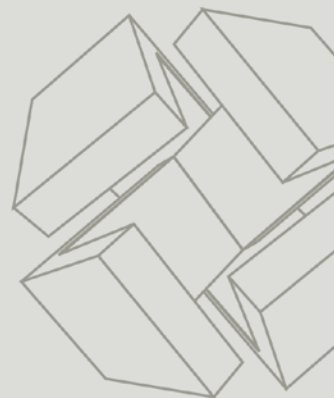




FUJITSU Manufacturing Industry Solution  
COLMINA 設計製造支援  
iCAD SX

# SDK ユーザーズガイド



---

## ご注意

本マニュアルは、“外国為替及び外国貿易管理法”に定める戦略物資関連技術が含まれています。  
したがって、本マニュアルを輸出する場合には、同法に基づく許可が必要とされます。  
なお、本マニュアルを廃棄する場合は、完全に消去してください。

# は じ め に

本書は、iCAD SX を使用して 2 次元を対象としたコマンドを開発する方法を説明しています。

iCAD SX を使用して初めてコマンドを開発する方でも、本書により、簡単にコマンド開発できるよう具体例をもとに、各作業項目について詳しく説明しています。

また本書は、iCAD SX を使用してコマンドを開発する方のために作られており、C 言語の文法に関する知識を有する方を前提としております。iCAD SX 基本コマンドを操作して、画面の遷移、コマンドの動きを理解してから、本書を読むことをおすすめします。

コマンド定義体、コマンドリスト、プロセスパラメタ、図形処理ライブラリ、プロセスリストの文法につきましては「SDK リファレンスガイド」を参照してください。

また、メニューのカスタマイズにつきましては「ユーザズガイド」をご覧ください。

開発に必要なソフトウェアなどは、『動作環境および入出力データ』（MAN フォルダ配下の environment\_and\_io\_data.pdf）をご覧ください。

1998年 4月	初 版
1998年11月	第 2 版
1999年11月	第 3 版
2000年 4月	第 4 版 (V3 L1)
2000年11月	第 5 版 (V3 L2)
2001年 6月	第 6 版 (V3 L3)
2006年 2月	第 7 版 (V5 L2)
2006年10月	第 8 版 (V5 L3)
2007年 7月	第 9 版 (V5 L4)
2008年12月	第10版 (V6 L1)
2009年12月	第11版 (V6 L2)
2010年12月	第12版 (V7 L1)
2012年 5月	第13版 (V7 L2)
2013年 7月	第14版 (V7 L3)
2015年 2月	第15版 (V7 L4)
2016年 2月	第16版 (V7 L5)
2017年 3月	第17版 (V7 L6)
2018年 9月	第18版 (V7 L7)
2020年 8月	第19版 (V8 L1)

# 目 次

<b>第 1 章 コマンド開発の前に</b> .....	1 - 1
ユーザコマンド開発 .....	1 - 2
コマンド操作例 .....	1 - 3
システム構造図 .....	1 - 5
コマンドの作成手順 .....	1 - 6
コマンドの組み込み図 .....	1 - 7
<b>第 2 章 コマンド作成例</b> .....	2 - 1
作成するコマンドの概要 .....	2 - 2
コマンドの設計 .....	2 - 3
コマンド定義体の作成 .....	2 - 6
コマンドリストの作成 .....	2 - 9
プロセスの作成 .....	2 - 10
プロセスリストの作成 .....	2 - 20
コンパイル、リンク .....	2 - 21
メッセージの登録 .....	2 - 24
コマンドの登録 .....	2 - 26
<b>第 3 章 コマンド定義体</b> .....	3 - 1
コマンド定義体の概要 .....	3 - 2
シンタックス・ダイアグラム .....	3 - 4
コマンド定義体の作成方法 .....	3 - 8
コマンド定義体の開始と終了 .....	3 - 8
詳細メニューの設定 .....	3 - 8
項目入力領域の設定 .....	3 - 10
データ入力 .....	3 - 11
GOの入力 .....	3 - 11
入力データタイプによる分岐 .....	3 - 11
繰り返しの指定 .....	3 - 13
プロセスの実行 .....	3 - 14
入力促進メッセージの表示 .....	3 - 14
コマンド定義体の呼び出し .....	3 - 14

グローバル領域への設定と参照 .....	3 - 1 5
値の設定 .....	3 - 1 5
値の参照 .....	3 - 1 5
特殊なオペレーションの指定 .....	3 - 1 6
注意事項 .....	3 - 1 8

<b>第4章 コマンドリスト .....</b>	<b>4 - 1</b>
コマンドリストの概要 .....	4 - 2

<b>第5章 プロセス .....</b>	<b>5 - 1</b>
プロセス作成手順 .....	5 - 2
データを受け取る方法(プロセスパラメタ) .....	5 - 3
図形処理ライブラリ(基本編)の利用方法 .....	5 - 5

<b>第6章 図形処理ライブラリ(応用編)の利用方法 .....</b>	<b>6 - 1</b>
要素の種類 .....	6 - 2
要素識別番号 .....	6 - 3
2次元要素 .....	6 - 5
グループ .....	6 - 9
グループの概要 .....	6 - 9
グループ作成方法 .....	6 - 1 1
実像部品 .....	6 - 1 3
実像部品の概要 .....	6 - 1 3
実像部品作成方法 .....	6 - 1 5
属性管理 .....	6 - 1 8
要素編集 .....	6 - 2 2
プリミティブ編集 .....	6 - 2 6
要素・プリミティブ読み込み .....	6 - 3 0
VS 管理 .....	6 - 3 4
ウインドウ制御 .....	6 - 3 5
復帰情報 .....	6 - 3 7

<b>第7章 図形処理ライブラリ(コマンド制御編)の利用方法</b>	7 - 1
UNDO／REDO	7 - 2
UNDO／REDO 機能概要	7 - 2
UNDO／REDO 作成方法	7 - 3
ラバーバンド	7 - 5
ラバーバンド機能概要	7 - 5
ラバーバンド作成方法	7 - 5
ドラッグング	7 - 6
ドラッグング機能概要	7 - 6
ドラッグング作成方法	7 - 6
ダイアログボックス	7 - 7
ダイアログボックスの概要	7 - 7
ダイアログボックス表示のしくみ	7 - 8
ダイアログボックス作成方法	7 - 9
コンパイル、リンク	7 - 10
ダイアログボックスのログ出力	7 - 17
ダイアログボックスのログ出力の概要	7 - 17
ダイアログボックスのログ出力を組み込んだプロセスの作成方法	7 - 18
 <b>第8章 プロセスリスト</b>	 8 - 1
プロセスリストの概要	8 - 2
 <b>第9章 コンパイル、リンク</b>	 9 - 1
コンパイル	9 - 2
コマンド定義体のコンパイル	9 - 2
コマンドリストのコンパイル	9 - 2
プロセスのコンパイル	9 - 3
プロセスリストのコンパイル	9 - 3
リソースのコンパイル	9 - 4
リンク	9 - 5
コマンドの組み込み図	9 - 7

第 10 章 メッセージ .....	10 - 1
メッセージの登録 .....	10 - 2
メッセージの出力方法 .....	10 - 5
第 11 章 バッチプログラムの作成 .....	11 - 1
バッチプログラムの作成手順 .....	11 - 2
使用可能なライブラリー一覧 .....	11 - 2
基本編 .....	11 - 2
応用編 .....	11 - 2
注意事項 .....	11 - 2
コンパイル、リンク .....	11 - 3
実行 .....	11 - 4
バッチプログラムの組み込み図 .....	11 - 5
第 12 章 トラブルシューティング .....	12 - 1
ユーザコマンドが正常に動作しない時の原因 .....	12 - 2
プロセスにミスがあった場合の原因追求方法 .....	12 - 3

---

# 第 1 章 コマンド開発の前に

ここでは、コマンドを初めて作る人のために、必要な知識について説明しています。

初めての方は、本章を熟読の上、次の章へお進みください。



---

## ユーザコマンド開発

コマンド開発者はシステムで用意されているコマンド開発機能を使い、ユーザコマンドを開発することができます。ユーザコマンドは基本コマンドと同様に、開発・操作することができます。

本章では、コマンドの操作手順、コマンドの処理の流れ、コマンドの作成手順について説明します。

## コマンド操作例

コマンド開発者が開発するコマンドは、基本コマンドと同じ方法で操作します。どのような操作手順で動くのか、例をあげて説明します。

次のような線を描く場合を例として示します。

＜操作例＞ 基準になる座標値から、指定の長さの線を引く

(1) コマンドメニュー「基本線」を指定します。

(2) 詳細メニュー「有限」を指定します。

画面が右図のようになります。

始点座標 始点要素という入力促進メッセージが出力されます。

	基本線		
	基本線		
	2点間	通過点	
	水平	垂直	
	有限	無限	
	折り返し		
角度		<input type="text" value="0.00"/>	
始点座標 始点要素			

(3) 始点座標を入力します。

(4) 線分長に「150」を入力します。

画面が右図のようになります。

終点座標 終点要素という入力促進メッセージが出力されます。

×	基本線		
	基本線		
	2点間	通過点	
	水平	垂直	
	有限	無限	
	折り返し		
線分長		<input type="text" value="150"/>	
終点座標 終点要素			

(5) 終点座標を入力します。

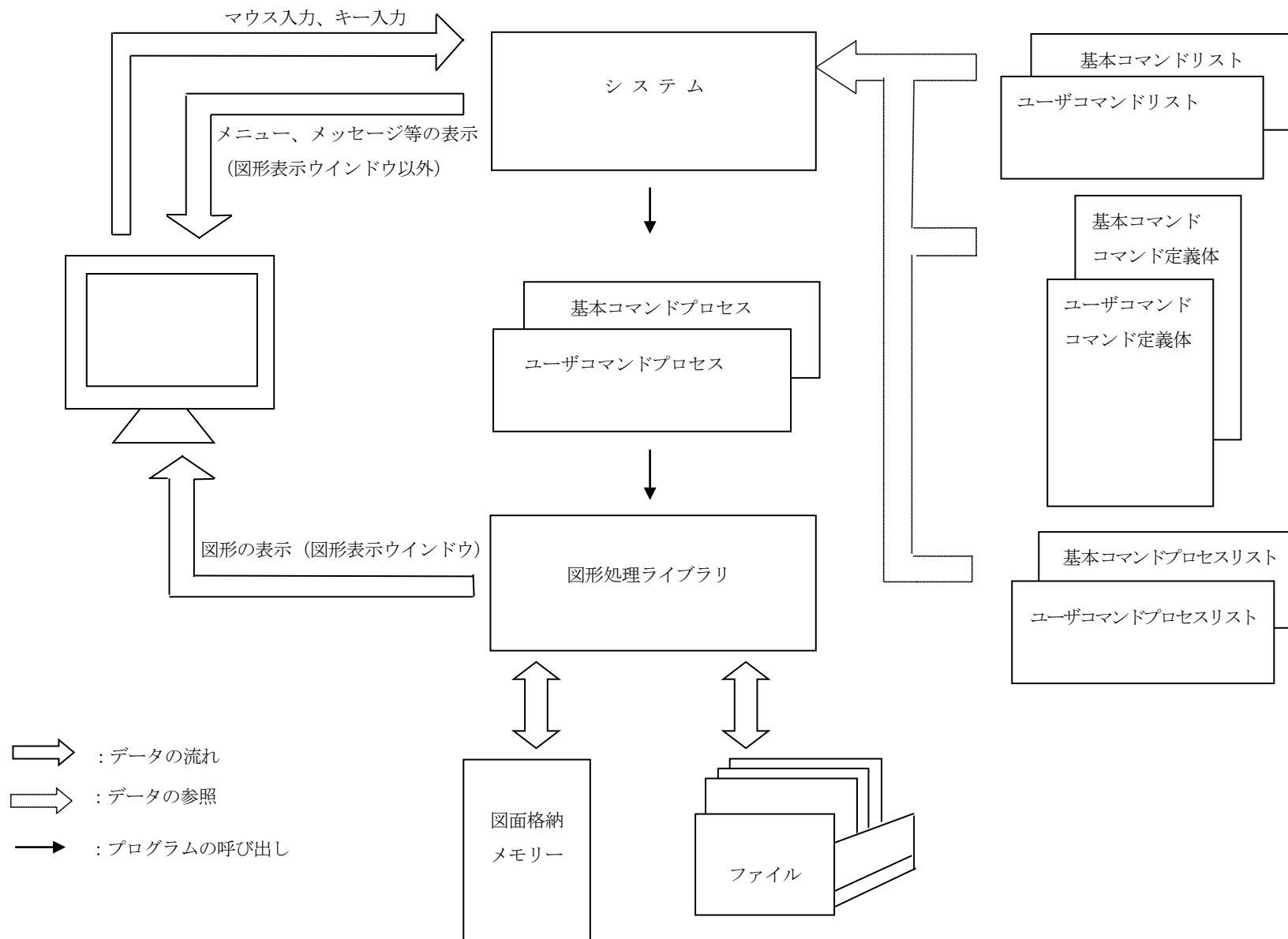
始点座標から「150」の長さの線が終点座標方向へ引かれます。

<div><div>×</div><div><div></div></div><div>×</div></div>		基本線	
		基本線	
		2点間	通過点
		水平	垂直
		有限	無限
		折り返し	
角度 0.00			
G0<消去> (始点座標 始点要素)			

このように、オペレータはまずコマンドを選び、次にコマンドの機能を選び、入力促進メッセージに従ってデータを入力します。

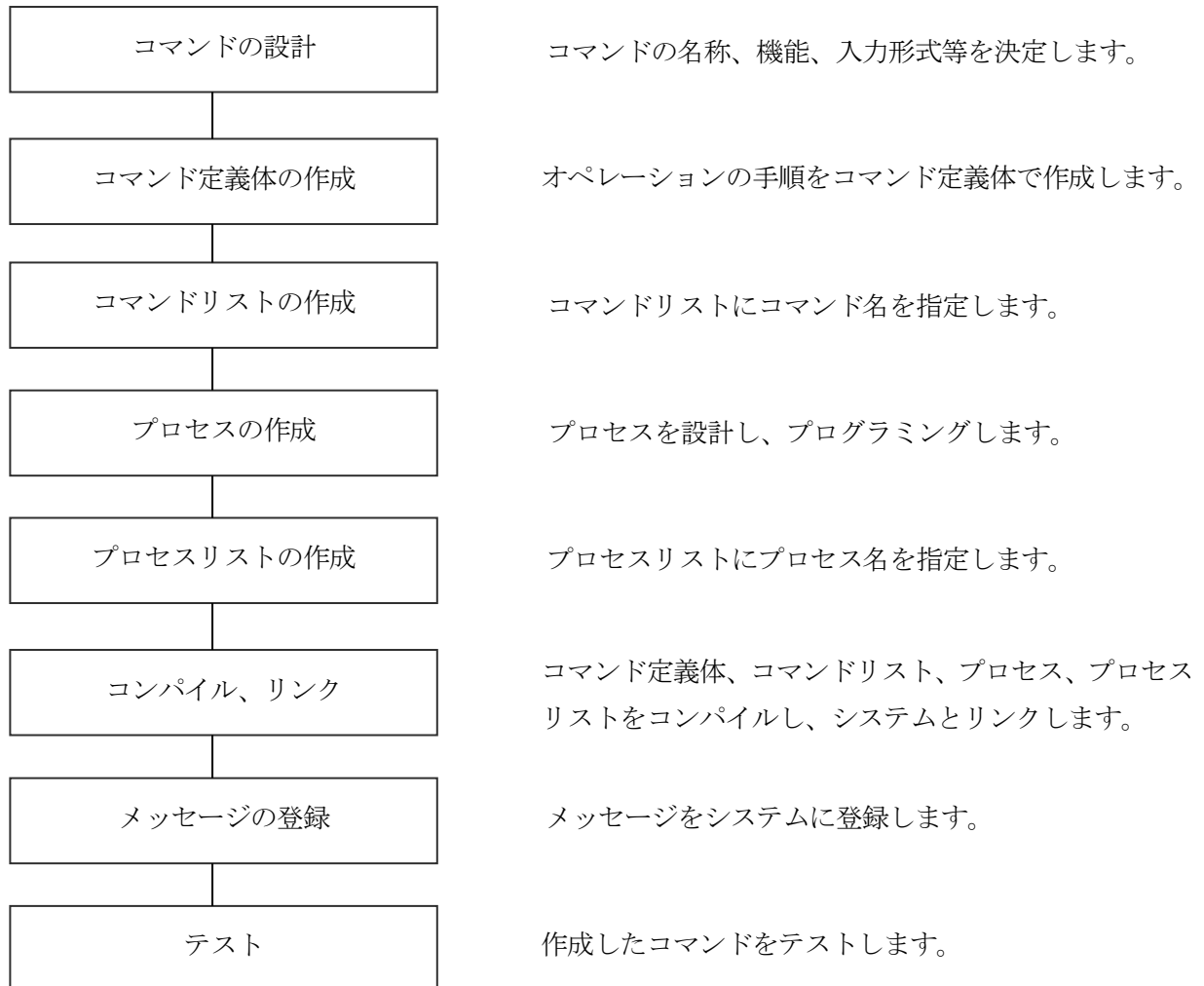
# システム構造図

システムがコマンドを解釈し、データを受け取り、図形を作成し、表示するまでの内部の処理の流れを示しています。



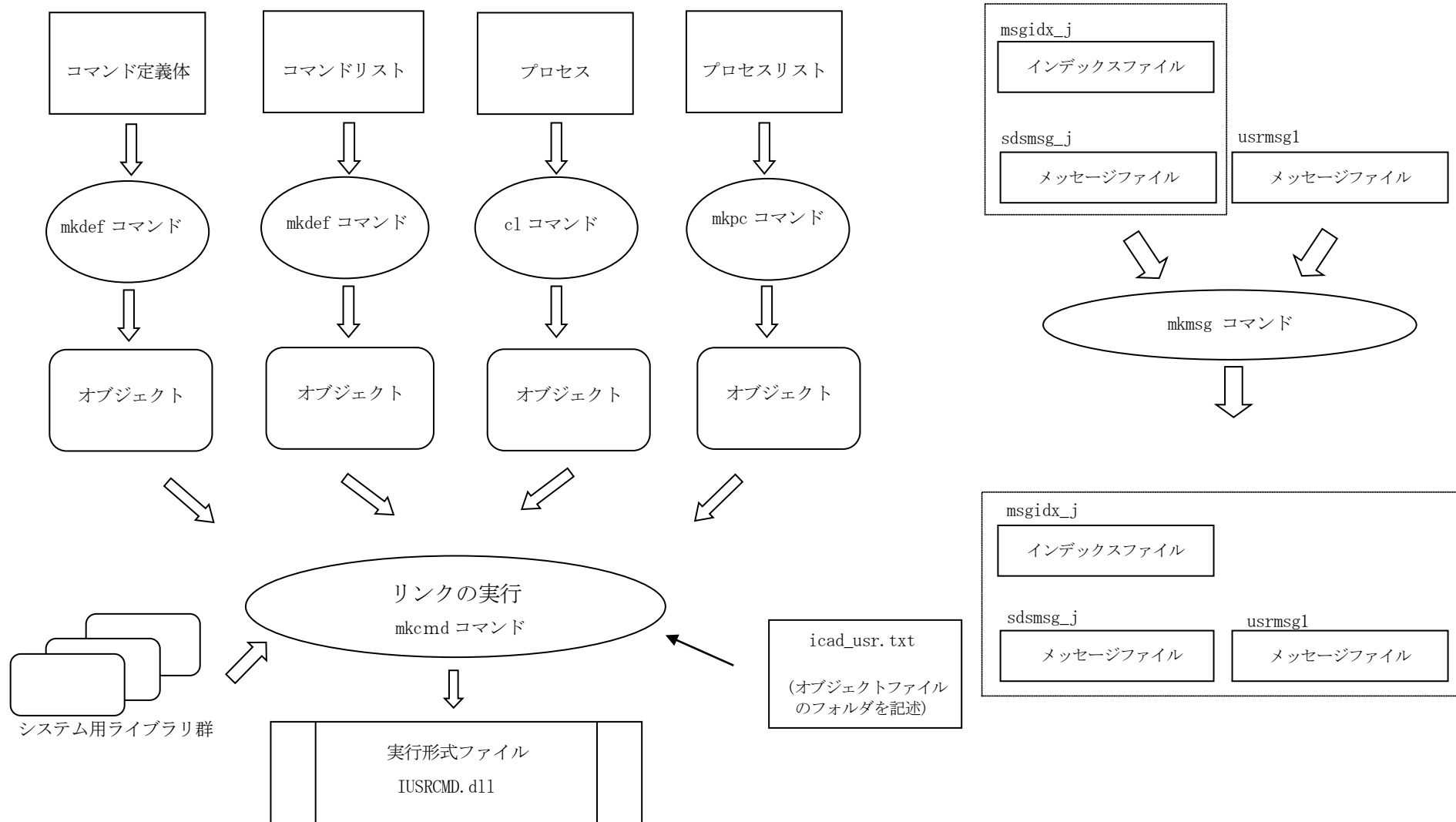
## コマンドの作成手順

コマンドの作成手順と、その処理について以下に示します。



## コマンドの組み込み図

コマンドをコンパイル、リンクしてシステムに組み込む手順を示しています。



---

## 第2章 コマンド作成例

ここでは、水平垂直線コマンドを具体例として、コマンド作成方法を作業順・項目別に説明しています。

初めての方は、本章を熟読の上、次の章へお進みください。

## 作成するコマンドの概要

まず最初に、コマンド名・機能・操作イメージの3点を考えます。

(1) コマンド名

水平垂直線 (HVLINE)

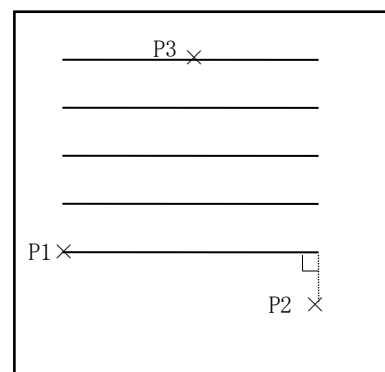
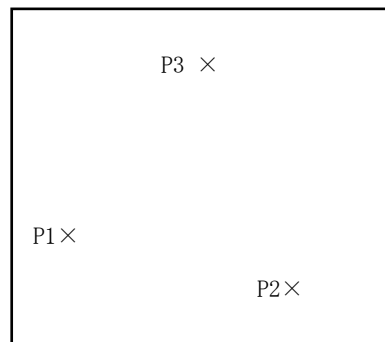
(2) 機能

水平線または垂直線を引きます。

(3) 操作イメージ (例：水平指定・作成本数=5)

P1 と P2 で第1線分を作成し、P3 まで等間隔に指定本数の平行線を作成します。

POS1—POS2—POS3



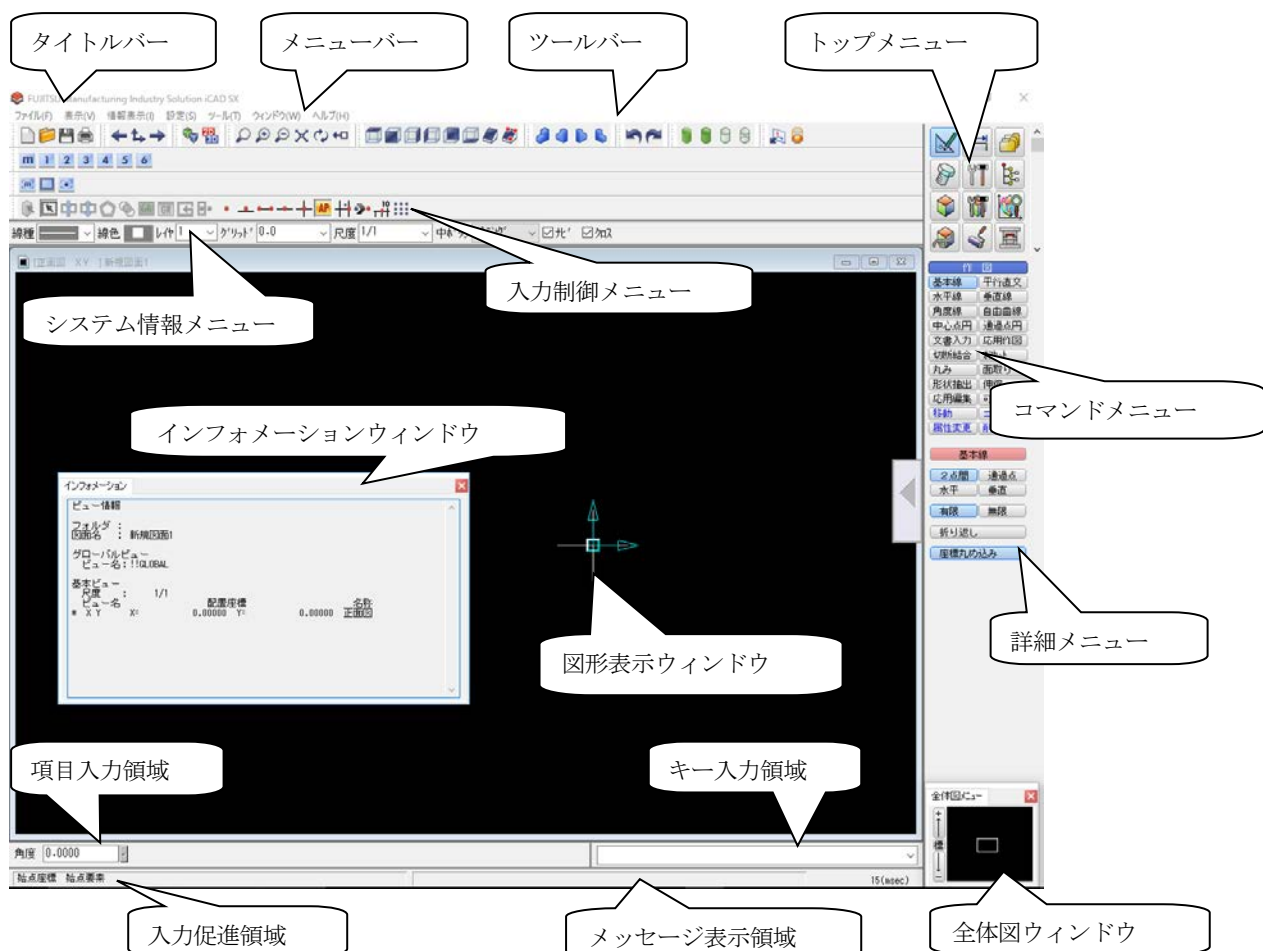


## コマンドの設計

コマンドを作成する時には、以下のことを決定しておく必要があります。

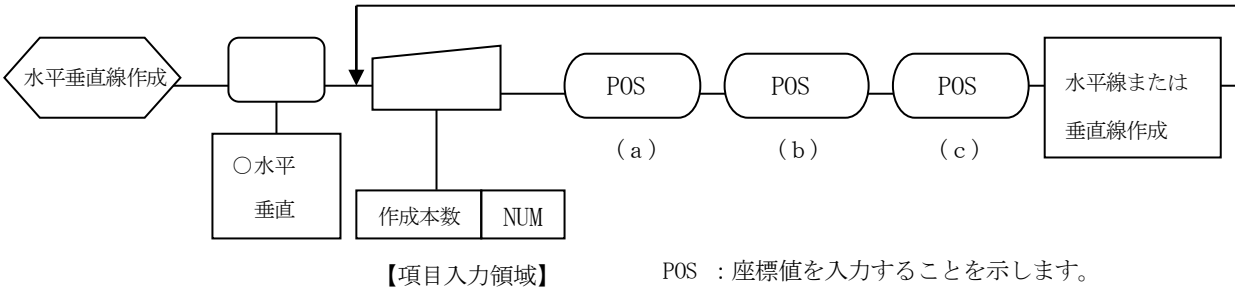
- (1) コマンド名
- (2) 機能概要
- (3) 操作イメージ
- (4) 入力促進メッセージ
- (5) 詳細機能
- (6) 詳細メニュー
- (7) 項目入力領域
- (8) 処理対象要素
- (9) 詳細機能オペレーション
- (10) 注意・制限事項

コマンドの設計中に出てくる画面の領域については、以下を参照してください。



- (1) コマンド名  
水平垂直線 (HVLINE)
- (2) 機能概要  
3 点を入力し、初めの 2 点で第 1 線分を決定し、第 3 点目まで等間隔に指定本数の平行線を作成します。作成された線分は、1 本で一つの線分とします。
- (3) 操作イメージ

【詳細メニュー】



- (4) 入力促進メッセージ
- (a) 「始点座標」
- (b) 「終点座標」
- (c) 「作画位置」
- (5) 詳細機能
- ① 1 点目及び 2 点目で作成した水平線に対して、平行線を作成します。
- ② 1 点目及び 2 点目で作成した垂直線に対して、平行線を作成します。

(6) 詳細メニュー

メニュー	キーワード	初期値	説明
水平	HOR	○	水平線を引きます。
垂直	VER		垂直線を引きます。

(7) 項目入力領域

項目	キーワード	初期値	説明
作成本数	NUM	2(整数)	線分を等間隔で何本引くかを指定します。 ( $2 \leq \text{NUM} \leq 999$ )

(8) 処理対象要素

2 次元でのみ実行可能です。

(9) 詳細機能オペレーション

ウインドウ	メニュー	オペレーション
	<div>水平垂直線</div> <div>水平</div> <div>垂直</div>	<p>詳細メニューから「水平」を選択します。</p> <p>項目入力領域に作成本数 5 を入力します。</p> <ul style="list-style-type: none"> <li>入力促進メッセージ表示「始点座標」 水平線の始点座標を指定します。(P1)</li> <li>入力促進メッセージ表示「終点座標」 水平線の終点座標を指定します。(P2)</li> </ul> <p>入力促進メッセージ表示「作画位置」 水平線を作成する方向と範囲を指定します。(P3)</p>
	<div>水平垂直線</div> <div>水平</div> <div>垂直</div>	<ul style="list-style-type: none"> <li>指定された水平線が作成されます。</li> </ul>

(10) 注意・制限事項

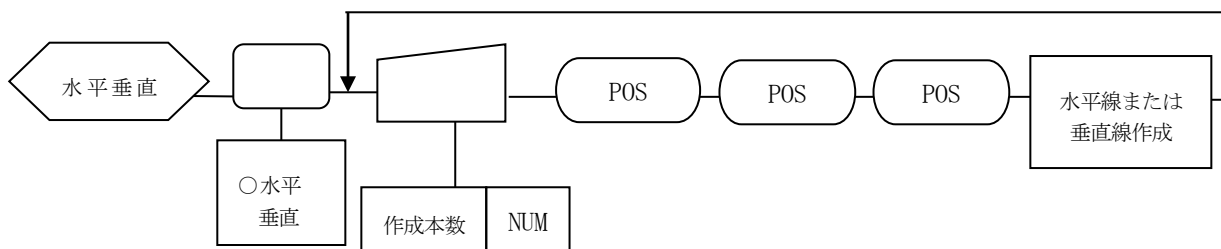
- ① 対象となる 3 点は同一ウインドウ上にあること。
- ② 1 点目と 2 点目は同一点であってはいけない。
- ③ 1、2、3 点が水平(垂直)線上にあってはいけない。
- ④ 等間隔に直線を作成する方向は 1、2 点で作成した直線に対し 3 点目の方向になる。

## コマンド定義体の作成

コマンド開発者は、コマンド定義体を使用して、コマンドの操作手順及びコマンド開発者の作成したプロセスの呼び出しを指示します。コマンド定義体の詳細は「第3章 コマンド定義体」で説明します。

コマンド定義体はシンタックス・ダイアグラムをもとにプログラミングします。シンタックス・ダイアグラムの詳細は「第3章 コマンド定義体 シンタックス・ダイアグラム」で説明します。

水平垂直線コマンドのコマンド定義体を以降に示します。



\*\*\*\*\*

\* コマンド名：H V L I N E      機能：水平線または垂直線の作成を行う

\*\*\*\*\*

\*

HVLINE    PROC    \_\_\_\_\_ ①

\*

      MENU      SUBA=1,PGCNTL=NO    \_\_\_\_\_ ②

          ITEM    (CMD1,1),\*,HIT=NO,MODE=OFF,COLOR=0, \_\_\_\_\_ ③

                  TEXT='水平垂直線',TXTYP=JEF

      MENUEND    \_\_\_\_\_ ④

\*

      MENU      SUBA=2,PGCNTL=NO

          GROUPS    DEFAULT=1,RESET=NO    \_\_\_\_\_ ⑤

              ITEM    (HOR,1),LBL00,COLOR=0,TXTYP=JEF,TEXT='水平' \_\_\_\_\_ ⑥

              ITEM    (VER,2),LBL00,COLOR=0,TXTYP=JEF,TEXT='垂直'

          GROUPE    \_\_\_\_\_ ⑦

      MENUEND

\*

LBL00    CONTINUE    \_\_\_\_\_ ⑧

      REPEAT    \_\_\_\_\_ ⑨

      STATUS    STHVL,WAIT=NO,RESET=NO    \_\_\_\_\_ ⑩

          STFLD    NUM,TYPE=I,FORM=I3,INIT=2,DISP=YES, \_\_\_\_\_ ⑪

		TXTYP=JEF, TEXT=' 作成本数'	
	STEND	_____	⑫
*			
	PROMPT	TXTYP=JEF, TEXT=' 始点座標' _____	⑬
	POS	HITRNG=2, DMDFR=NONE _____	⑭
	PROMPT	TXTYP=JEF, TEXT=' 終点座標'	
	POS	HITRNG=2, DMDFR=NONE	
	PROMPT	TXTYP=JEF, TEXT=' 作画位置'	
	POS	HITRNG=2, DMDFR=NONE	
*			
	CALL	3998, TPAR00, PARM=ALL _____	⑮
*			
	LEAVE	_____	⑯
	PROCEND	_____	⑰

#### 水平垂直線コマンドのコマンド定義体の説明

##### ① PROC 文

コマンド名を定義し、コマンド定義体の開始を示します。

##### ② MENU 文

詳細メニューの開始を示します。

##### ③ ITEM 文

詳細メニューの項目を指定します。詳細メニュー画面の 1 行目にコマンド名「水平垂直線」を表示することを指定します。

##### ④ MENUEND 文

詳細メニューの終了を示します。

##### ⑤ GROUPS 文

詳細メニューのグループの開始を示します。以降の ITEM 文で指定した項目は排他となります。

##### ⑥ ITEM 文

詳細メニューの項目（キーワード）を指定します。水平線の詳細メニューを「水平」、キーワードを「HOR」、垂直線の詳細メニューを「垂直」、キーワードを「VER」とします。初期値は「水平」とします。

##### ⑦ GROUPE 文

詳細メニューのグループの終了を示します。

##### ⑧ CONTINUE 文

分岐先のラベルを指定します。

⑨ REPEAT 文

オペレーションの繰り返しの開始を示します。

⑩ STATUS 文

項目入力領域の開始を示します。

⑪ STFLD 文

項目入力領域を指定します。項目名を「作成本数」、キーワードを「NUM」とし、データ属性を整数型の「I」とします。初期値は2とします。

⑫ STEND 文

項目入力領域の終了を示します。

⑬ PROMPT 文

入力促進メッセージの出力を指示します。（始点座標）

⑭ POS 文

座標値データの入力を指示します。（1点目）

⑮ CALL 文

プロセスを呼び出します。データ入力終了後に呼び出すプロセス名を TPAR00、プロセス番号を 3998 とします。

TPAR00 をプロセス名とした時、プロセスをプログラミングする場合は名前（関数名）を tpar00\_ と指定します。

⑯ LEAVE 文

オペレーションの繰り返しの終了を示します。

⑰ PROCEND 文

コマンド定義体の終了を示します。

## コマンドリストの作成

コマンドリストにコマンド名「水平垂直線 (HVLINE)」を指定します。  
コマンドリストの詳細は「第4章 コマンドリスト」で説明します。

@CMDL3	COMLIST	_____	①
	COMENTRY	HVLINE _____	②
	COMLEND	_____	③

### コマンドリストの説明

#### ① COMLIST 文

コマンドリストの開始を示します。

コマンドリストの名前を@CMDL3 とします。

#### ② COMENTRY 文

コマンド名を指定します。

コマンド定義体の PROC 文で指定したコマンド名「HVLINE」を記述します。

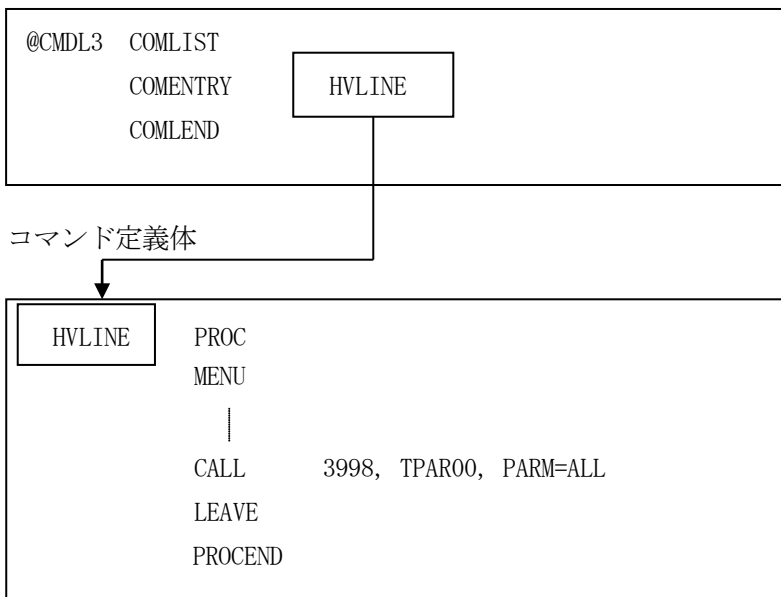
既存のコマンドリストにコマンド名を追加する場合は本文を付け加えてください。

#### ③ COMLEND 文

コマンドリストの終了を示します。

コマンドリストとコマンド定義体の結びつきを以下に示します。

### コマンドリスト



## プロセスの作成

システムでは、作図プログラム・編集プログラムなど、いろいろな種類のプログラムを図形処理ライブラリとして用意しています。これらのプログラムを利用し、入力データから図形を作成していきます。

dilin2\_（2次元の線要素を作成する）をプロセスで呼び出すことにより、図形表示ウインドウに線分が1本作成されます。水平垂直線コマンドのように何本も線分を描くためには、必要な本数だけ dilin2\_を呼び出します。

水平垂直線コマンドのプロセスの作成について以降に示します。

### (1) プログラム設計

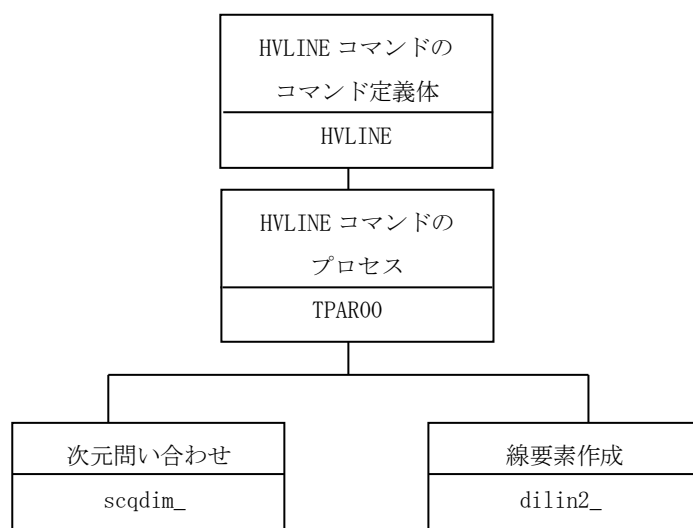
#### ① コマンド名

水平垂直線（HVLIN）

#### ② 機能概要

水平線または垂直線を作成する。

#### ③ プログラム構造図



### (2) プログラム説明

入力されたデータは、以下のようにプロセスパラメタに設定され、プロセスに渡されます。

プロセスパラメタの詳細は「第5章 プロセス」で説明します。

TPAR00 をプロセス名とした時、プロセスをプログラミングする場合は名前（関数名）を tpar00\_と指定します。

tpar00\_

3 点の座標から水平線または垂直線を作成します。

```
void tpar00_( long ircode[], long code[], double pos[], long ent[], double dir[], long valuei[],  
             double valuer[], unsigned char *text, long status[], long count[], long pose[], long  
             ente[], long dire[], long texte[], long global[])
```



## 引数（プロセスパラメタ）

プロセスの引数（プロセスパラメタ）の個数及び順序はコマンド定義体により決まり、以下の通りとなります。

out	ircode[3]	復帰情報
		[ 0 ] : 復帰コード
		[ 1 ] : メッセージ番号
		[ 2 ] : 詳細コード
in	code[2]	識別番号
		[ 0 ] : 未使用
		[ 1 ] : 詳細メニュー識別番号
		1 = HOR（水平）
		2 = VER（垂直）
in	pos[9]	座標値
		[ 0 ] : X1 座標
		[ 1 ] : Y1 座標
		[ 2 ] : 未使用
		[ 3 ] : X2 座標
		[ 4 ] : Y2 座標
		[ 5 ] : 未使用
		[ 6 ] : X3 座標
		[ 7 ] : Y3 座標
		[ 8 ] : 未使用
in	ent[]	未使用
in	dir[]	未使用
in	valuei[]	未使用
in	valuer[]	未使用
in	text[]	未使用
in	status[1]	項目入力領域情報
		[ 0 ] : 作成本数
in	count[]	未使用
in	pose[]	未使用
in	ente[]	未使用
in	dire[]	未使用
in	texte[]	未使用
in	global[]	未使用

## 注意事項

2次元でのみ動作します。

作成本数は2～999の範囲で指定します。

(3) 外部参照プログラム

- ① scqdim\_      次元を問い合わせます。
- ② dilin2\_      線要素を作成します。

**scqdim\_**

現在の検索対象の次元を得ます。

```
void scqdim_(long *odim)
```

引数

out *odim	現在の検索対象の次元が以下のいずれかで返されます。
2	: 2次元の要素のみを検索対象とします。
3	: 2次元以外の要素を検索対象とします。

**dilin2\_**

線要素を作成します。

```
void dilin2_(long *type, long *entid, double pmdata[5], long length[4], long  
              *pesadr, long ircode[2])
```

引数

in *type	作成する要素のタイプ番号 (2 または $101 \leq *type \leq 200$ ) 2 を指定した場合、基本コマンドで作成した線要素となります。 101~200 を指定した場合、ユーザセグメントとなります。
in *entid	作成する要素につけるユーザ識別番号 (1 以上) 0 を指定した場合、要素にはユーザ識別番号はつけられません。
in pmdata[5]	作成する線データを以下の形式で指定します。 [ 0 ] : 始点 X 座標 [ 1 ] : 始点 Y 座標 [ 2 ] : 単位ベクトルの X 成分 [ 3 ] : 単位ベクトルの Y 成分 [ 4 ] : 線分の長さ(正值)
in length[4]	線データ長 [ 0 ] : 5 を指定します。 [ 1 ] : 0 を指定します。 { [ 3 ] : 0 を指定します。
out *pesadr	作成された線の要素識別番号
out ircode[2]	復帰情報 [ 0 ] : 復帰コード [ 1 ] : 詳細コード

備考

- $\text{pmdata}[2] * \text{pmdata}[2] + \text{pmdata}[3] * \text{pmdata}[3] = 1.0$   
 $\text{pmdata}[4] = 0.0$  の場合、無限線となります。
- 要素の線種、線幅、線色はカレントの設定値に従います。

(4) エラーメッセージ一覧

メッセージ番号	エラー内容
13001	入力データに規定範囲を超えた値が入力されました。
13002	指定された点が同一点のため処理できません。
13003	操作次元が 3 次元なので実行できません。
13004	指定された 3 点が同一直線上にあるため線分が作成されません。
13005	図面格納領域が不足しました。
13006	コマンド処理中にエラーが発生しました。

水平垂直線コマンドのプロセス tpar00\_ のプログラミング例を示します。

```

/*****/
/*
/*      t p a r 0 0 _
/*
/*
/*      1. 機能概要
/*      水平線または垂直線を作成する
/*
/*
/*      2. 呼び出し形式
/*      void      tpar00_( long ircode[], long code[], double pos[],
/*                  long ent[], double dir[], long valuei[],
/*                  double valuer[], unsigned char *text, long status[],
/*                  long count[], long pose[], long ente[], long dire[],
/*                  long texte[], long global[] )
/*
/*
/*      3. 引数の説明
/*      o          long          ircode[]  : 復帰情報コード返答領域
/*
/*                  [0] : 復帰コード
/*
/*                  [1] : メッセージ番号
/*
/*                  [2] : 詳細コード
/*      i          long          code[]    : 識別番号
/*
/*                  [0] : 未使用
/*
/*                  [1] : 詳細メニュー識別番号
/*      i          double        pos[]     : 座標値データ
/*
/*                  [0] : 始点のX座標
/*
/*                  [1] : 始点のY座標
/*
/*                  [2] : 未使用
/*
/*                  [3] : 終点のX座標
/*
/*                  [4] : 終点のY座標
/*
/*                  [5] : 未使用
/*
/*                  [6] : 作画座標のX座標
/*
/*                  [7] : 作画座標のY座標
/*
/*                  [8] : 未使用
/*      i          long          ent[]     : 未使用
/*
/*      i          double        dir[]     : 未使用
/*
/*      i          long          valuei[]  : 未使用
/*

```

```

/*      i      double      valuer[] : 未使用      */
/*      i      unsigned char *text : 未使用      */
/*      i      long      status[] : 項目入力データ      */
/*                                  [0] : 作成本数      */
/*      i      long      count[] : 未使用      */
/*      i      long      pose[] : 未使用      */
/*      i      long      ente[] : 未使用      */
/*      i      long      dire[] : 未使用      */
/*      i      long      texte[] : 未使用      */
/*      i      long      global[] : 未使用      */
/*                                  */
/* 4. 呼出し条件      */
/*                                  */
/* 5. 備考      */
/*                                  */
/* 6. 復帰値      */
/*                                  */
/*****
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*****
/* マクロ定義      */
/*****
#define ZERO 1.0e-6

/*****
/* 外部参照関数宣言      */
/*****
/* 2次元／3次元を問い合わせる */
extern void scqdim_(long *);

/* 2次元の線要素を作成する */
extern void dilin2_(long *, long *, double [], long [], long *, long []);

```

```

/*****
/* 関数宣言 */
/*****
void tpar00_(long ircode[], long code[], double pos[], long ent[],
             double dir[], long valuei[], double valuer[], unsigned char *text,
             long status[], long count[], long pose[], long ente[],
             long dire[], long texte[], long global[])
{

/*****
/* 内部変数定義 */
/*****
    long    dim, i, lcnt, rtcode[2];
    double  dist, xy[9];
    long    type, entid, length[4], pesadr;
    double  pmdata[5];

/*****
/* 初期化 */
/*****

    ircode[0] = 0;
    ircode[1] = 0;
    ircode[2] = 0;
    type = 2;
    entid = 0;
    length[0] = 5;
    length[1] = 0;
    length[2] = 0;
    length[3] = 0;

/*****
/* 次元チェック */
/*****

    scqdim_( &dim );
    if( dim == 2 ){

```

```

/*****
/* パラメータチェック */
*****/

    for( i = 0; i <= 8; i++ ){
        xy[i] = pos[i];
    }
    if( code[1] != 1 && code[1] != 2 ){
        ircode[0] = 8;
        ircode[1] = 13001;
    } else if( status[0] < 2 || status[0] > 999 ){
        ircode[0] = 8;
        ircode[1] = 13001;
    } else if( (fabs( xy[0] - xy[3] ) < ZERO) &&
                (fabs( xy[1] - xy[4] ) < ZERO) ){
        ircode[0] = 8;
        ircode[1] = 13002;
    } else{
/*****
/* コマンドチェック */
*****/
/*****
/* 垂直線創成 */
*****/

        if( code[1] == 2 ){
            dist = xy[6] - xy[0];
            if( fabs( dist ) < ZERO ){
                ircode[0] = 8;
                ircode[1] = 13004;
                goto L_10000;
            } else{
                pmdata[1] = xy[1];          /* 始点のY座標 */
                pmdata[2] = 0.0e0;          /* X軸の傾き */
                if(xy[4] > xy[1])
                    pmdata[3] = 1.0e0;      /* Y軸の傾き */
                else
                    pmdata[3] = -1.0e0;      /* Y軸の傾き */
                pmdata[4] = fabs( xy[4] - xy[1] ); /* 線分の長さ */
                for( lcnt = 0; lcnt <= status[0] - 1; lcnt++ ){
                    /* 始点のX座標 */

```

```

        pmdata[0] = xy[0] + dist * lcnt / (status[0] - 1);
        dilin2_(&type, &entid, pmdata, length, &pesadr, rtcode);

        if( rtcode[0] != 0 ) goto L_9000;
    }
    goto L_10000;
}

/*****
/* 水平線創成
*****/

    } else{
        dist = xy[7] - xy[1];
        if( fabs( dist ) < ZERO ){
            ircode[0] = 8;
            ircode[1] = 13004;
            goto L_10000;
        } else{
            pmdata[0] = xy[0];          /* 始点のX座標 */
            if(xy[3] > xy[0])
                pmdata[2] = 1.0e0;      /* X軸の傾き */
            else
                pmdata[2] = -1.0e0;     /* X軸の傾き */
            pmdata[3] = 0.0e0;          /* Y軸の傾き */
            pmdata[4] = fabs( xy[3] - xy[0] ); /* 線分の長さ */
            for( lcnt = 0; lcnt <= status[0] - 1; lcnt++ ){
                /* 始点のY座標 */
                pmdata[1] = xy[1] + dist * lcnt / (status[0] - 1);
                dilin2_(&type, &entid, pmdata, length, &pesadr, rtcode);
                if( rtcode[0] != 0 ) goto L_9000;
            }
            goto L_10000;
        }
    }

L_9000:
    ircode[0] = 8;
    ircode[1] = 13005;
    if( rtcode[0] != 8 )
        ircode[1] = 13006;
}
} else{

```



---

```
        ircode[0] = 8;

        ircode[1] = 13003;
    }
L_10000:
    return;
}
```

## プロセスリストの作成

プロセスリストにプロセス名 (TPAR00) を指定します。  
プロセスリストの詳細は「第 8 章 プロセスリスト」で説明します。

@PROL3	PROLIST	_____	①
	PRONO	3998, TPAR00 _____	②
	PROLEND	_____	③

### プロセスリストの説明

#### ① PROLIST 文

プロセスリストの開始を示します。  
プロセスリストの名前を@PROL3 とします。

#### ② PRONO 文

プロセス名とプロセス番号の対応を指定します。  
コマンド定義体の CALL 文で指定した「3998, TPAR00」を記述します。  
既存のプロセスリストに、プロセスを追加する場合は、本文を付け加えてください。

#### ③ PROLEND 文

プロセスリストの終了を示します。

3 つの定義体とプロセスの結びつきを以下に示します。

#### コマンドリスト

@CMDL3	COMLIST	
	COMENTRY	HVLINE
	COMLEND	

#### コマンド定義体

HVLINE	PROC	
	CALL	3998, TPAR00, PARM=ALL
	LEAVE	
	PROCEND	

#### プロセスリスト

@PROL3	PROLIST	
	PRONO	3998, TPAR00
	PROLEND	

#### プロセス

```
void tpar00_(long ircode[3], ... )
```

C 言語でプログラミングする場合、名前 (関数名) は tpar00\_ です。

## コンパイル、リンク

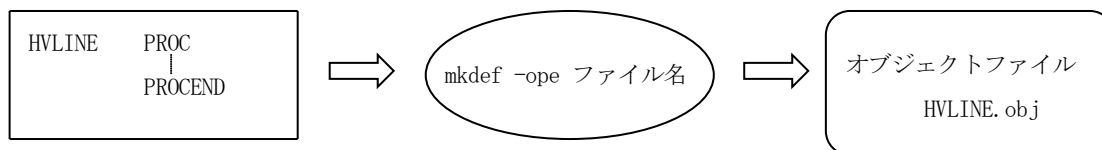
コマンドを組み込むには、作成したコマンド定義体、コマンドリスト、プロセス、プロセスリストをコンパイルし、システムとリンクします。

コンパイル、リンク用のコマンドはシステムで用意されており、このコマンドを実行します。

- (1) コマンド定義体、コマンドリスト、プロセス、プロセスリストのコンパイルを以下に示します。

### コマンド定義体のコンパイル

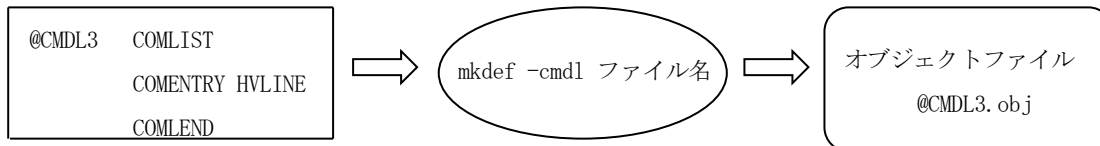
HVLINE



- ① コマンドプロンプトより以下のコマンドを実行します。  
`cd %ICADDIR%\usr¥opt¥obj`  
`mkdef -ope ..¥src¥HVLINE`
- ② カレントディレクトリにオブジェクトファイル (HVLINE.obj) が作成されます。

### コマンドリストのコンパイル

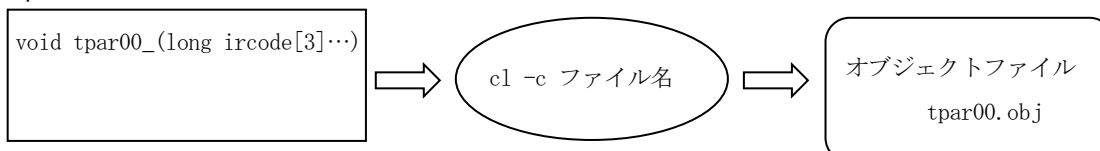
@CMDL3



- ① コマンドプロンプトより以下のコマンドを実行します。  
`cd %ICADDIR%\usr¥opt¥obj`  
`mkdef -cmdl ..¥src¥@CMDL3`
- ② カレントディレクトリにオブジェクトファイル (@CMDL3.obj) が作成されます。

### プロセスのコンパイル

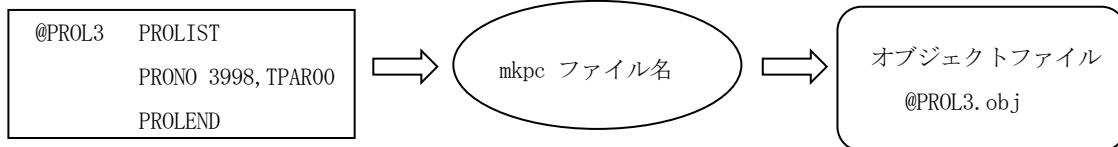
tpar00.c



- ① コマンドプロンプトより以下のコマンドを実行します。  
`cd %ICADDIR%\usr¥process¥obj`  
`cl -c ..¥src¥tpar00.c`
- ② カレントディレクトリにオブジェクトファイル (tpar00.obj) が作成されます。

## プロセスリストのコンパイル

@PROL3



- ① コマンドプロンプトより以下のコマンドを実行します。

```
cd %ICADDIR%\¥usr¥process¥obj
```

```
mkpc .. ¥src¥@PROL3
```

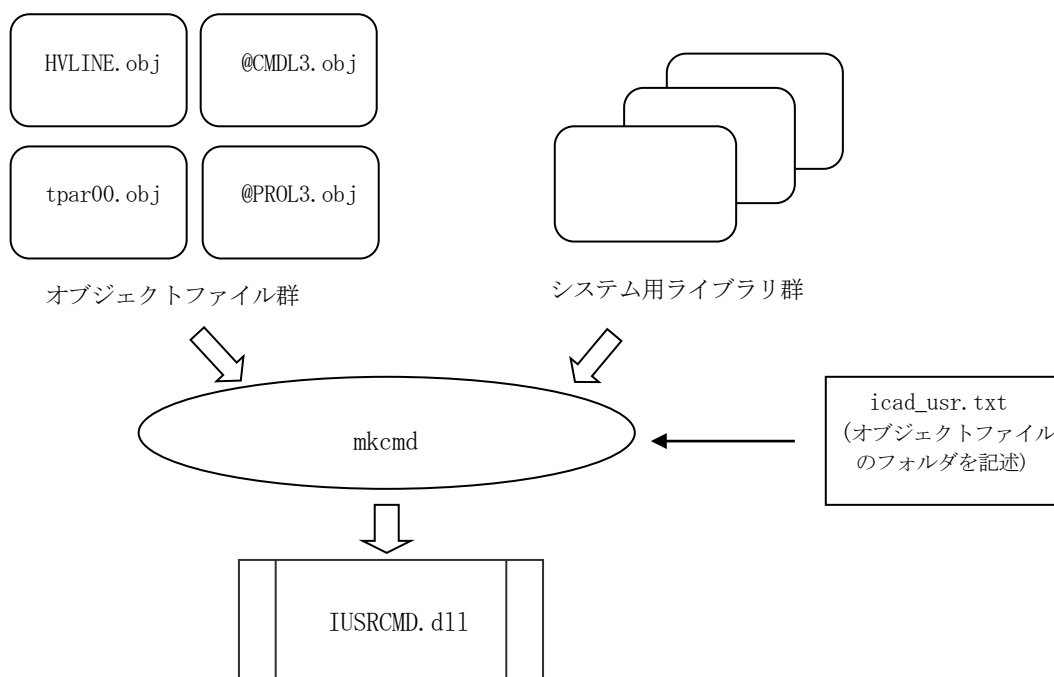
- ② カレントディレクトリにオブジェクトファイル (@PROL3. obj) が作成されます。

- (2) 作成したオブジェクトファイルが存在するディレクトリおよびファイル名を「 icad\_usr. txt 」に記述して、%ICADDIR%\¥USER¥BIN 配下に保存します。

%ICADDIR%\¥USER¥BIN がない場合は、%ICADDIR%\¥USER¥BIN を作成してから保存します。

; オブジェクト、ライブラリ		
.. ¥usr¥opt¥obj¥HVLIN. obj	—————	コマンド定義体
.. ¥usr¥opt¥obj¥@CMDL3. obj	—————	コマンドリスト
.. ¥usr¥process¥obj¥tpar00. obj	—————	プロセス
.. ¥usr¥process¥obj¥@PROL3. obj	—————	プロセスリスト
;		

- (3) 作成したオブジェクトファイルをシステムとリンクし、コマンドに組み込む方法を以下に示します。



- ① コマンドプロンプトより以下のコマンドを実行します。

mkcmd options

説明

options      ご利用の Visual Studio のバージョンを指定します。

Visual Studio のバージョン	options
Visual Studio 2015	-VS14
Visual Studio 2017	-VS15
Visual Studio 2019	-VS16

- ② %ICADDIR%\USER\BIN 配下に実行形式のファイルが作成されます。

## メッセージの登録

コマンドにてエラーを検出した場合などは、メッセージを出力して処理を中止する必要があります。本システムではメッセージをファイルに登録し、プログラムから独立した方式を採用しています。

メッセージの登録は、最初にメッセージファイルを作成し、それを入力データとしてインデックスファイルに登録します。メッセージについての詳細は「第 10 章 メッセージ」で説明します。

以下にメッセージファイル作成例を示します。

メッセージ番号 13001

エラー内容 入力データに規定範囲を超えた値が入力されました。

//3001,2,1,0,0,1,0

⇐ 制御本文行

① ② ③④⑤ ⑥ ⑦ ⑧

0,0,58,'MSG13001 入力データに規定範囲を超えた値が入力されました。' ⇐ メッセージ本文行

⑨ ⑩ ⑪ ⑫

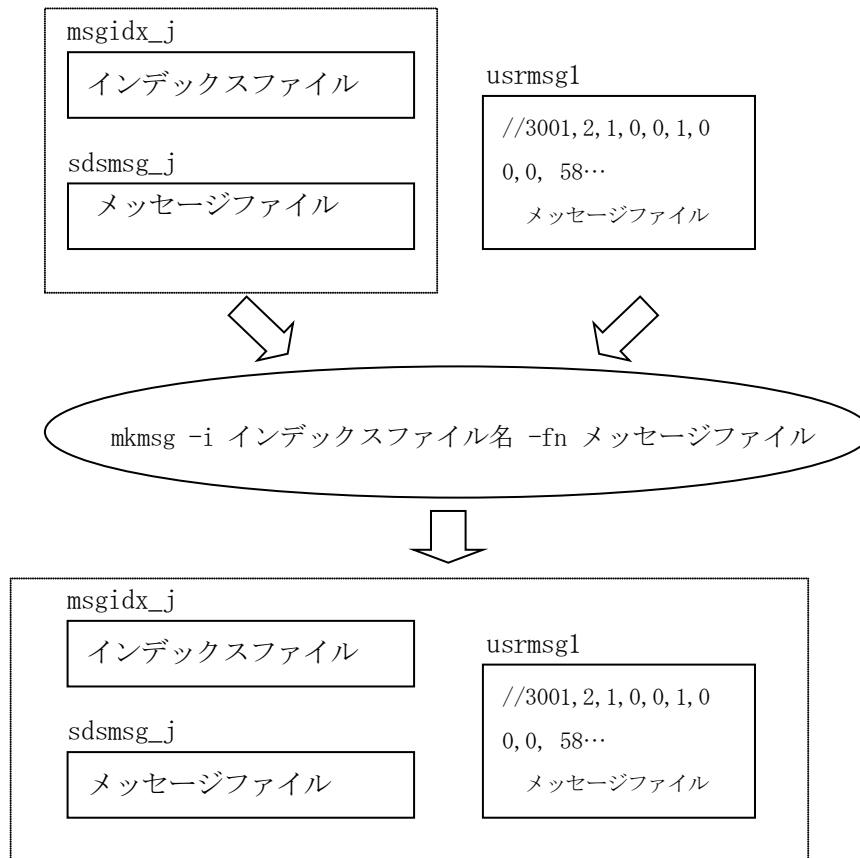
### (1) 制御情報行

- ① 「//」を記述します。
- ② メッセージ番号を指定します。
- ③ エラーレベル「2=処理続行不可能なレベル」を指定します。
- ④ 出力メッセージの色「1=赤」を指定します。
- ⑤ ベルの制御「0=鳴らさない」を指定します。
- ⑥ 追加情報の大きさを指定します。
- ⑦ 出力メッセージの行数「1」を指定します。
- ⑧ 追加情報「0=追加情報なし」を指定します。

### (2) メッセージ本文行

- ⑨ 追加情報がない場合は「0」を指定します。
- ⑩ 追加情報がない場合は「0」を指定します。
- ⑪ 出力メッセージのバイト数を指定します。
- ⑫ メッセージを記述します。

作成したメッセージファイル(usrmsg1)はmkmsg コマンドにより、インデックスファイルに登録します。  
インデックスファイル (msgidx\_j) 及びメッセージファイル (sdsmmsg\_j) はシステムであらかじめ用意されています。



- ① コマンドプロンプトより以下のコマンドを実行します。

```
cd %ICADDIR%\¥msg
```

```
mkmsg -i msgidx_j -f1 usrmsg1
```

- ② カレントディレクトリに変更されたインデックスファイルが作成されます。

作成したメッセージは、必要性が生じた箇所にて出力します。  
プロセスパラメタ ircode によりメッセージを出力する方法を以下に示します。

```
long int code[2], ircode[2];  
if( code[1] != 1 && code[1] != 2 ){  
    ircode[0] = 8;  
    ircode[1] = 13001;  
}
```

詳細は「プロセスの作成」の中のプログラミング例を参照してください。

## コマンドの登録

新しいユーザコマンドをコマンドメニューへ登録する手順及び方法については「ユーザーズガイドメニューのカスタマイズ」を参照してください。



「ユーザーズガイド」は本システムがインストールされているフォルダ配下”MAN”フォルダにPDFデータ (user\_guide.pdf) でご提供しております。



---

## 第3章 コマンド定義体

ここでは、コマンド定義体で指定できる内容別に具体的な使用方法について説明しています。

## コマンド定義体の概要

コマンド開発者は、コマンドのオペレーション手順を記述したコマンド定義体を作成する必要があります。オペレーション手順とは、どのようなタイプのデータをどのような順序で入力し、どのように処理をするかという一連の手順のことです。

コマンド定義体を作成するには以下のことを決定しておく必要があります。

### (1) コマンド名

日本語名（8文字以内）と英語名（6文字以内）

コマンドが持つ機能が名前から判断できるようなものにします。

例えば、日本語名「水平垂直線」英語名「HVLIN」 というコマンド名にすれば、水平垂直線を作成するコマンドであることが推定できます。

### (2) 機能概要

どのような機能を持つコマンドであるかを記述します。

### (3) シンタックス・ダイアグラム

シンタックス・ダイアグラムとは、コマンドの操作手順を図式化して表現したものです。

### (4) 入力促進メッセージ

入力促進領域に出力するメッセージを記述します。

日本語文字と英文字を混在しては指定できません。

### (5) 詳細機能

コマンドの持つ詳細機能をすべて記述します。

### (6) 詳細メニュー

詳細メニューに表示されるコマンドの各機能を選択するための項目名（選択項目名）を以下の形式で記述します。

①	②	③	④
メニュー	キーワード	初期値	説明
選択項目 A	KEYA	○	処理 A を実行します。
選択項目 B	KEYB		処理 B を実行します。

#### ① メニュー

詳細メニューに表示される選択項目名を記述します。

#### ② キーワード

各選択項目名に対応するキーワードを記述します。

英字で始まる 8 文字以内の英数字とします。

#### ③ 初期値

コマンド入力時の選択項目（排他項目）の状態を示します。

－排他項目

最初にコマンドを実行した時に、システムが自動的に選択する（初期設定状態）項目に「○」を記入します。なお詳細メニュー上ではこの項目が押された状態となります。

#### ④ 説明

各選択項目に対応する機能を記述します。

## (7) 項目入力領域

項目入力領域に表示される項目の、入力値について以下の形式で記述します。

①	②	③	④
項目	キーワード	初期値	説明
項目 1	KEY1	0.0000 (実数)	項目 1 の説明
項目 2	KEY2	0.0000 (実数)	項目 2 の説明

### ① 項目

項目入力領域に表示される項目名を記述します。

### ② キーワード

各項目名に対応するキーワードを記述します。

英字で始まる 8 文字以内の英数字とします。

### ③ 初期値

最初にコマンドを実行した時の各項目に対する初期値を示します。通常は、各項目に対し値が入力されると、その入力値が引き継がれます。括弧内は値の形式を示します。この形式には整数・実数・文字形式があります。

### ④ 説明

各項目入力データに関する説明を記述します。

入力値の規定範囲・最大値・最小値などを明確にします。

## (8) 処理対象要素

処理、操作の対象となる次元・要素を記述します。

## (9) 詳細機能オペレーション

コマンドの機能別に、実際にオペレーションしたものと同一イメージで記述します。

画面の変化、データの入力順序がわかるようにします。

## (10) 注意・制限事項

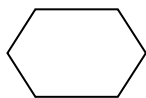
コマンドを実行するうえでの注意・制限事項を記述します。

## シンタックス・ダイアグラム

シンタックス・ダイアグラムとは、オペレーションの流れを図式化して表現したものです。「コマンド定義体の作成」とあわせてご覧ください。

以下にシンタックス・ダイアグラムで使用する記号とその意味を説明します。

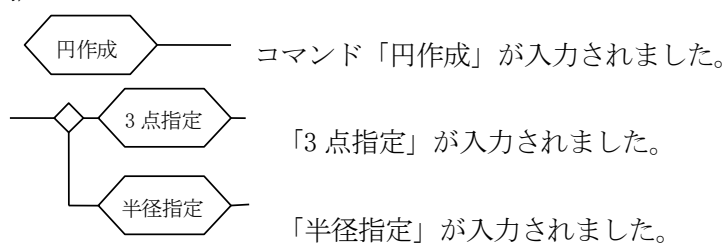
①



選択項目のいずれかが入力されたことを示します。

入力された選択項目を記号の中に記述します。

(例)



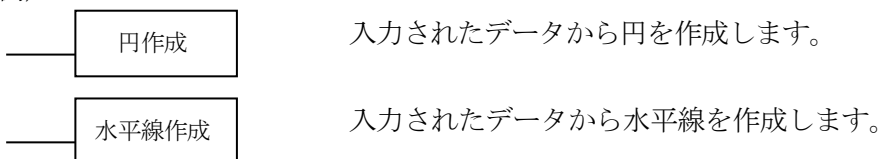
②



プロセスの実行を示します。

プロセスの機能を記号の中に記述します。

(例)

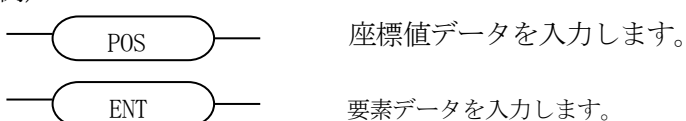


③

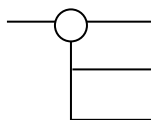


座標値、要素データのいずれかの入力要求を表します。

(例)



④

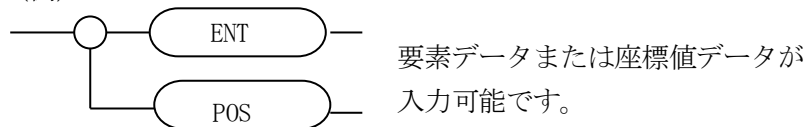


データタイプによる分岐を表します。

入力可能なデータが複数個存在することを示しています。

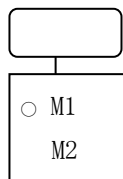
いずれかのデータを選んで入力します。

(例)



HOLD の場合は  を使います。

⑤



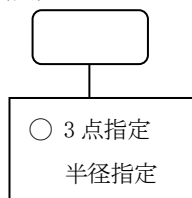
詳細メニュー領域のメニュー表示と入力要求を表します。

初期値は、○印をつけて表します。

メニュー項目 …… M1, M2

初期値 …………… M1

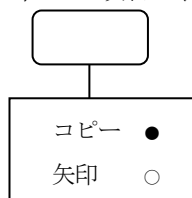
(例)



メニュー項目 …… 3点指定, 半径指定

初期値 …………… 3点指定

ON/OFF 項目の表示

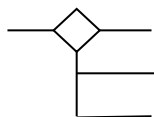


メニュー項目            初期値

コピー                    ON

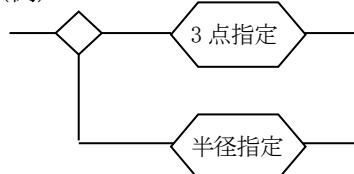
矢印                      OFF

⑥



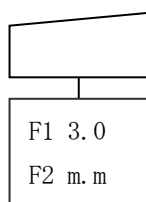
選択項目による分岐を示します。

(例)



「3点指定」「半径指定」の選択により分岐します。

⑦

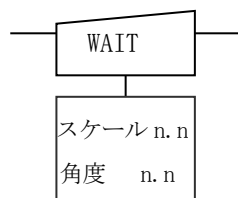


項目入力領域へのメニュー表示と入力要求を表します。

フィールド …… F1, F2

初期値 …………… F1 の 3.0

(例)

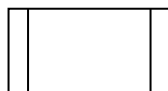


フィールド …… スケール, 角度

初期値 …………… なし

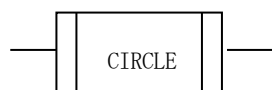
WAIT は WAIT 指定であることを表します。

⑧



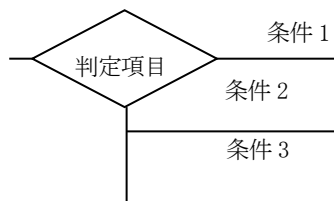
他のコマンド定義体の呼び出しを表します。

(例)



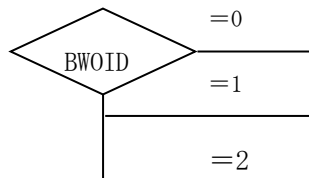
コマンド定義体 CIRCLE を呼び出します。

⑨

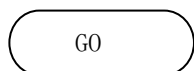


グローバル領域の値による分岐を表します。

(例)



⑩



**GO**の入力待ちを表します。

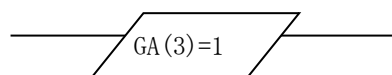
⑪



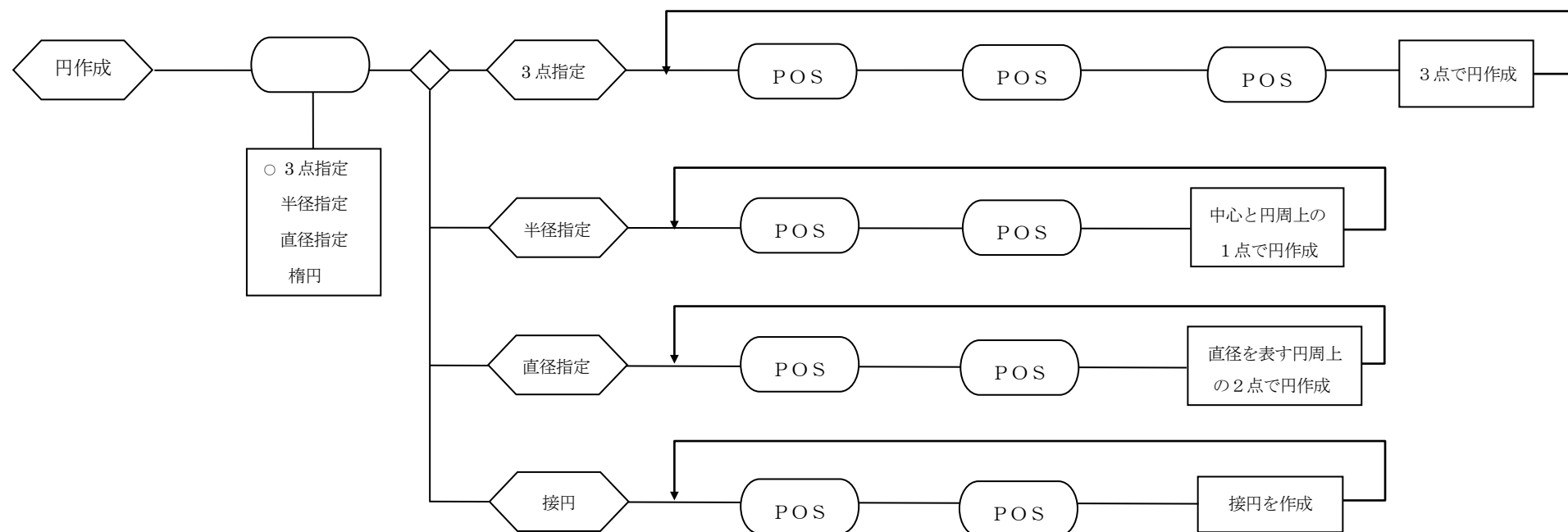
グローバル領域への値の設定等の処理を表します。

処理の要約を記号の中に記述します。

(例)



例：円作成コマンドのシンタックス・ダイアグラム



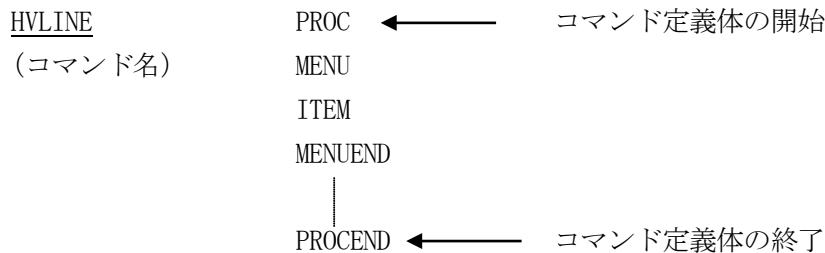
# コマンド定義体の作成方法

## ■ コマンド定義体の開始と終了

コマンド定義体を作成する場合、最初に PROC 文、最後に PROCEND 文を記述します。その間に、オペレーション手順を指定する他の文が記述できます。

PROC 文に記述された名前が、コマンド名です。

コマンド定義体の構造を以下に示します。



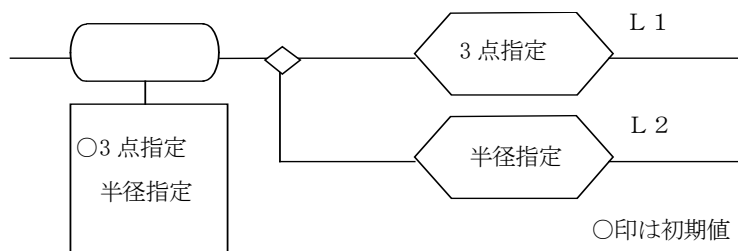
## ■ 詳細メニューの設定

記述内容

- ・ メニュー項目 (キーワード)
- ・ メニュー項目ごとの属性、指定可／不可、文字サイズ、色
- ・ メニュー項目間の排他関係
- ・ 初期値の指定
- ・ プロセスに渡す識別番号

メニューの指定は、MENU 文で始まり MENUEND 文で終わります。メニュー内で指定するキーワード、表示文字などは ITEM 文を使用します。以下に例を示します。

(例1) 選択により分岐するメニュー



```
MENU
GROUPS
  ITEM ( HIT, 1 ), L1, TEXT='3点指定', TXTYP=JEF
  ITEM ( R, 2 ), L2, TEXT='半径指定', TXTYP=JEF
GROUPE
MENUEND
```

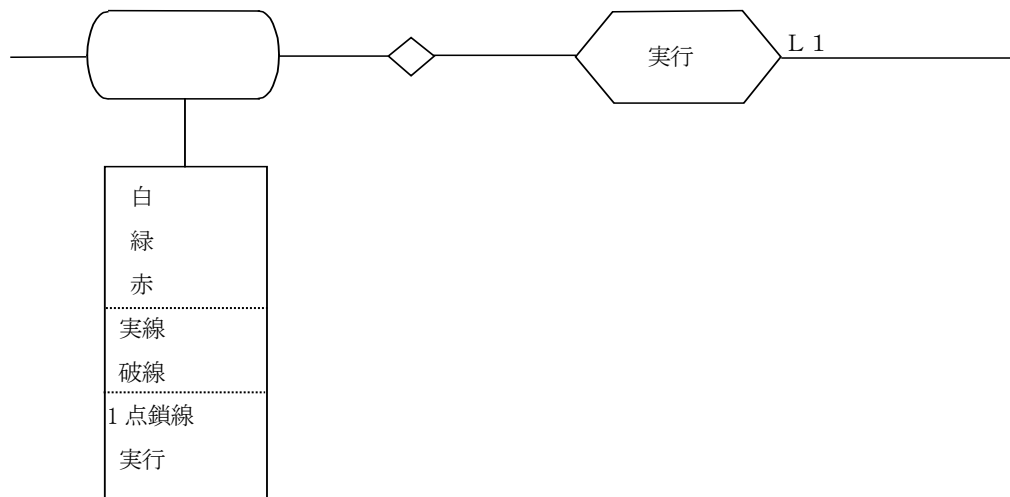
同一グループに指定された ITEM は  
選択項目です。

分岐先の指定  
識別番号



(例 2) オペレーションが遷移しないメニュー

選択されても分岐しない場合は、分岐先のラベルを省略します。この場合、コマンド定義体の実行は MENU 文の位置にとどまり、データの入力待ちとなります。



MENU

GROUPS

ITEM ( WHT, 1 ), TEXT='白', TXTYP=JEF

ITEM ( GRE, 2 ), TEXT='緑', TXTYP=JEF

ITEM ( RED, 3 ), TEXT='赤', TXTYP=JEF

GROUPE

GROUPS

ITEM ( SOL, 1 ), TEXT='実線', TXTYP=JEF

ITEM ( DAS, 2 ), TEXT='破線', TXTYP=JEF

ITEM ( OCH, 3 ), TEXT='1点鎖線', TXTYP=JEF

GROUPE

GROUPS

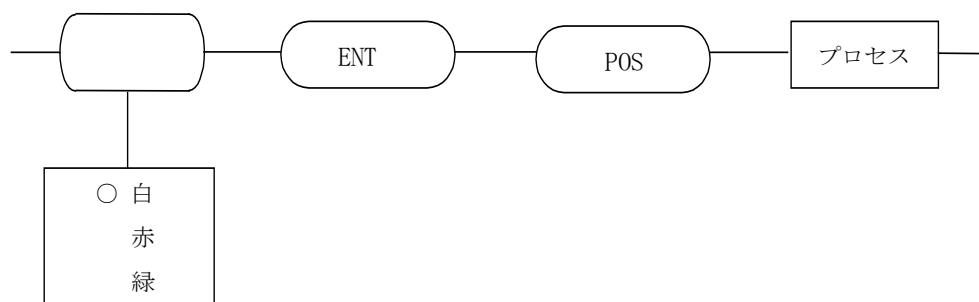
ITEM ( EXEC, 10 ), L1, TEXT='実行', TXTYP=JEF

GROUPE

MENUEND

(例 3) オペレーションが変化しないメニュー

オペレーションの途中でいつでも指定でき、かつオペレーションの状態が変化しないような項目を指定するには、分岐先指定に '\*' を記述します。分岐先指定に '\*' を指定する場合には、グループ内の ITEM すべてについて '\*' としなければなりません。



プロセス呼出し以前ならどの位置にいても指定できます。指定によりオペレーション位置は変化しません。

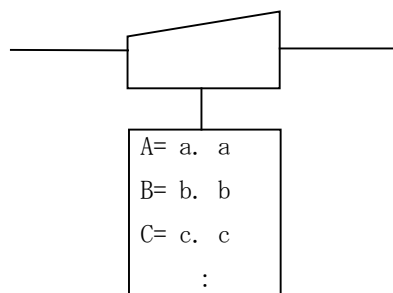
```
MENU
  GROUPS      DEFAULT=1
  ITEM        ( WHITE, 1 ), *, TEXT='白', TXTYP=JEF
  ITEM        ( RED, 2 ), *, TEXT='赤', TXTYP=JEF
  ITEM        ( GREEN, 3 ), *, TEXT='緑', TXTYP=JEF
  GROUPE
MENUEND
```

## ■ 項目入力領域の設定

記述内容

- ・ 項目入力領域のキーワード
- ・ 項目入力領域のデータタイプ（実数、整数、文字）
- ・ 項目入力領域の初期値

項目入力領域の指定は、STATUS 文で始まり、STEND 文で終わります。実際のフィールドの指定は STFLD 文を使用します。以下に記述例を示します。



```
STATUS
  STFLD  A,  TYPE=R, INIT=10.0
  STFLD  B,  TYPE=R, INIT=1.0
  STFLD  C,  TYPE=R, INIT=0.0
  :
STEND
```

項目入力領域には、表示のみして次を実行するものと、データ入力されるまで次の処理を実行しないものとの2種類があります。

- ・ 表示のみの場合

STATUS 文で WAIT=NO を指定します。

項目入力領域に対するデータは以降のどの時点でも入力でき、しかもオペレーションの遷移はしません。

- ・ データ入力待ちの場合

STATUS 文で WAIT=YES を指定します。

項目入力領域に対するデータは以降のどの時点でも入力できますが、オペレーションは STATUS 文の記述位置から再実行されます。

## ■ データ入力

コマンド処理のために必要なデータを入力するコマンド定義体について説明します。

記述内容

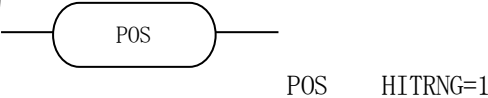

- ・ 座標値入力 ..... ヒットレンジ
- ・ 要素入力 ..... ヒットレンジ、レイヤマスク、要素タイプマスク、プリミティブタイプマスク

データ入力をするために、データのタイプに応じて以下の各文が用意されています。

- ・ 座標値データ : POS 文
- ・ 要素データ : ENT 文

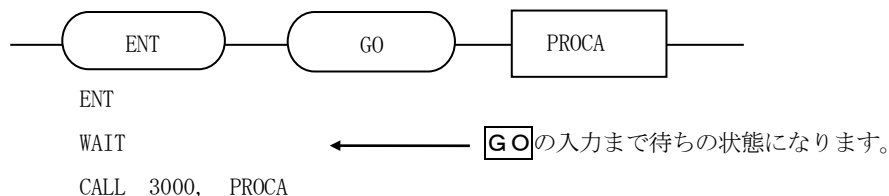
システムは上記の文を実行すると、オペレータからのデータ入力を待ちます。オペレータがデータを入力するまでコマンドは待ち状態となります。

各文の記述例を以下に示します。

- ・ 座標値入力  

- ・ 要素入力  


## ■ GO の入力

**GO** の入力待ちは、WAIT 文を使用します。システムは **GO** が入力されるまで、次の文を実行しません。以下に例を示します。



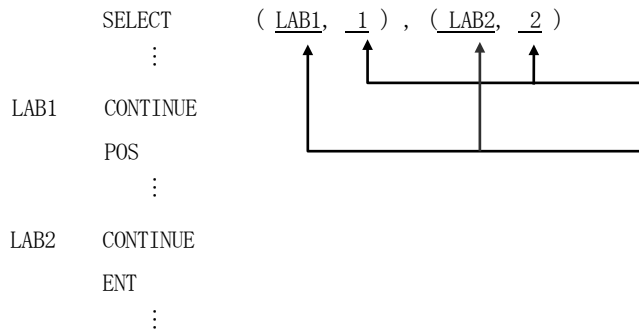
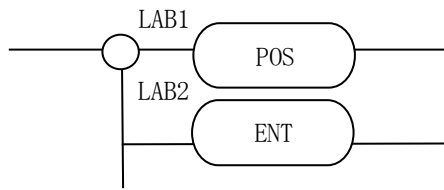
## ■ 入力データタイプによる分岐

入力データタイプによる分岐の指定は、SELECT 文を使用します。

記述内容

- ・ 入力可能なデータタイプ
- ・ プロセスに渡す識別番号

SELECT 文実行時には指定されたタイプのデータ入力が可能となり、入力されたデータのタイプに従って指定された処理に分岐します。このため、分岐先で同じタイプのデータ入力指定をしてはいけません。SELECT 文の例を次に示します。



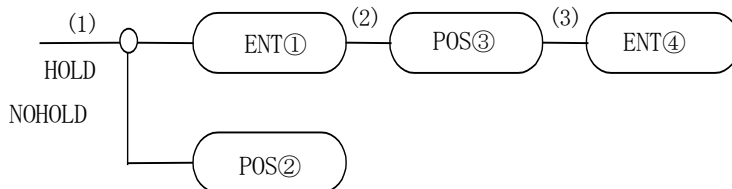
データ入力文とデータタイプの関係を次に示します。

POS 文 : 座標値データ  
 ENT 文 : 要素データ  
 MENU 文 : 選択項目データ  
 STATUS 文 : 項目入力データ  
 WAIT 文 : **GO**

SELECT 文には HOLD 指定と NOHOLD 指定の二つがあります。

- HOLD 指定  
 SELECT 文による分岐成立後も分岐条件となるデータ入力が可能です。
- NOHOLD 指定  
 SELECT 文による分岐成立後には各分岐先のデータ入力はできません。

SELECT 文と入力可能データ及びデータ入力時のオペレーションの変化について以下に説明します。  
 入力可能データの変化

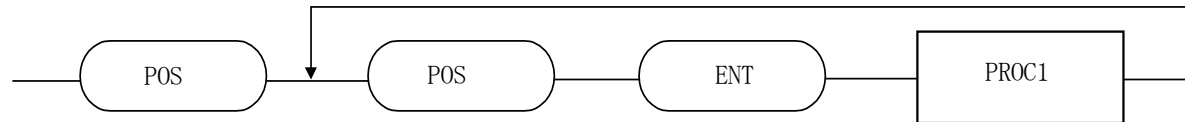


入力可能なデータ

位置	HOLD 指定	NOHOLD 指定
(1)	ENT①, POS②	ENT①, POS②
(2)	POS③, ENT①	POS③
(3)	ENT④, POS②	ENT④

## ■ 繰り返しの指定

### (1) 同一手順の繰り返し



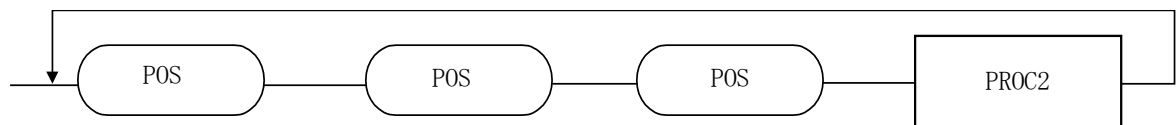
```

POS
REPEAT
  POS
  ENT
  CALL 2500, PROC1
LEAVE
  
```

永久に繰り返します。他コマンド等が入力された時に、  
ループからぬけます。

REPEAT 文から LEAVE 文までの間で入力されたデータは REPEAT 文に戻った時点でクリアされます。

### (2) 固定回数のデータ入力

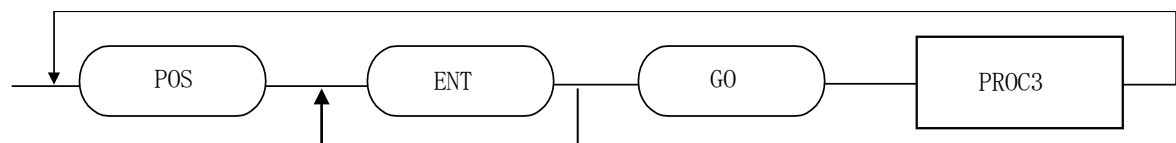


```

REPEAT
  LOOPS REPEAT=3
  POS
  LOOPE
  CALL 2501, PROC2
LEAVE
  
```

3 回繰り返したのち、ループからぬけます。

### (3) 不定回数のデータ入力



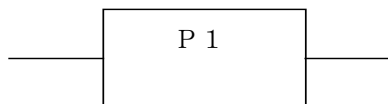
```

REPEAT
  POS
  LOOPS REPEAT=*
  ENT
  LOOPE
  CALL 2502, PROC3
LEAVE
  
```

**GO** 入力によりループからぬけます。

## ■ プロセスの実行

プロセスの呼出しは、CALL 文を使用します。以下に例を示します。



```
CALL    3100,  P1
CALL    3200,  P2,  PARM= (POS)
プロセス番号 プロセス名 プロセスパラメタ
```

プロセス番号はプロセスを識別するために与えられる番号であり、システムで一意となる様にコマンド開発者が割り当てます。

## ■ 入力促進メッセージの表示

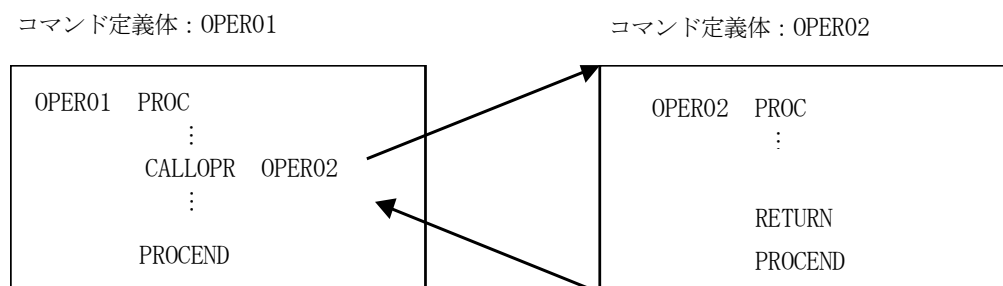
オペレータへのメッセージの表示は、PROMPT 文を使用します。  
記述例を以下に示します。

```
PROMPT  TXTYP=JEF, TEXT='始点座標'
PROMPT  DEL
PROMPT  MSGNO=1000
```

TEXT 指定の場合は入力促進領域にメッセージが表示されます。  
また、MSGNO 指定するとメッセージ表示領域に、メッセージを表示することもできます。

## ■ コマンド定義体の呼び出し

コマンド定義体から他のコマンド定義体を呼び出すには、CALLOPR 文を使用します。また、呼出し元のコマンド定義体へ復帰するには RETURN 文を使用します。



呼び出されたコマンド定義体から呼出し元に復帰する時点で、呼び出されたコマンド定義体で表示された詳細メニュー及び項目入力領域 (STATUS) は消去され、入力されたデータはクリアされます。

## ■ グローバル領域への設定と参照

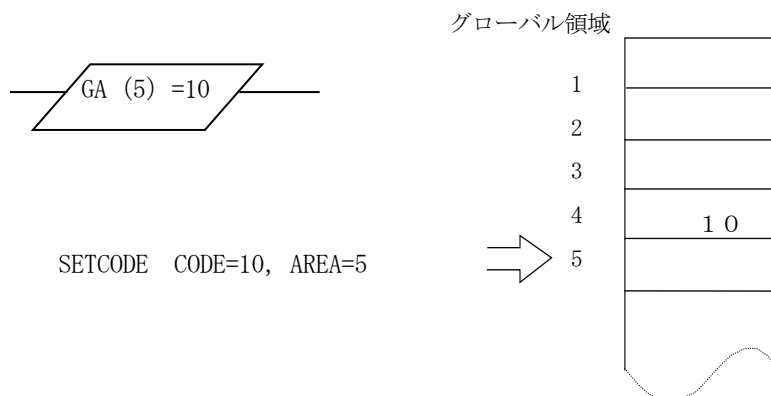
グローバル領域とは、コマンド定義体とプロセス間で共通な情報を保持しておくための領域です。グローバル領域は4バイト整数型の配列として扱われます。ただし、配列番号は1から始まります。以下の文により、グローバル領域の内容を参照したり、変更したりすることができます。

設定	GROUPS	AREA
	ITEM	AREA
	SELECT	AREA
	SETCODE	AREA
参照	CASE	AREA
	GROUPS	DEFAULT
	ITEM	MODE
	STFLD	INIT

グローバル領域の大きさを越えた値を指定した場合には、コマンドの実行時にエラーとなります。値の設定、参照例を次に示します。

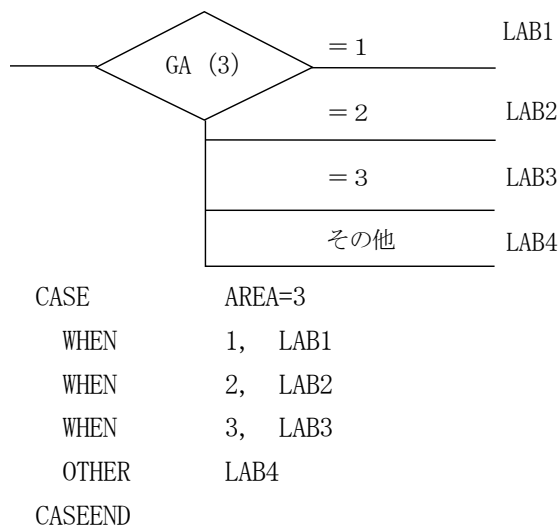
### ・ 値の設定

グローバル領域への値の設定は、SETCODE 文を使用します。記述例を示します。



### ・ 値の参照

グローバル領域の値により、次の処理を変更したいときには、CASE 文を使用します。

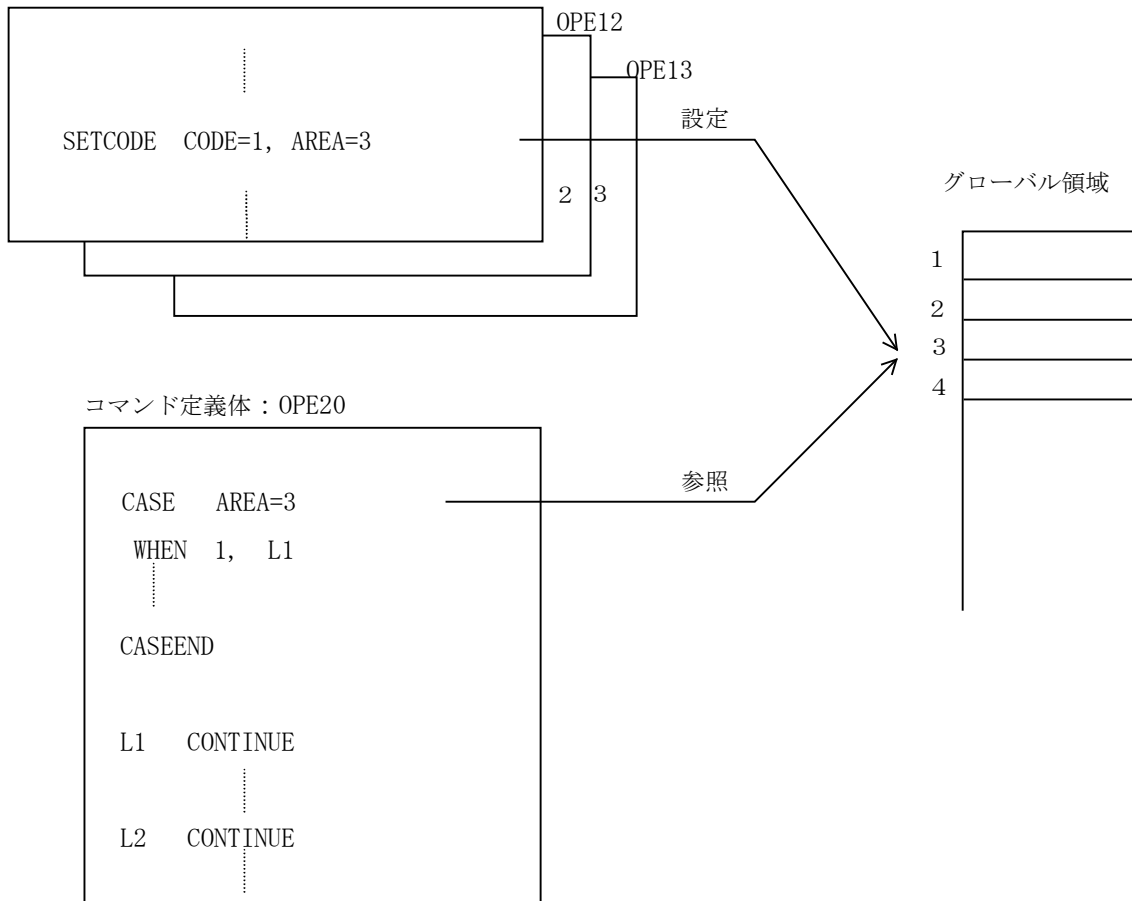


## ■ 特殊なオペレーションの指定

### (1) コマンド定義体間での情報の受渡し

以前のコマンド定義体の実行によって、以降のオペレーションの手順を変更したい場合。

コマンド定義体：OPE11



グローバル領域の値で処理を変更します。

```

CASE AREA=3
  WHEN 1, L1 .....OPE11 を実行した場合
  WHEN 2, L2 .....OPE12 を実行した場合
  WHEN 3, L3 .....OPE13 を実行した場合
CASEEND
:
L 1 CONTINUE
:
L 2 CONTINUE
:
L 3 CONTINUE
:
  
```

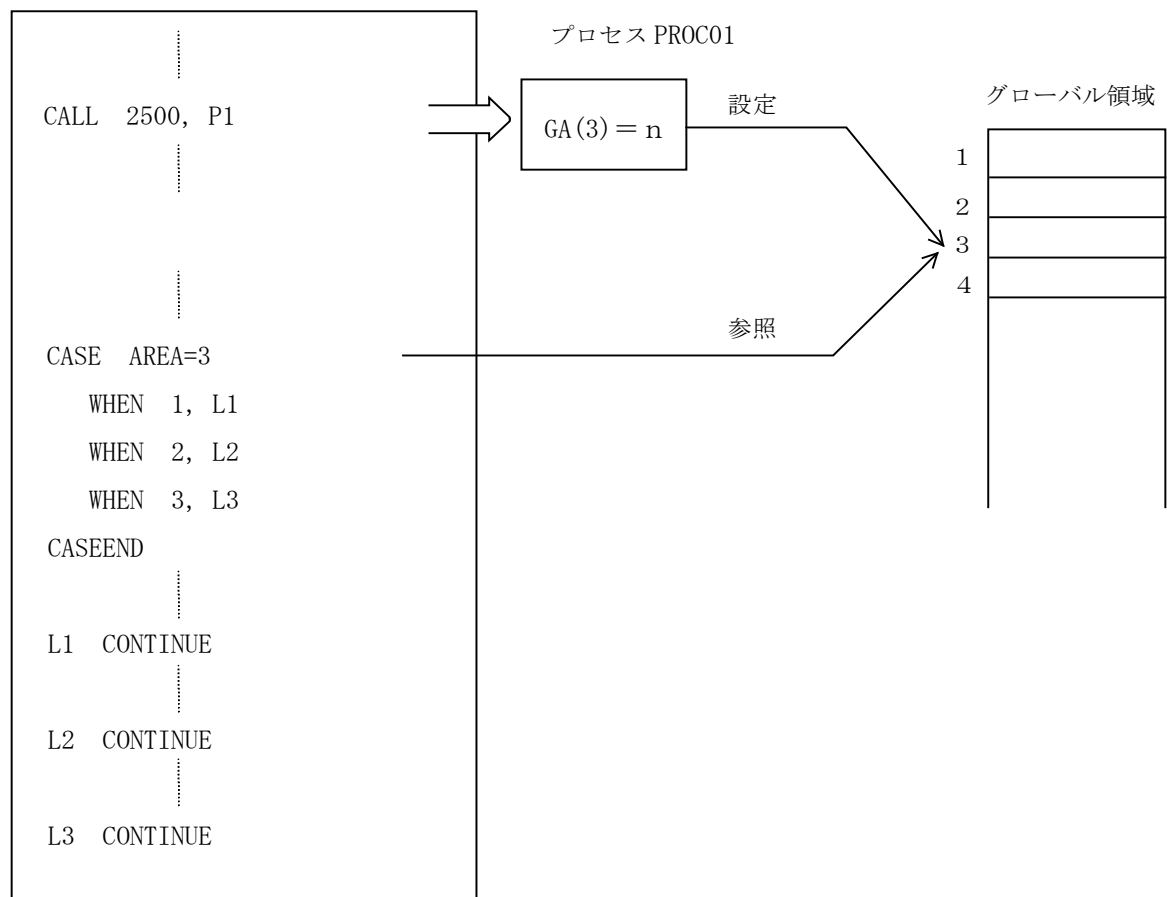


(2) プロセスとコマンド定義体間での情報の受渡し

プロセスの実行結果によって、以降のオペレーション手順を変更する場合、情報をグローバル領域で受渡します。

例を以下に示します。

コマンド定義体

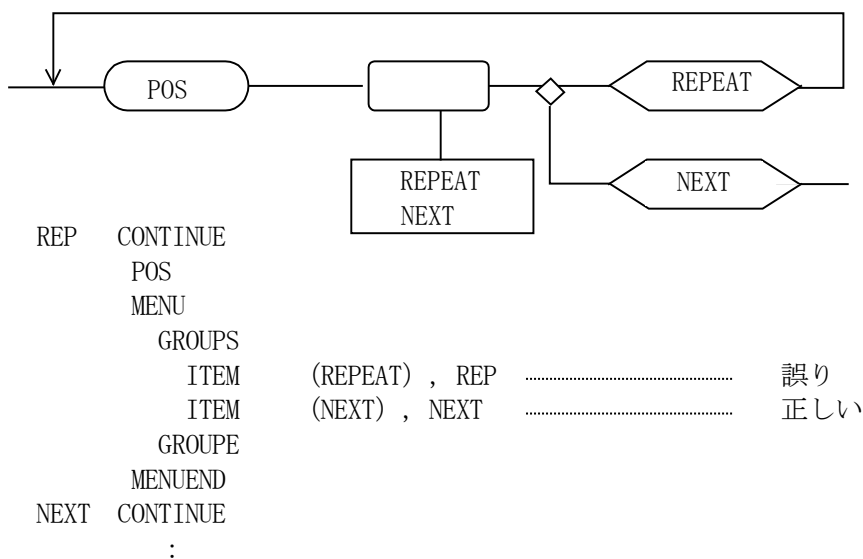


## ■ 注意事項

コマンド定義体を作成する際の注意事項を説明します。

### (1) 分岐先指定について

MENU 文 (ITEM)、CASE 文 (WHEN, OTHER)、SELECT 文による分岐は、オペレーションが順次先に進むことを想定しているため、以前のオペレーションの途中で飛び込むような分岐指定をした場合、その動作は保証されません。例を以下に示します。



### (2) SELECT 文について

SELECT 文の分岐条件として、MENU 文、STATUS 文を使用するときには、以下のことに充分注意してください。

STATUS 文では WAIT=YES を指定するようにしてください。

MENU 文では、すべての GROUPS 文に対して、DEFAULT を省略し RESET=YES を指定するようにしてください。

システムは、SELECT 文のそれぞれの分岐先に対して、データ入力 (POS, ENT) による入力待ちが発生するまで処理を進めます。このため、STATUS、MENU 文の指定によっては、意図した動作と異なる場合が発生します。例を以下に示します。

```

SELECT ( L 1, 1 ), ( L 2, 2 )
L1 CONTINUE
MENU
  GROUPS RESET=NO
  ITEM A, *
  ITEM B, *
  GROUPE
MENUEND
CALL 3000, P3000
LEAVE
L2 CONTINUE
POS
:
```

2 回目以降の実行時は、最新の入力データにより次の処理が実行されてしまいます。以後 LEAVE まで入力要求がないため、動作が異常になります。

### (3) 繰返し指定での注意事項

LOOPS～LOOPE 文、REPEAT～LEAVE 文の間でデータ入力待ちが発生しない様なコマンド定義体を作成してはいけません。コマンド定義体が永久にループします。

MENU 文、STATUS 文を記述する場合には特に注意が必要です。例を以下に示します。

→	REPEAT		
	MENU		
	GROUPS	RESET=NO	
	ITEM	(KYWD1) , *	1 回目の実行時には DEFAULT が
	ITEM	(KYWD2) , *	ないためデータ入力待ちとな
	GROUPE		りますが、一旦入力するとルー
	MENUEND		プ状態になります。
	CALL	3000, P3000	
└─	LEAVE		

### (4) 座標値データ・要素データのエコーの制御

#### ① エコーの表示

座標値データの入力時に '＊' マークが表示され、要素データの入力時には検索された要素に色があえ表示がされます。

#### ② エコーの消去

座標値データ・要素データ入力時のエコー表示が消去されるタイミングを以下に示します。

- ・ CALL 文によりプロセスを呼び出す前
- ・ LEAVE 文を実行したとき
- ・ プロセスからの復帰コードが 8 以上のとき
- ・ PROCEND 文を実行したとき
- ・ CALLOPR 文により呼び出されたコマンド定義体の RETURN 文を実行したとき
- ・ データ入力により上位遷移が起ったとき
  - 別コマンドを入力したとき
  - 上位詳細メニューを入力したとき (\*項目は含みません)
  - 上位項目を入力したとき (NOWAIT 指定は含みません)
  - HOLD 指定の SELECT 文の分岐データを入力

---

## 第4章 コマンドリスト

ここでは、作成したコマンドをコマンドリストに記述する方法について説明しています。

## コマンドリストの概要

コマンド開発者が開発したコマンドは、すべてコマンドリストに英語名で記述します。コマンドリストに記述することで、「コマンド」としてシステムに認識されます。

コマンドリストによる定義は、以下のように COMLIST 文で開始し、COMLEND 文で終了します。この間にコマンド名の数だけ COMENTRY 文を記述します。

既存のコマンドリストにコマンド名を追加する場合は COMENTRY 文を追加してください。

例を以下に示します。

次のコマンド名を記述します。

LINE、CIRCLE、ARC、POINT、MARKER、NOTE、MOVE、ROTATE、MIRROR、ERASE

コマンドリスト名は @CMDL3 とします。

```
@CMDL3      COMLIST
              COMENTRY  LINE
              COMENTRY  CIRCLE
              COMENTRY  ARC
              COMENTRY  POINT
              COMENTRY  MARKER
              COMENTRY  NOTE
              COMENTRY  MOVE
              COMENTRY  ROTATE
              COMENTRY  MIRROR
              COMENTRY  ERASE
              COMLEND
```

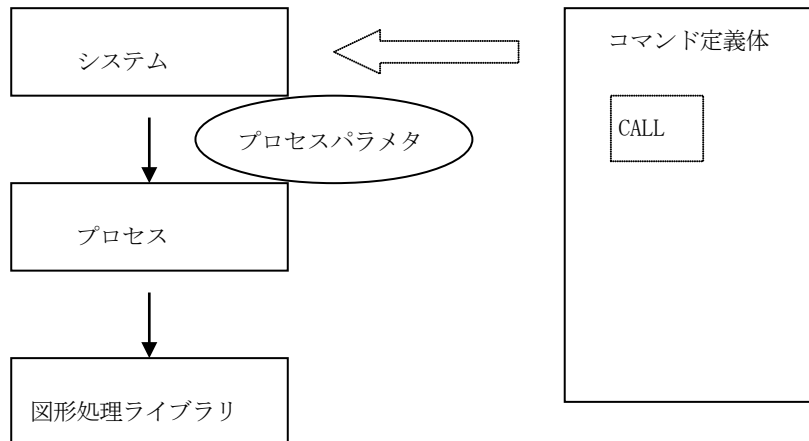
---

## 第5章 プロセス

ここでは、プロセスの作成手順、プロセスパラメタ、図形処理ライブラリ(基本編)について説明しています。

## プロセス作成手順

コマンドの機能を実現しているのはプロセスであり、これはコマンド開発者が作成します。コマンドが入力されると、以下のように対応したコマンド定義体が発行され、CALL 文で指定されたプロセスがシステムより呼び出されます。プロセス呼出し時には、指定されたデータがプロセスパラメタとしてシステムからプロセスに渡されます。プロセスでは、受け取ったデータをもとにコマンド固有の処理をします。



プロセスの作成手順を以下に示します。

- ① プロセスの設計をします。(プログラム設計書)  
プロセスでは、入力データを受取り、データを解釈して、作図等を行います。
- ② プログラミング  
プログラム設計書に沿ってプロセスをプログラミングします。

プロセスは、オペレータが入力したデータをシステムから受取り、コマンド固有の処理をするプログラムです。

プロセスは、以下の様な処理をします。

- ① システムから渡されたデータの加工
- ② 図形の作成、操作、編集
- ③ アプリケーション固有の処理

また、プロセスは以下の様なデータを受取ることができます。

- (1) 識別番号
- (2) 座標値
- (3) 要素情報
- (4) 項目入力領域情報
- (5) グローバル領域の情報

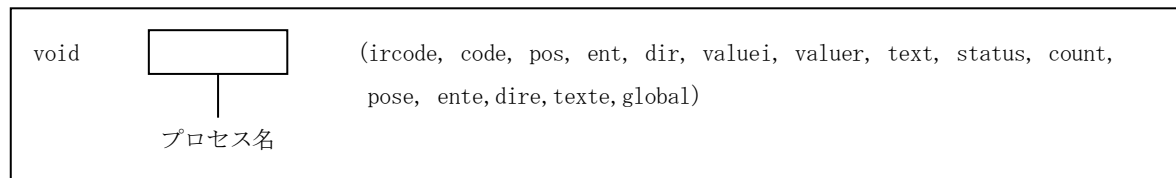
これらのデータは、コマンド定義体のプロセス呼出しで指定されたものだけが、プロセスパラメタとして通知されます。(4)、(5)については、図形処理ライブラリを利用して受取することもできます。

## データを受け取る方法（プロセスパラメタ）

プロセスがデータをシステムから受け取る時の方法を述べます。

オペレータがマウスを使って画面上の 2 点を指定すると、システムはそれを (X、Y) 座標値に変換します。その座標値はプロセスパラメタ領域に設定され、プロセスに通知されます。

以下に示す形式で、プロセスの呼び出しをします。



各プロセスパラメタの説明を次に述べます。

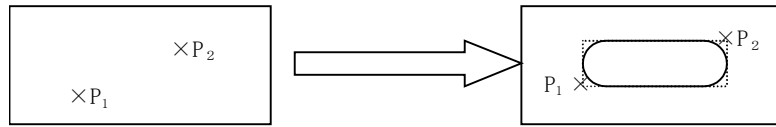
	プロセスパラメタ名	内 容
out	ircode[3]	プロセスの復帰情報
in	code[]	識別番号
in	pos[]	座標値
in	ent[]	要素情報
in	dir[]	未使用
in	valuei[]	未使用
in	valuer[]	未使用
in	text[]	未使用
in	status[]	項目入力領域情報
in	count[]	各パラメタの大きさ
in	pose[]	pos 入力時の付加情報
in	ente[]	ent 入力時の付加情報
in	dire[]	未使用
in	texte[]	未使用
in/out	global[]	グローバル領域

プロセスでは以上のプロセスパラメタの内、コマンド処理に必要な情報のみ使用してください。

また dir、valuei、valuer、text、dire、texte は未使用パラメタであるので、プロセスでその内容を参照したり変更したりしてはいけません。



2 点を入力して以下の図形を作成する DEMO10 コマンド例を以下に示します。



このコマンドの記述例とプログラムを示します。

### ①コマンド定義体

```
DEMO10  PROC
        LOOPS REPEAT=2
        POS
    LOOPE
    CALL  3010, PROC10, PARM=POS
    PROCEND
```

↑  
POS データをプロセス  
パラメタとして指定します

システムは POS 入力ごとに座標値  
をプロセスパラメタ領域に設定します

### ②システムのプロセスパラメタ領域

```
ircode[ 0 ]
      [ 1 ]
      [ 2 ]
```

```
pos  [ 0 ] : X1 座標    1 点目の座標値
      [ 1 ] : Y1 座標    (P1)
      [ 2 ] : 未使用
      [ 3 ] : X2 座標    2 点目の座標値
      [ 4 ] : Y2 座標    (P2)
      [ 5 ] : 未使用
```

- ・プロセスの実行結果をシステムに通知します
- ・プロセスの第一パラメタとして必ず必要です

コマンド定義体で指定したデータがプロセスパラメタとして通知されます

### ③プロセス

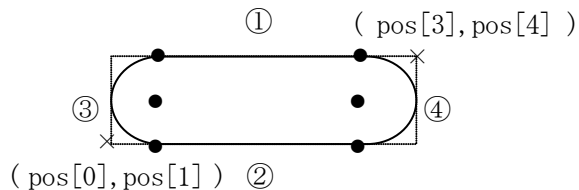
```
void proc10_ (long ircode[3], double pos[])
{
    long  ircode[3];
    double pos[6];
    :
}
```

## 図形処理ライブラリ（基本編）の利用方法

システムでは、作図プログラム・編集プログラムなど、いろいろな種類のプログラムを図形処理ライブラリとして用意しています。これらのプログラムを利用し、入力データから図形を作成していきます。図形処理ライブラリを用いることにより、簡単に2次元要素を作成することができます。

次の図形処理ライブラリを使用して、以下の図形を作成する例を示します。

- ・ dilin2\_ 線要素作成
- ・ diarc2\_ 円弧要素作成



```
r=(pos[4]-pos[1])/2.0;
type=2;entid=0;                                     線要素①作成
pmdata[0]=pos[0]+r;pmdata[1]=pos[4];pmdata[2]=1.0;pmdata[3]=0.0;pmdata[4]=pos[3]-pos[0]-2.0*r;
length[0]=5;length[1]=0;length[2]=0;length[3]=0;
dilin2_(&type,&entid,pmdata,length,&pesadr,ircode);
type=2;entid=0;                                     線要素②作成
pmdata[0]=pos[0]+r;pmdata[1]=pos[1];pmdata[2]=1.0;pmdata[3]=0.0;pmdata[4]=pos[3]-pos[0]-2.0*r;
length[0]=5;length[1]=0;length[2]=0;length[3]=0;
dilin2_(&type,&entid,pmdata,length,&pesadr,ircode);
type=5;entid=0;                                     円弧要素③作成
pmdata[0]=pos[0]+r;pmdata[1]=pos[1]+r;pmdata[2]=r;pmdata[3]=PAI/2.0;pmdata[4]=PAI;
length[0]=5;length[1]=0;length[2]=0;length[3]=0;
diarc2_(&type,&entid,pmdata,length,&pesadr,ircode);
type=5;entid=0;                                     円弧要素④作成
pmdata[0]=pos[3]-r;pmdata[1]=pos[4]-r;pmdata[2]=r;pmdata[3]=PAI/2.0;pmdata[4]=-PAI;
length[0]=5;length[1]=0;length[2]=0;length[3]=0;
diarc2_(&type,&entid,pmdata,length,&pesadr,ircode);
```

---

## 第6章 図形処理ライブラリ(応用編) の利用方法

ここでは、図形処理ライブラリ(応用編)の利用方法について説明しています。

## 要素の種類

コマンドで図形の作成、移動、コピー、削除などの図形操作をする単位を要素といいます。  
要素には以下のものがあります。

### 要素

- 2次元要素
  - ・点、線、円、円弧、スプライン、その他作図要素
  - ・長さ寸法線、角度寸法線、径寸法線、面取り寸法線、角/長円/角穴寸法線、円弧長寸法線、注記、風船、矢印、文字列、シンボル/矢視/切断線/、ハッチング、仕上記号、表面粗さ、幾何公差、溶接記号、デルタ、その他製図要素
  - ・記号
  - ・ユーザセグメント(2次元)
- 配置子図
- 写像部品
- 実像部品
- グループ

### 2次元要素

2次元要素は一般に複数個のプリミティブから構成されます。このプリミティブとは図形要素の最小単位であり、システムで用意されています。コマンド開発者は独自のプリミティブを作成することはできませんが、プリミティブを自由に組み合わせて固有のユーザセグメント(2次元要素)を作成することができます。2次元要素は2次元プリミティブ、任意情報プリミティブから構成されます。2次元要素は図形処理ライブラリを利用して作成します。

- ・ 基本コマンドで作成した場合



線要素：線プリミティブ1個で構成されています。  
円弧要素：円弧プリミティブ1個で構成されています。

- ・ ユーザコマンドで作成した場合



ユーザセグメント：  
線プリミティブ2個と円弧プリミティブ2個で構成されています。

## 要素識別番号

各要素には要素識別番号が付加されており、システムは要素識別番号により要素を特定します。コマンドはこの要素識別番号により図形処理ライブラリに対し「移動する」「削除する」などの指示をします。

要素識別番号には以下の2種類があります。

### 要素識別番号

#### システム識別番号(負の値)

システムが全要素に対し、自動的に付加します。

#### ユーザ識別番号(正の値)

ユーザコマンドで必要なときに付加します。

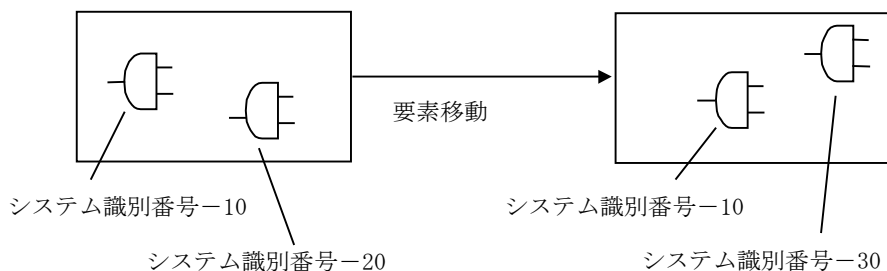
要素識別番号は以下に示す要素作成プログラムで指定できます。

```
sscert2_(&type,&visi,&disp,&rcode);  
)  
ssend2_(&segno,&mode,&isegno,&rcode);
```

↑ ユーザ識別番号 (in)      ↑ システム識別番号 (out)

### (1) システム識別番号

システム識別番号は、要素を識別するためシステムが自動的に要素に付加する負の値です。ssend2\_の isegno の引数の領域に返ってきます。要素を移動させることにより、同じ要素でもシステム識別番号は異なった値となります。

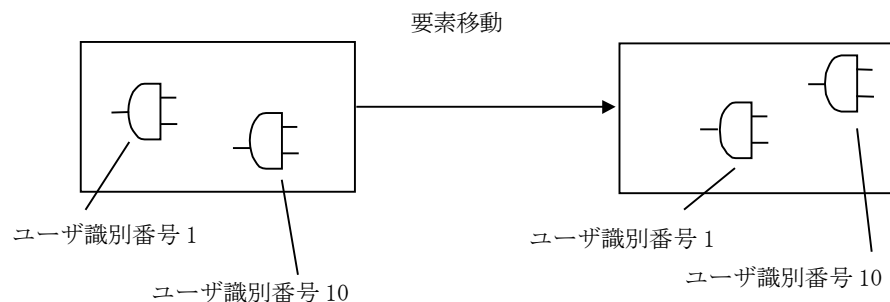


上左図で右側の要素（システム識別番号-20）を移動する場合、オペレータは右側の要素を指示します。すると、システム識別番号-20の要素が指示されたと認識され、システム識別番号-20の要素を移動するという指示を送ります。移動することで、システム識別番号は-20ではなく上右図のように別の値が付加されます。

### (2) ユーザ識別番号

ユーザ識別番号は、要素作成時に任意に正の値(1以上の整数)を付けることができ、ssend2\_の segno の引数で指定します。別の要素に同じユーザ識別番号を付加することはできません。なお、ユーザ識別番号の登録個数は最大 1000000 です。

ユーザ識別番号は永久に保証され、その値は変わることがありません。



上左図で右側の要素（ユーザ識別番号 10）を移動する場合、オペレータは右側の要素を指示します。するとユーザ識別番号 10 の要素が指定されたと認識され、ユーザ識別番号 10 の要素を移動するという指示を送ります。移動後もユーザ識別番号は 10 のままとなります。

ユーザ識別番号は、要素作成時に図形処理ライブラリにより付加したり、一度付加した値を変更することもできます。さらに、ユーザ識別番号を付加していなかった要素に対して新しくユーザ識別番号を付加することや、ユーザ識別番号をつけない要素にすることもできます。

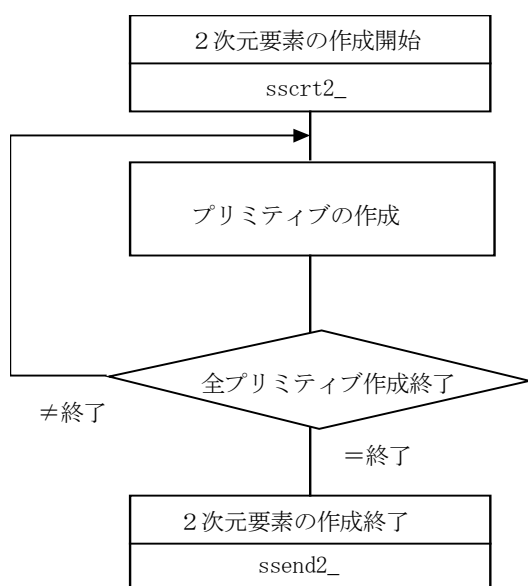
ただし、ユーザ識別番号は付加した番号をユーザコマンドで管理する必要があり、そのためプログラムの複雑性が増すことになります。ユーザ識別番号は必要性を認識した上で利用してください。

## 2次元要素

図形処理ライブラリを利用してプリミティブを作成し、それを1つ又は複数個まとめて2次元要素とします。2次元要素の作成は、2次元要素の作成開始プログラムで始まり、作成終了プログラムの呼び出しで終わります。

この間に任意のプリミティブを作成すると、それを2次元要素として登録することができます。

各2次元要素には要素タイプ番号を指定することができ、その要素タイプ番号により、2次元要素の種類を識別することができます。要素タイプ番号は1～255を利用します。1～100、201～255はシステムで使用し、ユーザセグメントを作成する場合は101～200を使用します。



2次元要素を作成するには、2次元要素の作成開始を宣言し、その2次元要素を構成するプリミティブを必要なだけ作成し、2次元要素の作成終了を宣言します。

### (1) 注意

1つの2次元要素内には、最大255個のプリミティブを作成できます。

2次元要素の大きさは最大64KBです。

2次元要素の作成は開始宣言 (sscr2\_) で始まり、終了宣言 (ssend2\_) で終わります。

sscr2\_～ssend2\_の間でエラーが生じた場合、必ずssend2\_を呼ばなければなりません。

### (2) 画面の表示

作成した2次元要素を図面格納メモリに格納すると同時に、画面上に表示するか否かを指定できます。

表示中レイヤの2次元要素のみ表示されます。

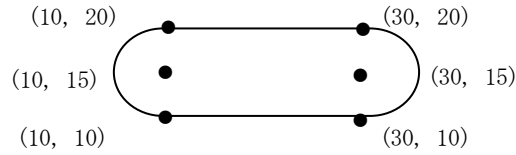
画面上への表示はssend2\_のタイミングで実行されます。

### (3) 作成例

図形処理ライブラリを用いて図形を作成する例を示します。

## ① 2次元要素作成例 1

次に示す図形を作成します。



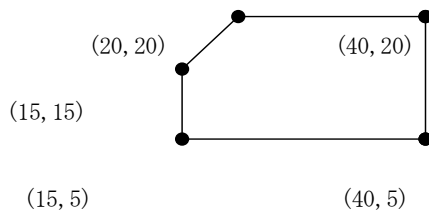
```

type=101;visi=1;disp=1;                                2次元要素作成開始宣言
ssert2_(&type,&visi,&disp,&rcode);
xstart=10.0;ystart=20.0;xdir=1.0;ydir=0.0;length=20.0;atrid=0;mode=1; 線プリミティブの作成
splin2_(&xstart,&ystart,&xdir,&ydir,&length,&atrid,&mode,&rcode);
xstart=10.0;ystart=10.0;xdir=1.0;ydir=0.0;length=20.0;atrid=0;mode=1; 線プリミティブの作成
splin2_(&xstart,&ystart,&xdir,&ydir,&length,&atrid,&mode,&rcode);
x=10.0;y=15.0;r=5.0;th1=PAI/2.0;th2=PAI;atrid=0;mode=1;      弧プリミティブの作成
sparc2_(&x,&y,&r,&th1,&th2,&atrid,&mode,&rcode);
x=30.0;y=15.0;r=5.0;th1=PAI/2.0;th2=-PAI;atrid=0;mode=1;     円弧プリミティブの作成
sparc2_(&x,&y,&r,&th1,&th2,&atrid,&mode,&rcode);
segno=0;mode=0;                                              2次元要素作成終了宣言
ssend2_(&segno,&mode,&isegno,&rcode);

```

## ② 2次元要素作成例 2

四角形の左上を面取りした 2次元要素を作成します。



```

type=102;visi=1;disp=1;                                2次元要素作成開始宣言
ssert2_(&type,&visi,&disp,&rcode);
xstart=20.0;ystart=20.0;xdir=1.0;ydir=0.0;length=20.0;atrid=0;mode=1;
splin2_(&xstart,&ystart,&xdir,&ydir,&length,&atrid,&mode,&rcode);
xstart=40.0;ystart=5.0;xdir=0.0;ydir=1.0;length=15.0;atrid=0;mode=1;
splin2_(&xstart,&ystart,&xdir,&ydir,&length,&atrid,&mode,&rcode);
xstart=15.0;ystart=5.0;xdir=1.0;ydir=0.0;length=25.0;atrid=0;mode=1;
splin2_(&xstart,&ystart,&xdir,&ydir,&length,&atrid,&mode,&rcode);
xstart=15.0;ystart=5.0;xdir=0.0;ydir=1.0;length=10.0;atrid=0;mode=1;
splin2_(&xstart,&ystart,&xdir,&ydir,&length,&atrid,&mode,&rcode);
xstart=15.0;ystart=15.0;xdir=0.71;ydir=0.71;length=7.07;atrid=0;mode=1;
splin2_(&xstart,&ystart,&xdir,&ydir,&length,&atrid,&mode,&rcode);
segno=0;mode=0;
ssend2_(&segno,&mode,&isegno,&rcode);

```

線プリミティブの作成

2次元要素作成終了宣言



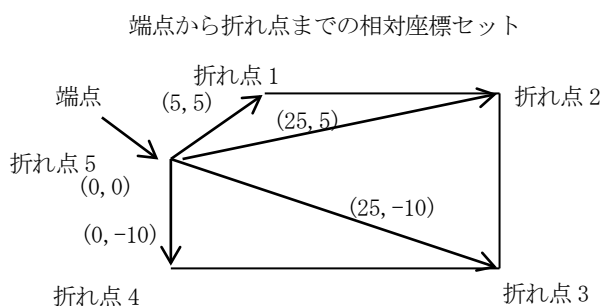
このように2次元要素を線プリミティブで作成できます。また、次のように折れ線プリミティブでも作成できます。

```

type=102;visi=1;disp=1;                                2次元要素作成開始宣言
sscr2_(&type,&visi,&disp,&ircode);

xarray[0]=5.0;yarray[0]=5.0;xarray[1]=25.0;yarray[1]=5.0;xarray[2]=25.0;yarray[2]=-10.0;
xarray[3]=0.0;yarray[3]=-10.0;xarray[4]=0.0;yarray[4]=0.0;
xstart=15.0;ystart=15.0;numpt=5;incrm=1;atrid=0;mode=1;
sply2_(&xstart,&ystart,xarray,yarray,&numpt,&incrm,&atrid,&mode,&ircode);
segno=0;mode=0;                                          折れ線プリミティブの作成
ssend2_(&segno,&mode,&isegno,&ircode);                  2次元要素作成終了宣言

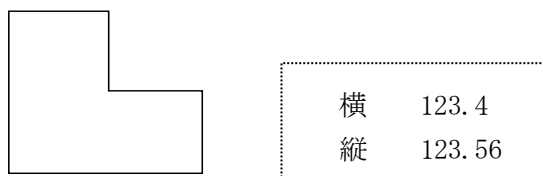
```



2次元要素の構成プリミティブはコマンド開発者が決定します。

### ③ 任意数値プリミティブ作成例

2次元要素を作成する時に、画面上には表示されませんが、部品サイズなどを付加情報として一緒に覚えておくことができます。これは、任意数値プリミティブを2次元要素作成時に一緒に付加すればよく、この手順について以下に例を示します。



```

type=103;visi=1;disp=1;                                2次元要素作成開始宣言
sscr2_(&type,&visi,&disp,&ircode);

プリミティブ作成処理 (splin2_等)

form=1;numarry[0]=123.4;numarry[1]=123.56;n=16;          任意数値プリミティブ作成
spauxn_(&form,numarry,&n,&ircode);

segno=0;mode=0;                                          2次元要素作成終了宣言
ssend2_(&segno,&mode,&isegno,&ircode);

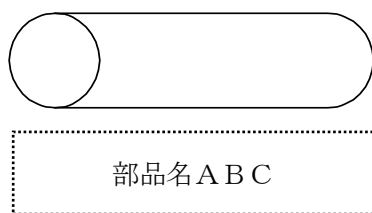
```

#### 注意

任意数値プリミティブは必ず非表示で、画面には表示されません。内容は8バイト単位です。  
spdget\_により、任意数値プリミティブの内容を見ることができます。

#### ④ 任意文字プリミティブ作成例

2次元要素を作成する時に、画面上には表示されませんが、部品名などを付加情報として一緒に覚えておくことができます。これは、任意文字プリミティブを2次元要素作成時に一緒に付加すればよく、この手順について以下に例を示します。



```
type=104;visi=1;disp=1;  
sscr2_(&type,&visi,&disp,&rcode);
```

2次元要素作成開始宣言

プリミティブ作成処理 (splin2\_等)

```
form=1;txtarry="ABC";n=8;  
spauxt_(&form,txtarry,&n,&rcode);  
segno=0;mode=0;  
ssend2_(&segno,&mode,&isegno,&rcode);
```

任意文字プリミティブ作成

2次元要素作成終了宣言

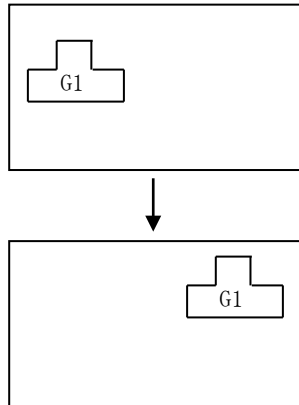
#### 注意

任意文字プリミティブは必ず非表示属性で、画面には表示されません。内容は8バイト単位です。  
spdget\_により、任意文字プリミティブの内容を見ることができます。

# グループ

## ■ グループの概要

要素群をグループ化することにより、移動コマンド等でグループ単位の処理ができます。

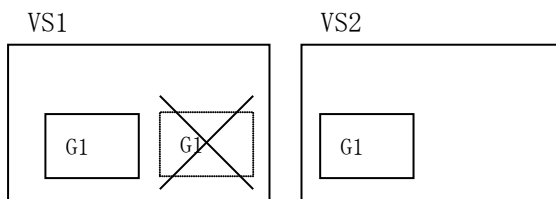


- ① 作成した図形をコマンドメニューの「グループ」コマンドでグループ化します。
- ② 「移動」コマンドを選択します。
- ③ 入力制御メニューの「グループ」で画面上の図形をヒットします。
- ④ 移動先を指定します。
- ⑤ グループ化された図形が指定された場所へ移動します。

グループの特徴として以下のものがあげられます。

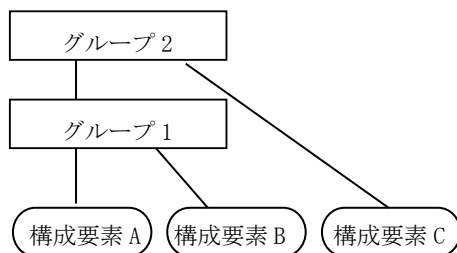
### (1) グループの名前

グループにはそれぞれ名前があり、1つのVS内に同一名のグループをもつことはできません。ただし、他のVS上であれば可能です。



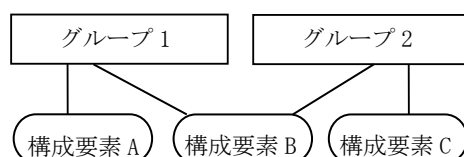
### (2) グループの階層化

以下に示すようなグループの階層化はできません。



### (3) 構成要素の重複

1つの要素は複数のグループに属することができます。



(4) グループの数

グループの数は1つのVSあたり500個までです。

(5) グループの構成要素数

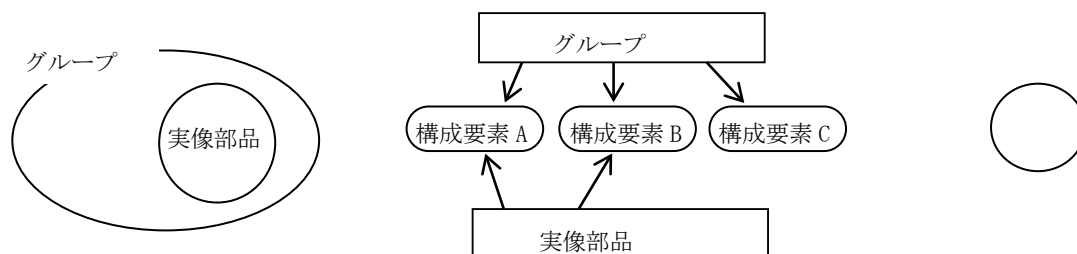
構成要素数に制限はありません。VSの中に存在するすべての要素をグループの構成要素とすることができます。

(6) グループの構成要素

2次元要素、配置子図、写像部品

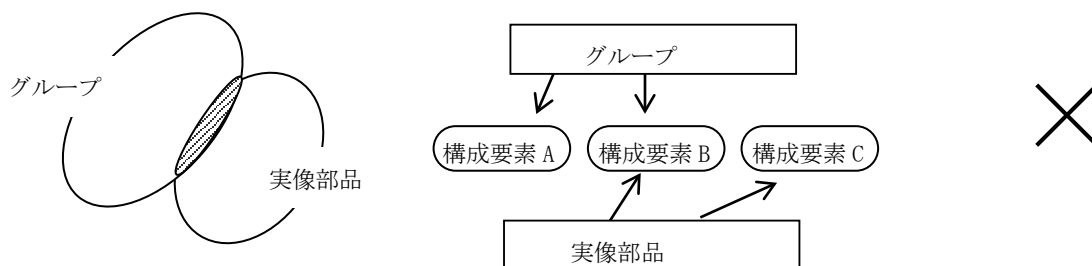
(7) グループと実像部品の関係

グループと実像部品は以下の関係だけが成り立ちます。

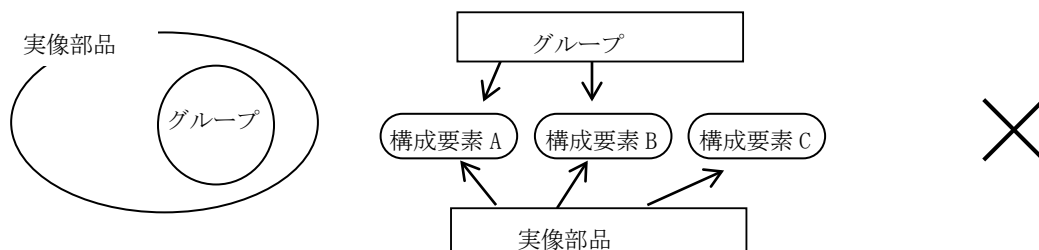


グループと実像部品で以下の関係は成り立ちません。

- ・グループと実像部品で一部同じ構成要素を共有

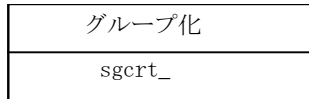


- ・実像部品構成要素 > グループ構成要素



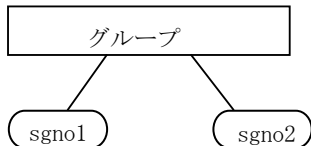
## ■ グループ作成方法

### (1) グループ化



指定された要素群をグループ化します。

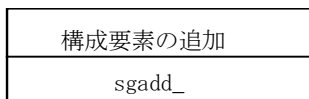
例：要素識別番号が sgn01 の 2 次元要素と sgn02 の 2 次元要素をグループ化します。



```

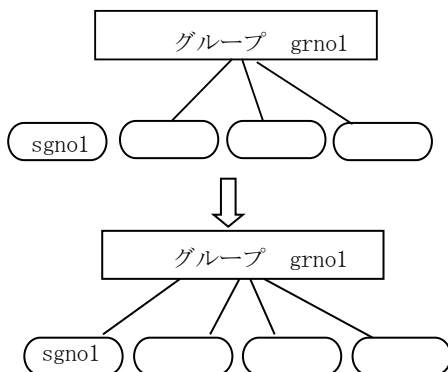
entids[0]=sgn01;entids[1]=sgn02;
grpid=0;n=2;dim=2;
sgcrt_(&grpid,entids,&n,&dim,&entid,&rcode);
  
```

### (2) 構成要素の追加



1 度作成したグループに対して、要素を追加することができます。

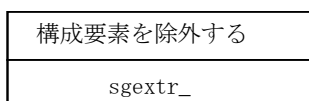
例：グループの要素識別番号が grn01 のグループに要素識別番号が sgn01 をもつ要素を追加します。



```

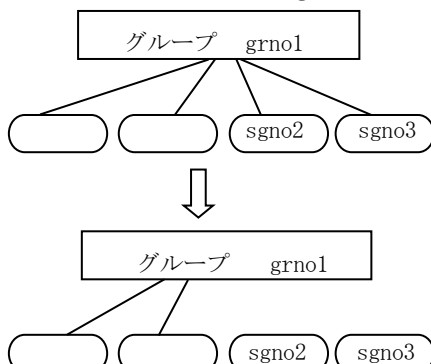
grpid=grn01;entids=sgn01;n=1;
sgadd_(&grpid,&entids,&n,&entid,&rcode);
  
```

### (3) 構成要素の除外



1 度作成したグループに対して、要素を除外することができます。

例：要素識別番号が sgn02 をもつ要素と、sgn03 を持つ要素を除外します。



```

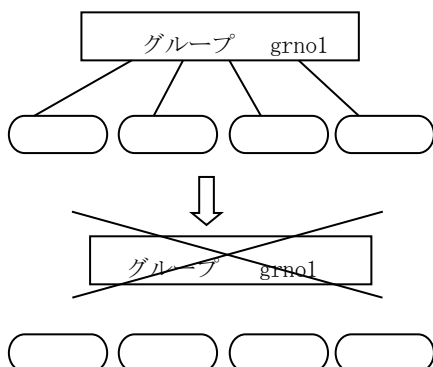
grpid=grn01;entids[0]=sgn02;entids[1]=sgn03;n=2;
sgextr_(&grpid,entids,&n,&rcode);
  
```

#### (4) グループの解除



グループは、グループの要素識別番号を指定して解除することができます。

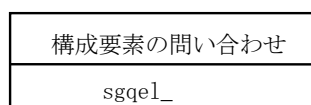
例：グループの要素識別番号が grno1 であるグループを解除します。



```
grpid=grno1;  
sgfree_(&grpid,&rcode);
```

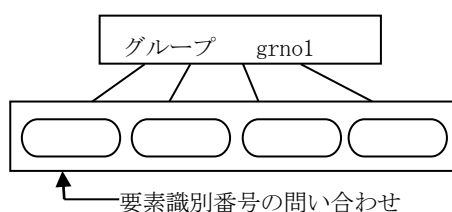
このとき、グループは削除されます。

#### (5) 構成要素の問い合わせ



グループの要素識別番号を指定して構成要素の要素識別番号を問い合わせることができます。

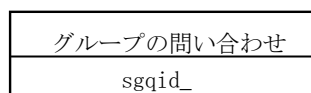
例：要素識別番号が grno1 を持つグループの構成要素である要素識別番号を問い合わせます。



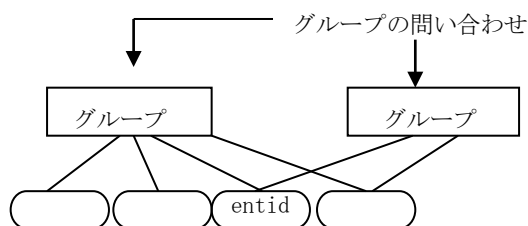
```
grpid=grno1;size=10;start=1;
```

グループ内の構成要素を指定してグループの要素識別番号を問い合わせることができます。

#### (6) グループの問い合わせ



例：要素識別番号が entid である要素が属するグループの要素識別番号を問い合わせます。

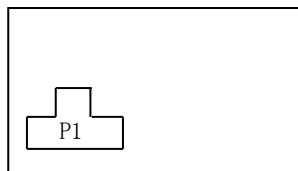


```
segid=entid;  
sgqid_(&segid,grpids,&size,&rcode);
```

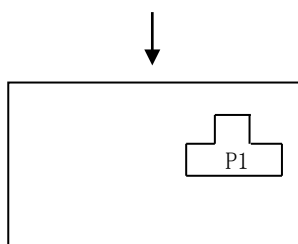
# 実像部品

## ■ 実像部品の概要

要素群を実像部品化することにより、移動コマンド等で実像部品単位の処理ができます。



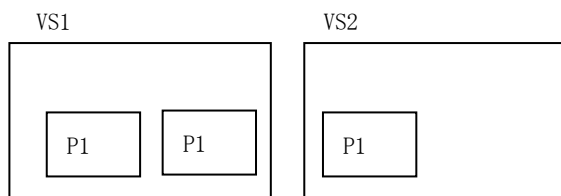
- ① 「移動」 コマンドを選択します。
- ② 画面上の図形をヒットします。
- ③ 移動先を指定します。
- ④ 実像部品化された図形が指定された場所へ移動します。



実像部品の特徴として以下のものがあげられます。

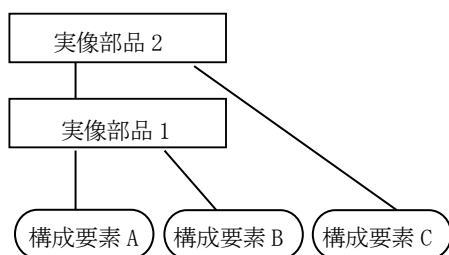
### (1) 実像部品の名前

実像部品にはそれぞれ名前があり、1 つの VS 内に同一名の実像部品を複数もつことができます。



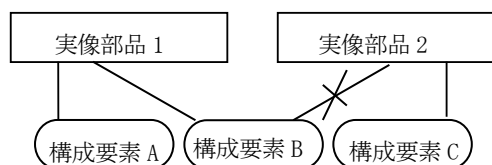
### (2) 実像部品の階層化

以下に示すような実像部品の階層化はできません。



### (3) 構成要素の重複

1 つの要素は複数の実像部品に属することはできません。



(4) 実像部品の数

実像部品の数に制限はありません。

(5) 実像部品の構成要素数

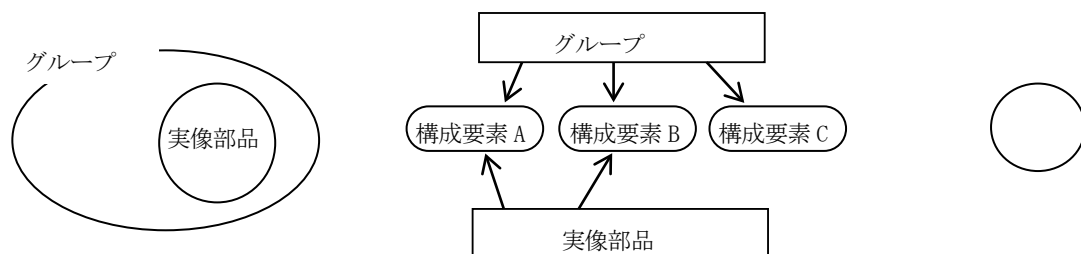
構成要素数に制限はありません。VS の中に存在するすべての要素を実像部品の構成要素とすることができます。

(6) 実像部品の構成要素

2次元要素

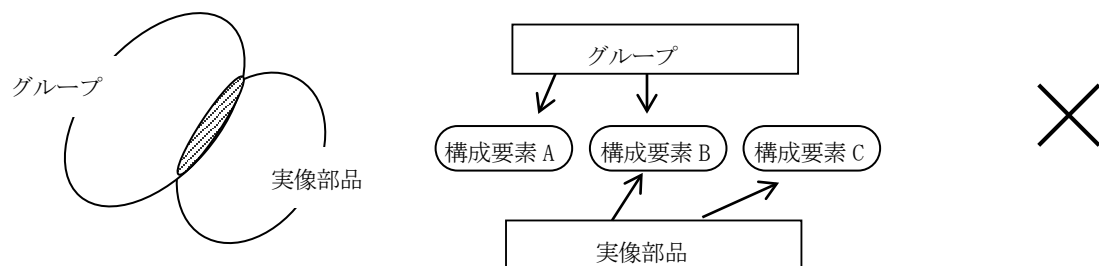
(7) グループと実像部品の関係

グループと実像部品は以下の関係だけが成り立ちます。

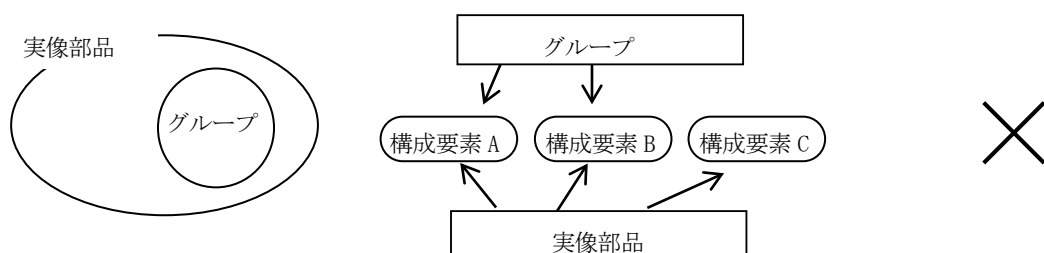


グループと実像部品で以下の関係は成り立ちません。

- ・グループと実像部品で一部同じ構成要素を共有



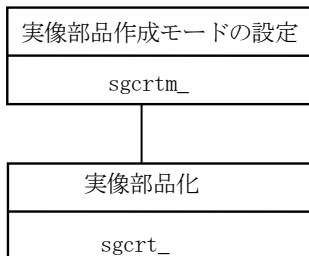
- ・実像部品構成要素 > グループ構成要素





## ■ 実像部品作成方法

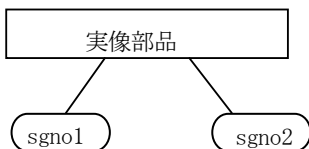
### (1) 実像部品化



実像部品作成モードを設定します。  
設定しなかった場合、グループ作成モードとなります。

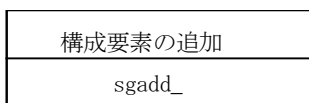
指定された要素群を実像部品化します。

例：要素識別番号が sgn01 の 2 次元要素と sgn02 の 2 次元要素を実像部品化します。



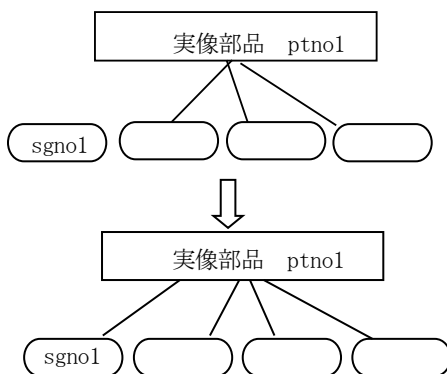
```
mode=1;
sgcrtm_(&mode,&rcode);
entids[0]=sgn01;entids[1]=sgn02;grpid=0;n=2;dim=2;
sgcrt_(&grpid,entids,&n,&dim,&ientid,&rcode);
```

### (2) 構成要素の追加



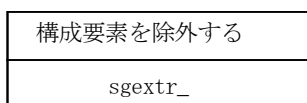
1 度作成した実像部品に対して、要素を追加することができます。

例：実像部品の要素識別番号が ptn01 の実像部品に要素識別番号が sgn01 をもつ要素を追加します。



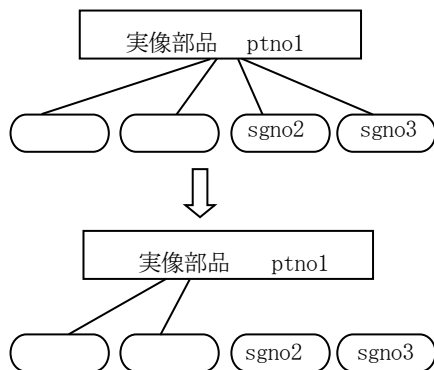
```
prtid=ptn01;entids=sgn01;n=1;
sgadd_(&prtid,&entids,&n,&ientid,&rcode);
```

### (3) 構成要素の除外



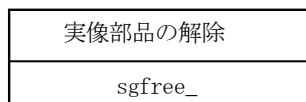
1 度作成した実像部品に対して、要素を除外することができます。

例：要素識別番号が sgn02 をもつ要素と、sgn03 を持つ要素を除外します。



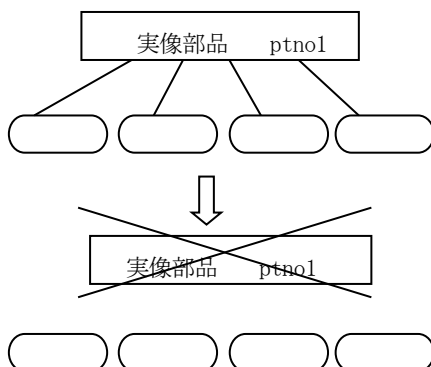
```
prtid=ptn01;entids[0]=sgn02;entids[1]=sgn03;n=2;
sgextr_(&prtid,entids,&n,&rcode);
```

#### (4) 実像部品の解除



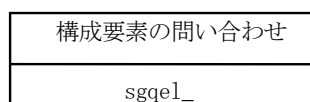
実像部品は、実像部品の要素識別番号を指定して解除することができます。

例：実像部品の要素識別番号が ptn01 である実像部品を解除します。



このとき、実像部品は削除されます。

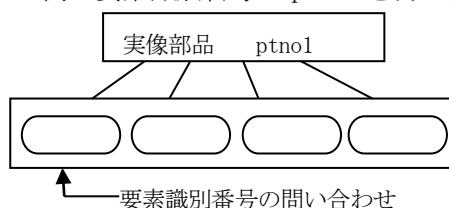
#### (5) 構成要素の問い合わせ



```
prtid=ptn01;
sgfree_(&prtid,&rcode);
```

実像部品の要素識別番号を指定して構成要素の要素識別番号を問い合わせることができます。

例：要素識別番号が ptn01 を持つ実像部品の構成要素である要素識別番号を問い合わせます。



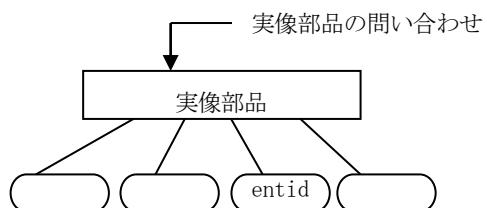
```
prtid=ptn01;size=10;start=1;
```

(6) 実像部品の問い合わせ

実像部品の問い合わせ
sgqids_

実像部品内の構成要素を指定して実像部品の要素識別番号を問い合わせることができます。

例：要素識別番号が entid である要素が属する実像部品の要素識別番号を問い合わせます。



```
segid=entid;  
sgqids_(&segid, grpid, &n, &rcode);
```

## 属性管理

2次元の線要素を作成する場合でも、線種は実線か点線か、色は緑か白か、線の太さは太いか細いかなどの形式があり、これらの形式を属性といいます。ここでは、属性とはどういうものか、また、属性管理をする場合の図形処理ライブラリの利用方法について説明します。

### (1) 属性

属性には、図形属性、文字属性があり、これはプリミティブに対して設定することができます。このため、2個以上のプリミティブからなる2次元要素の場合は、プリミティブ毎に属性を設定、変更することができます。図形属性、文字属性を管理する領域を属性テーブルといい、属性テーブルに設定された属性値で、要素（プリミティブ）が作成されます。

#### ① 属性の種類

#### ② 属性テーブルの形式と使用方法

について述べます。

#### ① 属性の種類





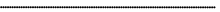
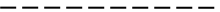
##### ・ 図形属性

図形属性には次の3種類があります。

#### 線 幅

線幅コード	線の太さ	形状
1	太 線	
2 (初期値)	中 線	
3	細 線	

#### 線 種

線種コード	線 種	表示形式
1 (初期値)	実 線	
2	破 線	
3	一点鎖線	
4	二点鎖線	
5	点 線	
6	長 破 線	

## 線の色

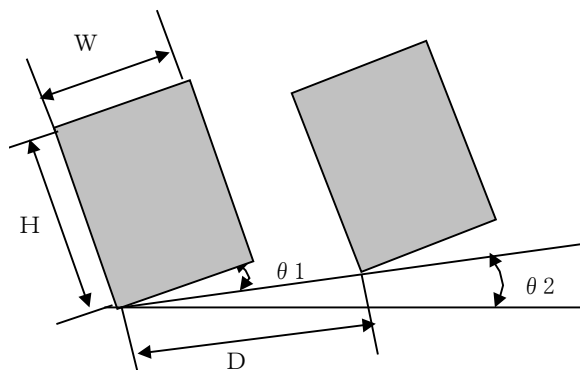
色コード	色
1	白
2	赤
3(初期値)	緑
4	黄
5	青
6	マゼンダ
7	シアン
8	灰色
9	橙色
10	薄緑
11	黄緑
12	空色
13	紫
14	青緑
15	薄橙
16	黒
17~31	拡張色 1~15

既に作成されているプリミティブの属性を問い合わせるためには `sqprm_` を利用します。

### ・ 文字属性

文字属性には次の 5 種類があります。(線種は実線のみ)

文字の高さ	.....	H	(初期値 3.2mm)
文字の幅	.....	W	(初期値 2.75mm)
文字の間隔	.....	D	(初期値 3.2mm)
文字の傾き (角)	.....	$\theta 1$	(初期値 0.0 度)
文字列の傾き (度)	.....	$\theta 2$	(初期値 0.0 度)



## ②属性テーブル

### ・ 図形属性テーブル

図形属性テーブルは全部で4つあります。基本コマンドではテーブル1を使用しており、これを参照してカレントの線の太さ、種類、色で要素を作成することができます。ユーザコマンドではテーブル0、2、3に設定して要素を作成することができます。

1	線の太さ	線の太さ	線の太さ	線の太さ
2	線の種類	線の種類	線の種類	線の種類
3	色	色	色	色
4	0を指定する	0を指定する	0を指定する	0を指定する
	ユーザ用 0	システム用 1	ユーザ用 2	ユーザ用 3

テーブル番号は左から0、1、2、3となります。

### ・ 文字属性テーブル

文字属性テーブルは全部で4つあります。ユーザコマンドではテーブル0、1、2、3に設定して要素を作成することができます。

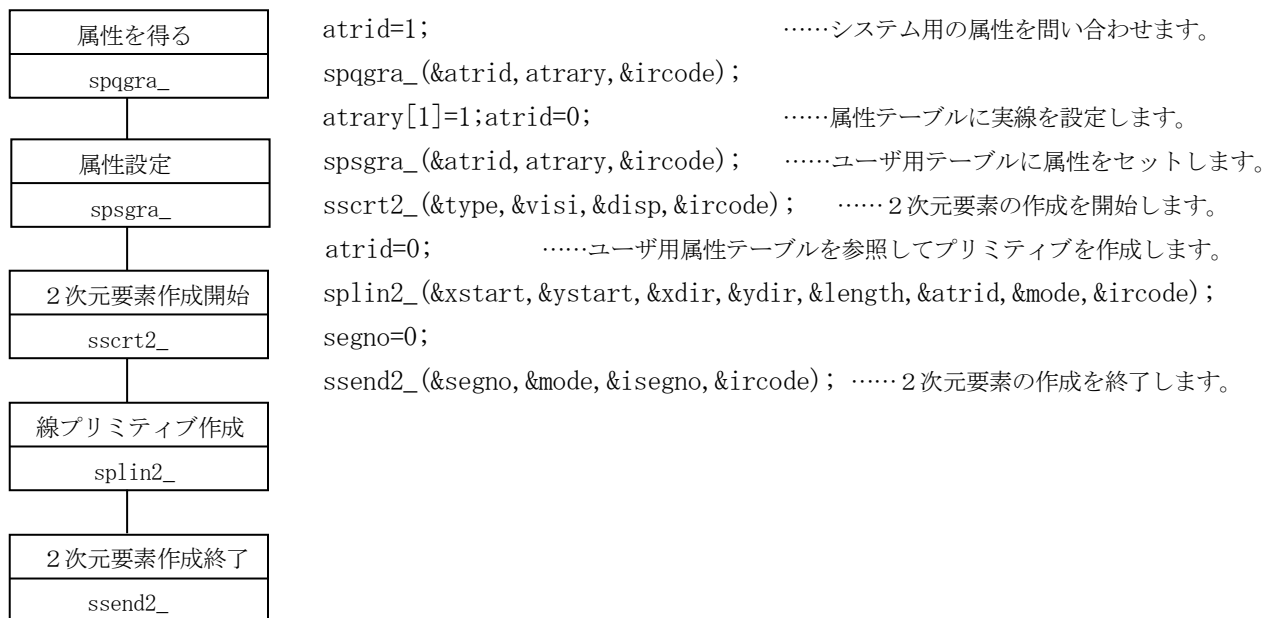
1	文字の高さ	文字の高さ	文字の高さ	文字の高さ
2	文字の幅	文字の幅	文字の幅	文字の幅
3	文字の間隔	文字の間隔	文字の間隔	文字の間隔
4	文字の傾き	文字の傾き	文字の傾き	文字の傾き
5	文字列の傾き	文字列の傾き	文字列の傾き	文字列の傾き
6	文字の色	文字の色	文字の色	文字の色
7	文字の線幅	文字の線幅	文字の線幅	文字の線幅
	ユーザ用 0	ユーザ用 1	ユーザ用 2	ユーザ用 3

テーブル番号は左から0、1、2、3となります。

## (2) 属性設定、問い合わせ

ユーザコマンドで属性テーブルに設定したり、現在の属性値を問い合わせる図形処理ライブラリの利用方法を説明します。

例：実線の線プリミティブで構成される 2 次元要素を作成します。



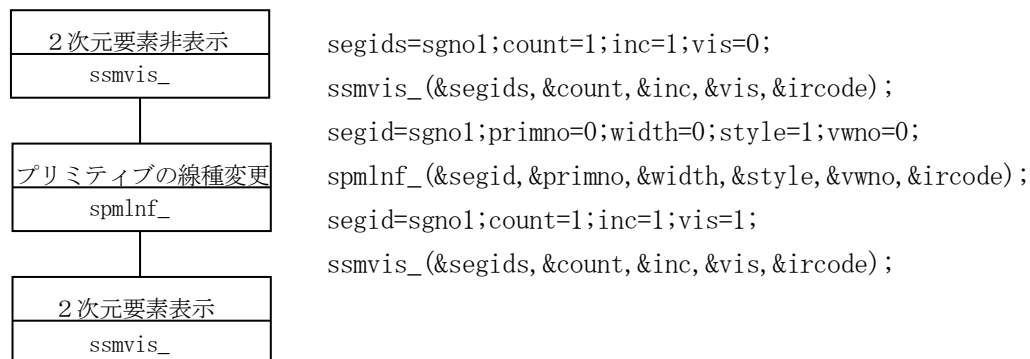
作成、編集された 2 次元要素の表示、検索属性は、spqprm\_により問い合わせます。

## (3) 属性変更

プリミティブの属性を変更する場合のプログラムを以下に示します。

- spmcol\_ プリミティブの線色変更
- spmlnf\_ プリミティブの線種変更

例：要素識別番号が sgno1 である 2 次元要素の全プリミティブの線種を実線にします。



### 注意

属性を変更する場合は、2 次元要素を非表示にした方がよいでしょう。表示にしたまま属性を変更すると spmlnf\_/spmcol\_が呼び出されるたびに画面に表示され、無駄な処理が多くなります。

## 要素編集

既に作成されている要素に対して、移動、回転などの図形配置変更やコピー、削除、表示属性変更などをする場合の図形処理ライブラリの利用方法について説明します。

編集の対象となる要素は、要素識別番号によって指示します。要素識別番号はシステムにより、プロセスに通知されるのでこれを利用します。

### (1) 要素の削除

不用となった要素を削除する場合は、要素識別番号をキーにして要素を削除します。

要素の削除
semdl_t_

#### 注意

2次元要素を指定したとき、指定要素のみが削除されます。

削除した2次元要素が表示レイヤであれば、消去されます。

例：要素識別番号が sgno1、sgno2 の2次元要素を削除します。

```
entids[0]=sgno1;entids[1]=sgno2;nument=2;incrm_t=1;mode=0;  
semdl_t_(entids,&nument,&incrm_t,&mode,&rcode);
```

### (2) 要素の移動、回転、ミラー

要素識別番号を指定して、要素の移動、回転、ミラーが行えます。

#### 注意

任意数値、任意文字プリミティブは対象外です。


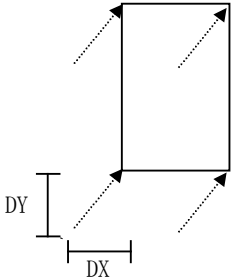
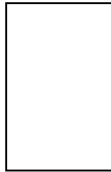
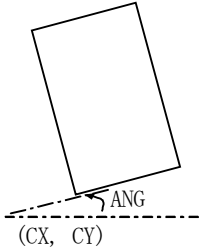
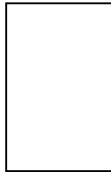
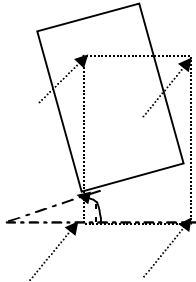
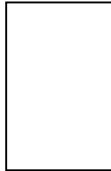
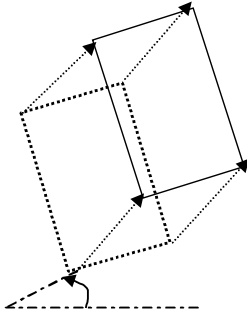
2次元要素を指定したとき、指定の要素のみが処理されます。

#### ① 移動、回転

要素の移動・回転
semr2_

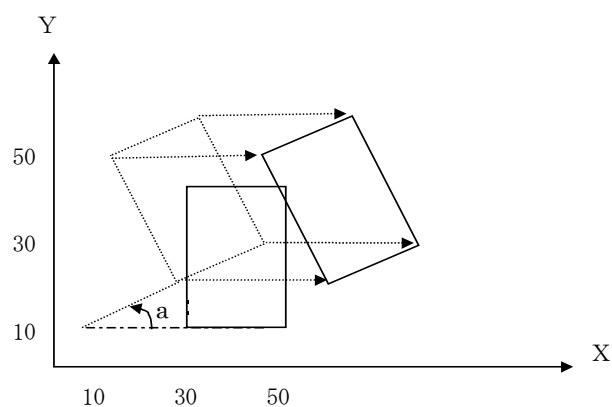
semr2\_などを用いて処理する場合、平行移動、回転移動、平行移動して回転移動、回転移動して平行移動の4種類の処理手順を指定することができます。



処理手順	MODE の値	処理前の図形	処理後の図形
平行移動	1		
回転移動	2		
平行移動して 回転移動	3		
回転移動して 平行移動	4		

例：要素識別番号が、segno である 2 次元要素を以下のように回転移動して平行移動します。

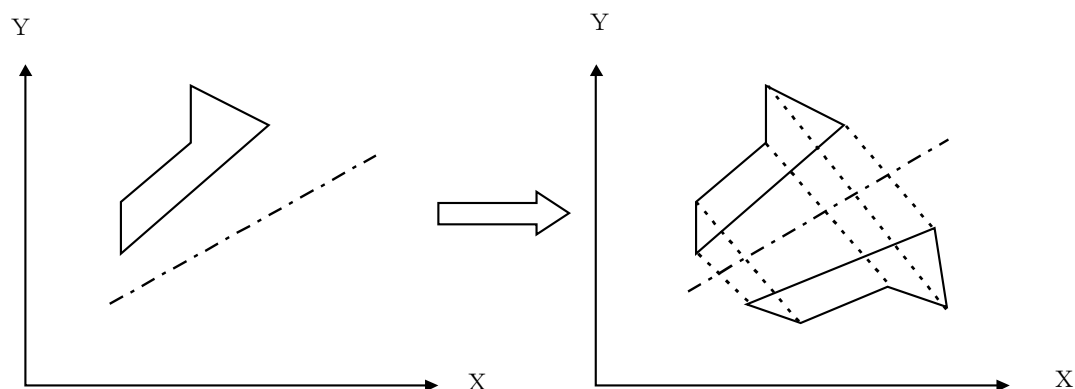
```
entids=segno;nument=1;incrmt=1;mode=4;dx=20.0;dy=0.0;ang=a;cx=5.0;cy=10.0;erase=1;
semtr2_(&entids,&nument,&incrmt,&mode,&dx,&dy,&ang,&cx,&cy,&erase,&rcode);
```



## ②ミラー

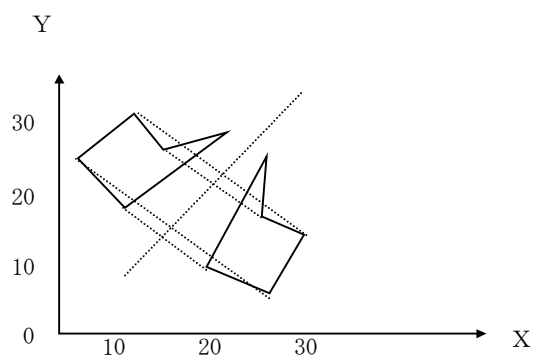
要素のミラー
semmr2_

semmr2\_を用いて、ミラー処理するとき、鏡面となる線分を指定します。



例：要素識別番号が segno である要素を以下のようにミラーします。

```
entids=segno;nument=1;incrmt=1;mx=10.0;my=10.0;mvx=0.71;myv=0.71;erase=1;
semmr2_(&entids,&nument,&incrmt,&mx,&my,&mvx,&myv,&erase,&rcode);
```



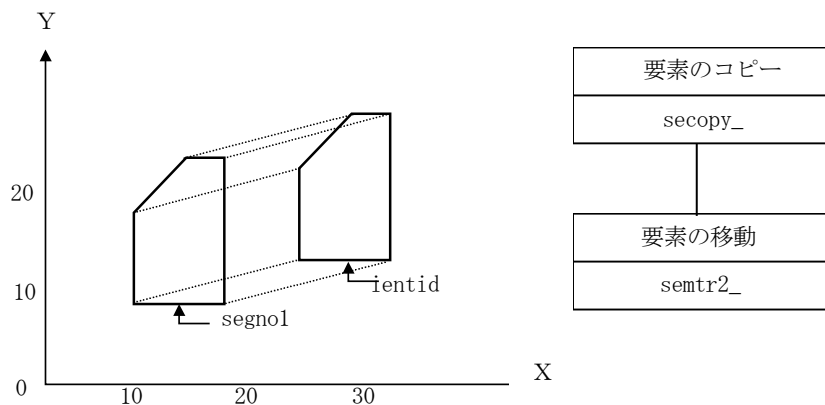
### (3) コピー

要素をコピーをする場合は、コピーする要素識別番号をキーにして、同一レイヤ内にコピーします。

要素のコピー
secopy_

例：要素識別番号が segno1 である要素をコピーし、以下のように移動します。

```
oentid=segno1;nentid=0;mode=1;  
secopy_(&oentid,&nentid,&mode,&ientid,&ircode);  
entids=ientid;nument=1;incrmt=1;mode=1;dx=15.0;dy=5.0;erase=1;  
semt2_(&entids,&nument,&incrmt,&mode,&dx,&dy,&ang,&cx,&cy,&erase,&ircode);
```

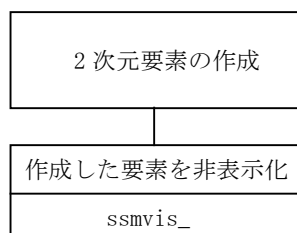


### (4) 表示属性の変更

作成済みの 2 次元要素に対して、表示／非表示属性を変更できます。

- ssmvis\_ 2 次元要素の表示／非表示属性の変更

例：要素識別番号が segno1 である 2 次元要素と segno2 である 2 次元要素を非表示にします。



```
segids[0]=sgno1;segids[1]=sgno2;count=2;inc=1;vis=0;  
ssmvis_(segids,&count,&inc,&vis,&ircode);
```

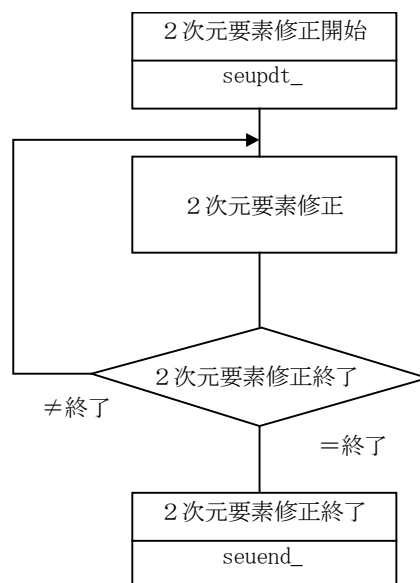
## プリミティブ編集

2次元要素の中のプリミティブを削除したり、追加するなどの変更ができます。2次元要素の中のプリミティブは `spsget_`、`spdget_` により読み込むことができます。  
変更できる項目を以下にあげます。

### プリミティブの変更

<code>seupdt_</code>	要素の修正開始
<code>seuend_</code>	要素の修正終了
<code>seuskp_</code>	プリミティブのスキップ
<code>seucpy_</code>	プリミティブのコピー

ここでは、2次元要素内のプリミティブの追加、削除、置換の方法について以下に示します。2次元要素の修正開始 (`seupdt_`) から、修正終了 (`seuend_`) の間でプリミティブを編集します。



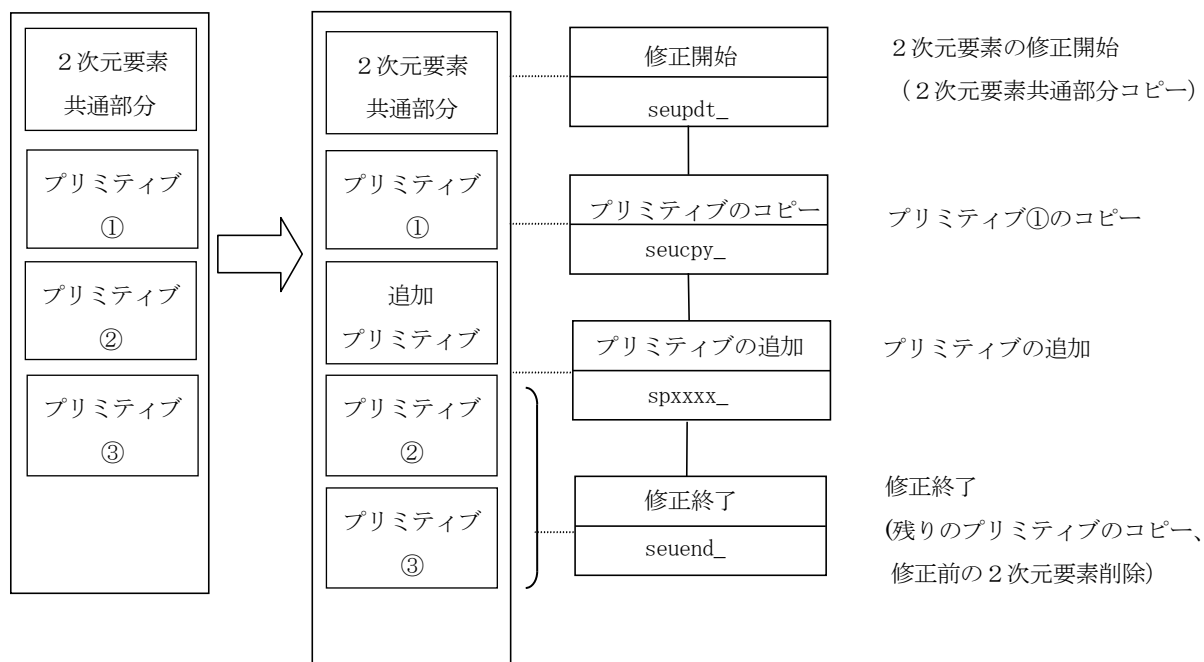
- `seupdt_`では、指定された要素の修正開始を宣言します。
- `seuskp_`では、順に読み込まれたプリミティブを読み飛ばし、修正後の2次元要素に追加しません。
- `seucpy_`では、順に読み込まれたプリミティブを修正後の2次元要素に追加します。
- `seuend_`では、修正前の2次元要素に未処理のプリミティブがあれば、それらをすべて修正後の2次元要素にコピーし、修正前の2次元要素を削除して、修正を終了します。

### 2次元要素を修正する上での注意点

`seupdt_`～`seuend_`の間でエラーが発生した場合、必ず `seuend_` を呼ばなければなりません。この場合、修正したものは無視されます。

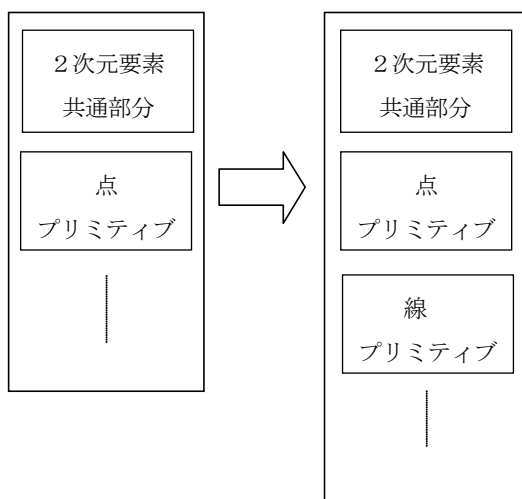
### (1) プリミティブの追加

2次元要素中にプリミティブ追加する場合、2次元要素内のプリミティブをコピーした後、必要なプリミティブを追加します。



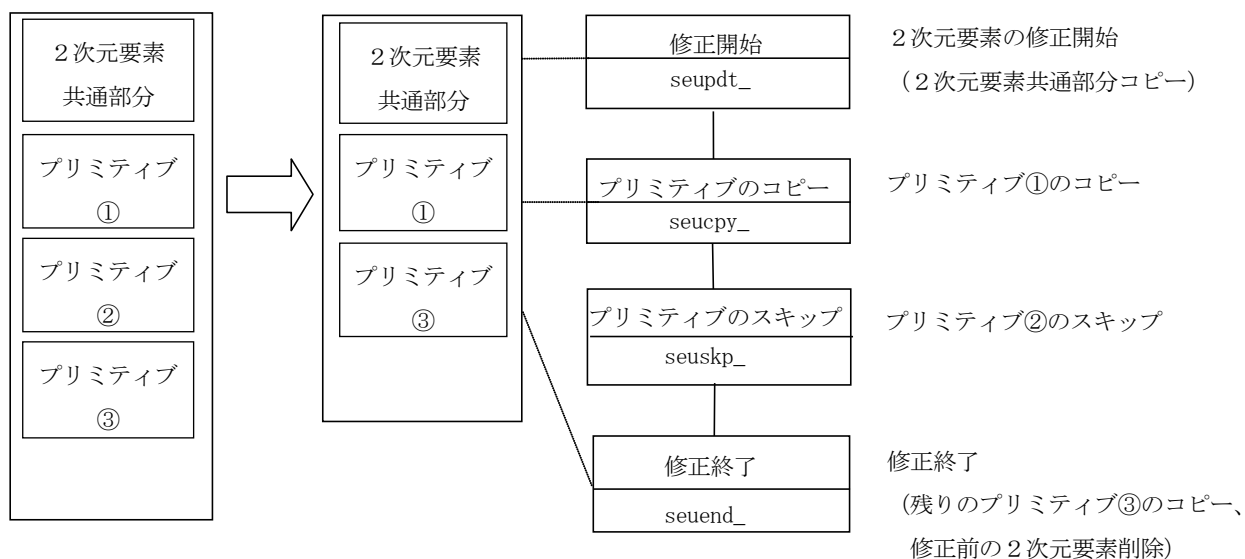
例：要素識別番号が `sgnol` である2次元要素に線プリミティブを追加します。

```
segid=sgnol;type=0;erase=0;
seupdt_(&segid,&type,&erase,&rcode);
seucpy_(&ptype,&rcode);
xstart=10.0;ystart=10.0;xdir=1.0;ydir=0.0;length=5.0;atrid=0;mode=1;
splin2_(&xstart,&ystart,&xdir,&ydir,&length,&atrid,&mode,&rcode);
seuend_(&isegno,&rcode);
```



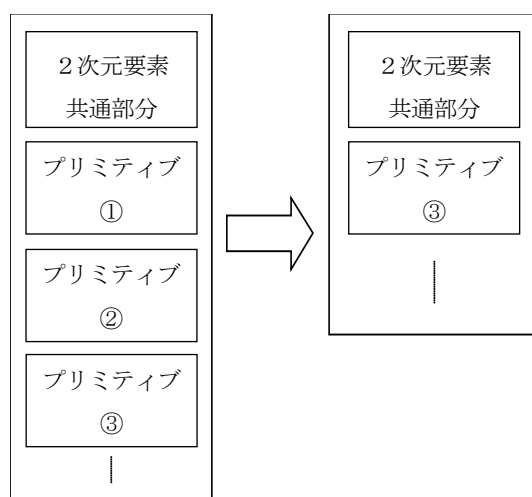
## (2) プリミティブの削除

2次元要素中の不必要なプリミティブを削除する場合、必要な2次元要素内のプリミティブをコピーした後、不必要なプリミティブを削除します。



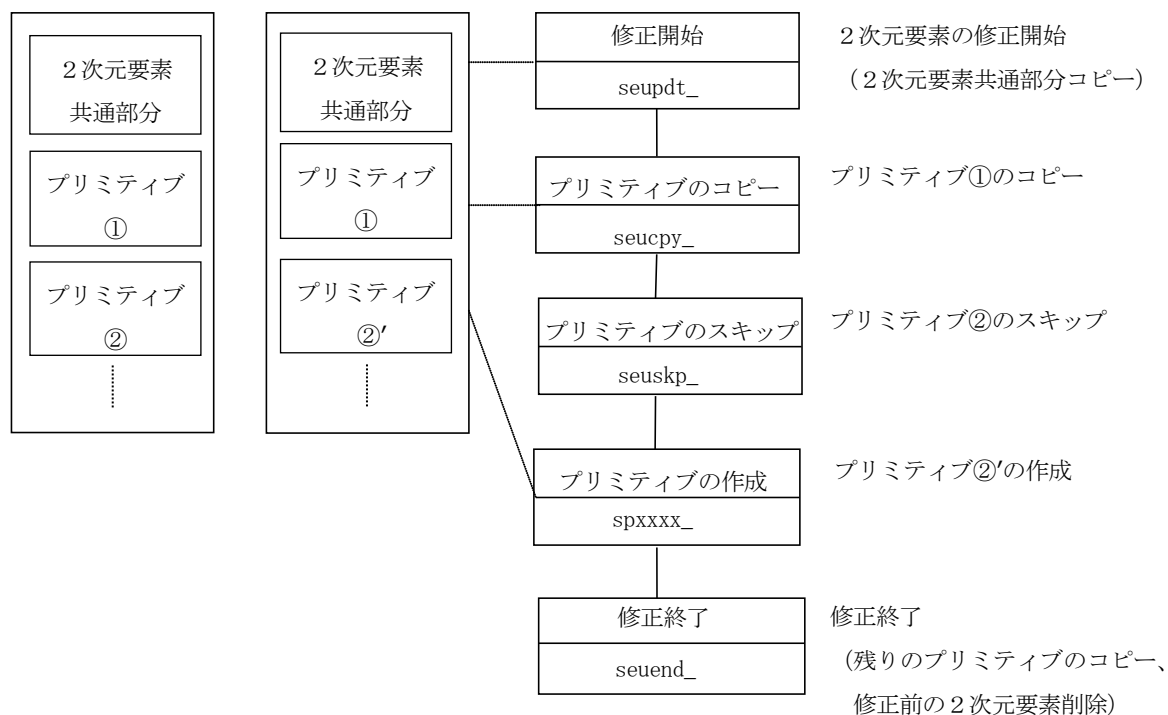
例：要素識別番号がsgno1である2次元要素中のプリミティブ①、プリミティブ②を削除します。

```
segid=sgno1;type=0;erase=0;
seupdt_(&segid,&type,&erase,&ircode);
seuskp_(&ircode);
seuskp_(&ircode);
seuend_(&isegno,&ircode);
```



### (3) プリミティブの置換

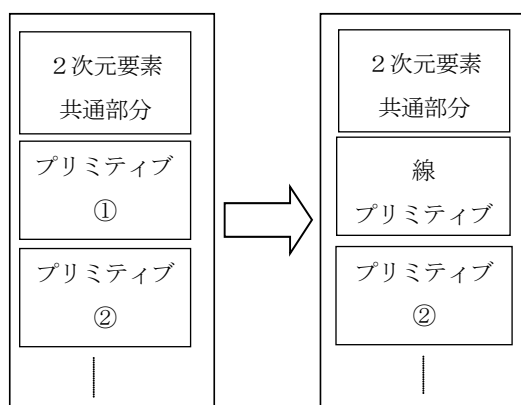
2次元要素中の不必要なプリミティブを新たに作成するプリミティブで置換する場合、不必要なプリミティブを削除した後、必要なプリミティブを作成します。



例：要素識別番号がsgno2である2次元要素中のプリミティブ①を線プリミティブで置換します。

```

segid=sgno2;type=0;erase=0;
seupdt_(&segid,&type,&erase,&rcode);
seuskp_(&rcode);
splin2_(&xstart,&ystart,&xdir,&ydir,&length,&atrid,&mode,&rcode);
seuend_(&isegno,&rcode);
  
```



## 要素・プリミティブ読み込み

図面格納メモリーに保存された要素の情報を得る場合の、図形処理ライブラリの利用方法について説明します。

### (1) 要素情報の読み込み

要素情報を得る	要素情報を得ます。
ssread_	

引数

in \*entid            情報を得る要素の要素識別番号を指定します。

out infary[16]    要素情報の返答領域

    [ 0 ] : 識別コード

          0 = 2次元要素

          2 = グループまたは実像部品

          3 = 配置子図または写像部品

         100 = 削除要素

    [ 1 ] : ユーザ識別番号

          >0 : ユーザ識別番号

          =0 : ユーザ識別番号が未割当

    [ 2 ] : 2次元

          0 = 2次元

    [ 3 ] : 属する VS/WF 番号

    [ 4 ] : レイヤ番号

    [ 5 ] : 要素タイプ番号

    [ 6 ] : 要素の表示属性

          0 = 非表示

          1 = 表示

    [ 7 ] : 未使用

    [ 8 ] : 未使用

    [ 9 ] : 要素分類

          グループまたは実像部品の一部であるか否かを示します。

          0 = どちらにも属しません。

          2 = グループまたは実像部品の一部です。

    [ 10 ] : 未使用

    [ 11 ] : 要素を構成するプリミティブ数

    [ 12 ] : 要素長

    [ 13 ] : 未使用

    [ 14 ] : 未使用



[ 15 ] : 実像部品識別コード(本引数は infary[0]=2 の時のみ有効)

0 = 実像部品ではありません。

1 = 実像部品

例：要素識別番号が sgnol である要素情報を読み込みます。

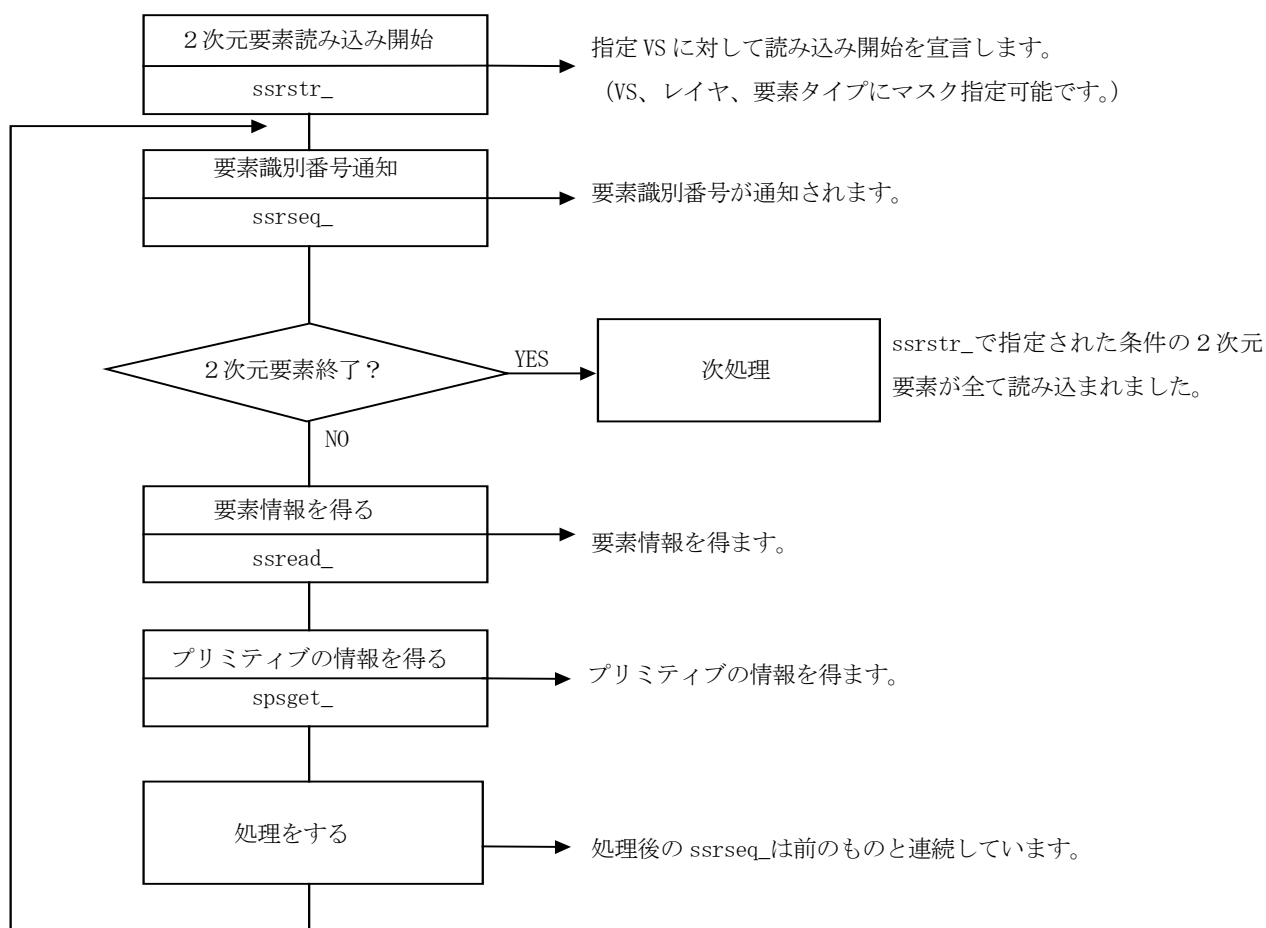
```
entid=sgnol;
```

```
ssread_(&entid, infary);
```

↑ 要素情報が読み込まれる領域

## (2) 要素の読み込み

指定した VS・レイヤ・タイプの要素を順次読み込みます。



ssread\_を呼び出した直後にその要素の削除、コピー、あるいは別の要素を作成することができます。

例：VS 番号が 1、レイヤ番号が 30, 31 の 2 次元要素のうち、タイプが 2（線）である全ての要素の情報を得ます。

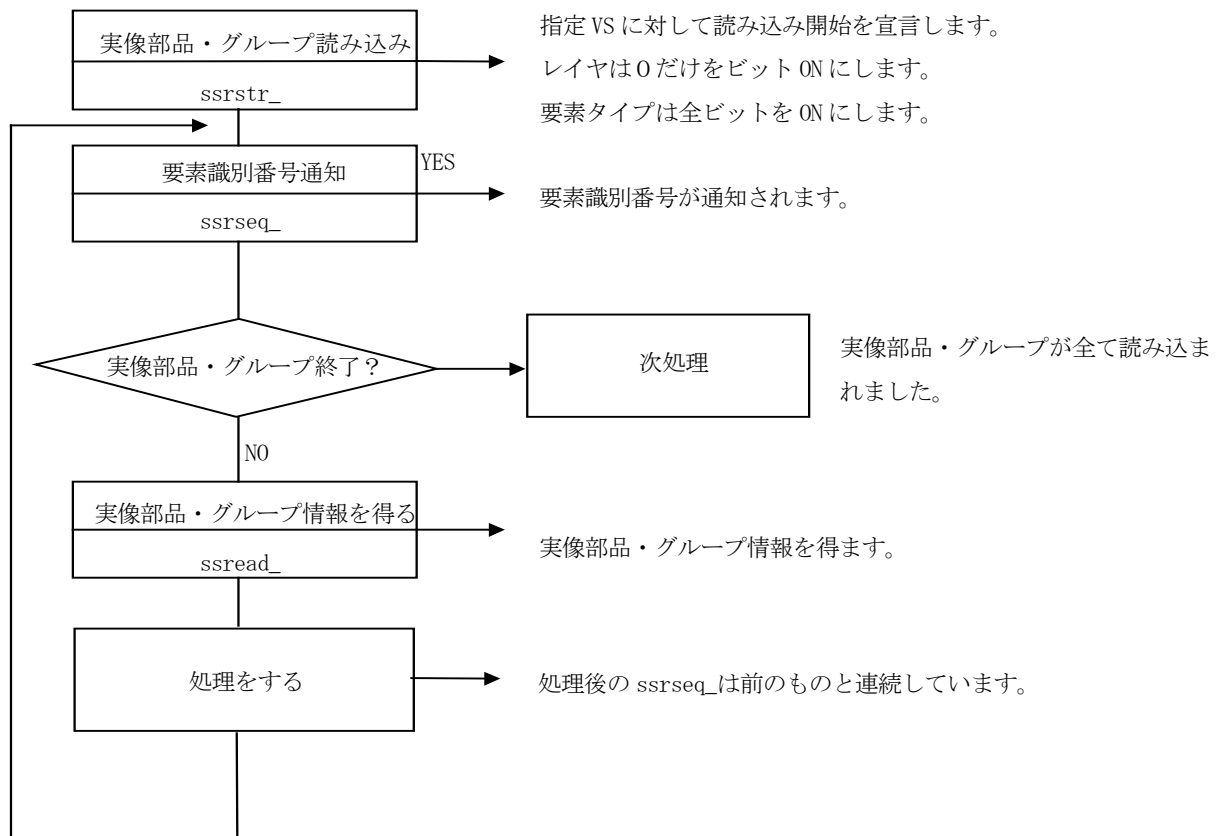
```

for(i=0;i<32;i++){
    class[i]=0x00;
    type[i]=0x00;
}
class[3]=0x03; /*レイヤ 3 0, 3 1*/
type[0]=0x20; /*タイプ 2      */
dim=2;
vswfno=1;
ssrstr_(&dim,&vswfno,class,type,&ircode);
L_10:
    ssrseq_(&segid,&ircode);
    if(segid==0)
        goto PROC_END; /*2次元要素の読み込みを終了します*/
    ssread_(&segid,infary);
    /*処理をします*/
    goto L_10;
PROC_END:

```

### (3) 実像部品・グループの読み込み

指定した VS の実像部品・グループ要素を順次読み込みます。



ssread\_を呼び出した直後にその実像部品・グループの削除、コピー、あるいは別の要素を作成することができます。

例：VS 番号が1の実像部品・グループの情報を得ます。

```
    for(i=0;i<32;i++){
        class[i]=0x00;
        type[i]=0xff;
    }
    class[0]=0x80; /*レイヤ0*/
    dim=2;
    vswfno=1;
    ssrstr_(&dim,&vswfno,class,type,&ircode);
L_10:
    ssrseq_(&segid,&ircode);
    if(segid==0)
        goto PROC_END; /*実像部品・グループの読み込みを終了します*/
    ssread_(&segid,infary);
        if(infary[15]==1) { /*実像部品の処理*/ }
        else                { /*グループの処理*/ }
    goto L_10;
PROC_END:
```

## VS 管理

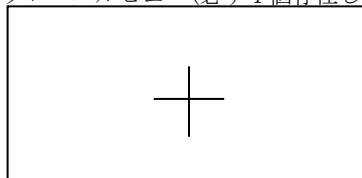
VS とは 2 次元要素を格納する器で、縦横無限大の大きさで、x 座標、y 座標を持ちます。2 次元要素を作成する前にあらかじめ利用する VS を開設しておく必要があります。VS は最大 500 個開設できますが、基本コマンドでワーク VS として 3 個利用しているため、常時開設できるのは 497 個です。システム起動時は通常 VS が 2 つ(グローバルビュー、基本ビュー)開設されています。また 2 次元要素を作成する VS を設定することができ、設定するとそれ以降に作成される要素はすべて、ここで設定された VS に格納されます。ユーザコマンドで VS を特に設定しない場合には、そのユーザコマンドが実行された時点で設定されていた VS が対象となります。

VS 名は 8 文字以内の以下の文字列とします。

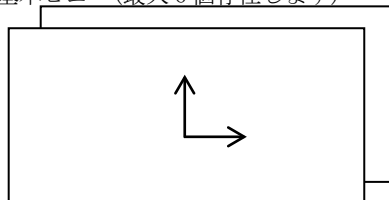
ASCII コード(半角文字列)    A～Z   a～z   0～9   #   @   .   %   (   )   [   ]   \_   {   }   -  
ただし、@はユーザコマンドで使用できません  
S-JIS コード(全角日本語文字列)

VS には以下の 4 種類があります。

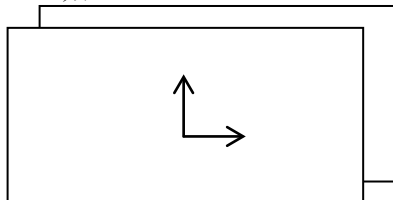
グローバルビュー(必ず 1 個存在します)



基本ビュー(最大 6 個存在します)



ローカルビュー



子図



### (1) VS の開設

- ・グローバルビュー、基本ビュー、ローカルビューはユーザコマンドで開設することはできません。
- ・子図

VS の開設
svopnv_

```
vsname[]="KZ01  ";  
svopnv_(vsname,&vsno,&ircode);
```

### (2) 格納する VS を設定します

格納する VS の設定
sssvs_

```
sssvs_(&vsno,&ircode);
```

### (3) 格納する VS を問い合わせます

格納する VS の問い合わせ
ssqvs_

```
ssqvs_(&vsno);
```

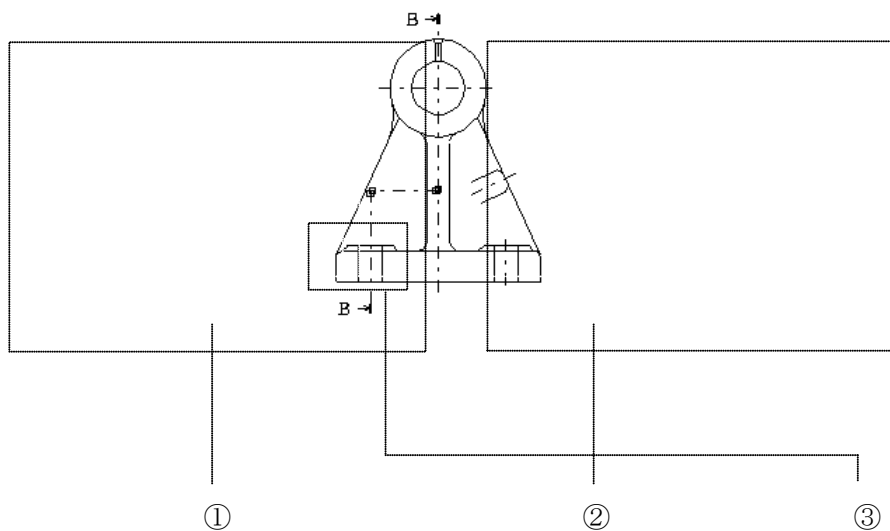
## ウィンドウ制御

VS に要素が作成されると同時に、システムは作成された要素を画面に表示します。システムでは VS をカメラで撮影し、映像を画面に表示しています。

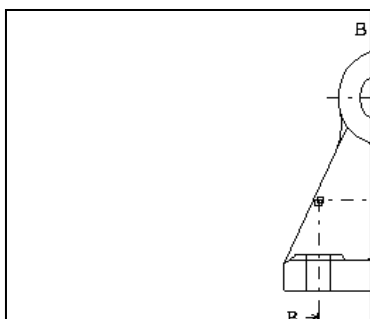
システム起動時は通常 VS が 2 つ(グローバルビュー、基本ビュー)開設されており、グローバルビューの原点と画面の中心が一致し、カメラの倍率も 1 となるように設定されています。しかし、VS は縦横とも無限大の大きさをもつため、全部を一度に表示することは不可能です。このため、以下のようにカメラを左右、上下に移動したり、倍率を変えたりして VS の中のある一部分を画面に表示します。

画面には、最大 15 個までウィンドウを開くことができます。

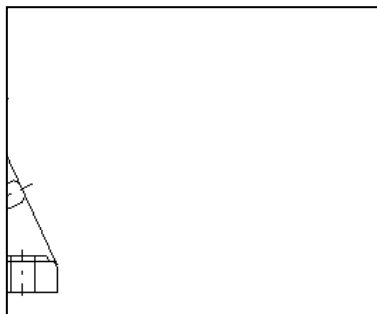
VS(無限大)



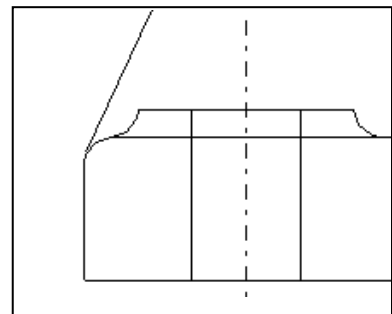
①



②



③(拡大)

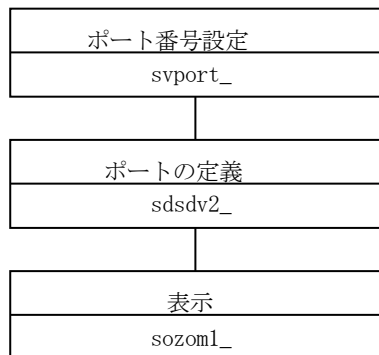


VS の表示・操作をする場合の図形処理ライブラリの利用方法を説明します。

### (1) VS を画面に表示します

画面にどの VS を表示するかはポートを定義することにより指定します。

ポートの番号は 1 ～ 1 2 8 の整数で指定します。

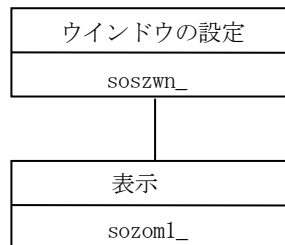
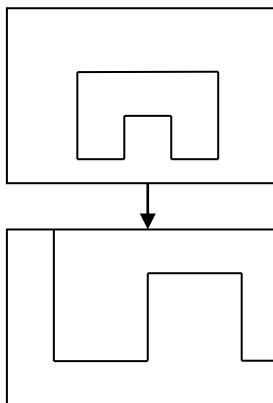


```
vpno=1;vsno=1;wfno=0;vwno=0;
svport_(&vpno,&vsno,&wfno,&vwno,&ircode)

devcnt=1;vplist=1;
sdsdv2_(&devcnt,&devflg,&x,&y,&vplist,&ircode)

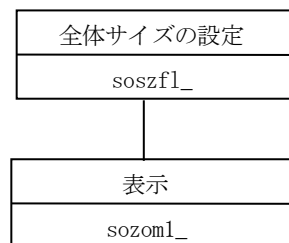
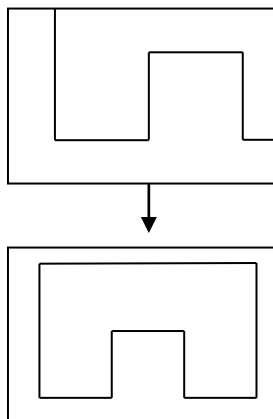
sozom1_(&ircode);
```

### (2) ズームアップ・パン



```
soszwn_(&pdno,&mode,&factor,&cx,&cy,&ircode);
sozom1_(&ircode);
```

### (3) 全体図を表示します



```
soszf1_(&pdno,&mode,&ircode);
sozom1_(&ircode);
```

## 復帰情報

復帰情報には、復帰コードと詳細コードがあります。  
復帰コードは、各プログラムの出力引数として返されます。  
詳細コードは、scqerr\_により問い合わせます。

例：

```
vsname[]="@KZ01  ";
svopnv_(vsname, &vsno, &ircode);
if (ircode!=0) {
    scqerr_(&detail);
    goto ERROR_PROC;
}
```

↑ 復帰コード

↑ 詳細コード

---

## 第7章 図形処理ライブラリ(コマンド制御編)の利用方法

ここでは、図形処理ライブラリ(コマンド制御編)の利用方法について説明しています。



## ■ UNDO／REDO 機能概要

UNDO／REDO 機能とは以下の通りです。

- ・ UNDO 機能      オペレーション中に、作成・編集された 図形要素 を元の状態に戻します。
- ・ REDO 機能      UNDO 機能により元に戻された 図形要素 を、作成・編集された状態に戻します。

UNDO／REDO の対象はコマンドオペレーション自体ではなく、図形要素の作成・編集です。

UNDO／REDO の対象例

(1) 図形要素の作成例

「基本線」コマンドで、2 点を指定して線分を作成します。

(2) 図形要素の削除例

「削除」コマンドで、一度に複数の図形要素を削除します。

(3) 図形要素の編集例

「切断・結合」コマンドで、指定した要素をトリムします。

UNDO／REDO の対象外例

(1) 「ウインドウを開く」コマンドでウインドウを開きます。

(2) 「情報表示 要素」コマンドで要素情報を見ます。

(3) 「検索要素 要素」コマンドで検索要素タイプを指定します。

## ■ UNDO／REDO 作成方法

UNDO／REDO 機能を利用する為には、コマンド開発者は各コマンドの UNDO／REDO データをシステムに登録する必要があります。

UNDO／REDO データは、次のプログラムをプロセスの先頭で呼び出すことにより登録が開始され、終了が宣言されるまで続きます。(UNDO／REDO の 1 回の単位として登録されます。)

```
iurbgn_    登録開始
iurend_    登録終了
```

以下に例を示します。

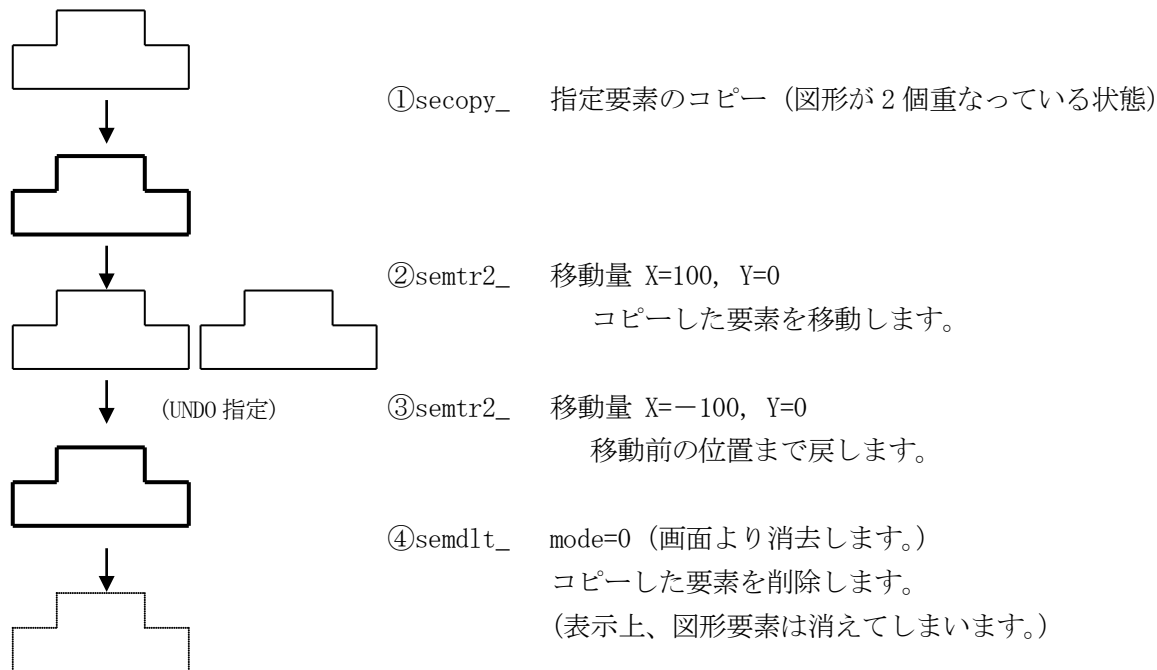
### (1) 図形要素をコピーするコマンドの場合

プロセス		
iurbgn_	登録開始	
secopy_	指定要素のコピー	DO／REDO 1 回の単位として登録されます
semtr2_	コピーした要素の移動	
iurend_	登録終了	

上記の様に、プロセス内で、iurbgn\_から iurend\_の間に呼び出された図形処理ライブラリの操作が、UNDO／REDO の 1 回の単位として登録されます。

また、UNDO は登録された順番と逆の順番に復元していくために、図形要素の表示順番に注意する必要があります。

(1) の UNDO の動きは以下ようになります。

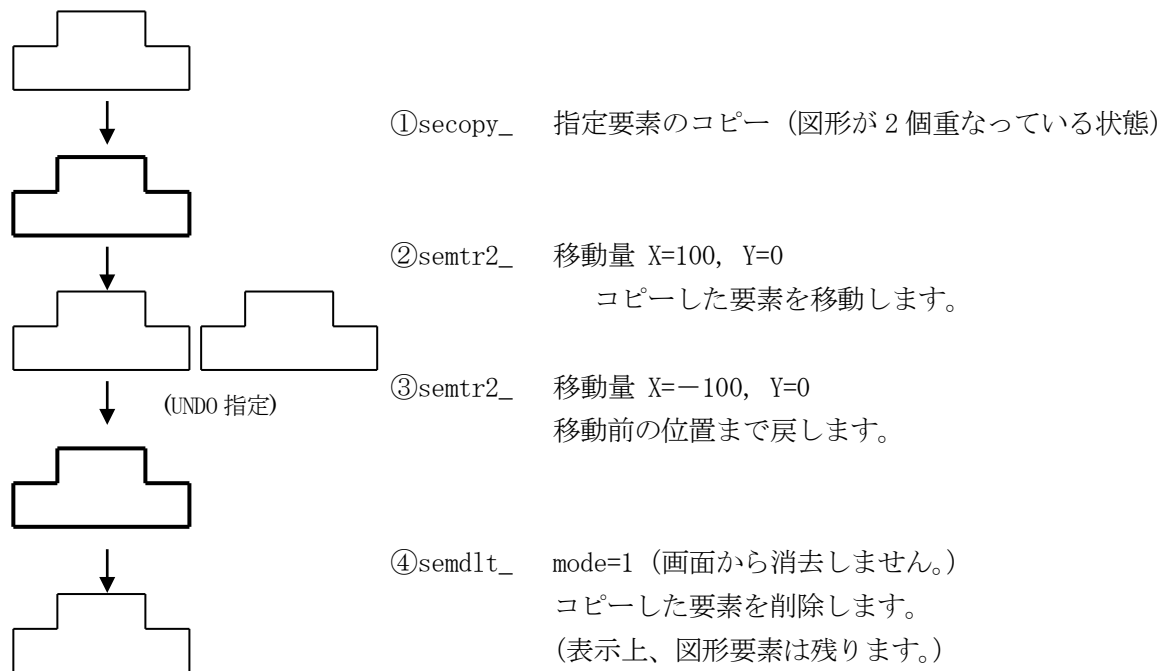


UNDO 実行時には、secopy\_に対応するプログラムが semdlt\_ ( … ,&mode, … ) mode=0 となり、最後にコピーした要素を削除するため、表示上コピー元の図形要素が消えてしまいます。但し、図形データとしては残っています。この様な場合は、iurvis\_により、UNDO 登録データの表示属性を変更することで、表示上も、元の図形要素が消えなくなります。次に例を示します。

(2) (1)で iurvis\_を使用して、UNDO の表示属性を変更します。

プロセス	
iurbgn_	登録開始
visual=0	0=非表示
iurvis_(&visual)	表示属性を非表示に変更
secopy_	指定要素をコピー
visual=-1	-1=設定解除
iurvis_(&visual)	表示属性を図形処理ライブラリ依存
semtr2_	コピーした要素を移動
iurend_	登録終了

(2)の UNDO の動き



上記の様に iurvis\_で表示属性を変更することで、UNDO 実行時には secopy\_に対応するプログラムが semdlt\_ ( ... ,&mode,... ) mode=1 となり、元の図形要素は表示上残ります。

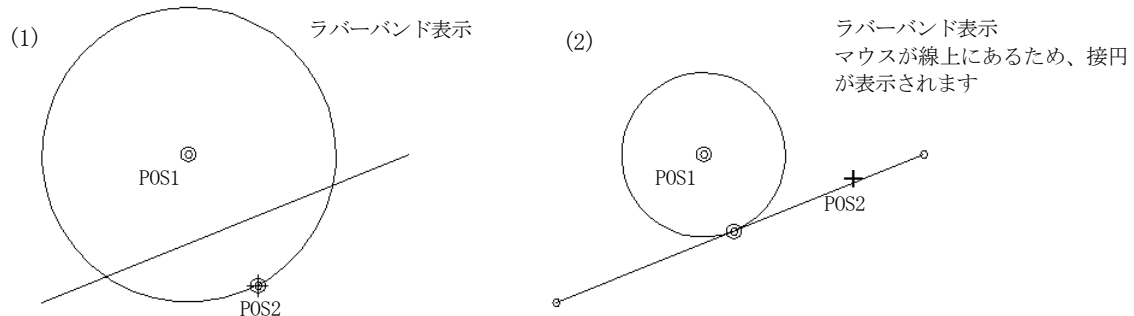
(1)のように、同一座標上で図形要素を操作する場合には、元の図形要素が表示上消えてしまうことがあるので、注意する必要があります。

# ラバーバンド

## ■ ラバーバンド機能概要

ラバーバンド機能とは、座標値入力時に作成される図形の形状を前もって画面上に表示する機能です。従ってオペレータはあらかじめ、作成される図形の形状を確認し、座標値を入力することができます。ドラッグ機能と異なりマウスの位置により、表示する形状を変更することができます。

中心点円コマンドで円を作成する場合



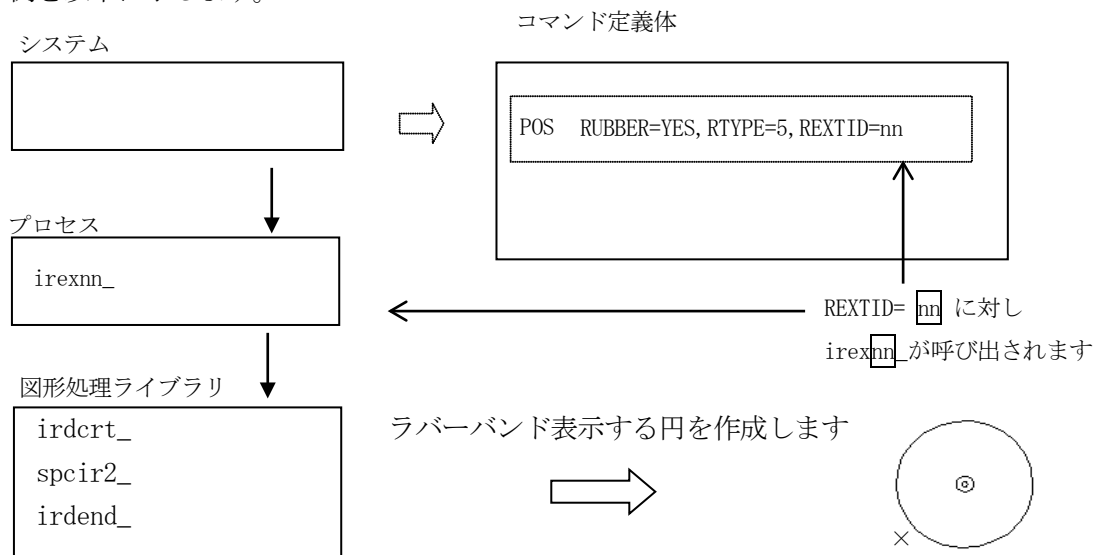
座標値の入力時、作成される円の形状がラバーバンド機能により表示されます。

## ■ ラバーバンド作成方法

コマンド開発者は座標値入力(POS)待ちの時、ラバーバンド機能を使用することができます。以下のようにコマンド定義体で POS RUBBER=YES、RTYPE=5、REXTID=nn が指定されるとシステムより対応するプロセスが呼び出されます。

プロセスは irdcrt\_~irdend\_で、表示する図形を作成します。

例を以下に示します。



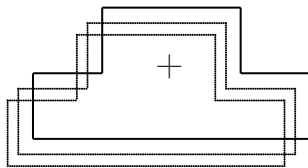
座標値入力 (POS) 待ちの時、マウスが動くたびに irexnn\_は何回も呼び出されます。ラバーバンドは、システムに自動的に消去されます。

# ドラッグング

## ■ ドラッグング機能概要

ドラッグング機能とは、座標値入力時に作成される図形の形状を前もって画面上に表示する機能です。従ってオペレータはあらかじめ、作成される図形の形状を確認し、座標値を入力することができます。ラバーバンド機能と異なりマウスの位置により、表示する形状を変更することはできません。

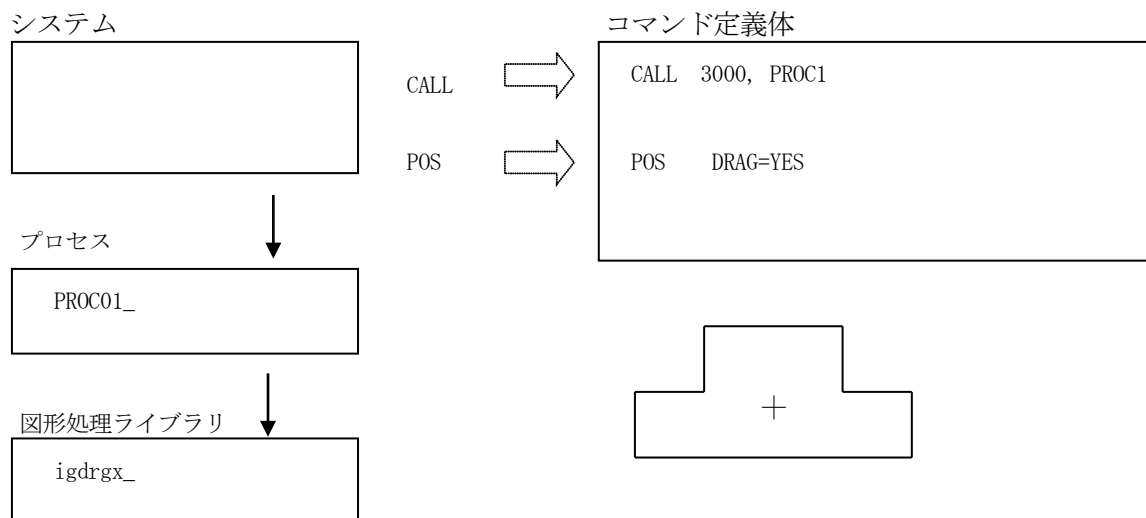
移動コマンドで図形を移動する場合



選択した図形と同じ形状がマウスに追従します。

## ■ ドラッグング作成方法

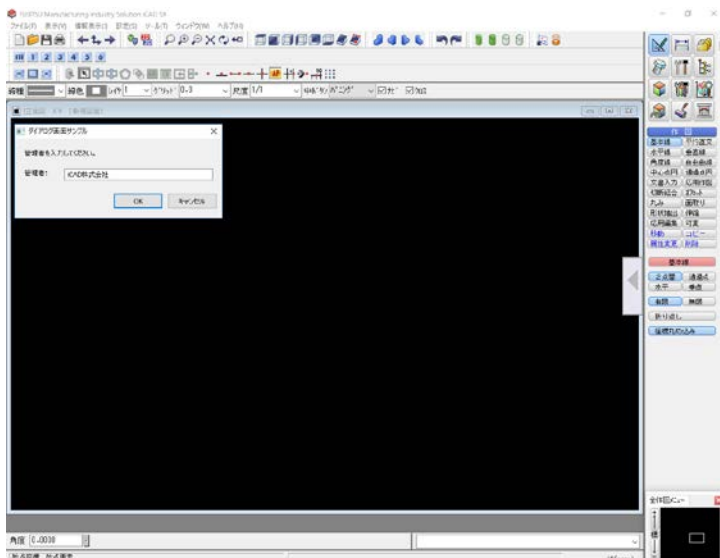
コマンド開発者は座標値入力(POS)待ちの時、ドラッグング機能を使用することができます。以下のようにまずコマンド定義体でCALLを指定すると、システムより対応するプロセスが呼び出されます。本プロセス内ではドラッグングする要素識別番号をigdrgx\_に通知します。その後POS DRAG=YESが指定されると、システムはマウスに追従させて通知された要素識別番号の図形を表示します。例を以下に示します。



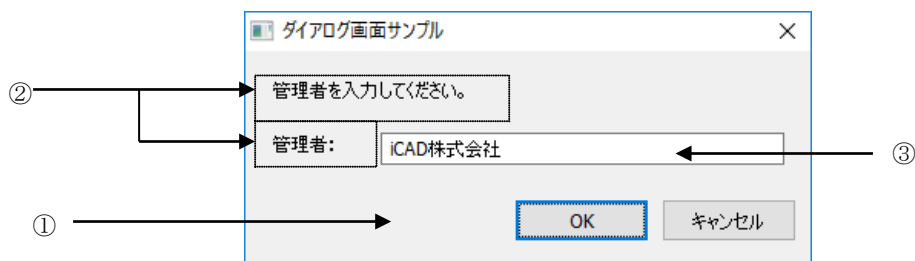
# ダイアログボックス

## ■ ダイアログボックスの概要

ダイアログボックスは、コマンドの中でオペレータに対して情報を表示したり、文字入力や選択を促したい時に使用すると便利です。



ダイアログボックスにはいくつかのコントロールを組み込むことができ、オペレータはこれらのコントロールを操作して情報の入力、オプションの選択をすることができます。以下にコントロールの例を示します。



- |                |                                |
|----------------|--------------------------------|
| ① プッシュボタン      | オペレータに OK またはキャンセルを選択させるコントロール |
| ② スタティックコントロール | オペレータに知らせたい情報を表示するコントロール       |
| ③ エディットコントロール  | オペレータに情報を入力させるためのコントロール        |

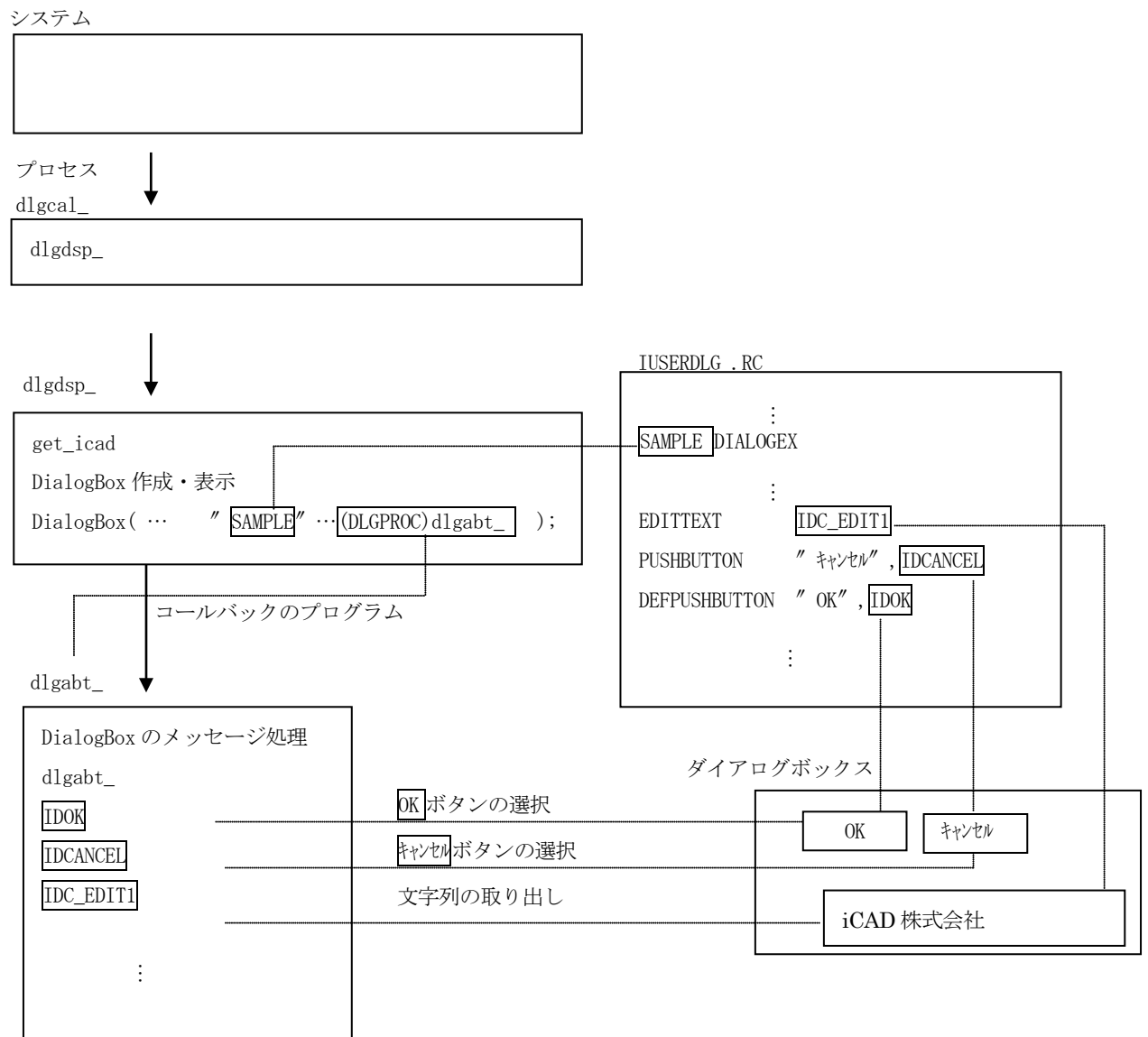
### 注意

ダイアログボックスを使用する場合は必ず、モーダルダイアログボックスを使用しなければなりません。

モーダルダイアログボックスとは、オペレータが **OK** または **キャンセル** ボタンを押してダイアログボックスを終了させない限り、処理が進まないダイアログボックスのことです。

## ■ ダイアログボックス表示のしくみ

ダイアログボックスの「SAMPLE」を作成して表示するしくみを以下に示します。



## ■ ダイアログボックス作成方法

### (1) ダイアログボックスの作成、リソースへの組み込み

詳細は、「VisualStudio」のオンラインマニュアルを参照してください。

### (2) プロセスの作成

#### ① ダイアログボックスの作成・表示処理プログラム

プログラミング例は次頁を参照してください。

#### ② ダイアログボックス作成・表示プログラム

ダイアログボックスを使用するためにはシステムのウインドウハンドルとインスタンスが必要です。

get\_icad 関数で取得します。

get\_icad

システムのウインドウハンドルとインスタンスを取得します。

void get\_icad(HWND \*icad\_hwnd, HANDLE \*icad\_inst)

引数

out \*icad\_hwnd システムのウインドウハンドル

out \*icad\_inst システムのインスタンス

プログラミング例は次頁を参照してください。

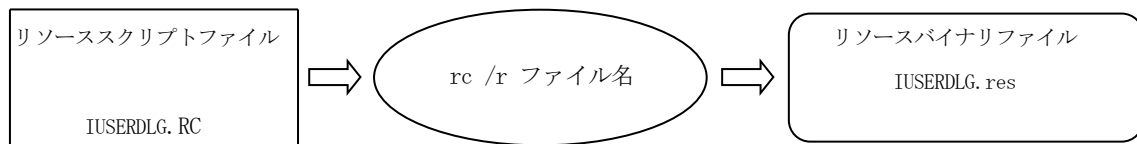
#### ③ ダイアログボックスのメッセージ処理プログラム

プログラミング例は次頁を参照してください。



## ■ コンパイル、リンク

- (1) 新規ダイアログボックスを作成した IUSERDLG.RC をコンパイルする方法を以下に示します。



### rc コマンド

リソーススクリプトファイルからリソースバイナリファイルを作成します。

rc /r ファイル名

### 説明

IUSERDLG.RC をコンパイルして IUSERDLG.res を作成します。

- ① コマンドプロンプトより以下のコマンドを実行します。

```
cd %ICADDIR%\usr%rc
rc /r IUSERDLG.rc
```

- ② カレントディレクトリに、置き換えられたリソースバイナリファイル(IUSERDLG.res)が作成されます。

- (2) プロセスのコンパイル

プロセスのコンパイルについては「第9章 コンパイル、リンク」を参照してください。

- (3) リンク

作成したリソースバイナリファイルが存在するディレクトリおよびファイル名を「icad\_user.txt」に記述して、%ICADDIR%\USER%BIN 配下に保存します。

%ICADDIR%\USER%BIN がない場合は、%ICADDIR%\USER%BIN を作成してから保存します。

詳細については「第9章 コンパイル、リンク」を参照してください。

```

/*****
/*
/*      d l g c a l _
/*
/*
/*      1. 機能概要
/*      ダイログボックスを作成・表示処理
/*
/*      2. 呼び出し形式
/*      void  dlscal_( ircod )
/*
/*
/*      3. 引数の説明
/*      i/o      long      ircod[3]      :   復帰値
/*
/*      4. 呼出し条件
/*
/*      5. 備考
/*
/*      6. 復帰値
/*
*****/
#include <stdio.h>
#include <windows.h>
#include <memory.h>

/*****
/*  外部参照関数宣言
/*
*****/
extern long dlgdsp_(char *);

/*****
/*  関数宣言
/*
*****/
void dlscal_(ircod)
    long ircod[3];
{
    char str[81];
    ircod[0] = 0;
    ircod[1] = 0;
    ircod[2] = 0;
    memset( str, 0x00, sizeof(str) );

```

---

```
/* **** */
/*   ダイアログボックスの作成・表示関数の呼出   */
/* **** */
    dlgdsp_( str );
}
```

```

/*****
/*
/*      d l g d s p _
/*
/*
/*      1. 機能概要
/*      ダイログボックスを作成・表示
/*
/*      2. 呼び出し形式
/*      long  dlgdsp_( str )
/*
/*
/*      3. 引数の説明
/*      o      char      *str      :   入力された文字列
/*
/*      4. 呼出し条件
/*
/*      5. 備考
/*
/*      6. 復帰値
/*      NULL : 正常終了
/*      EOF  : キャンセル又は異常終了
/*
*****/

#include <stdio.h>
#include <string.h>
#include <stdio.h>
#include <windows.h>
#include "resource.h"

char wk_str[81];

/*****
/*      外部参照関数宣言
/*
*****/

extern BOOL CALLBACK      dlgabt_( HWND, UINT, WPARAM, LPARAM );
extern void               get_icad( HWND, HANDLE );

/*****
/*      関数宣言
/*
*****/

long dlgdsp_( str )

```

```

char *str;
{
    HWND    hwnd;
    HANDLE  hinst;
    long     dlgb;

    /* iCAD SX のウィンドウハンドルとインスタンスを取得する */
    get_icad( &hwnd, &hinst );

    /* ダイアログボックスを作成・表示 */
    dlgb = DialogBox( hinst, /* iCAD SX のインスタンス */
                     "SAMPLE", /* 表示する DialogBox のテンプレート名 */
                     hwnd, /* iCAD SX のウィンドウハンドル */
                     (DLGPROC)dlgabt_ ); /* DialogBox のメッセージ処理関数 */

    if( (dlgb == FALSE) || (dlgb == EOF) ) /* キャンセル又は異常終了 */
        return(EOF);

    strcpy( str, wk_str );
    return( (long)NULL );
}

```

```

/*****
/*
/*      d l g a b t _
/*
/*
/*      1. 機能概要
/*      ダイアログボックスのメッセージ処理
/*
/*
/*      2. 呼び出し形式
/*      BOOL CALLBACK dlgabt_( hwnd, message, wParam, lParam )
/*
/*
/*      3. 引数の説明
/*      i          HWND          hwnd    :   ウィンドウハンドル
/*      i          UINT          message :   メッセージ I D
/*      i          WPARAM        wParam   :   ワードパラメータ
/*      i          LPARAM        lParam   :   ロングワードパラメータ
/*
/*
/*      4. 呼出し条件
/*
/*
/*      5. 備考
/*
/*
/*      6. 復帰値
/*      TRUE   :   ダイアログ処理有り
/*      FALSE  :   ダイアログ処理無し
/*
/*****/
#include <stdio.h>
#include <memory.h>
#include <windows.h>
#include "resource.h"

/*****
/*      外部変数宣言
/*
/*****/
extern char wk_str[81];

/*****
/*      関数宣言
/*
/*****/
BOOL CALLBACK dlgabt_( hwnd, message, wParam, lParam )
HWND    hwnd;

```

```

UINT    message;
WPARAM  wParam;
LPARAM  lParam;
{
    long    wkleng=81;

    switch( message ) {
        case WM_INITDIALOG:                                /* DialogBox の初期化 */
            memset( wk_str, 0, wkleng );
            GetUserName( wk_str, &wkleng );                /* ユーザ名の取得 */
            SetDlgItemText( hwnd, IDC_EDIT1, wk_str);
            /* エディットコントロールの識別番号↑ ユーザ名をエディットコントロールに設定 */
            SetFocus( hwnd );                               /* DlgBox にキーボードフォーカスを設定 */
            return TRUE;

        case WM_COMMAND:
            if( HIWORD(wParam) == BN_CLICKED ) { /* マウスをクリックされた */
                if( LOWORD(wParam) == IDOK ) {
                    /* プッシュボタンの識別番号↑ クリックされたのはOKボタン */
                    GetDlgItemText( hwnd, IDC_EDIT1, wk_str, wkleng );
                    /* エディットコントロールの識別番号↑ エディットコントロールの文字列の取出す */
                    EndDialog( hwnd, TRUE ); /* DialogBox の処理終了させる */
                    return TRUE;
                }
                if( LOWORD(wParam) == IDCANCEL ) {
                    /* プッシュボタンの識別番号↑ クリックされたのはキャンセルボタン */
                    EndDialog( hwnd, FALSE ); /* DialogBox の処理終了させる */
                    return TRUE;
                }
                return FALSE;
            }
            break;
    }
    return FALSE;
}

```

# ダイアログボックスのログ出力

## ■ ダイアログボックスのログ出力の概要

通常、システムでのオペレータの操作はプロセスに特別な関数を組み込まなくてもコマンドログを出力しています。

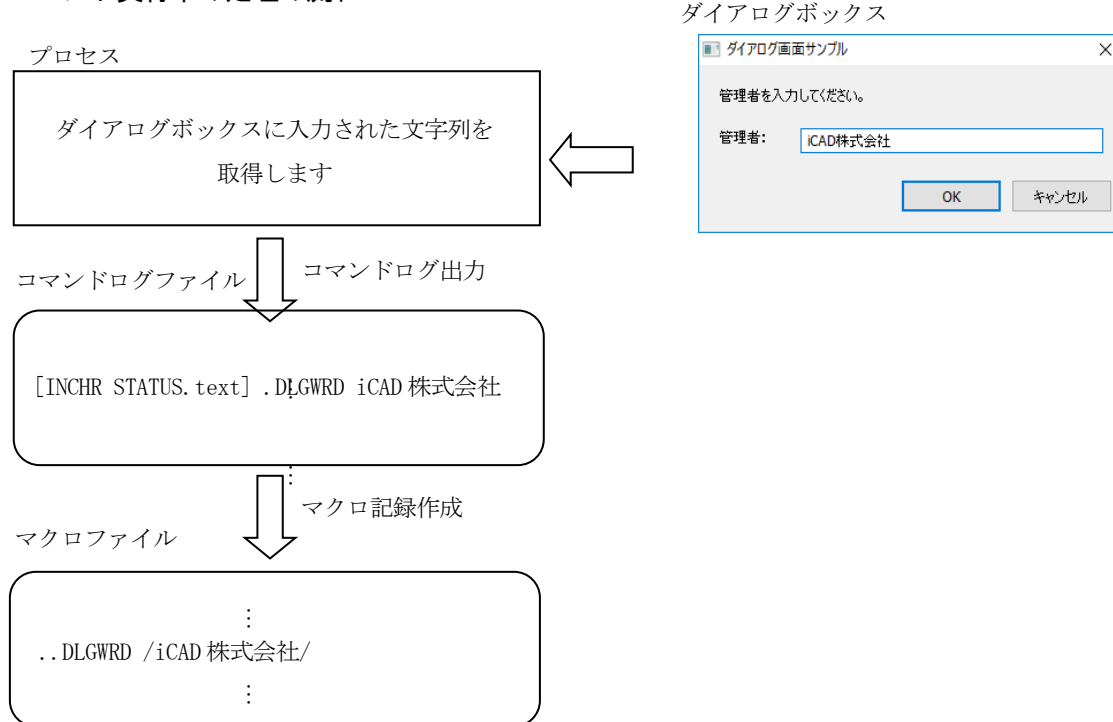
しかし、ダイアログボックスを使用しているコマンドはダイアログボックスの部分がシステムとは独立して動いているため、コマンドログを出力するには、プロセスにコマンドログ出力用の関数を組み込むことが必要となります。

これを組み込むことで、ダイアログボックスを使用しているコマンドからもマクロ記録を取ることができ、同じ処理をVBI/Fを利用して再実行することができます。

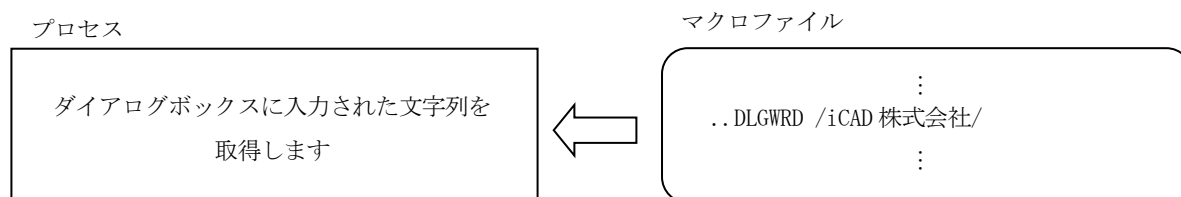
ダイアログボックスで入力された文字列を取得するというプロセスを例に処理の流れを以下に示します。

(入力された文字列は「iCAD 株式会社」)

### コマンド実行中の処理の流れ



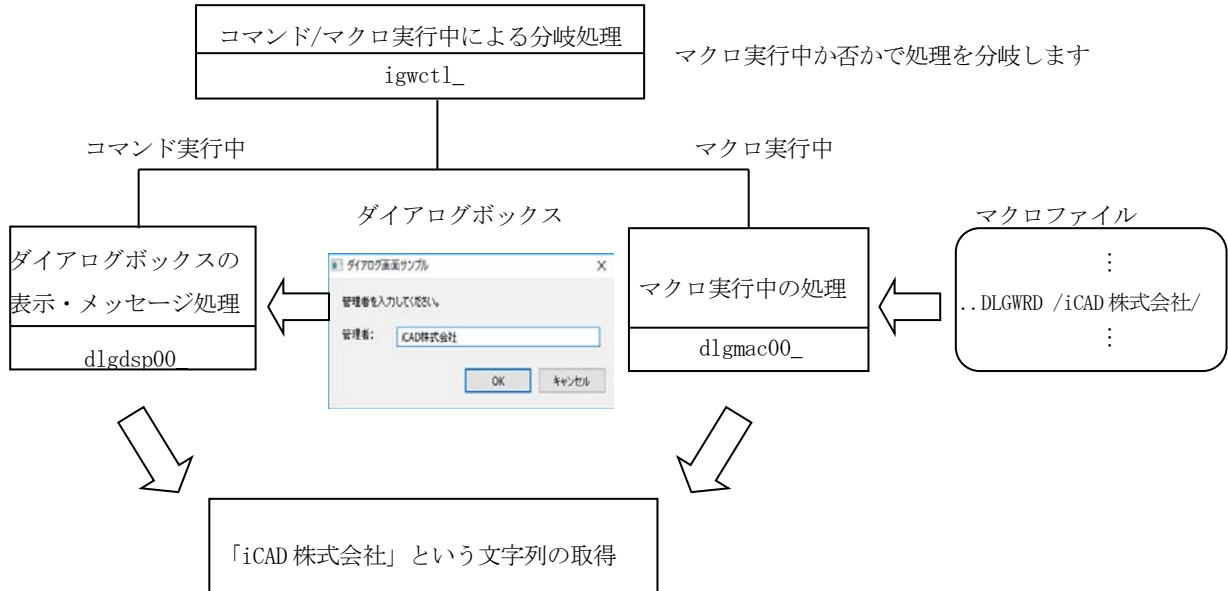
### マクロ実行中の処理の流れ





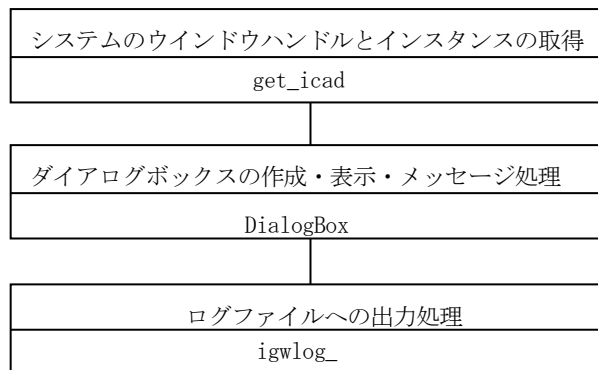
## ■ ダイアログボックスのログ出力を組み込んだプロセスの作成方法

① ダイアログボックスに入力された文字列取得プログラム(dlgcal00\_)を作成します。



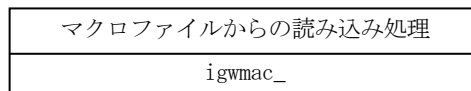
プログラミング例は次頁を参照してください。

② ダイアログボックスの作成・表示・メッセージ処理プログラム(dlgdsp00\_)を作成します。



プログラミング例は次頁を参照してください。

③ ダイアログのマクロ実行中である処理プログラム(dlgmac00\_)を作成します。



マクロ実行中の処理は自動的にログファイルに出力されます。

プログラミング例は次頁を参照してください。

```

/*****
/*
/*      d l g c a l 0 0 __
/*
/*
/*      1. 機能概要
/*      ダイログボックスの作成・表示処理
/*
/*      2. 呼び出し形式
/*      void  dlgcal00_( str, ircode )
/*
/*
/*      3. 引数の説明
/*      o      char      str[81]      :   文字列データ
/*      o      long      ircode[3]    :   復帰情報
/*
/*      [0] : 復帰コード
/*      [1] : メッセージ番号
/*      [2] : 詳細コード
/*
/*      4. 呼出し条件
/*
/*      5. 備考
/*
/*      6. 復帰値
/*
*****/

#include <stdio.h>
#include <string.h>
#include <windows.h>
#include <memory.h>

/*****
/*  外部参照関数宣言
*****/

extern void igwctl_();          /* コマンド／マクロ実行中での分岐処理 */
extern void dlgdsp00_();       /* ダイアログボックスの作成・表示
extern void dlgmac00_();       /* マクロ実行中の処理

```

```

/*****/
/* 構造体定義 */
/*****/

/* 文字列データ格納テーブル */

typedef struct {
    char    dlg_str[81];
} DLG_TBL;

/*****/
/* 関数宣言 */
/*****/
void dlgcal00_( str, ircode )
    char    str[81];
    long    ircode[3];
{
/*****/
/* 内部変数定義 */
/*****/
    long        i;
    long        wnfunc[5];
    long        mcfunc[5];
    DLG_TBL     data;
    long        ircd[3];

/*****/
/* 初期化 */
/*****/
    ircode[0] = 0;
    ircode[1] = 0;
    ircode[2] = 0;
    ircd[0] = 0;
    ircd[1] = 0;
    ircd[2] = 0;
    memset( (char *)&data, 0x00, sizeof(data) );

```

```

/*****
/* ダイアログボックス処理及びマクロ処理関数の登録処理 */
*****/

/* N U L Lで初期化 */
for ( i=0; i < 5; i++ ){
    wnfunc[i] = (long)NULL;
    mcfunc[i] = (long)NULL;
}

/* ダイアログボックス処理関数の登録 */
wnfunc[0] = (long)dlgdsp00_;

/* マクロ処理関数の登録 */
mcfunc[0] = (long)dlgmac00_;

/*****
/* 文字列データを取り出す */
*****/

/* コマンド／マクロ実行中での処理の分岐 */
igwctl_( wnfunc, mcfunc, &data, ircd);
if ( ircd[0] != 0 ){
    if (ircd[0] == 4 ){
        ircode[0] = 4;
        /* 「キャンセル」 ボタンにより終了 */
    } else {
        /* 異常終了 */
        ircode[0] = 8;
        ircode[1] = 13006;
    }
    return;
}
memset( str, 0x00, sizeof(str) );
strcpy( str, data.dlg_str );
return;
}

```

```

/*****
/*
/*      d l g d s p 0 0 __
/*
/*
/*      1. 機能概要
/*      ダイアログボックスを作成・表示
/*
/*
/*      2. 呼び出し形式
/*      long   dlgdsp00_( data, ircode )
/*
/*
/*      3. 引数の説明
/*      o      dlg_tbl      *data   : 文字列データ格納テーブル
/*      o      long         *ircode  : 復帰情報
/*                                     [0] : 復帰コード
/*                                     [1] : メッセージ番号
/*                                     [2] : 詳細コード
/*
/*
/*      4. 呼出し条件
/*
/*
/*      5. 備考
/*
/*
/*      6. 復帰値
/*
*****/
#include <stdio.h>
#include <string.h>
#include <memory.h>
#include <windows.h>
#include "resource.h"

/*****
/*      外部参照関数宣言
*****/
extern BOOL CALLBACK   dlgabt_( HWND, UINT, WPARAM, LPARAM ); /* メッセージ処理 */
extern void            get_icad(); /* iCAD SX のウィンドウハンドルとインスタンスの取得 */
extern void            igwlog_(); /* ログ出力処理 */

```

```

/*****/
/* 構造体定義 */
/*****/

/* 文字列データ格納テーブル */
typedef struct {
    char    dlg_str[81];
} DLG_TBL;

/* マクロデータ格納テーブル */
typedef struct {
    char    itmnam[8];
    long    format;
    void    *data;
} DLG_MACRO;

static DLG_TBL wkdata;

/*****/
/* デファイン定義 */
/*****/
#define    DLG_OK            0
#define    DLG_CANCEL       1
#define    F_CHAR            1          /* char のデータ型 */
#define    F_LONG           3          /* long のデータ型 */

/*****/
/* 関数宣言 */
/*****/
void dlgdsp00_( data, ircode )
    DLG_TBL    *data;
    long       ircode[3];
{

```

```

/*****
/* 内部変数定義
/*****

    HWND          hwnd;
    HANDLE         hinst;
    long           dlgb;
    long           i;
    long           dmy;
    long           ircd[3];
    DLG_MACRO      itm[] = {
                        { "CANCEL", F_LONG, &dmy },
                        { "DLGWRD", F_CHAR, &data->dlg_str },
                        { "",          0,      (void *)NULL } };

/*****
/* 初期化
/*****

    ircode[0] = 0;
    ircode[1] = 0;
    ircode[2] = 0;
    ircd[0] = 0;
    ircd[1] = 0;
    ircd[2] = 0;

/*****
/* iCAD SX のウィンドウハンドルとインスタンスを取得する
/*****

    get_icad( &hwnd, &hinst );

/*****
/* ダイアログボックスを作成・表示
/*****

    dlgb = DialogBox( hinst,          /* iCAD SX のインスタンス      */
                    "SAMPLE",        /* 表示する DialogBox のテンプレート名 */
                    hwnd,            /* iCAD SX のウィンドウハンドル    */
                    (DLGPROC)dlgabt_ ); /* DialogBox のメッセージ処理関数    */

    if ( dlgb == DLG_CANCEL ) {      /* 「キャンセル」ボタンにより終了 */
        /* CANCEL をログファイルへ出力 */
        igwlog_( itm[0].itmnam, &itm[0].format, itm[0].data, ircd );
        if ( ircd[0] != 0 ) {        /* ログ出力の異常終了 */
            ircode[0] = 8;
        } else {

```

```

        ircode[0] = 4;
    }
    return;
} else if ( dlgb == DLG_OK ){          /* 「OK」 ボタンにより終了      */
    /* 文字列データをテーブルへ格納 */
    strcpy ( data->dlg_str, wkdata.dlg_str );
    for ( i=1; itm[i].format != 0; i++ ){
        /* データを項目毎にログファイルへ出力 */
        igwlog_( itm[i].itmnam, &itm[i].format, itm[i].data, ircd );
        if ( ircd[0] != 0 ){          /* ログ出力の異常終了      */
            ircode[0] = 8;
            return;
        }
    }
} else {                               /* ダイアログの異常終了      */
    ircode[0] = 8;
}
return;
}

```

```

/*****
/*
/*
/* 関数名      : dlgabt_()
/*
/* 機能概要   : ダイアログボックスの各メッセージ毎に処理を振り分ける
/*
/* return    : TRUE   : ダイアログ処理有り
/*             FALSE : ダイアログ処理無し
/*
/* input     : HWND    hwnd;   シンボル選択ダイアログのハンドル
/*             UINT    message; メッセージID
/*             WPARAM  wParam;   ワードパラメータ
/*             LPARAM  lParam;   ロングワードパラメータ
/*
/*
*****/

```



```

BOOL CALLBACK dlgabt_( hwnd, message, wParam, lParam )

HWND    hwnd;
UINT    message;
WPARAM  wParam;
LPARAM  lParam;
{
    char    wk_str[81];
    long    wkleng=81;

    switch( message ) {
        case WM_INITDIALOG:                                /* DialogBox の初期化 */
            memset( wk_str, 0, wkleng );
            GetUserName( wk_str, &wkleng );                /* ユーザ名を取得 */
            /* ユーザ名をエディットコントロールに設定 */
            SetDlgItemText( hwnd, IDC_EDIT1, wk_str );
            /* DlgBox にキーボードフォーカスを設定 */
            SetFocus( hwnd );
            return TRUE;

        case WM_COMMAND:
            if( HIWORD(wParam) == BN_CLICKED ) { /* マウスがクリックされた */
                if( LOWORD(wParam) == IDOK ) { /* クリックされたのはOKボタン */
                    /* エディットコントロールの文字列を取出す */
                    GetDlgItemText( hwnd, IDC_EDIT1, wk_str, wkleng );
                    /* 文字列データをテーブルへ格納 */
                    strcpy ( wkdata.dlg_str, wk_str );
                    EndDialog( hwnd, DLG_OK ); /* DialogBox の処理を終了させる */
                    return TRUE;
                }
                if( LOWORD(wParam) == IDCANCEL ) { /* クリックされたのはキャンセルボタン */
                    EndDialog( hwnd, DLG_CANCEL ); /* DialogBox の処理を終了 */
                    return TRUE;
                }
            }
            return FALSE;
        }
    }
    break;
}
return FALSE;
}

```

```

/*****
/*
/*      d l g m a c 0 0 __
/*
/*
/*      1. 機能概要
/*      ダイアログボックスの表示・処理コマンドのマクロ実行中の処理
/*
/*
/*      2. 呼び出し形式
/*      void    dlgmacc00_( data, ircode )
/*
/*
/*      3. 引数の説明
/*      o      dlg_tbl      *data      :   文字列データ格納テーブル
/*      o      long      *ircode      :   復帰情報
/*
/*      [0] : 復帰コード
/*      [1] : メッセージ番号
/*      [2] : 詳細コード
/*
/*
/*      4. 呼出し条件
/*
/*
/*      5. 備考
/*
/*
/*      6. 復帰値
/*
/*****/
#include <stdio.h>
#include <string.h>

/*****
/*      外部参照関数宣言
/*****/
extern void      igwmac_(); /* マクロデータ読込処理

```

```

/*****/
/* 構造体定義 */
/*****/

/* 文字列データ格納テーブル */
typedef struct {
    char    dlg_str[81];
} DLG_TBL;

/* マクロデータ格納テーブル */
typedef struct {
    char    itmnam[8];
    long    format;
    void    *data;
} DLG_MACRO;

/*****/
/* デファイン定義 */
/*****/
#define    F_CHAR        1                /* char のデータ型 */
#define    F_LONG        3                /* long のデータ型 */

/*****/
/* 関数宣言 */
/*****/
void dlmac00_( data, ircode )
DLG_TBL    *data;
long       ircode[3];
{
/*****/
/* 内部変数定義 */
/*****/
    long        i;
    char        cmdnam[8];
    long        dmy;
    long        ircd[3];
    DLG_MACRO   itm[] = {
        { "CANCEL", F_LONG, &dmy },
        { "DLGWRD", F_CHAR, &data->dlg_str },
        { "",      0,      (void *)NULL } };

```

```

/*****
/* 初期化 */
*****/
    ircode[0] = 0;
    ircode[1] = 0;
    ircode[2] = 0;
    ircd[0] = 0;
    ircd[1] = 0;
    ircd[2] = 0;
    memset( cmdnam, 0x00, sizeof(cmdnam) );

/*****
/* マクロデータの読込処理 */
*****/
/* マクロデータ読込処理関数の呼出 */
for ( i=1; itm[i].format != 0; i++ ){
    /* データを各項目毎にマクロファイルより読込む */
    igwmac_( cmdnam, itm[i].itmnam, &itm[i].format, itm[i].data, ircd );
    if ( ircd[0] != 0 ){
        if ( ircd[0] == 4 ){          /* CANCEL キーワードの検出 */
            ircode[0] = 4;
        } else {                    /* 異常終了 */
            ircode[0] = 12;
        }
        return;
    }
}
return;
}

```

---

## 第8章 プロセスリスト

ここでは、作成したプロセスをプロセスリストに記述する方法について説明しています。

## プロセスリストの概要

コマンド開発者が開発したプロセスは、すべてプロセスリストに記述します。プロセスリストに記述することで、「プロセス」としてシステムに認識されます。

プロセスリストによる定義は、以下のように PROLIST 文で開始し、PROLEND 文で終了します。この間にプロセスの数だけ PRONO 文を記述します。

既存のプロセスリストにプロセスを追加する場合は PRONO 文を追加してください。

例を以下に示します。

次のプロセスを記述します。

プロセスリスト名は@PROL3 とします。

プロセス名	プロセス番号
PLIN00	3001
PCIR00	3002
PARC00	3003
PPNT00	3004
PMRK00	3005
PNOT00	3006
PMOV00	3007
PROT00	3008
PMIR00	3009
PERS00	3010

@PROL3	PROLIST
PRONO	3001, PLIN00
PRONO	3002, PCIR00
PRONO	3003, PARC00
PRONO	3004, PPNT00
PRONO	3005, PMRK00
PRONO	3006, PNOT00
PRONO	3007, PMOV00
PRONO	3008, PROT00
PRONO	3009, PMIR00
PRONO	3010, PERS00
	PROLEND

---

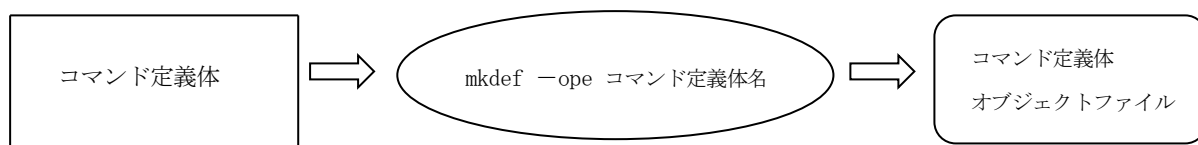
## 第9章 コンパイル、リンク

ここでは、作成したコマンド定義体、コマンドリスト、プロセス、プロセスリストをコンパイルし、システムとリンクする方法について説明しています。

# コンパイル

コマンドを組み込むには、作成したコマンド定義体、コマンドリスト、プロセス、プロセスリストをコンパイルします。コンパイル用のコマンドはシステムで用意されており、このコマンドを実行します。

## ■ コマンド定義体のコンパイル



### mkdef コマンド

コマンド定義体からオブジェクトファイルを作成します。

`mkdef -ope コマンド定義体名`

#### 説明

<code>-ope</code>	必ず <code>-ope</code> と指定します。
コマンド定義体名	コマンド定義体名を指定します。

例：コマンド定義体名= HVLINE

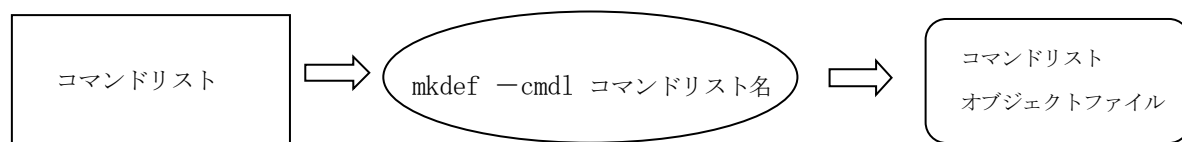
- ① コマンドプロンプトより以下のコマンドを実行します。

`cd %ICADDIR%\%usr%\opt\%obj`

`mkdef -ope ..\%src%\HVLINE`

- ② カレントディレクトリにオブジェクトファイル (HVLINE.obj) が作成されます。

## ■ コマンドリストのコンパイル



### mkdef コマンド

コマンドリストからオブジェクトファイルを作成します。

`mkdef -cmdl コマンドリスト名`

#### 説明

<code>-cmdl</code>	必ず <code>-cmdl</code> と指定します。
コマンドリスト名	コマンドリスト名 (@CMDL3)を指定します

例：コマンドリスト名= @CMDL3

- ① コマンドプロンプトより以下のコマンドを実行します。

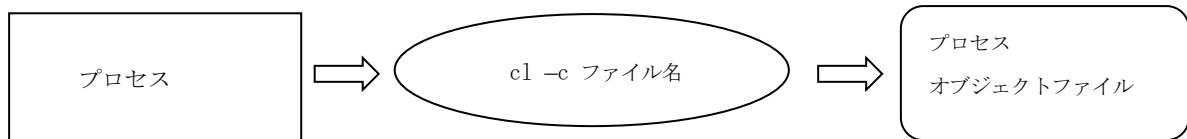
`cd %ICADDIR%\%usr%\opt\%obj`

`mkdef -cmdl ..\%src%\@CMDL3`

- ② カレントディレクトリにオブジェクトファイル (@CMDL3.obj) が作成されます。



## ■ プロセスのコンパイル



### cl コマンド

プロセスからオブジェクトファイルを作成します。

cl -c ファイル名

#### 説明

-c                      必ず -c と指定します。(コンパイルだけ行います)

ファイル名            プロセスとして作成した .c ソースを指定します。

例：ファイル名= tpar00.c

- ① コマンドプロンプトより以下のコマンドを実行します。

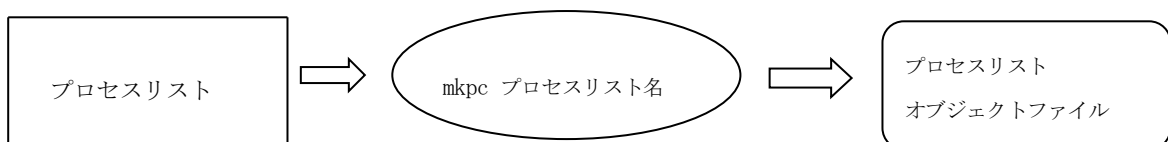
```
cd %ICADDIR%\usr¥process¥obj
```

```
cl -c ..¥src¥tpar00.c
```

- ② カレントディレクトリにオブジェクトファイル (tpar00.obj) が作成されます。

cl コマンドの詳細は、「Visual Studio」のオンラインマニュアルを参照してください。

## ■ プロセスリストのコンパイル



### mkpc コマンド

プロセスリストよりオブジェクトファイルを作成します。

mkpc プロセスリスト名

#### 説明

プロセスリスト名            プロセスリスト名 (@PROL3) を指定します。

例：プロセスリスト名= @PROL3

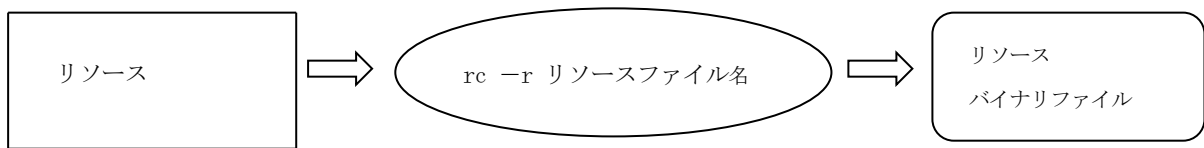
- ① コマンドプロンプトより以下のコマンドを実行します。

```
cd %ICADDIR%\usr¥process¥obj
```

```
mkpc ..¥src¥@PROL3
```

- ② カレントディレクトリにオブジェクトファイル (@PROL3.obj) が作成されます。

## ■ リソースのコンパイル



### rc コマンド

リソースからリソースバイナリファイルを作成します。

`rc -r リソースファイル名`

### 説明

`-r`                                      必ず `-r` と指定します。  
リソースファイル名                      ダイアログを作成したリソースを指定します。

例：リソースファイル名= `IUSERDLG.rc`    リソースファイル存在ディレクトリ=`%ICADDIR%\usr\%rc`

① コマンドプロンプトより以下のコマンドを実行します。

`cd %ICADDIR%\usr\%rc`

`rc -r IUSERDLG.rc`

② カレントディレクトリにリソースバイナリファイル (`IUSERDLG.res`) が作成されます。

## リンク

コンパイルしたコマンド定義体、コマンドリスト、プロセス、プロセスリストをシステムとリンクします。リンク用のコマンドはシステムで用意されており、このコマンドを実行します。

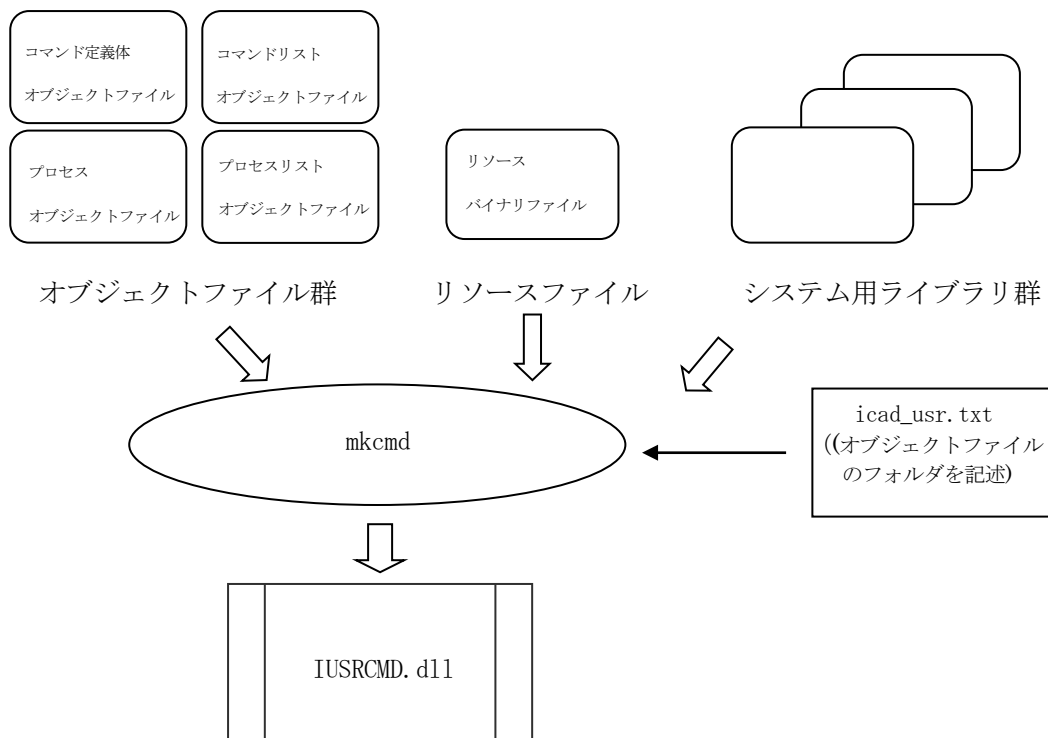
作成したオブジェクトファイルが存在するディレクトリおよびファイル名を「icad\_usr.txt」に記述して%ICADDIR%\¥USER¥BIN 配下に保存します。%ICADDIR%\¥USER¥BIN がない場合は、%ICADDIR%\¥USER¥BIN を作成してから保存します。

```
; オブジェクト、ライブラリ
.. ¥usr¥opt¥obj¥*.obj  _____ コマンド定義体/コマンドリスト
.. ¥usr¥process¥obj¥*.obj _____ プロセス/プロセスリスト
.. ¥usr¥rc¥*.res       _____ リソース
;
```

「icad\_usr.txt」は%ICADDIR%\¥BIN 配下に提供されています。

リンクは通常%ICADDIR%\¥USER¥BIN 配下にある「icad\_usr.txt」が使用されます。%ICADDIR%\¥USER¥BIN 配下に「icad\_usr.txt」がない場合、%ICADDIR%\¥BIN 配下にある「icad\_usr.txt」が使用されます。

作成したオブジェクトファイルをシステムとリンクし、コマンドに組み込む方法を以下に示します。



## mkcmd コマンド

コマンドのリンクをします。

mkcmd options

### 説明

options

ご利用の「Visual Studio」のバージョンを指定します。

Visual Studio のバージョン	options
Visual Studio 2015	-VS14
Visual Studio 2017	-VS15
Visual Studio 2019	-VS16

例：「Visual Studio 2019」をご利用の場合

- ① コマンドプロンプトより以下のコマンドを実行します。

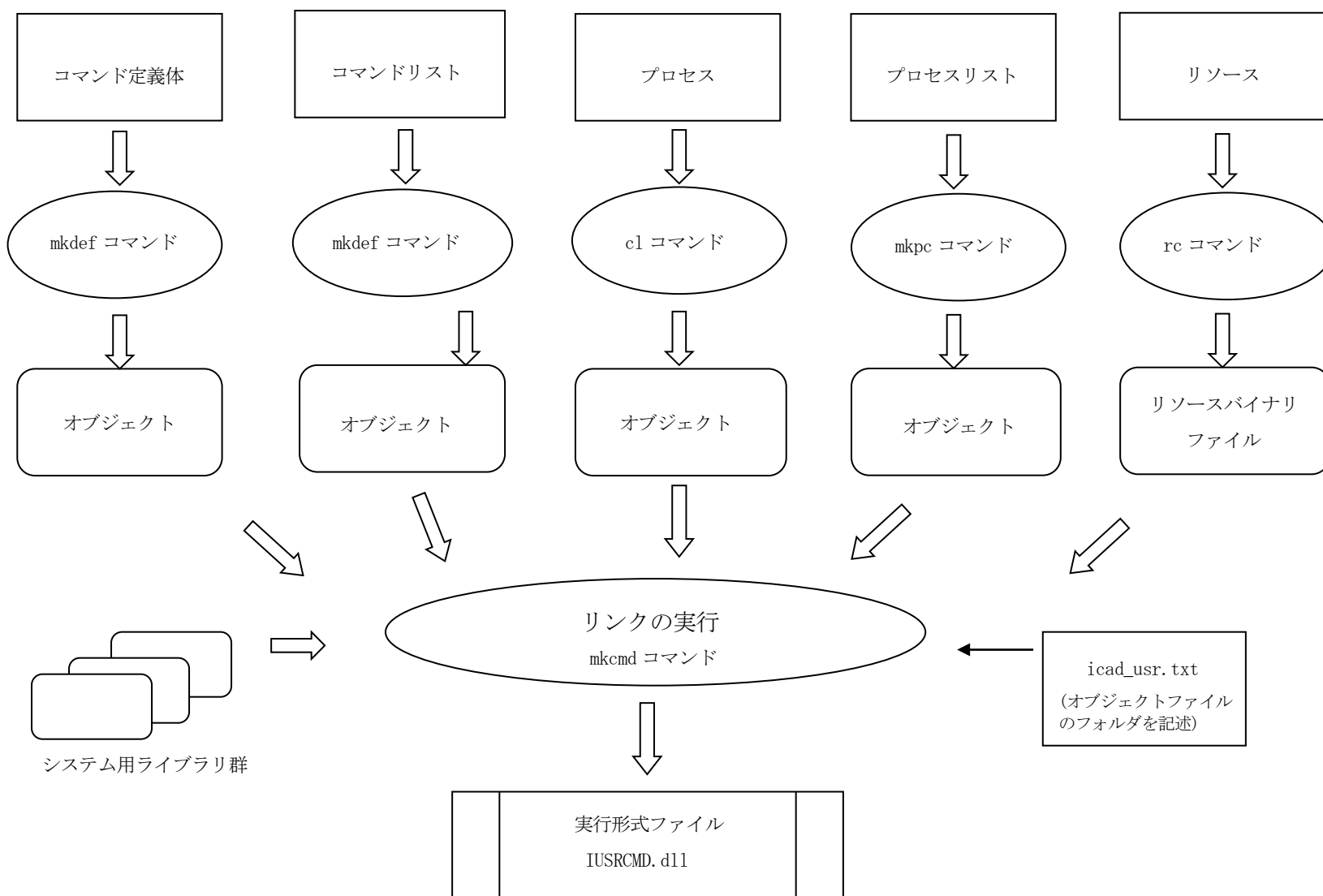
mkcmd -VS16

- ② %ICADDIR%\USER\BIN 配下に実行形式のファイル (IUSRCMD.d11) が作成されます。

IUSRCMD.d11 は、%ICADDIR%\USER\BIN 配下と%ICADDIR%\BIN 配下がありますが、%ICADDIR%\USER\BIN 配下の IUSRCMD.d11 が優先して参照されます。

%ICADDIR%\BIN 配下の IUSRCMD.d11 は削除、名前の変更等しないでください。削除、名前の変更等をおこなった場合、システムが正常に起動できません。

## コマンドの組み込み図



---

# 第 10 章 メッセージ

ここでは、メッセージ表示領域に表示させるメッセージの登録、出力方法について説明しています。

## メッセージの登録

コマンド実行中にエラーが発生した場合など、オペレータに対して通知するため、メッセージ表示領域にメッセージを表示させます。表示されたメッセージは正しいデータが入力された後、次のデータの入力待ちになった時に画面上から消去されます。

本システムでは、メッセージをファイルに登録し、プログラムから独立した方式を採用しています。メッセージの登録は、最初にメッセージファイルを作成し、それを入力データとしてインデックスファイルに登録します。

メッセージファイルには、制御情報行とメッセージ本文行があり、以下の形式にそって作成します。

-----1-----2-----3-----	
//nnnn, l, c, b, il, cml, d	⇐ 制御情報行
sw, ew, mb, ' MSG13001 入力データに規定範囲を超えた値が入力されました。'	⇐ メッセージ本文行

制御情報行は、「//」に続いて以下の制御情報を「,」で区切って指定します。

### 制御情報行の形式一覧

Nnnn (第 3～6 カラム)	メッセージ番号 1～19999 の下 4 桁を指定します。 上 1 桁は登録時 (mkmsg) に指定します。4 桁に満たない場合は 0 を詰めてください。 1～9999 は基本コマンドで使用するため、ユーザコマンドでは 10000～19999 が使用可能です。
L (第 8 カラム)	エラーレベルを指定します。 ここで指定されたレベルが imsgpt_ の出力引数に設定されます。 エラーのレベル =0 通知レベル =1 警告レベル(処理続行可能なレベル) =2 エラーレベル(処理続行不可能なレベル) =3 重大エラーレベル(処理続行不可能なレベル) =4 その他
C (第 10 カラム)	出力メッセージの色を指定します。 =1 赤 =2 緑 =3 白
B (第 12 カラム)	ベルの制御指定をします。 =0 ベルを鳴らしません。 =1 ベルを鳴らします。
Il (第 14～15 カラム)	追加情報の大きさをワード数(バイト数 / 4)で指定します。ワード数は、0～19 の範囲です。 2 桁に満たない場合は、空白を左に詰めてください。
cml (第 17～19 カラム)	出力するメッセージの行数を指定します。 必ず "1" を指定してください。(左 2 桁に空白を詰めます。)
D (第 21 カラム)	追加情報の指定をします。 =0 追加情報がありません。 =1 追加情報があります。 (メッセージ本文中に追加情報の出力フィールドがある場合のみ指定します。)

## メッセージ本文行の形式一覧

sw, ew (第 1～5 カラム)	メッセージ出力時の引数として指定された追加情報の、何ワード目から何ワード目までを追加情報として出力するか指定します。 追加情報がない場合は、sw, ew 共に 0 を指定します。 2 桁に満たない場合は、空白を左に詰めてください。
mb (第 7～8 カラム)	メッセージのバイト数(0～78)を指定します。 バイト数は、追加情報も含めたものとします。 2 桁に満たない場合は、空白を左に詰めてください。
メッセージ (第 10 カラム～)	固定の文字列を出力する場合、「' 」で文字列を囲んで記述します。 固定の文字列に加え、任意の追加情報を出力する場合、追加情報のワード数を nA4 で記述します。 A4 は 4 バイトの文字列、n はその倍数を意味します。例) 3A4 であれば 3*4=12 バイトの文字列

例を以下に示します。

- (1) 固定の文字列「MSG13001 入力データに規定範囲を超えた値が入力されました。」を出力する場合

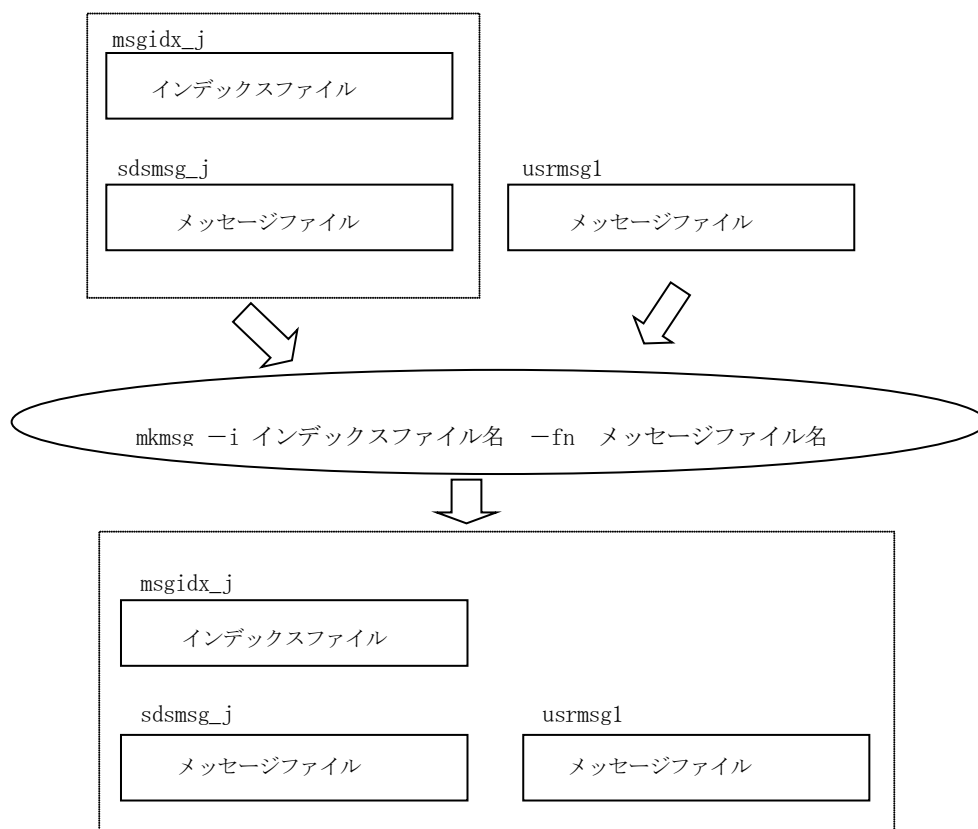
```
//3001, 2, 1, 0, 0, 1, 0
0, 0, 58, ' MSG13001 入力データに規定範囲を超えた値が入力されました。'
```

- (2) 固定の文字列に加え、任意の追加情報「負値－1 0 0」を出力する場合

```
//3001, 2, 1, 0, 3, 1, 1
1, 3, 70, ' MSG13001 入力データに規定範囲を超えた値が入力されました。', 3A4
```

「負値－1 0 0」はにて指定します。

作成したメッセージファイル(usrmsg1)はmkmsg コマンドによりインデックスファイルに登録します。  
インデックスファイル(msgidx\_j) 及びメッセージファイル(sdsmsg\_j) はシステムであらかじめ用意されています。





## mkmsg コマンド

メッセージファイルを登録します

mkmsg -i インデックスファイル名 -f1 メッセージファイル名

## 説明

-i	必ず -i と指定します。
インデックスファイル名	インデックスファイル名 (msgidx_j) を指定します。
-f1	必ず -f1 と指定します。
メッセージファイル名	メッセージファイル名 (usrmsg1) を指定します。

① %ICADDIR%¥msg 配下の msgidx\_j をバックアップします。

```
copy msgidx_j msgidx_j.old
```

② コマンドプロンプトより以下のコマンドを実行します。

```
cd %ICADDIR%¥msg
```

```
mkmsg -i msgidx_j -f1 usrmsg1
```

③ カレントディレクトリにインデックスファイル(usridx\_j)が作成されます。

## メッセージの出力方法

作成したメッセージは、必要性が生じた箇所にて出力します。

出力の方法には、次の3つがあります。コマンド開発者はどの方法でメッセージを出力するかを、必要に応じて決定してください。

以下に例を示します。

- (1) プロセスパラメタ `ircode` (復帰情報) によりメッセージを出力します。この場合、追加情報は付加できません。

```
long int code[2], ircode[3];
if( code[1] != 1 && code[1] != 2 ){
    ircode[0] = 8;
    ircode[1] = 13001;
}
```

プロセスパラメタ `ircode[1]` でメッセージ番号を指定します。

注意) `ircode[0]=4, 8, 12` の時、メッセージが出力されます。

- (2) `imsgpt_` を利用してメッセージを出力します。この場合、追加情報が付加できます。

```
long int msgno=13001;
char    insmsg[4];
long int n=0;
long int msglvl, ircode;
imsgpt_( &msgno, insmsg, &n, &msglvl, &ircode );
```

- (3) `imwrit_` を使用してメッセージを出力します。この場合、メッセージファイルは利用しません。

```
static char wrtdat[59]=" MSG13001 入力データに規定範囲を超えた値が入力されました。";
long int    lwrtdt=58;
long int    color=1;
long int    ircode;
imwrit_(wrtdat, &lwrtdt, &color, &ircode);
```

指定されたメッセージはメッセージ表示領域に表示されます。

---

# 第 1 1 章 バッチプログラムの作成

ここでは、複数の図面に対して一括処理する等のバッチプログラム（たとえば、記号の配置など）を作る人のために、必要な知識について説明しています。

## バッチプログラムの作成手順



バッチプログラムの作成手順は、`%ICADDIR%\usr\process\src\ubatch.c` 及び、「第6章 図形処理ライブラリの利用方法」を参照してください。

### ■ 使用可能なライブラリー一覧

バッチプログラム内で使用可能なライブラリは、『SDKリファレンスガイド』の「付録：バッチプログラムで使用可能なライブラリ」を参照してください。

### ■ 注意事項

バッチプログラムを作成する際は、以下の注意事項に従ってください。

- (1) バッチプログラムの最初に、`sbinit_` を呼び出す必要があります。
- (2) バッチプログラムの最後に、`sbterm_` を呼び出す必要があります。

ただし、`sbinit_` が正常終了しなかった場合は、呼び出してはいけません。

- (3) バッチプログラムをリンクするリンク環境と、起動する環境のバージョン／レベルは同じでなければなりません。このため、バッチプログラム作成後にレベルアップを行う場合は、レベルアップ後の環境にて再リンクする必要があります。

## コンパイル、リンク



作成したバッチプログラムのコンパイルは、「第9章 コンパイル、リンク」「プロセスのコンパイル」を参照してください。

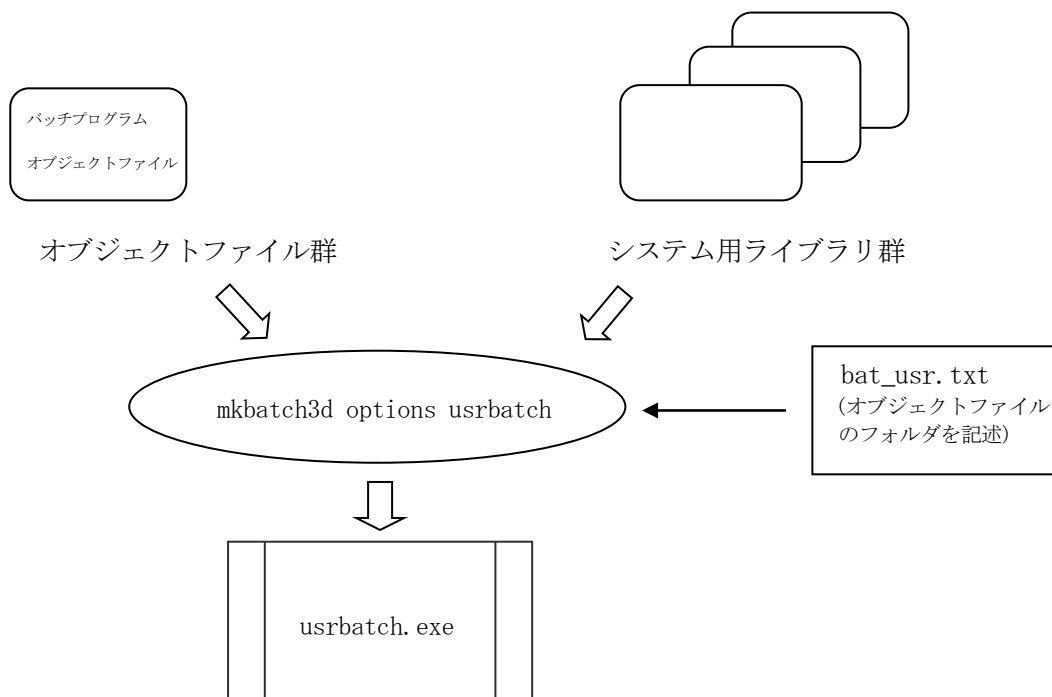
コンパイルしたバッチプログラムをシステムとリンクします。

リンク用のコマンドはシステムで用意されており、このコマンドを実行します。

作成したオブジェクトファイルが存在するフォルダおよびファイル名を「bat\_usr.txt」に記述します。

```
; オブジェクト、ライブラリ  
.. %usr%process%obj%*.obj ————— バッチプログラム  
;
```

作成したオブジェクトファイルをシステムとリンクし、コマンドに組み込む方法を以下に示します。



## mkbatch3dコマンド

バッチプログラムのリンクをします。

```
mkbatch3d options usrbatch
```

### 説明

options           ご利用の「Visual Studio」のバージョンを指定します。

Visual Studioのバージョン	options
Visual Studio 2015	-VS14
Visual Studio 2017	-VS15
Visual Studio 2019	-VS16

usrbatch           バッチプログラムの実行ファイルを指定します。

例：「Visual Studio 2019」をご利用の場合

- ① コマンドプロンプトより以下のコマンドを実行します。

```
mkbatch3d -VS16 usrbatch
```

- ② %ICADDIR%\bin配下に実行形式のファイル（usrbatch.exe）が生成されます。

## 実行

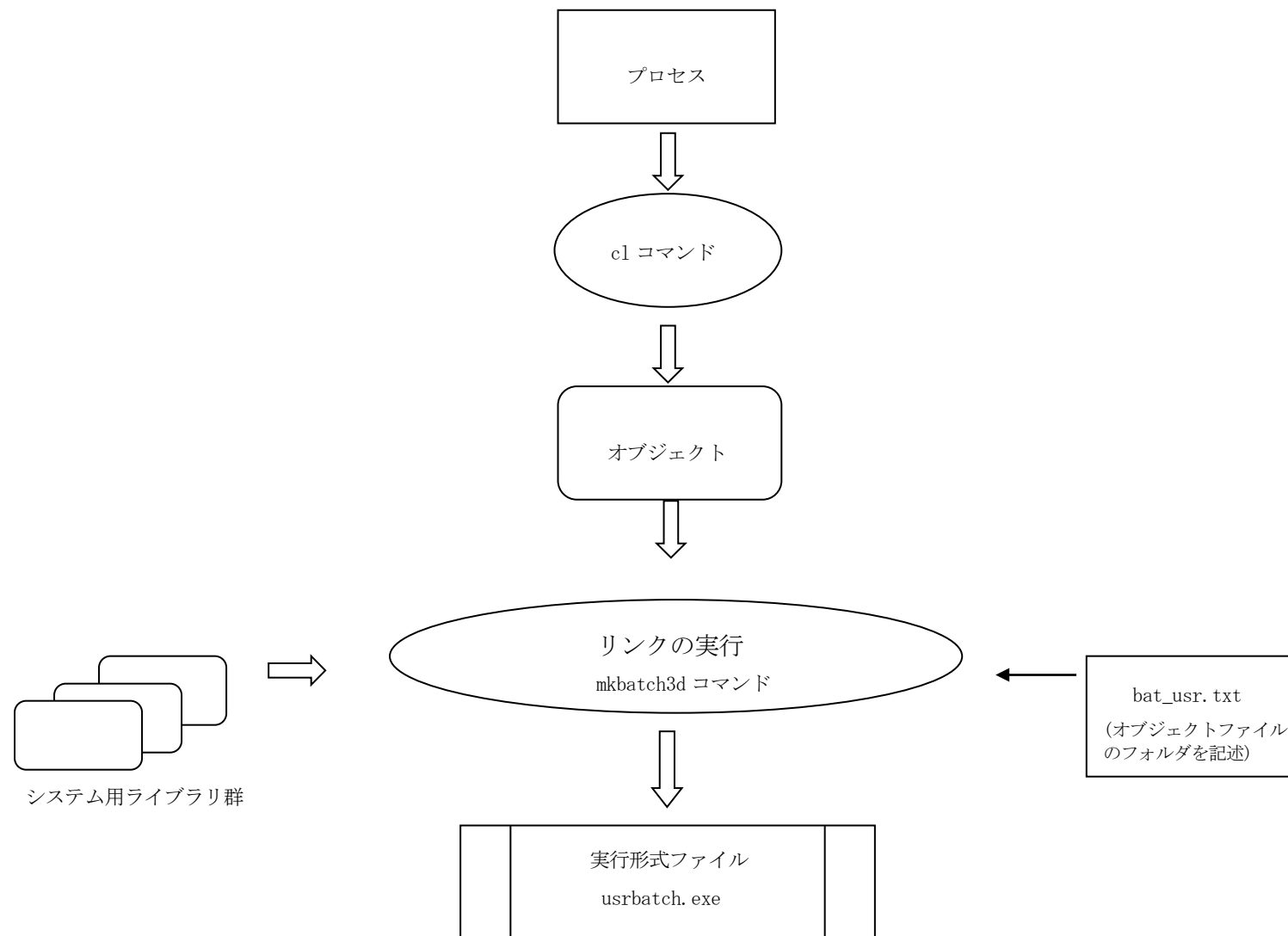
バッチプログラムの実行方法を説明します。

ここでは、実行形式のファイル名をusrbatch.exeとして説明します。

1. エクスプローラを起動します。
2. バッチプログラムの実行形式のファイルが格納されているフォルダ(%ICADDIR%\bin)を開きます。
3. usrbatch.exeをダブルクリックします。

バッチプログラムが実行されます。

## バッチプログラムの組み込み図



---

## 第 1 2 章 トラブルシューティング

ここでは、コマンド開発機能を使い、ユーザコマンドを作成、実行したときに発生する問題に対処する方法について説明しています。

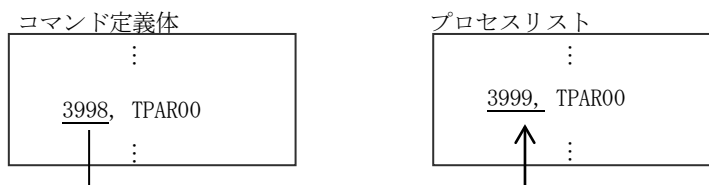


## ユーザコマンドが正常に動作しない時の原因

ユーザコマンドが組込まれているにも関わらずコマンド開発者が意図する動作をしない場合には、主に以下の様な原因が考えられます。

- (1) ユーザコマンドを入力したときに「MSG00017 コマンド名またはキーワードが不当です。」のメッセージが表示される場合  
(原因)
  - ・ コマンドリストに記述されているユーザコマンド名に誤りがあります。
- (2) ユーザコマンドを入力したときに「MSG02407 指定プロセスが存在しません。」のメッセージが表示される場合  
(原因)
  - ・ コマンド定義体に記述されているプロセス番号とプロセスリストに記述されているプロセス番号が違います。

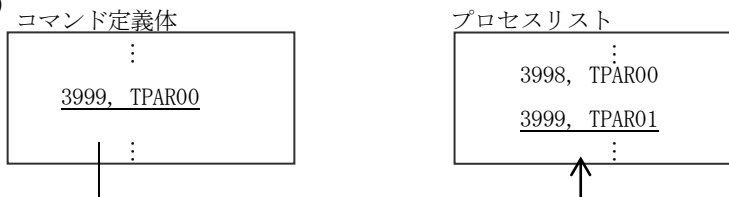
(例)



コマンド定義体とプロセスリストはプロセス番号でリンクしていますが、プロセスリストにはプロセス番号「3998」が存在しません。

- (3) コマンド開発者が意図するプロセスと違うプロセスが動く場合  
(原因)
  - ・ コマンド定義体に記述されているプロセス名とプロセスリストに記述されているプロセス名が違います。

(例)



コマンド定義体とプロセスリストはプロセス番号でリンクするため、プロセス名が違っててもプロセス番号「3999」で指定されたプロセスが動きます。

- (4) ユーザコマンドがコマンド開発者の意図するように動作しない場合  
(原因)
  - ・ プロセス(設計、プログラミング)にミスがあります。

## プロセスにミスがあった場合の原因追求方法

ユーザコマンドが正常に動作しない原因がプロセスにある場合には、デバッガを使ってプロセス内の動きを確認していきます。

プロセスをデバッガより参照するためには、あらかじめプロセスのコンパイル、リンク時に、シンボリックデバッグ情報を付加しておく必要があります。

### (1) プロセスのコンパイル

コマンドプロンプトより以下のコマンドを実行します。

#### cl コマンド

プロセスからシンボリックデバッグ情報付きのオブジェクトファイルを作成します。

cl -c -Zi ファイル名(プロセス)

#### 説明

-c           必ず -c と指定します。(コンパイルだけ実行します。)

-Zi           シンボリックデバッグ情報付きのオブジェクトが作成されます。

### (2) リンク

シンボリックデバッグ情報付きの IUSRCMD.dll を作成します。

#### ① デバッグオプションファイル「icad\_dbg.txt」を編集します。

```
⋮  
;デバッグオプションリスト (デバッグしない場合は、以下の行の先頭にセミコロンを付けてください)  
/debug       ..... コメント文を実行文に変更します。(行の先頭のセミコロンを削除します)
```

#### ② リンクをします。

リンク方法については、本マニュアルの「第9章 コンパイル、リンク」を参照してください。

リンク実行後、%ICADDIR%\¥USER¥BIN 配下に IUSRCMD.dll が作成されます。

### (3) デバッガを使ってシステムを起動します。

デバッガを使ってシステムを起動する方法について説明します。

コマンドプロンプトより以下のコマンドを実行します。

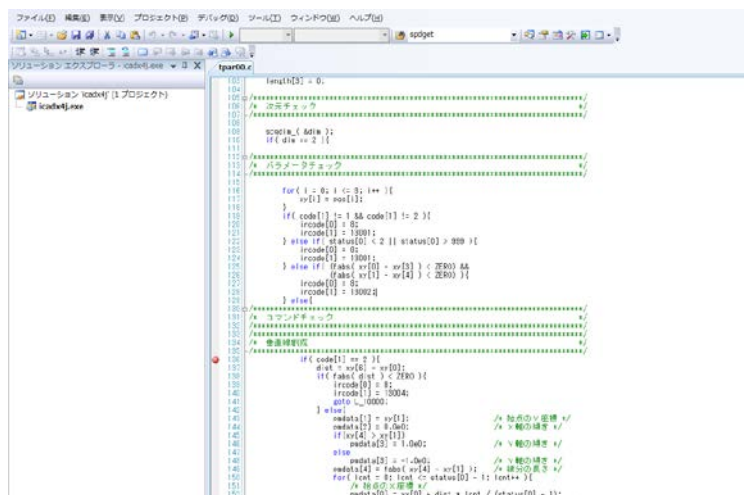
```
devenv icadx4j.exe
```

デバッガの使用方法につきましては、「Visual Studio」のヘルプ等を参照してください。

#### (4) デバッガの実行方法

「Visual Studio」を例にデバッガを使ってプロセスを参照する方法について説明します。  
詳細は、「Visual Studio」のヘルプ等を参照してください。

- ① プロセスを開きます。
- ② プロセス内で参照したい行にカーソルを移動します。
- ③ テキストエディタウインドウのポップアップメニューの[ブレークポイント(B)]を選択し  
[ブレークポイントの挿入(R)]を選択します。カーソルがあった位置に赤色の丸が表示されます。
- ④ [F5]キーを押すとシステムが起動します。



- ④ ユーザコマンドを実行し、ブレークポイントに到達するとデバッガがアクティブになります。  
プロセス内の動きを確認していくことで、ユーザコマンドが正常に動作しない原因を追求していきます。

