scriptで指定した文字セットのフォントデータが存在しない場合、EX\_FTIDのエラーとなる。この場合でも inf には有効な内容が格納される。

### 【エラーコード】

EX\_ADR : アドレス(inf)のアクセスは許されていない。  
EX\_FTID : フォントID/フォントは存在しない。  
EX\_FTD : フォントディスクリプタは存在しない。

## fget\_img

文字イメージ情報の取り出し

### 【形式】

WERR fget\_img(W fdesc, FDATA \*cimg, W size, W script, TC ch, UW mode)

### 【パラメータ】

|       |        |                    |
|-------|--------|--------------------|
| W     | fdesc  | フォントディスクリプタ        |
| FDATA | *cimg  | 文字イメージ情報           |
| W     | size   | cimgで指定した領域のバイトサイズ |
| W     | script | 文字セット              |
| TC    | ch     | 文字コード              |
| W     | mode   | 取り出す内容の指定          |

mode & FT\_IMAGE = 0:

ch で指定した文字の実際の文字イメージビットマップ以外のイメージ情報 ( 即ち、文字幅 / 高さの情報 ) を取り出す。

mode & FT\_IMAGE != 0:

ch で指定した文字の実際の文字イメージビットマップを含む全てのイメージ情報を取

り出す。

mode & FT\_SYS = 0:

文字イメージビットマップは cimg で指定した領域に格納される。

mode & FT\_SYS != 0:

文字イメージビットマップはマネージャが管理する領域に格納される。

## 【リターン値】

0 正常(関数値は情報を取り出した文字数 ( = 1 ) )  
<0 エラー(エラーコード)

## 【解説】

fdesc で選択したフォントセットの、script で指定した文字セット, ch で指定した文字コードのイメージ情報を取り出して、cimg で指定した領域に格納する。関数値は情報が得られた文字数として常に 1 が戻る。

cimg で指定した FDATA の構造体のサイズは、inf 配列の要素数が 1 のサイズである必要がある。size が sizeof ( FDATA ) に満たない場合はエラー ( EX\_PAR ) となり、文字のイメージ情報は一切取り出されない。

mode & FT\_IMAGE = 0 の場合、cimg に戻される内容の rowbytes, pixbits, image, ch[].frame, ch[].imgofs の値は保証されないため image で示されるメモリを参照してはならない。

mode & FT\_IMAGE != 0 の場合、cimg の指す領域は FDATA 構造体とともに実際の文字イメージビットマップを格納できる大きさでなければならない。size が文字イメージビットマップを格納できる大きさに満たない場合はエラー ( EX\_PAR ) となる。

mode & FT\_SYS != 0 の場合、得られたビットマップは、フォントマネージャが管理するメモリ領域上に存在する。同一のフォントディスクリプタによる新たな文字イメージのアクセスを行なうまでは存在していることが保証される。通常のアプリケーションは FT\_SYS を指定してはいけない。

script で指定した文字セットのフォントデータが存在しない場合、エラーとはならないが、すべての文字コードに対して未定義文字イメージが返る。

## 【エラーコード】

EX\_ADR : アドレス(cimg)のアクセスは許されていない。  
EX\_FTD : フォントディスクリプタは存在しない。  
EX\_PAR : パラメータが不正である(sizeが不正)。  
EX\_NOSPC : システムのメモリ領域が不足した。

### 3.9.4 標準フォントデータ形式

#### 全体の形式

#### 固定フォントの場合

固定フォントの場合は、フォントデータの全体の形式は以下の通りであり、各データブロックの順序、位置は任意である。

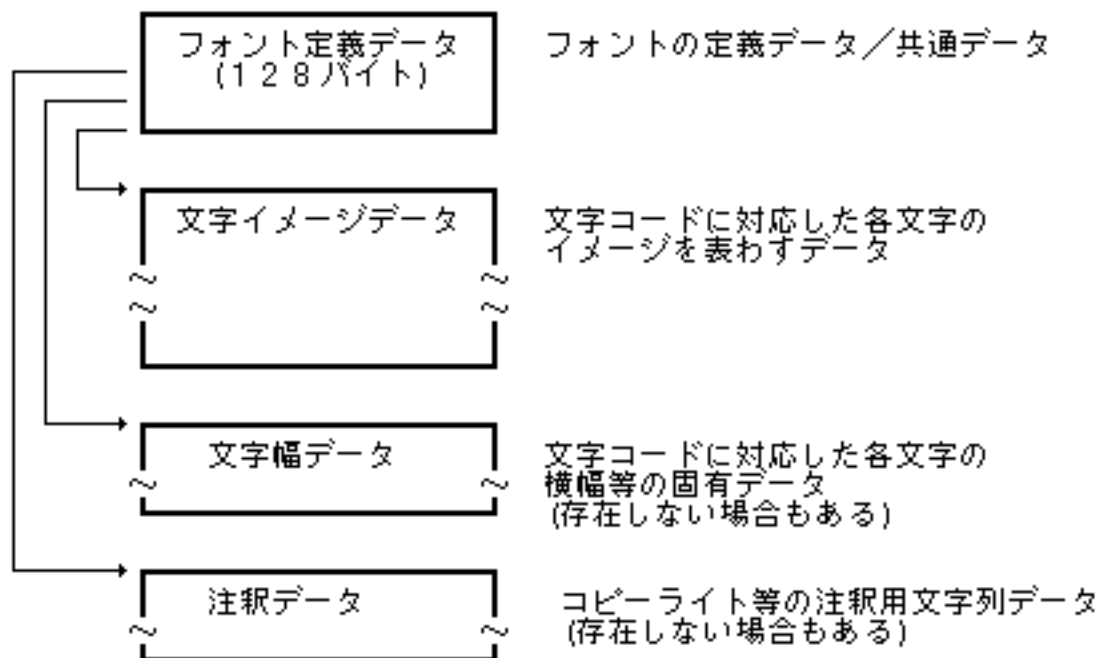


図 136 : フォントデータの全体構造

#### 動的フォントの場合

動的フォント、即ちフォントファイルの場合は、ファイルはフォント定義データおよび文字イメージデータのいずれかあるいは両方を含む一つ以上のレコードにより構成される。

レコードは、先頭がフォント定義データで始まるデータを格納したレコードタイプ 11 (フォントデータレコード) のレコード、および文字イメージのみを格納したレコードタイプ 15 ~ 31 (システムデータ/アプリケーションレコード) のいずれかである。

一つのフォントデータレコードには一つのフォント定義データを格納することができ、一つのフォントファイル中には複数のフォントデータレコードを格納することができる。通常は一つのフォントファイルには一つのフォントデータを格納するが、複数のフォントデータが一つの文字イメージを共有する場合、あるいは関連の強いフォントデータである場合は、一つのフォントファイル中に複数のフォントデータを格納する。

#### 文字イメージデータの分離

動的フォントの場合、文字イメージデータをフォント定義データとは別のレコードに格納することができる。この場合、文字イメージデータのレコードタイプはレコードサブタイプフォント定義データの offimage が 0 ~ 127 の場合、レコードタイプ 15 ~ 31 (システムデータ/アプリケーションレコード) とする。複数の文字イメージデータレコードを格納する場合は、レコードサブタイプにより区別する。

## 特定の文字イメージデータの指定

TrueType 形式の文字イメージデータの場合、複数の文字イメージデータが一つの文字イメージデータレコードに含まれる場合がある。この場合は、フォント定義データのレコードサブタイプにより区別する。

### 文字イメージデータ形式

#### 標準ドット形式

標準ドット形式の文字イメージデータは以下の2種類に分類される。

#### 固定イメージサイズ形式

固定されたイメージ幅で文字イメージを格納する

#### 可変イメージサイズ形式

文字毎に異なったイメージ幅で文字イメージを格納する

このうち、以下に示す固定イメージサイズ形式のみが使用できる。

固定イメージサイズ形式の文字イメージデータは以下に示す形式となる。即ち、ビットマップ上に、固定サイズの枠が規則正しく並んでいることになる。横に並ぶ文字数は  $(\text{rowbytes} \times 8 - \text{margin}) \div \text{width}$  となり、右端に空きがある場合もある。offset は文字イメージデータの先頭からのバイトオフセットとなる。



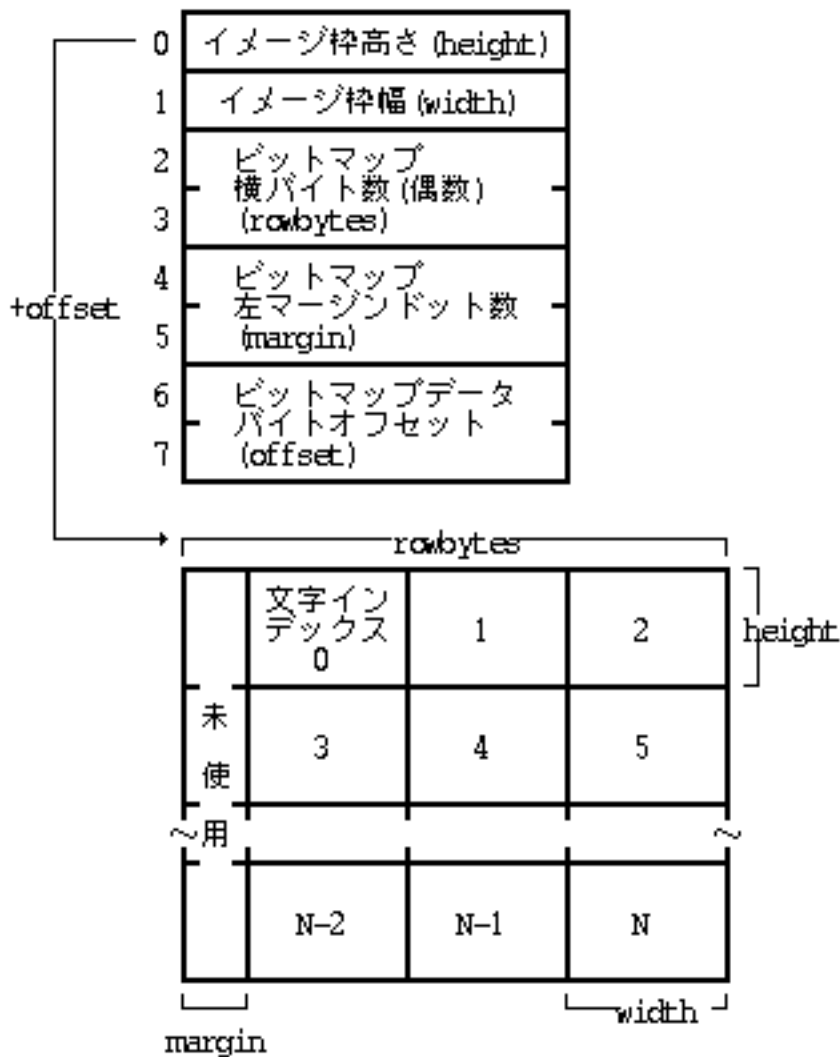


図 137: 標準ドット形式(固定イメージサイズ)

## TrueType 形式

TrueType 仕様の規定に従う。

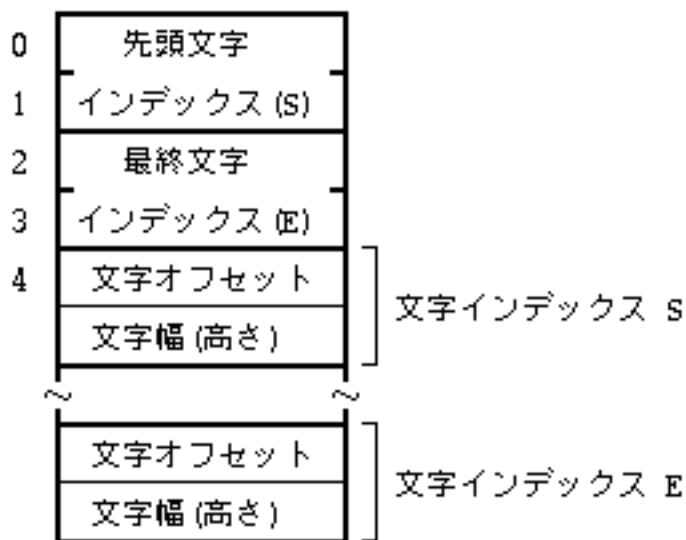
TrueType は Apple Computer, Inc の登録商標である。

## 文字幅等データ形式

### 3.1 文字幅データ

文字幅データは、比例ピッチフォントの場合に各文字の文字幅 (または高さ) と文字イメージオフセットを示すデータであり、以下に示す形式である。

このデータはフォントの定義域のすべての文字ではなく、定義域内の1つの連続した範囲に対して定義される。定義されていない文字に対しては、文字幅は最大文字幅、文字オフセットは0とみなされる。



文字オフセット: 符号付き -127~128  
 文字幅 (高さ) : 符号なし 0~255

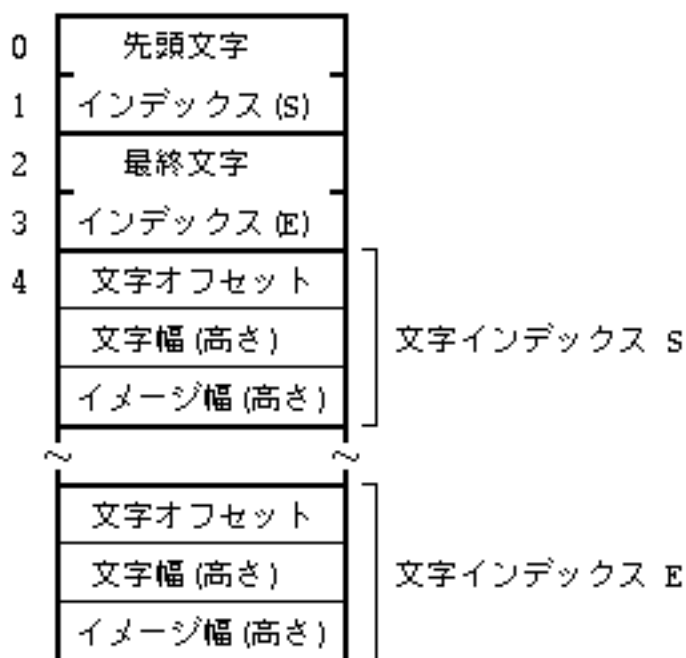
図 138: 標準文字幅データの形式

### イメージ幅付き文字幅データ

イメージ幅付き文字幅データは、比例ピッチフォントの場合に各文字の文字幅 (または高さ) と文字イメージオフセット、およびイメージ枠の幅 (または高さ) を示すデータであり、以下に示す形式である。

このデータはフォントの定義域のすべての文字ではなく、定義域内の1つの連続した範囲に対して定義される。定義されていない文字に対しては、文字幅は最大文字幅、文字オフセットは0、イメージ枠は文字幅と等しいとみなされる。

イメージ枠の幅は最大文字幅を超えてはいけない。



文字オフセット : 符号付き -127~128  
 文字幅 (高さ) : 符号なし 0~255  
 イメージ幅 (高さ): 符号なし 0~255

図 139 : イメージ幅指定付き文字幅データの形式

### 文字幅 / インデックスデータ

文字幅 / インデックスデータは、各文字の文字幅を示すデータおよび文字コードと文字イメージデータ内のインデックスの対応を定義するデータである。インデックスの定義が必要な場合に使用する。

文字幅は、文字幅 / インデックスデータの先頭からのバイトオフセットにより、文字幅データ等を指定する。

インデックスデータは、定義域内での文字の順序と、文字イメージデータ内での文字の順序が異なる場合に、文字コードと文字インデックスの対応を格納したデータを、文字幅 / インデックスデータの先頭からのバイトオフセットにより指定する。

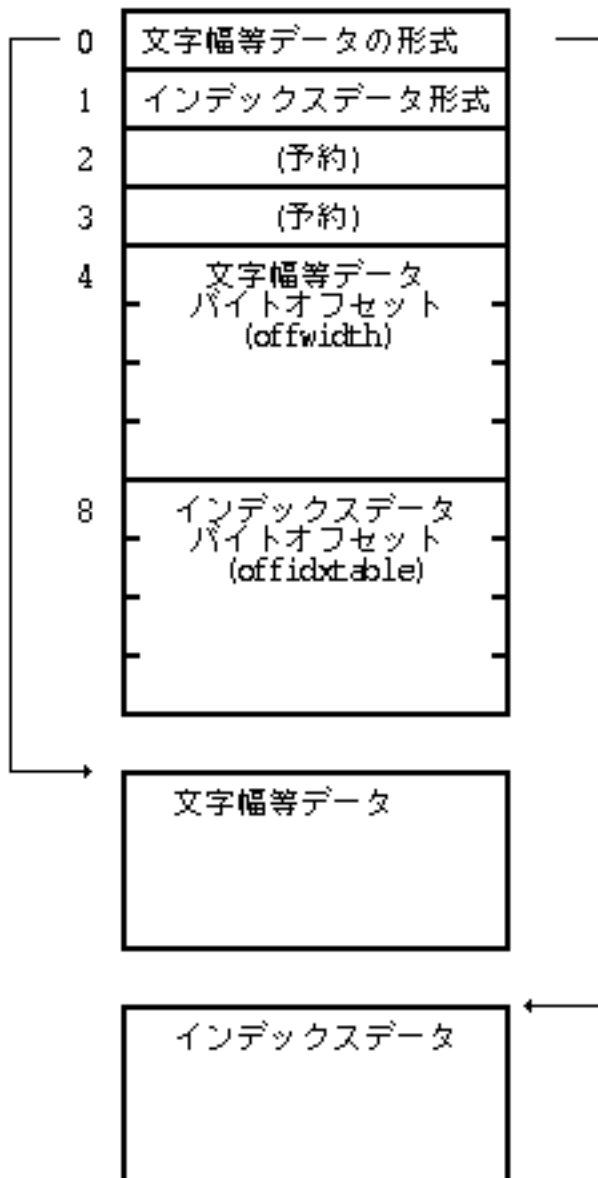


図 140 : 文字幅 / インデックスデータ

インデックスデータが存在する場合、文字コードから計算した文字インデックスによりインデックスデータを参照し、それにより得られた文字インデックスを使用して文字イメージデータ内の文字イメージにアクセスすることになる。インデックスデータ形式として、以下の間接インデックス形式がサポートされている。

間接インデックス形式は、フォント定義データが指定する最初の文字コードから、定義域内の文字全てについて文字インデックスを格納した配列である。特別に、文字インデックスが 0xffff の場合は、その文字が未定義であることを表す。

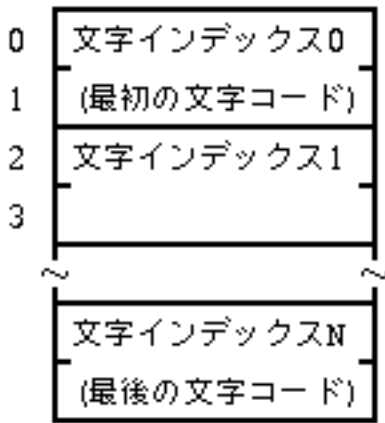


図 141 : 間接インデックス

---

[この章の目次にもどる](#)

[前頁:3.8 実身 / 仮身マネージャにもどる](#)

[次頁:3.10 TCP/IPマネージャにすすむ](#)

[この章の目次にもどる](#)

[前頁:3.9 フォントマネージャにもどる](#)

[次頁:3.11 印刷マネージャ仕様書にすすむ](#)

---

## 3.10 TCP/IPマネージャ

### 3.10.1 構成

TCP/IP マネージャは LAN を利用した TCP/IP 通信の機能を実現するモジュールでネットワーク上の別のマシンと TCP/IP プロトコルを用いて通信することができる。TCP/IP マネージャのシステムコールはソケットを意識したものとなっている。

通信はソケットを作成し、ソケットデスクリプタを使用して行なう。このソケットデスクリプタはファイルデスクリプタとは別の概念であり、TCP/IP マネージャのシステムコールではソケットデスクリプタのみを扱うことができる。

また、ソケットによる通信であっても、同一マシン内の異なるプロセス間の通信手段として利用することはできない。

### 3.10.2 TCP/IP

以下のデータを定義する。

```
/* プロトコルファミリ */
```

```
#define PF_UNSPEC    0        /* unspecified protocol family */
#define PF_INET      2        /* TCP/IP and related */
```

```
/* アドレスファミリ
*/
```

```
#define AF_UNSPEC    0        /* unspecified address family */
#define AF_INET      2        /* TCP/IP and related */
```

```
/* ソケットタイプ
*/
```

```
#define SOCK_STREAM  1        /* stream socket */
#define SOCK_DGRAM   2        /* datagram socket */
#define SOCK_RAW     3        /* raw-protocol interface */
```

```
/* getsockopt(), setsockopt() のオプション
*/
```

```
#define SOL_SOCKET   0xffff   /* options for socket level */
#define IPPROTO_TCP  0x0001   /* options for TCP level */
#define IPPROTO_IP   0x0002   /* options for IP level */
```

```
#define SO_DEBUG     0x0001   /* turn on debugging info recording */
#define SO_REUSEADDR 0x0004   /* allow local address reuse */
```

```

#define SO_KEEPALIVE      0x0008      /* keep connections alive */
#define SO_DONTROUTE     0x0010      /* just use interface addresses */
#define SO_BROADCAST     0x0020      /* permit sending of broadcast msgs */
#define SO_LINGER        0x0080      /* linger on close if data present */
#define SO_OOBINLINE     0x0100      /* leave received OOB data in line */
#define SO_SNDBUF        0x1001      /* send buffer size */
#define SO_RCVBUF        0x1002      /* receive buffer size */
#define SO_ERROR         0x1007      /* get error status and clear */
#define SO_TYPE          0x1008      /* get socket type */

#define TCP_MAXSEG       0x2000      /* maximum segment size */
#define TCP_NODELAY      0x2001      /* immediate send() */

#define IP_OPTIONS       0x0001      /* IP header options */

/*  recv, send のオプション */

#define MSG_OOB          0x01        /* send or receive out of band data */
#define MSG_PEEK         0x02        /* take data but leave it */
#define MSG_DONTROUTE    0x04        /* do not route */

/*  fcntl のオプション */

#define O_NDELAY         0x04        /* non-blocking */
#define FNDELAY          O_NDELAY    /* synonym */
#define F_GETFL         3           /* get flags */
#define F_SETFL         4           /* set flags */

/*  ioctl のオプション */

#define SIOCATMARK      7           /* check for out of bound data */

/*  アドレス */
struct sockaddr {
    unsigned short sa_family; /* address family */
    char sa_data[14];        /* up to 14 bytes of direct address */
};

/*  インターネットアドレス */
struct in_addr {
    unsigned long s_addr;
};

/*  ソケットアドレス */
struct sockaddr_in {
    short sin_family;
    unsigned short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};

```

```

struct iovec {                /* address and length */
    char *iov_base;          /* base */
    int iov_len;            /* size */
};

/* recvmsg, sendmsg 用メッセージヘッダ */
struct msghdr {              /* Message header for recvmsg and sendmsg. */
    char *msg_name;         /* optional address */
    int msg_namelen;        /* size of address */
    struct iovec *msg_iov;   /* scatter/gather array */
    int msg_iovlen;         /* # elements in msg_iov */
    char *msg_accrights;     /* access rights sent/received */
    int msg_accrightslen;
};

struct hostent {             /* structure for gethostbyname */
    char *h_name;           /* official name of host */
    char **h_aliases;       /* alias list */
    int h_addrtype;         /* host address type */
    int h_length;           /* length of address */
    char **h_addr_list;     /* list of addresses from name server */
#define h_addr h_addr_list[0] /* address, for backward compatibility */
};

struct servent {             /* structure for getservbyname */
    char *s_name;           /* official service name */
    char **s_aliases;       /* alias list */
    int s_port;             /* port # */
    char *s_proto;          /* protocol to use */
};

struct linger {              /* structure for the SO_LINGER option */
    int l_onoff;            /* zero=off, nonzero = on */
    int l_linger;           /* linger time, in seconds */
};

#define FD_SETSIZE 256
typedef struct fd_set {      /* Bit mask for select() */
    int fds_bits[FD_SETSIZE/sizeof(int)];
} fd_set;

struct timeval {             /* Timeout format for select() */
    long tv_sec;            /* seconds */
    long tv_usec;          /* microseconds */
};

```

### 3.10.3 エラーコード

以下のエラーコードが定義されている。

```
#define EX_HOSTUNREACH ((-401) << 16) /* ホストが見つからない */
#define EX_TIMEDOUT ((-402) << 16) /* TCPIP タイムアウト */
#define EX_CONNABORTED ((-403) << 16) /* 接続がアボートされた */
#define EX_NOBUFS ((-404) << 16) /* TCPIP 内部領域不足 */
#define EX_BADF ((-405) << 16) /* ソケットが無効 */
#define EX_WOULDBLOCK ((-407) << 16) /* ブロックされる処理がある */
#define EX_MSGSIZE ((-408) << 16) /* 送信メッセージの細分化ができない */
#define EX_DESTADDRREQ ((-409) << 16) /* 送信先アドレス指定が必要 */
#define EX_PROTOTYPE ((-410) << 16) /* プロトコルが不正 */
#define EX_NOPROTOOPT ((-411) << 16) /* 利用できないプロトコル指定 */
#define EX_PROTONOSUPPORT ((-412) << 16) /* 認識されていないオプション指定 */
#define EX_SOCKETNOSUPPORT ((-413) << 16) /* サポートされていないソケットタイプ */
#define EX_OPNOTSUPP ((-414) << 16) /* サポートされていない動作指定 */
#define EX_PFNOSUPPORT ((-415) << 16) /* サポートされていないプロトコルファミリ */
#define EX_AFNOSUPPORT ((-416) << 16) /* サポートされていないアドレスファミリ */
#define EX_ADDRINUSE ((-417) << 16) /* アドレスは使用中 */
#define EX_ADDRNOTAVAIL ((-418) << 16) /* アドレスが利用できない */
#define EX_NETDOWN ((-419) << 16) /* ネットワーク機能が無効 */
#define EX_NETUNREACH ((-420) << 16) /* ネットワークが見つからない */
#define EX_NETRESET ((-421) << 16) /* ネットワークがリセットされた */
#define EX_CONNRESET ((-422) << 16) /* 接続がリセットされた */
#define EX_ISCONN ((-423) << 16) /* ソケットは接続済み */
#define EX_NOTCONN ((-424) << 16) /* ソケットは接続されていない */
#define EX_SHUTDOWN ((-425) << 16) /* 送信は禁止されている */
#define EX_CONNREFUSED ((-426) << 16) /* 接続が拒絶された */
#define EX_HOSTDOWN ((-427) << 16) /* ホストがダウンしている */
#define EX_ALREADY ((-428) << 16) /* 既に処理中の動作 */
#define EX_INPROGRESS ((-429) << 16) /* 非ブロック化ソケットで接続中 */
```

### 3.10.4 システムコール

#### so\_start

TCP/IPの初期化

#### 【形式】

ERR so\_start (W arg)

#### 【パラメータ】

W arg 特殊用途の引数



## 【リターン値】

=0 正常  
<0 エラー(エラーコード)

## 【解説】

TCP/IP 通信を初期化する。通常はアプリケーションプロセスが TCP/IPMGR のその他のシステムコールを呼び出すと暗黙的に TCP/IP 通信の初期化が行なわれる。

# so\_finish

TCP/IPの終了宣言

## 【形式】

ERR so\_finish (W arg)

## 【パラメータ】

W arg 0 終了宣言  
<0 特殊用途

## 【リターン値】

0 正常  
<0 エラー

## 【解説】

TCP/IP の終了宣言。  
プロセス終了時に自動的に呼び出されるので通常は、明示的に呼び出す必要はない。

## 【エラーコード】

ER\_BUSY TCP/IP ロック中。  
EX\_NOSPT TCP/IP を使用していないのに so\_finish を呼び出した。

# so\_accept

ソケットへの接続受け付け

## 【形式】

WERR so\_accept (W s, SOCKADDR \*nam, W \*namlen)

### 【パラメータ】

W s ソケットデスクリプタ  
SOCKADDR \*nam アドレスが戻される  
W \*namlen アドレスのバイト長

### 【リターン値】

0 新規生成されたソケットデスクリプタ  
<0 エラーコード

### 【解説】

listen 状態にあるソケットに対して接続要求を待ち、要求があったときに最初の要求を取りだし、ソケット s と同じ属性の新しいソケットを生成しそのデスクリプタを戻す。

接続要求元のアドレスとその長さが、nam および namlen に戻される。

元のソケット s は、次の要求を受け付けるためにオープンされたままとなる。

### 【エラーコード】

EX\_ADR パラメータでアクセスが許されていない領域を参照している。  
EX\_PAR パラメータが不正(ソケットデスクリプタの値が不正)  
EX\_BADF ソケットが無効  
EX\_WOULDBLOCK ソケットが非ブロック化されていて、接続要求がない  
EX\_NOBUFS 内部領域不足

## so\_bind

ソケットへの名前のバインド

### 【形式】

ERR so\_bind (W s, SOCKADDR \*nam, W namlen)

### 【パラメータ】

W s ソケット  
SOCKADDR \*nam アドレス  
W namlen アドレスのバイト長

### 【リターン値】

= 0 正常  
< 0 エラー

### 【解説】

名前なしのソケットに `nam` および `namlen` で指定した名前を割り当てる。

### 【エラーコード】

EX\_ADR            パラメータでアクセスが許されていない領域を参照している。  
EX\_PAR            パラメータが不正(ソケットデスクリプタの値が不正)。  
EX\_BADF           ソケットが無効。  
EX\_FAULT          名前が不正。  
EX\_ADDRNOTAVAIL 名前(アドレス)が利用できない。  
EX\_ADDRINUSE     名前(アドレス)は使用中。

## so\_close

ソケットを閉じる

### 【形式】

ERR so\_close (W s)

### 【パラメータ】

Ws ソケットデスクリプタ

### 【リターン値】

= 0 正常  
< 0 エラー

### 【解説】

ソケットをクローズする。

### 【エラーコード】

EX\_PAR            パラメータが不正(ソケットデスクリプタの値が不正)  
EX\_BADF           ソケットが無効。

## so\_connect

ソケットへの接続の開始

## 【形式】

ERR so\_connect (W s, SOCKADDR \*nam, W namlen)

## 【パラメータ】

|          |        |            |
|----------|--------|------------|
| W        | s      | ソケットデスクリプタ |
| SOCKADDR | *na m  | アドレス       |
| W        | namlen | アドレスのバイト長  |

## 【リターン値】

= 0 正常  
< 0 エラー

## 【解説】

s が SOCK\_DGRAM 型ソケットの場合は、ソケットに関連づけられる通信先のアドレスを指定する。

s が SOCK\_STREAM 型ソケットの場合は、指定されたアドレスへの接続を行なう。

## 【エラーコード】

|                |                              |
|----------------|------------------------------|
| EX_PAR         | パラメータが不正(ソケットデスクリプタの値が不正)    |
| EX_ADR         | パラメータでアクセスが許されていない領域を参照している。 |
| EX_BADF        | ソケットが無効。                     |
| EX_FAULT       | 名前(アドレス)が不正。                 |
| EX_HOSTUNREACH | 接続先が不正。                      |
| EX_INPROGRESS  | ソケットが非ブロック型で、接続がすぐに完了しない。    |
| EX_TIMEDOUT    | タイムアウト。                      |
| EX_CONNABORTED | 接続がアボートされた。                  |

# so\_fcntl

ソケットの制御

## 【形式】

WERR so\_fcntl (W s, W cmd, W arg)

## 【パラメータ】

|   |     |            |
|---|-----|------------|
| W | s   | ソケットデスクリプタ |
| W | cmd | 以下         |

F\_GETFL デスクリプタ状態フラグを取得する  
F\_SETFL デスクリプタ状態フラグを設定する

W arg フラグ値

### 【リターン値】

0 フラグ値(F\_GETFL)  
<0 エラー

### 【解説】

ソケットデスクリプタの状態フラグを取得/設定する。

有効なフラグ値は O\_NDELAY のみである。

O\_NDELAY を指定されたソケットは非ブロック状態となり、ソケットに関連したシステムコールでは待ちに入らなくなる。

### 【エラーコード】

EX\_PAR パラメータが不正(ソケットデスクリプタの値が不正)  
EX\_BADF ソケットが無効。

## so\_gethostbyname

名前によるホスト情報の取得

### 【形式】

ERR so\_gethostbyname (B \*nam, HOSTENT \*hp, B \*buf)

### 【パラメータ】

|         |      |       |
|---------|------|-------|
| B*      | nam  | ホスト名  |
| HOSTENT | *hp  | ホスト情報 |
| B       | *buf | 作業領域  |

### 【リターン値】

=0 正常  
<0 エラー

### 【解説】

nam で指定された名前を持つホストに関する情報を hp で指定された領域に格納する。  
B buf[HBUFLE] で定義される作業領域を指定する必要がある。

hp が指す領域には以下の情報が戻される。

```

struct hostent {
    char *h_name;           /* structure for gethostbyname */
    char **h_aliases;      /* official name of host */
    int h_addrtype;        /* alias list */
    int h_length;          /* host address type */
    char **h_addr_list;    /* length of address */
    #define h_addr h_addr_list[0] /* list of addresses from nameserver */
                                /* address, for backward compatibility */
};

```

**h\_name**       ホストの正式名  
**h\_aliases**   ホストのエイリアス名(実装では NULL が戻る)  
**h\_addrtype**   アドレスタイプ(常に AF\_INET が戻る)  
**h\_length**     アドレスのバイト長  
**h\_addr\_list**  ホストのネットワークアドレスのリストへのポインタ

### 【エラーコード】

**EX\_ADR**        パラメータでアクセスが許されていない領域を参照している。  
**EX\_HOSTUNREACH**  指定されたホストが見つからない。  
**EX\_TIMEOUT**    タイムアウト。

## so\_gethostbyaddr

アドレスによるホスト情報の取得

### 【形式】

ERR so\_gethostbyaddr (B \*addr, W type, W len, HOSTENT \*hp, B \*buf)

### 【パラメータ】

|         |       |                         |
|---------|-------|-------------------------|
| B       | *addr | アドレス                    |
| W       | type  | アドレスタイプ(AF_INET のみ指定可能) |
| W       | len   | アドレスのバイト長               |
| HOSTENT | *hp   | ホスト情報                   |
| B       | *buf  | 作業領域                    |

### 【リターン値】

= 0  正常  
 < 0  エラー

### 【解説】

アドレスで指定されたホストに関する情報を hp で指定された領域に格納する。  
B buf[HBUFLE] で定義される作業領域を指定する必要がある。

### 【エラーコード】

EX\_ADR            パラメータでアクセスが許されていない領域を参照している。  
EX\_HOSTUNREACH 指定されたホストが見つからない

## ERR so\_getpeername

接続相手の名前の取得

### 【形式】

ERR so\_getpeername (W s, SOCKADDR \*nam, W \*namlen)

### 【パラメータ】

|          |         |            |
|----------|---------|------------|
| W        | s       | ソケットデスクリプタ |
| SOCKADDR | *nam    | アドレス       |
| W        | *namlen | アドレスのバイト長  |

### 【リターン値】

= 0 正常  
< 0 エラー

### 【解説】

ソケットに接続されている相手のアドレスを戻す。  
namlen は nam が指す領域のサイズを指定する。実行後は nam に戻されたアドレスのバイト長が戻される。

### 【エラーコード】

EX\_ADR            パラメータでアクセスが許されていない領域を参照している。  
EX\_PAR            パラメータが不正(ソケットデスクリプタの値が不正)  
EX\_BADF           ソケットが無効。  
EX\_FAULT          名前(アドレス)が不正。

## so\_getsockname

ソケット名の獲得

## 【形式】

ERR so\_getsockname (W s, SOCKADDR \*nam, W \*namlen)

## 【パラメータ】

W s ソケットデスクリプタ  
SOCKADDR \*nam アドレス  
W \*namlen アドレスのバイト長

## 【リターン値】

= 0 正常  
< 0 エラー

## 【解説】

指定したソケットに対する現在の名前(アドレス)を戻す。  
namlen は nam のサイズを指定する。実行後は name に戻されたアドレスのバイト長が戻される。

## 【エラーコード】

EX\_ADR パラメータでアクセスが許されていない領域を参照している。  
EX\_PAR パラメータが不正(ソケットデスクリプタの値が不正)  
EX\_BADF ソケットが無効。  
EX\_FAULT 名前(アドレス)が不正。

# so\_getsockopt

ソケットオプションの取得

## 【形式】

ERR so\_getsockopt (W s, W level, W optnam, B \*optval, W \*optlen)

## 【パラメータ】

W s ソケットデスクリプタ  
W level レベル  
W optnam オプション  
B \*optval オプション値  
W \*optlen オプションのバイト長

## 【リターン値】



= 0 正常  
< 0 エラー

## 【解説】

ソケットに関するオプションを取得する。

level およびオプション

IPPROTO\_IP :

IP\_OPTIONS optval, optlen で IP オプションを指定

IPPROTO\_TCP :

TCP\_NODELAY send 時にメッセージをただちに送信するか否かの状態取得

TCP\_MAXSEG 最大メッセージ長を取得

SOL\_SOCKET :

|              |                       |
|--------------|-----------------------|
| SO_DEBUG     | デバッグ情報の記録フラグを取得       |
| SO_REUSEADDR | ローカルアドレスの再使用状態を取得     |
| SO_KEEPALIVE | (未サポート)               |
| SO_DONTROUTE | 発信メッセージのためのバイパスルートの決定 |
| SO_BROADCAST | ブロードキャストメッセージの送信許可    |
| SO_LINGER    | データがある場合のクローズの延期      |
| SO_OOBINLINE | 帯域内の帯域外データの受信状態       |
| SO_SNDBUF    | 出力のためのバッファサイズを取得      |
| SO_RCVBUF    | 入力のためのバッファサイズを取得      |
| SO_ERROR     | ソケットのエラーの取得およびクリア     |
| SO_TYPE      | ソケットのタイプを取得           |

## 【エラーコード】

|               |                              |
|---------------|------------------------------|
| EX_ADR        | パラメータでアクセスが許されていない領域を参照している。 |
| EX_PAR        | パラメータが不正(ソケットデスクリプタの値が不正)    |
| EX_BADF       | ソケットが無効。                     |
| EX_FAULT      | 不正なオプション指定。                  |
| EX_NOPROTOOPT | 指定したレベルで認識されていないオプション。       |

## so\_setsockopt

ソケットオプションの設定

ERR so\_setsockopt (W s, W level, W optnam, B \*optval, W optlen)

## 【パラメータ】

W s ソケットデスクリプタ  
W level レベル  
W optnam オプション  
B \*optval オプション値  
W \*optlen オプションのバイト長

### 【リターン値】

= 0 正常  
< 0 エラー

### 【解説】

ソケットに関するオプションを設定する。  
レベルとオプションについては [so\\_getsockopt](#) を参照すること。  
SOL\_SOCKET レベルのオプションでは、各オプションがトグルされる。

IPPROTO\_IP :

IP\_OPTIONS optval, optlen で IP オプションを指定

IPPROTO\_TCP :

TCP\_NODELAY send 時にメッセージをただちに送信するよう指定

SOL\_SOCKET :

|              |                           |
|--------------|---------------------------|
| SO_DEBUG     | デバッグ情報の記録フラグをトグル          |
| SO_REUSEADDR | ローカルアドレスの再使用状態をトグル        |
| SO_KEEPAIVE  | (未サポート)                   |
| SO_DONTROUTE | 発信メッセージのためのバイパスルートの決定のトグル |
| SO_BROADCAST | ブロードキャストメッセージの送信許可をトグル    |
| SO_LINGER    | データがある場合のクローズの延期を設定       |
| SO_OOBINLINE | 帯域内の帯域外データの受信状態をトグル       |
| SO_SNDBUF    | 出力のためのバッファサイズを設定          |
| SO_RCVBUF    | 入力のためのバッファサイズを設定          |

### 【エラーコード】

|               |                              |
|---------------|------------------------------|
| EX_ADR        | パラメータでアクセスが許されていない領域を参照している。 |
| EX_PAR        | パラメータが不正(ソケットデスクリプタの値が不正)    |
| EX_BADF       | ソケットが無効。                     |
| EX_FAULT      | 不正なオプション指定。                  |
| EX_NOPROTOOPT | 指定したレベルで認識されていないオプション。       |

## so\_ioctl

デバイスの制御

## 【形式】

WERR so\_ioctl (W s, W request, W arg)

## 【パラメータ】

W s ソケットデスクリプタ  
W request 要求  
W arg 引数

## 【リターン値】

正常  
0 エラー

## 【解説】

ソケットに関して特殊な処理を行なう。

request  
FIONREAD :

ソケットの受信バッファにたまっているメッセージのバイト数を arg を W\* とみなしてそこに戻す。

SIOCATMARK :

ソケットが帯域外データを受信しているとき arg を W\* とみなしてそこに 1 を戻す。

## 【エラーコード】

EX\_ADR パラメータでアクセスが許されていない領域を参照している。  
EX\_PAR パラメータが不正(ソケットデスクリプタの値が不正)  
EX\_BADF ソケットが無効。

# so\_listen

ソケット接続のためのlisten

## 【形式】

ERR so\_listen (W s, W backlog)

## 【パラメータ】

W s ソケットデスクリプタ  
W backlog 最大接続要求待ちキュー数

## 【リターン値】

=0 正常  
<0 エラー

## 【解説】

指定したソケットを listen 状態にする。

## 【エラーコード】

EX\_PAR パラメータが不正(ソケットデスクリプタの値が不正)  
EX\_BADF ソケットが無効。

# so\_read

ソケットからの受信

## 【形式】

WERR so\_read (W s, B \*buf, W len)

## 【パラメータ】

W s ソケットデスクリプタ  
B \*buf 受信バッファ  
W len 受信バッファのバイト長  
u

## 【リターン値】

0 受信したバイト数  
<0 エラー

## 【解説】

ソケットからメッセージを受信する。  
so\_recv(s, buf, len, 0) と等しい。

## 【エラーコード】

EX\_ADR パラメータでアクセスが許されていない領域を参照している。  
EX\_PAR パラメータが不正(ソケットデスクリプタの値が不正)  
EX\_BADF ソケットが無効。

EX\_WOULDBLOCK ソケットが非ブロック化されていて、受信データがない。  
EX\_TIMEDOUT タイムアウト。  
EX\_CONNABORTED 接続がアボートされた。  
EX\_OPNOTSUP サポートされていない動作指定

## so\_recv

ソケットからの受信

### 【形式】

WERR so\_recv (W s, B \*buf, W len, W flags)

### 【パラメータ】

W s ソケットデスクリプタ  
B \*buf 受信バッファ  
W len 受信バッファのバイト長  
W flags 以下の OR を指定することができる  
MSG\_OOB 帯域外データを読み取る  
MSG\_PEEK ソケット上にあるデータを「のぞき見」する。  
データは廃棄されず、それ以降の受信処理では、  
同一データが戻る。

### 【リターン値】

0 受信したバイト数  
<0 エラー

### 【解説】

ソケットからメッセージを受信する。メッセージがないときは非ブロック状態の場合は EX\_WOULDBLOCK を戻し、それ以外では待ちに入る。

### 【エラーコード】

EX\_ADR パラメータでアクセスが許されていない領域を参照している。  
EX\_PAR パラメータが不正(ソケットデスクリプタの値が不正)  
EX\_BADF ソケットが無効。  
EX\_WOULDBLOCK ソケットが非ブロック化されていて、受信データがない。  
EX\_TIMEDOUT タイムアウト。  
EX\_CONNABORTED 接続がアボートされた。  
EX\_OPNOTSUP サポートされていない動作指定(flag 指定が無効)

# so\_recvfrom

ソケットからの受信

## 【形式】

WERR so\_recvfrom (W s, B \*buf, W len, W flags, SOCKADDR \*from, W \*flen)

## 【パラメータ】

|          |          |  |
|----------|----------|--|
| W        | s        | ソケットデスクリプタ   |
| B        | *buf     | 受信バッファ   |
| W        | len      | 受信バッファのバイト長  |
| W        | flags    | 以下の OR を指定することができる   |
|          | MSG_OOB  | 帯域外データを読み取る  |
|          | MSG_PEEK | ソケット上にあるデータを「のぞき見」する。<br>データは廃棄されず、それ以降の受信処理では、<br>同一データが戻る。 |
| SOCKADDR | *from    | アドレス   |
| W        | *flen    | アドレスのバイト長  |

## 【リターン値】

0 受信したバイト数  
<0 エラー

## 【解説】

ソケットからメッセージを受信する。メッセージがないときは非ブロック状態の場合は EX\_WOULDBLOCK を戻し、それ以外では待ちに入る。  
from が NULL でないときは、メッセージの発信元のアドレスが戻される。\*flen はアドレスのサイズを指定し、実行後は実際のアドレスの長さが戻される。

## 【エラーコード】

|                |                              |
|----------------|------------------------------|
| EX_ADR         | パラメータでアクセスが許されていない領域を参照している。 |
| EX_PAR         | パラメータが不正(ソケットデスクリプタの値が不正)    |
| EX_BADF        | ソケットが無効。                     |
| EX_WOULDBLOCK  | ソケットが非ブロック化されていて、受信データがない。   |
| EX_TIMEOUT     | タイムアウト。                      |
| EX_CONNABORTED | 接続がアボートされた。                  |
| EX_OPNOTSUP    | サポートされていない動作指定(flag 指定が無効)   |

# so\_recvmsg

ソケットからの受信

## 【形式】

WERR so\_recvmsg (W s, MSGHDR \*msg, W flags)

## 【パラメータ】

|        |          |  |
|--------|----------|--|
| W      | s        | ソケットデスクリプタ   |
| MSGHDR | *msg     | メッセージヘッダー  |
| W      | flags    | 以下の OR を指定することができる   |
|        | MSG_OOB  | 帯域外データを読み取る  |
|        | MSG_PEEK | ソケット上にあるデータを「のぞき見」する。<br>データは廃棄されず、それ以降の受信処理では、<br>同一データが戻る。 |

## 【リターン値】

0 受信したバイト数  
<0 エラー

## 【解説】

ソケットからメッセージを受信する。メッセージがないときは非ブロック状態の場合は EX\_WOULDBLOCK を戻し、それ以外では待ちに入る。

構造体 msg\_hdr で指定することができる。

|              |                   |                   |
|--------------|-------------------|-------------------|
| caddr_t      | msg_name;         | オプションのアドレス        |
| int          | msg_namelen;      | アドレスサイズ           |
| struct iovec | *msg_iov;         | 配列の分散/収集          |
| int          | msg_iovlen;       | msg_iov にある構成要素の数 |
| caddr_t      | msg_accrights;    | 送信または受信されたアクセス権   |
| int          | msg_accrightslen; |                   |

msg\_name, msg\_namelen に相手先アドレスを指定する。指定不要のときは msg\_name は NULL を指定する。

msg\_iov により一回の要求で分散したいいくつかの場所に読み込むことができる。

## 【エラーコード】

|         |                              |
|---------|------------------------------|
| EX_ADR  | パラメータでアクセスが許されていない領域を参照している。 |
| EX_PAR  | パラメータが不正(ソケットデスクリプタの値が不正)    |
| EX_BADF | ソケットが無効。                     |

|                |                            |
|----------------|----------------------------|
| EX_WOULDBLOCK  | ソケットが非ブロック化されていて、受信データがない。 |
| EX_TIMEOUT     | タイムアウト。                    |
| EX_CONNABORTED | 接続がアボートされた。                |
| EX_OPNOTSUP    | サポートされていない動作指定(flag 指定が無効) |

## so\_select

ソケット状態の検査

### 【形式】

```
WERR    so_select (W nfds, fd_set *rfd, fd_set *wfd, fd_set *efd,
                  TIMEVAL *tmout)
```

### 【パラメータ】

|         |        |                        |
|---------|--------|------------------------|
| W       | nfds   | 最大ソケットデスク립タ数           |
| fd_set  | *rfd   | 受信準備完了を調べるソケットデスク립タの集合 |
| fd_set  | *wfd   | 送信準備完了を調べるソケットデスク립タの集合 |
| fd_set  | *efd   | 例外発生を調べるソケットデスク립タの集合   |
| TIMEVAL | *tmout | タイムアウト指定               |

### 【リターン値】

- >0 準備ができているデスク립タ数
- =0 タイムアウトした。
- <0 エラー

### 【解説】

指定したソケットデスク립タ中で、 rfd, wfd, efd 中で準備ができているソケットがあるかどうか検査する。 rfd, wfd, efd の内容は、準備ができているかどうかの結果が戻される。

rfd, wfd, efd は、関連するソケットが一つもない場合は、それぞれ NULL を指定することができる。

tmout が NULL のときは無限に待つ。 tmout が NULL でない場合は、そこで指定された時間だけ待つ。

### 【エラーコード】

EX\_ADR          パラメータでアクセスが許されていない領域を参照している。



# so\_write

ソケットからのメッセージ送信

## 【形式】

WERR so\_write (W s, B \*buf, W len)

## 【パラメータ】

W s ソケットデスクリプタ  
B \*buf 送信データ  
W len 送信データのバイト長

## 【リターン値】

0 送信したバイト数  
<0 エラー

## 【解説】

ソケットの接続先にメッセージを送信する。so\_send(s, buf, len, 0) に等しい。

## 【エラーコード】

|                |                                |
|----------------|--------------------------------|
| EX_ADR         | パラメータでアクセスが許されていない領域を参照している。   |
| EX_PAR         | パラメータが不正(ソケットデスクリプタの値が不正)      |
| EX_BADF        | ソケットが無効。                       |
| EX_SHUTDOWN    | 送信は禁止されている。                    |
| EX_CONNABORTED | 接続がアボートされた。                    |
| EX_MSGSIZE     | 送信メッセージサイズへの細分化ができなかった。        |
| EX_NOBUFS      | 内部領域不足。                        |
| EX_WOULDBLOCK  | ソケットが非ブロック化されているが、送信により待ちにはいる。 |
| EX_TIMEOUT     | タイムアウト。                        |

# so\_send

ソケットからのメッセージ送信

## 【形式】

WERR so\_send (W s, B \*buf, W len, W flags)

## 【パラメータ】

|   |         |                    |
|---|---------|--------------------|
| W | s       | ソケットデスクリプタ         |
| B | *buf    | 送信データ              |
| W | len     | 送信データのバイト長         |
| W | flags   | 以下の OR を指定することができる |
|   | MSG_OOB | 帯域外データを送信する。       |

### 【リターン値】

0 送信したバイト数  
 <0 エラー

### 【リターン値】

### 【解説】

ソケットの接続先にメッセージを送信する。

### 【エラーコード】

|                |                                |
|----------------|--------------------------------|
| EX_ADR         | パラメータでアクセスが許されていない領域を参照している。   |
| EX_PAR         | パラメータが不正(ソケットデスクリプタの値が不正)      |
| EX_BADF        | ソケットが無効。                       |
| EX_SHUTDOWN    | 送信は禁止されている。                    |
| EX_CONNABORTED | 接続がアボートされた。                    |
| EX_MSGSIZE     | 送信メッセージサイズへの細分化ができなかった。        |
| EX_NOBUFS      | 内部領域不足。                        |
| EX_WOULDBLOCK  | ソケットが非ブロック化されているが、送信により待ちにはいる。 |
| EX_TIMEOUT     | タイムアウト。                        |

## so\_sendto

ソケットからのメッセージ送信

### 【形式】

WERR so\_sendto (W s, B \*buf, W len, W flags, SOCKADDR \*to, W tolen)

### 【パラメータ】

|          |         |                    |
|----------|---------|--------------------|
| W        | s       | ソケットデスクリプタ         |
| B        | *buf    | 送信データ              |
| W        | len     | 送信データのバイト長         |
| W        | flags   | 以下の OR を指定することができる |
|          | MSG_OOB | 帯域外データを送信する。       |
| SOCKADDR | *to     | 送信先アドレス            |

W            tolen            送信先アドレスのバイト長

### 【リターン値】

0 送信したバイト数  
<0 エラー

### 【解説】

相手先アドレスを指定して、メッセージを送信する。

### 【エラーコード】

|                |                                |
|----------------|--------------------------------|
| EX_ADR         | パラメータでアクセスが許されていない領域を参照している。   |
| EX_PAR         | パラメータが不正(ソケットデスクリプタの値が不正)      |
| EX_BADF        | ソケットが無効。                       |
| EX_SHUTDOWN    | 送信は禁止されている。                    |
| EX_CONNABORTED | 接続がアボートされた。                    |
| EX_MSGSIZE     | 送信メッセージサイズへの細分化ができなかった。        |
| EX_NOBUFS      | 内部領域不足。                        |
| EX_WOULDBLOCK  | ソケットが非ブロック化されているが、送信により待ちにはいる。 |
| EX_TIMEOUT     | タイムアウト。                        |
| EX_HOSTUNREACH | 指定された送信先が見つからない。               |
| EX_DESTADDRREQ | 送信先アドレスの指定が必要。                 |

## so\_sendmsg

ソケットからのメッセージ送信

### 【形式】

WERR      so\_sendmsg (W s, MSGHDR \*msg, W flags)

### 【パラメータ】

|        |         |                    |
|--------|---------|--------------------|
| W      | s       | ソケットデスクリプタ         |
| MSGHDR | *msg    | メッセージデータ           |
| W      | flags   | 以下の OR を指定することができる |
|        | MSG_OOB | 帯域外データを送信する。       |

### 【リターン値】

0 送信したバイト数  
<0 エラー

### 【解説】

相手先アドレスを指定して、メッセージを送信する。

### 【エラーコード】

|                |                                |
|----------------|--------------------------------|
| EX_ADR         | パラメータでアクセスが許されていない領域を参照している。   |
| EX_PAR         | パラメータが不正(ソケットデスクリプタの値が不正)      |
| EX_BADF        | ソケットが無効。                       |
| EX_SHUTDOWN    | 送信は禁止されている。                    |
| EX_CONNABORTED | 接続がアボートされた。                    |
| EX_MSGSIZE     | 送信メッセージサイズへの細分化がてきなかった。        |
| EX_NOBUFS      | 内部領域不足。                        |
| EX_WOULDBLOCK  | ソケットが非ブロック化されているが、送信により待ちにはいる。 |
| EX_TIMEOUT     | タイムアウト。                        |
| EX_HOSTUNREACH | 指定された送信先が見つからない。               |

## so\_shutdown

全二重接続の部分シャットダウン

### 【形式】

ERR so\_shutdown (W s, W how)

### 【パラメータ】

|   |     |             |
|---|-----|-------------|
| W | s   | ソケットデスクリプタ  |
| W | how | これ以降の送受信の指定 |
|   | 0   | 以降の受信の禁止    |
|   | 1   | 以降の送信の禁止    |
|   | 2   | 以降の送受信の禁止   |

### 【リターン値】

### 【解説】

ソケットと対応する全二重通信路のすべてまたは一部をシャットダウンする。

### 【エラーコード】

|           |                           |
|-----------|---------------------------|
| EX_NOBUFS |                           |
| EX_PAR    | パラメータが不正(ソケットデスクリプタの値が不正) |
| EX_BADF   | ソケットが無効。                  |

# so\_socket

通信用の終点の作成

## 【形式】

WERR so\_socket (W domain, W type, W protocol)

## 【パラメータ】

|   |          |   |
|---|----------|---|
| W | domain   | 通信を行なうドメイン<br>PF_INET<br>PF_UNSPEC (PF_INET として扱われる)                                  |
| W | type     | 通信のタイプ<br>SOCK_STREAM<br>SOCK_DGRAM   |
| W | protocol | プロトコル<br>TCP (SOCK_STREAM タイプの時)<br>UDP (SOCK_DGRAM タイプの時)<br>ICMP (SOCK_DGRAM タイプの時) |

## 【リターン値】

0 ソケットデスクリプタ  
<0 エラー

## 【解説】

ソケットを作成しそのデスクリプタを戻す。

## 【エラーコード】

|                   |                        |
|-------------------|------------------------|
| EX_NOBUFS         | 内部領域不足。                |
| EX_PROTONOSUPPORT | サポートされていないタイプまたはプロトコル。 |

# so\_gethostname

ホスト名の取得

ERR so\_gethostname (B \*name, W nlen)

## 【パラメータ】

B \*name ホスト名を格納する領域  
W nlen name のバイト長

### 【リターン値】

= 0 正常  
< 0 エラー

### 【解説】

現在のホスト名を戻す。

### 【エラーコード】

EX\_ADR パラメータでアクセスが許されていない領域を参照している。

so\_sethostname

ホスト名の設定

### 【形式】

ERR so\_sethostname (B \*name, W nlen)

### 【パラメータ】

B \*name ホスト名  
W nlen ホスト名のバイト長

### 【リターン値】

= 0 正常  
< 0 エラー

### 【解説】

現在のホスト名を指定されたものに変更する。ホスト名は最大 31文字。

### 【エラーコード】

EX\_ADR パラメータでアクセスが許されていない領域を参照している。

---

[この章の目次にもどる](#)  
[前頁:3.9 フォントマネージャにもどる](#)  
[次頁:3.11 印刷マネージャ仕様書にすすむ](#)

## 3.11 印刷マネージャ

### 3.11.1 印刷マネージャの機能

印刷マネージャは印刷に関連する処理を統一的に扱うための外殻であり、印刷に関するアプリケーションからのプログラミングインタフェース、およびユーザインタフェースを統一することを目的としている。

印刷マネージャでは、TAD データを印刷用紙上にレイアウトして、指定されたプリンタに印刷する機能を備えているため、アプリケーションは、印刷する TAD データを用意するだけで印刷機能を実現することができるため、印刷に関する処理を大幅に軽減することができる。

印刷に関する全体の構成を示す。

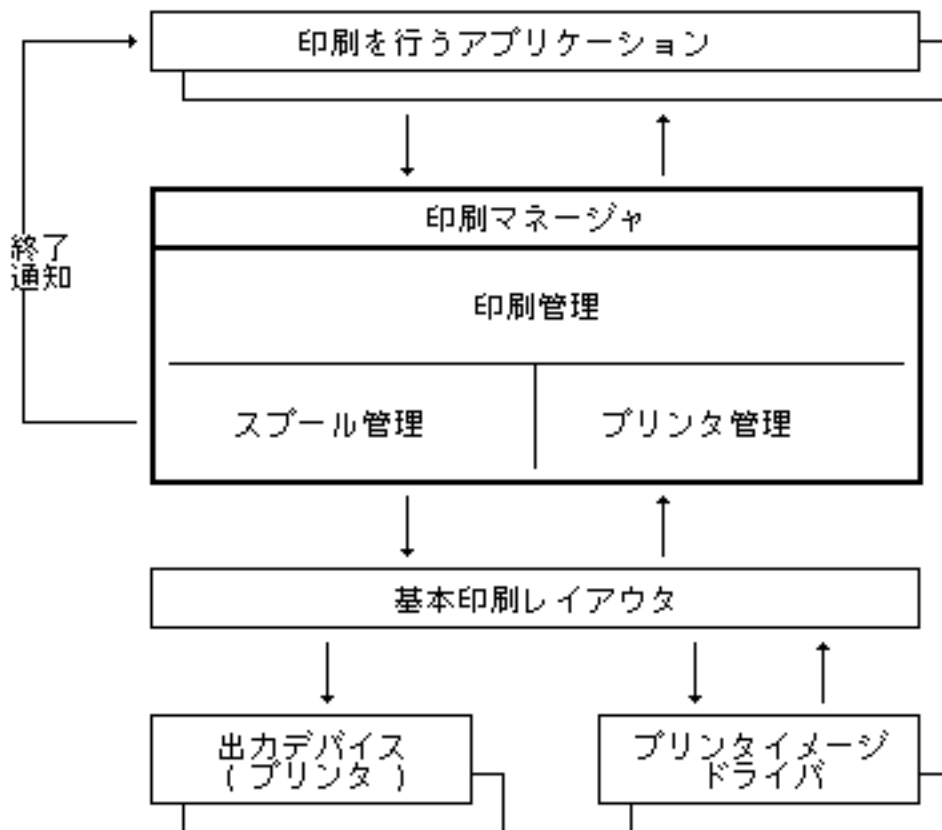


図 142 : 印刷の全体構成

印刷マネージャは、大きく以下の3つの機能を備えている。

印刷管理：

印刷管理では、印刷に関連する各種のパラメータの表示設定処理、TAD データをレイアウトして実際に印刷する処理、レイアウト結果を画面に表示するプレビュー処理などの、印刷に直接関連する各種の処理を行う。

スプール管理：

一般に印刷には時間がかかるため、印刷終了までアプリケーションを待たせることはできない。そのため、印刷要求をプリンタごとの待ち列に登録した時点で印刷処理を一旦終了させ、バックグラウンド処理により待ち列に入っている印刷要求を順番に処理していくスプール方式をとる。

スプール管理では、印刷要求の登録、削除、実行、印刷終了の通知、登録状態の取り出しなどのスプールに関する処理を行う。

プリンタ管理：

印刷を行うプリンタはあらかじめシステムに登録しておく必要がある。

プリンタ管理では、プリンタの登録、削除、各種の情報の設定や取り出しなどの処理を行う。

基本印刷レイアウトは、印刷管理を処理を実際に行うためのシステムアプリケーションであり、印刷マネージャから起動される。

プリンタイメージドライバは、プリンタ機種に依存した各種の情報の提供や、プリンタへ出力する印刷データの生成を行うシステムアプリケーションであり、基本印刷レイアウトから起動される。

基本印刷レイアウト、およびプリンタイメージドライバの処理、構成やインタフェースの詳細に関してはインプリメントに依存する。

## 3.11.2 印刷管理

### 3.11.2.1 レイアウト用紙と印刷用紙

TAD データをレイアウトするために想定されている仮想的な用紙をレイアウト用紙と呼び、TAD データ内の用紙指定付せんにより定義される。

印刷用紙とは、実際に印刷を行うために印刷時に指定する用紙であり、レイアウト用紙と一致することが普通であるが、レイアウト用紙とは異なる印刷用紙に印刷することもできる。このとき、拡大/縮小を行ったり、印刷位置をずらしたりすることができる。

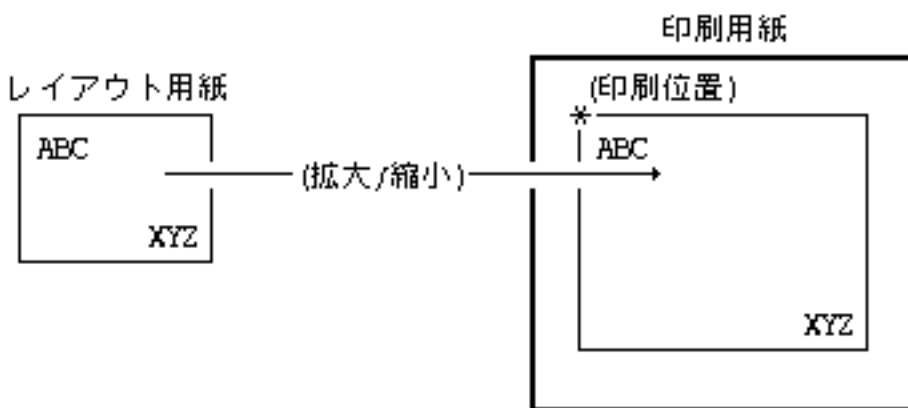


図 143：レイアウト用紙と印刷用紙

### 3.11.2.2 印刷管理の機能



印刷管理として、以下の機能を持つ。

プリンタ選択：

システムに登録されているプリンタの一覧をユーザに示し、対象とするプリンタを選択させる機能。

印刷パラメータ設定：

印刷するために必要となる印刷用紙などの各種のパラメータをユーザに示し、設定させる機能。

印刷：

選択したプリンタに対して、設定した各種の印刷パラメータにしたがって実際にスプール方式によりプリンタに印刷する機能。

実際の印刷終了後に通知を行うかどうか、および、対象のファイルを削除するかどうかの指定を行うことができる。

印刷するデータは、TAD データが基本であるが、印刷データも指定することができる。ここで、印刷データとは、プリンタにそのまま出力することにより印刷を行うことができるプリンタへのコマンド列データで、その内容はプリンタの機種に依存する。

印刷データを実際のプリンタに出力せずに、ファイルとして作成する機能も持つ。この場合はスプール方式は使用されない。

画面表示 (プレビュー)：

実際の印刷は行わずに、画面上に印刷結果を表示を行なう機能。おもに、印刷結果を事前に確認するために使用される。

これらの機能の詳細およびユーザインタフェースに関してはインプリメントに依存する。

### 3.11.2.3 印刷パラメータ

印刷パラメータは、印刷するための必要となる共通的なパラメータであり、以下にその内容の詳細を示す。

```
typedef struct {
    SIZE    size;          /* 用紙サイズ */
    PNT     offset;       /* 印刷位置指定 */
    UH      ratio;        /* 印刷倍率 */
    UH      spage;        /* 開始ページ指定 */
    UH      epage;        /* 終了ページ指定 */
    UH      ncopy;        /* 印刷部数 */
    UH      spec;         /* 印刷仕様 */
    UB      prnvobj;      /* 仮身印刷/展開 */
    UB      expvobj;      /* 仮身展開レベル */
    SIZE    layoutsz;     /* レイアウト用紙サイズ */
    RECT    ovlmgn;       /* オーバーレイマージン */
    UB      reserve[32]; /* 予約 */
} PR_PAR;
```

用紙サイズ：

印刷用紙の縦、横の大きさをミリメートル単位で指定する。

用紙サイズは用紙全体の大きさを設定するが、実際に印刷される領域は、プリンタに依存した上下左右の物理マージンを除いた印刷可能領域であるため、一般に用紙サイズより小さくなる。

size.h == 0 のときは、以下の特殊な指定が有効となる。

size.v = PR\_A4 : A4サイズ短辺方向行の書式  
PR\_A5 : A5サイズ短辺方向行の書式  
PR\_B4 : B4サイズ短辺方向行の書式  
PR\_B5 : B5サイズ短辺方向行の書式  
PR\_A4R : A4サイズ長辺方向行の書式  
PR\_A5R : A5サイズ長辺方向行の書式  
PR\_B4R : B4サイズ長辺方向行の書式  
PR\_B5R : B5サイズ長辺方向行の書式

印刷位置指定：

印刷用紙上で実際に印刷する上側、および左側のオフセットをミリメートル単位で指定する。

通常は0を指定するが、0以外を指定した場合は右下部分が印刷用紙をはみ出て印刷されないことがある。

印刷倍率：

印刷時の倍率をパーセント(%)で指定する。

レイアウト用紙にレイアウトした内容を指定した倍率で拡大/縮小した結果を、印刷用紙に印刷する。

レイアウト用紙の左上の位置を、印刷用紙の印刷位置指定で指定した位置に合わせて印刷するため、倍率の指定によっては、右下部分が印刷用紙をはみ出て印刷されなかったり、余白となることがある。

0は自動倍率を意味し、レイアウト用紙全体が印刷用紙全体に一致するように自動的に実際の倍率が計算される。レイアウト用紙と印刷用紙が同じときは100%の倍率となる。

通常は0を指定する。

開始/終了ページ指定：

印刷する開始ページ番号と終了ページ番号を指定する。

ページ番号は1から始まるが、開始ページに0を指定したときは、全ページを印刷することを意味する。

印刷部数：

印刷する部数を指定する。0を指定すると1とみなされる。通常は1を指定する。

印刷仕様：

印刷の品質に関する指定を行う。実際にどのように印刷されるかどうかは選択したブ

リントに依存する。

印刷品質：

|          |                 |
|----------|-----------------|
| PR_NORM  | 通常印刷            |
| PR_DRAFT | ドラフト印刷 (低品質、高速) |
| PR_FINE  | ファイン印刷 (高品質、低速) |

カラー / モノクロ印刷：

|          |       |
|----------|-------|
| PR_COLOR | カラー印刷 |
|----------|-------|

仮身印刷 / 展開：

仮身の印刷方法を指定する。

仮身を印刷するか無視するかの指定：

|            |          |
|------------|----------|
| PR_PRNVOBJ | 仮身を印刷する。 |
|------------|----------|

仮身を展開してその内容を印刷するか、仮身の形式で印刷するかの指定：

|            |              |
|------------|--------------|
| PR_EXPVOBJ | 仮身を展開して印刷する。 |
|------------|--------------|

仮身展開レベル：

仮身を展開して印刷する場合のネストの深さを指定する。

仮身印刷、および、仮身展開が指定されているときのみ有効となる。

仮身の属性として展開印刷が指定されている仮身のみが展開される。

レイアウト用紙サイズ：

レイアウト用紙の縦、横の大きさをミリメートル単位で指定する。

layouts.h == 0 のときは、layouts.v により、用紙サイズと同様の特殊な指定が有効となる。ただし、layouts.v == 0 のときは、レイアウト用紙サイズは未定義となり、用紙サイズと同じとみなされる。

通常は、TAD データ内の用紙指定付せんで定義されているレイアウト用紙と同じ大きさの指定を行う。

オーバーレイマージン：

レイアウト用紙上の上下左右のオーバーレイマージンの大きさをミリメートル単位で指定する。

通常は、TAD データ内の用紙指定付せんで定義されているオーバーレイマージンと同じ大きさの指定を行う。

### 3.11.3 スプール管理

#### 3.11.3.1 スプール管理の機能

スプール管理では、システムに登録されたプリンタごとにスプール処理のための待ち列を管理し、以下の機能を行う。この待ち列のことをスプールキューと呼ぶ。

印刷登録：

印刷の要求を選択したプリンタのプールキューに登録する。  
プリンタが印刷中でないときは、印刷処理を開始する。

印刷登録の削除：

プールキューに登録した印刷要求の登録を削除する。  
印刷中のときは、印刷処理を中断する。

印刷登録の状態取りだし：

プールキューに登録した印刷要求の現在の状態を取り出す。

印刷処理：

プールキューの先頭に登録されている、印刷要求を実際に処理して印刷を行う。

印刷を終了した時点で、以下の処理を行う。

- 通知指定があるとき、アプリケーションに対して終了通知を行う。
- 削除指定があるとき、対象ファイルを削除する。
- スプールキューの先頭から取り除き、次の要求があれば、その要求に対する印刷処理を開始する。

### 3.11.3.2 印刷登録状態

プールキューから取り出せる印刷登録状態の詳細を以下に示す。

```
typedef struct {
    W    prid;        /* 印刷登録 ID */
    W    pid;        /* 登録プロセス ID */
    UW   time;       /* 登録日時 */
    LINK lnk;       /* 登録ファイル */
    UH   mode;       /* 印刷モード */
    UH   stat;       /* 出力状態 */
    UH   rempage;    /* 残りページ数 */
    UH   remcopy;    /* 残り部数 */
} PR_STAT;
```

印刷登録 ID：

印刷登録を行ったときに割り当てられる内部的な ID を示す。  
印刷登録 ID を使用して、印刷登録の状態を取り出したり、登録を削除することができる。

登録プロセス ID：

印刷登録を行ったプロセスのプロセス ID を示す。

登録日時：

印刷登録したシステム時間を示す。

登録ファイル：

印刷登録の対象のファイルを示す。

印刷モード：

登録時の印刷モードであり、以下の通りである。

削除モード：

|          |                  |
|----------|------------------|
| PR_NODEL | 印刷終了後ファイルを削除しない。 |
| PR_DEL   | 印刷終了後ファイルを削除する。  |

通知モード：

|          |                    |
|----------|--------------------|
| PR_NOMSG | 印刷終了は通知しない。        |
| PR_MSG   | 印刷終了をメッセージにより通知する。 |

出力状態：

以下のいずれかの状態を示す。

|           |      |
|-----------|------|
| PR_WAIT   | 出力待ち |
| PR_ACTIVE | 出力中  |
| PR_DELETE | 削除中  |

残りページ数：

出力状態が出力中のとき、残りのページ数を示す。  
出力中でないときは、意味を持たない。

残り部数：

出力状態が出力中のとき、残りの部数を示す。  
出力中でないときは、意味を持たない。

### 3.11.3.3 印刷終了メッセージ

登録された印刷モードで、印刷終了の通知を指定されている場合、実際の印刷が終了した時点で、印刷登録を行ったアプリケーションプロセスに対して以下のメッセージが送信される。

```
#define MS_PRINT      MS_MNG5      /* 印刷メッセージタイプ */

typedef struct {
    W    type;          /* = MS_PRINT */
    W    size;         /* メッセージ本体サイズ (= 8) */
    W    prid;         /* 印刷登録 ID */
    W    code;         /* 終了コード(0: 正常) */
                          /* PR_CANCEL: 取り消された */
} PR_MESG;
```

正常に印刷が終了した場合は、終了コードに0が設定される。印刷が終了する前に取り消された場合は、終了コードにPR\_CANCELが設定される。

## 3.11.4 プリンタ管理

### 3.11.4.1 プリンタ管理の機能

システムには複数のプリンタを登録することができる。プリンタ管理では、プリンタに関する以下の機能を持つ。

プリンター一覧の取り出し：

システムに登録されているプリンタの一覧を取り出す。

プリンタの構成情報の取り出し：

プリンタに関連する構成情報を取り出す。

プリンタの構成情報の設定 (登録、削除)：

プリンタに関連する構成情報を設定(変更)、追加(登録)、削除する。

### 3.11.4.2 プリンタ構成情報

システムに登録されたプリンタは以下に示す内容のプリンタ構成情報を持つ。

```
#define L_DEVPAR    88      /* 出力デバイスパラメータ長さ */
#define L_HWPAR     160     /* プリンタ固有パラメータ長さ */
```

```
typedef struct {
    TC  prname[L_PRNM];      /* プリンタ名 */
    TC  hwname[L_PRNM];      /* プリンタ機種名 */
    TC  devname[L_DEVNM];    /* 出力デバイス名 */
    TC  drvname[L_FNM];      /* ドライバファイル名 */
    VB  devpar[L_DEVPAR];    /* 出力デバイスパラメータ */
    VB  hwpar[L_HWPAR];      /* プリンタ固有パラメータ */
} PR_CONF;
```

プリンタ名：

プリンタに付ける任意の名前。

この名前によりプリンタの区別や選択が行われる。

プリンタ機種名：

プリンタの機種を示す名前。

機種名はシステムに登録されているプリンタイメージドライバにより定義される。

出力デバイス名：

プリンタが実際に接続されている出力デバイスを示す名前。

システムに登録されている実際のデバイス名の他に、以下の特殊な指定がある。

|      |          |
|------|----------|
| file | ファイルへの出力 |
| none | 出力なし     |

ドライバファイル名：

プリンタの機種に対応したプリンタイメージドライバのファイル名。

このドライバにより、プリンタの機種に対応した印刷データを作成する。

出力デバイスパラメータ：

出力デバイス名に関連するパラメータ。  
出力デバイスの種別ごとに異なる構造となる。

プリンタ固有パラメータ：

プリンタに関連するパラメータ。  
プリンタの機種、およびインプリメントに依存した構造となる。プリンタイメージドライバにより操作される。

### 3.11.5 データ / 定数の定義

印刷パラメータ

```
typedef struct {
    SIZE    size;           /* 用紙サイズ(mm) */
    PNT     offset;        /* 印刷位置指定(mm) */
    UH      ratio;         /* 用紙倍率(%) (0:自動) */
    UH      spage;         /* 開始ページ指定(0:全体) */
    UH      epage;         /* 終了ページ指定(0:全体) */
    UH      ncopy;         /* 印刷部数 (0 は 1) */
    UH      spec;          /* 印刷仕様 */
    UB      prnvobj;       /* 仮身印刷/展開の有無 */
    UB      expvobj;       /* 仮身展開レベル */
    SIZE    layoutsz;      /* レイアウト用紙サイズ(mm) */
    RECT    ovlmgn;        /* オーバーレイマージン(mm) */
    UB      reserve[32];   /* 予約 */
} PR_PAR;
```

仮身印刷/展開の有無(prnvobj)

```
#define PR_PRNVOBJ 0x01      /* 仮身印刷 */
#define PR_EXPVBJ 0x02      /* 仮身展開 */
```

印刷仕様(spec)

```
#define PR_NORM 0x00        /* 通常印刷 */
#define PR_DRAFT 0x01       /* ドラフト印刷 */
#define PR_FINE 0x02        /* ファイン印刷 */
#define PR_COLOR 0x10       /* カラー印刷 */
```

標準用紙サイズ (size.h = 0 のときの size.v)

```
#define PR_A4 0x00a4        /* A4 縦方向 */
#define PR_A5 0x00a5        /* A5 縦方向 */
```

```
#define PR_B4      0x00b4      /* B4 縦方向 */
#define PR_B5      0x00b5      /* B5 縦方向 */
#define PR_A4R     0x80a4      /* A4 横方向 */
#define PR_A5R     0x80a5      /* A5 横方向 */
#define PR_B4R     0x80b4      /* B4 横方向 */
#define PR_B5R     0x80b5      /* B5 横方向 */
```

### 印刷パラメータ設定リターン値

```
#define PR_CANCEL  0x01      /* 設定取り消し */
#define PR_SET      0x02      /* 設定 */
#define PR_PREVIEW 0x03      /* 画面表示 */
#define PR_PRINT    0x04      /* 印刷 */
#define PR_FILEOUT  0x05      /* ファイル出力 */
```

### 印刷モード

```
#define PR_NOMSG   0x00      /* 印刷終了メッセージなし */
#define PR_MSG      0x01      /* 印刷終了メッセージあり */
#define PR_NODEL   0x00      /* 登録ファイルの削除なし */
#define PR_DEL      0x02      /* 登録ファイルの削除あり */
```

### 印刷登録状態

```
typedef struct {
    W    prid;      /* 印刷登録 ID */
    W    pid;      /* 登録プロセス ID */
    UW   time;     /* 登録日時 */
    LINK lnk;      /* 登録ファイル */
    UH   mode;     /* 印刷モード */
    UH   stat;     /* 出力状態 */
    UH   rempage;  /* 残りページ数 */
    UH   remcopy;  /* 残り部数 */
} PR_STAT;
```

### 出力状態(stat)

```
#define PR_WAIT    0      /* 出力状態: 出力待ち */
#define PR_ACTIVE  1      /* 出力状態: 出力中 */
#define PR_DELETE  2      /* 出力状態: 削除中 */
```

### 印刷終了通知メッセージ

```
#define MS_PRINT    MS_MNG5    /* 印刷メッセージタイプ */
```



```

typedef struct {
    W    type;           /* = MS_PRINT */
    W    size;          /* メッセージ本体サイズ (= 8) */
    W    prid;          /* 印刷登録 ID */
    W    code;          /* 終了コード(0: 正常) */
                          /* PR_CANCEL: 取り消された */
} PR_MSG;

```

### プリンタ情報

```

#define L_PRNM      20      /* プリンタ名の長さ */

```

```

typedef struct {
    TC  prname[L_PRNM];    /* プリンタ名 */
    TC  hwname[L_PRNM];   /* プリンタ機種名 */
    TC  devname[L_DEVNM]; /* 出力デバイス名 */
} PR_INFO;

```

### プリンタ構成情報

```

#define L_DEVPAR    88      /* 出力デバイスパラメータ長さ */
#define L_HWPAR     160     /* プリンタ固有パラメータ長さ */

```

```

typedef struct {
    TC  prname[L_PRNM];    /* プリンタ名 */
    TC  hwname[L_PRNM];   /* プリンタ機種名 */
    TC  devname[L_DEVNM]; /* 出力デバイス名 */
    TC  drvname[L_FNM];    /* ドライバファイル名 */
    VB  devpar[L_DEVPAR];  /* 出力デバイスパラメータ */
    VB  hwpar[L_HWPAR];    /* プリンタ固有パラメータ */
} PR_CONF;

```

## 3.11.6 印刷マネージャの関数

ここでは、印刷マネージャがサポートしている関数群について説明する。これらの関数は、外殻として提供される。

各関数は ERR 型または WERR 型の関数値をとり、何らかのエラーがあった場合は「負」のエラーコードが戻る。正常終了時には「0」または「正」の値が戻る。

各関数のエラーコードとしては、本章で示した以外にも、核や外殻でエラーが検出された場合は、そのエラーコードが直接戻る場合がある。

## rset\_par

印刷パラメータ設定

## 【形式】

WERR rset\_par(TC \*prname, PR\_PAR \*prpar, LINK \*fout)

## 【パラメータ】

TC \*prname プリンタ名 [入出力]  
PR\_PAR \*prpar 印刷パラメータ [入出力]  
LINK \*fout 出力ファイルへのリンク [出力]

## 【リターン値】

= PR\_CANCEL : 正常終了(設定を中止した)  
= PR\_SET : 正常終了(設定を行った)  
= PR\_PREVIEW : 正常終了(次に画面表示を行う)  
= PR\_PRINT : 正常終了(次に印刷を行う)  
= PR\_FILEOUT : 正常終了(次にファイル出力を行う)  
< 0 : エラー(エラーコード)

## 【解説】

prpar で指定した印刷パラメータを初期値として、印刷パラメータをユーザに表示し、設定させ、その結果を prpar に戻すとともに、選択されたプリンタ名を prname に戻す。

prname が空でなく、かつ登録されているプリンタ名のときは、プリンタ選択の初期値となる。そうでない場合は、最初に登録されているプリンタ名が初期値となる。

選択されたプリンタの出力デバイスがファイルのとき、または、ファイル出力が選択された場合は、出力ファイルが生成され、そのリンクが fout に戻される。生成された出力ファイルは、参照カウント 0 の空ファイルの状態となる。

関数値として、選択した操作を示す以下の値 ( $\geq 0$ ) が戻る。

PR\_CANCEL :

設定を中止した。  
prname, prpar の内容は更新されない。

PR\_SET :

設定を行った。  
prname, prpar の内容は更新されている。

PR\_PREVIEW :

設定を行い、次に画面表示を行う。  
更新された prname, prpar を使用して、次に rpvw\_fil() により画面表示を行う必要がある。

PR\_PRINT :

設定を行い、次に印刷を行う。

更新された prname, prpar を使用して、次に rprn\_fil() により印刷を行う必要がある。

PR\_FILEOUT :

設定を行い、次にファイル出力を行う。

更新された prname, prpar, fout を使用して、次に rout\_fil() によりファイル出力を行う必要がある。

### 【エラーコード】

EX\_ADR : アドレス(prname, prpar, fout)が不正。

EX\_NOEXS : プリンタは 1 つも存在しない。

## rprn\_fil

印刷の実行

### 【形式】

WERR rprn\_fil(TC \*prname, LINK \*lnk, PR\_PAR \*prpar, UW mode)

### 【パラメータ】

|        |         |                   |
|--------|---------|-------------------|
| TC     | *prname | プリンタ名 [入力]        |
| LINK   | *lnk    | 印刷するファイルのリンク [入力] |
| PR_PAR | *prpar  | 印刷パラメータ [入力]      |
| UW     | mode    | 印刷モード [入力]        |

### 【リターン値】

> 0 : 正常終了(印刷登録 ID)

< 0 : エラー(エラーコード)

### 【解説】

prname で指定したプリンタに対して、lnk で指定したファイルを prpar で指定した印刷パラメータにしたがって印刷を行う。

prpar == NULL の時は、ファイルの内容をプリンタに対する印刷データ列とみなし、ファイル内の全レコードの内容を順番にそのままプリンタへ出力する。

prpar != NULL の時は、ファイルの内容は TAD データとみなし、prpar で指定した印刷パラメータにしたがって、レイアウト処理を行い、生成した印刷データ列をプリンタへ出力する。

印刷を登録した時点で、実際に印刷される前にリターンし、印刷登録 ID をリターン値とし

て戻す。  
実際の印刷処理が終了するまでにファイルの内容を変更した時の印刷結果は保証されない。  
mode は以下を指定する。

mode : (PR\_NODEL || PR\_DEL) | (PR\_NOMSG || PR\_MSG)

PR\_NODEL :

印刷終了後ファイルを削除しない。

PR\_DEL :

印刷終了後ファイルを削除する。  
ただし、ファイルの参照カウントが0でない時は削除されない。

PR\_NOMSG :

印刷終了は通知しない。

PR\_MSG :

印刷終了をメッセージにより通知する。

### 【エラーコード】

EX\_ADR : アドレス(prname, lnk, prpar)が不正。  
EX\_NOEXS : 指定したプリンタは存在しない。  
その他 : ファイルアクセスエラー。

## rou\_t\_fil

ファイル出力の実行

### 【形式】

ERR rou\_t\_fil(TC \*prname, LINK \*lnk, PR\_PAR \*prpar, LINK \*fout)

### 【パラメータ】

|        |         |                  |
|--------|---------|------------------|
| TC     | *prname | プリンタ名 [入力]       |
| LINK   | *lnk    | 印刷するファイル [入力]    |
| PR_PAR | *prpar  | 印刷パラメータ [入力]     |
| LINK   | *fout   | 出力ファイルへのリンク [入力] |

### 【リターン値】

= 0 : 正常終了

<0 : エラー(エラーコード)

### 【解説】

prname で指定したプリンタに対して、Ink で指定した TAD データファイルを prpar で指定した印刷パラメータにしたがって、レイアウト処理を行い、生成した印刷データ列を fout で指定した出力ファイルに格納する。

処理を終了してからリターンする。

### 【エラーコード】

EX\_ADR : アドレス(prname, Ink, prpar, fout)が不正。  
EX\_NOEXS : 指定したプリンタは存在しない。  
その他 : ファイルアクセスエラー。

## rpvw\_fil

印刷の画面表示(プレビュー)

### 【形式】

ERR rpvw\_fil(TC \*prname, LINK \*Ink, PR\_PAR \*prpar)

### 【パラメータ】

TC \*prname プリンタ名 [入力]  
LINK \*Ink 画面表示するファイル [入力]  
PR\_PAR \*prpar 印刷パラメータ [入力]

### 【リターン値】

=0 : 正常終了  
<0 : エラー(エラーコード)

### 【解説】

prname で指定したプリンタに対して、Ink で指定した TAD データファイルを prpar で指定した印刷パラメータにしたがって、レイアウト処理を行った結果を画面表示(プレビュー)する。

画面表示を終了してからリターンする。

### 【エラーコード】

EX\_ADR : アドレス(prname, lnk, prpar)が不正。  
EX\_NOEXS : 指定したプリンタは存在しない。  
その他 : ファイルアクセスエラー。

## rcan\_ent

印刷登録の取り消し

### 【形式】

WERR rcan\_ent(TC \*prname, W prid)

### 【パラメータ】

TC \*prname プリンタ名 [入力]  
W prid 印刷登録 ID [入力]

### 【リターン値】

0 : 正常終了(印刷取り消し数)  
<0 : エラー(エラーコード)

### 【解説】

prname で指定したプリンタに対する印刷登録を取り消す。

取り消された印刷登録の印刷モードが PR\_DEL の場合、登録されたファイルは削除される。

prid == 0 のとき:

全ての印刷登録を取り消し、実際に取り消した登録数をリターン値として戻す。何も登録されていないときは0が戻る。

prname == NULL のときは、特別な動作として、登録されている全てのプリンタの全ての印刷登録を取り消す。

prid != 0 のとき:

prid で指定した印刷登録 ID の登録を取り消し、1をリターン値として戻す。

### 【エラーコード】

EX\_ADR : アドレス(prname)が不正。  
EX\_NOEXS : 指定したプリンタは存在しない。  
EX\_PAR : 印刷登録 ID は存在しない。

# rget\_ent

印刷登録状態の取り出し

## 【形式】

```
WERR    rget_ent(TC *prname, W prid, PR_STAT *stat, W cnt)
```

## 【パラメータ】

|         |         |                    |
|---------|---------|--------------------|
| TC      | *prname | プリンタ名 [入力]         |
| W       | prid    | 印刷登録 ID [入力]       |
| PR_STAT | *stat   | 印刷登録状態の取り出し領域 [出力] |
| W       | cnt     | 取り出し数 [入力]         |

## 【リターン値】

0 : 正常終了(印刷登録数)  
<0 : エラー(エラーコード)

## 【解説】

prname で指定したプリンタに対する印刷登録状態を取り出す。

cnt は stat の領域の数を示し、実際の登録数より少ない場合は、cnt 個のみ stat に取り出される。

cnt == 0 または stat == NULL のときは、取り出さずに登録数のみを返す。

prid == 0 のとき：

全ての登録状態を取り出し、実際の登録数をリターン値として返す。何も登録されていないときは、0 が返る。

prname == NULL, stat == NULL, cnt == 0 のときは、特別な動作として、登録されている全てのプリンタの印刷登録数の合計をリターン値として返す。

prid != 0 のとき：

prid で指定した印刷登録 ID の登録状態を取り出し、1 をリターン値として返す。

## 【エラーコード】

EX\_ADR : アドレス(prname, stat)が不正。  
EX\_NOEXS : 指定したプリンタは存在しない。  
EX\_PAR : 印刷登録 ID は存在しない。

# rlst\_prn

プリンター一覧の取り出し

## 【形式】

WERR     rlst\_prn(PR\_INFO \*info, W cnt)

## 【パラメータ】

PR\_INFO \*info    プリンタ情報の取り出し領域 [出力]  
W        cnt     取り出し数 [入力]

## 【リターン値】

0        : 正常終了(登録済みのプリンタ数)  
<0       : エラー(エラーコード)

## 【解説】

登録済みのプリンター一覧を info に取り出し、登録済みのプリンタ数をリターン値として戻す。

cnt は info の領域の数を示し、実際のプリンタ登録数より少ない場合は、cnt 個のみ info に取り出される。

cnt == 0 または、info == NULL の時は、取り出さずに、登録済みのプリンタ数を戻す。

## 【エラーコード】

EX\_ADR        : アドレス(info)が不正。

# rget\_cnf

プリンタ構成情報の取り出し

## 【形式】

WERR     rget\_cnf(TC \*prname, PR\_CONF \*cnf)

## 【パラメータ】



TC \*prname プリンタ名 [入力]  
PR\_CONF \*cnf プリンタ構成情報の取り出し領域 [出力]

### 【リターン値】

0 : 正常終了(プリンタ最大登録数)  
<0 : エラー(エラーコード)

### 【解説】

prname で指定したプリンタの構成情報を cnf に取り出し、プリンタの最大登録数をリターン値として戻す。

prname == NULL の時は、取り出さずに、プリンタの最大登録数を戻す。

### 【エラーコード】

EX\_ADR : アドレス(prname, cnf)が不正。  
EX\_NOEXS : 指定したプリンタは存在しない。

## rset\_cnf

プリンタ構成情報の設定/登録/削除

### 【形式】

ERR rset\_cnf(TC \*prname, PR\_CONF \*cnf)

### 【パラメータ】

TC \*prname プリンタ名 [入力]  
PR\_CONF \*cnf プリンタ構成情報 [入力]

### 【リターン値】

= 0 : 正常終了  
< 0 : エラー(エラーコード)

### 【解説】

prname で指定したプリンタの構成情報を cnf で指定した内容に設定する。

prname != NULL のとき :

cnf == NULL のとき (プリンタの登録削除) :

prname で指定したプリンタ構成情報を削除する。

praname で指定したプリンタに対して、印刷登録されているときはエラーとなる。

cnf == ((PR\_CONF)\*1) のとき (プリンタの登録位置を先頭に移動) :

prname で指定したプリンタ構成情報を先頭に移動する。

それ以外るとき (プリンタの登録内容の変更) :

prname で指定したプリンタ構成情報を cnf で指定した内容に変更する。

praname で指定したプリンタに対して、印刷登録されているときはエラーとなる。

prname == NULL のとき (プリンタの登録) :

cnf で指定したプリンタ構成情報を最後に追加する。

## 【エラーコード】

|          |                          |
|----------|--------------------------|
| EX_ADR   | : アドレス(prname, cnf)が不正。  |
| EX_LIMIT | : プリンタ登録数が制限を超えた。        |
| EX_EXS   | : 指定したプリンタに対して印刷登録されている。 |
| EX_NOEXS | : 指定したプリンタは存在しない。        |
| EX_PAR   | : パラメータが不正。              |
| その他      | : ファイルアクセスエラー。           |

---

[この章の目次にもどる](#)

[前頁:3.10 TCP/IPマネージャにもどる](#)